30th International Conference on Types for Proofs and Programs

TYPES 2024, June 10–14, 2024, Copenhagen, Denmark

Edited by

Rasmus Ejlers Møgelberg
Benno van den Berg

LIPICS

Editors

Rasmus Ejlers Møgelberg
IT University of Copenhagen, Denmark
mogel@itu.dk

Benno van den Berg
University of Amsterdam, The Netherlands
b.vandenberg3@uva.nl

ACM Classification 2012
Theory of computation → Categorical semantics; Theory of computation → Constructive mathematics; Theory of computation → Denotational semantics; Theory of computation → Type theory; Theory of computation → Lambda calculus; Software and its engineering → Functional languages

ISBN 978-3-95977-376-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-376-8.

Publication date
July, 2025

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.TYPES.2024.0

ISBN 978-3-95977-376-8 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

To the memory of Peter Aczel and Thomas Streicher.

Contents

Papers

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The TYPES meetings are a forum to present new and ongoing work in all aspects of type theory and its applications, especially in formalized and computer assisted reasoning and computer programming. This volume constitutes the post-proceedings of the 30th International Conference on Types for Proofs and Programs, TYPES 2024, that was held at the IT University of Copenhagen, Denmark, from 10 to 14 June 2024.

The meetings from 1990 to 2008 were annual workshops corresponding to five consecutive EU-funded networking projects. Since 2009, TYPES has been run as an independent conference series. Previous TYPES meetings were organised by Antibes (1990), Edinburgh (1991), Båstad (1992), Nijmegen (1993), Båstad (1994), Torino (1995), Aussois (1996), Kloster Irsee (1998), Lökeberg (1999), Durham (2000), Berg en Dal near Nijmegen (2002), Torino (2003), Jouy-en-Josas near Paris (2004), Nottingham (2006), Cividale del Friuli (2007), Torino (2008), Aussois (2009), Warsaw (2010), Bergen (2011), Toulouse (2013), Paris (2014), Tallinn (2015), Novi Sad (2016), Budapest (2017), Braga (2018), Oslo (2019), Turin (2020), Leiden (2021), Nantes (2022), Valencia (2023). The 2020 and 2021 editions were virtual, because of the SARSCoV-2 pandemics.

The TYPES areas of interest include, but are not limited to: Foundations of type theory and constructive mathematics; Homotopy type theory; Applications of type theory; Dependently typed programming; Industrial uses of type theory technology; Meta-theoretic studies of type systems; Proof assistants and proof technology; Automation in computer-assisted reasoning; Links between type theory and functional programming; Formalizing mathematics using type theory; Type theory in linguistics.

The TYPES conferences are all based on contributed talks based on short abstracts; reporting work in progress and work presented or published elsewhere. A post-proceedings volume is prepared after the conference, whose papers must represent unpublished work. Submitted papers to the post-proceedings are subject to a full peer-review process.

The conference program of TYPES 2024 consisted of 57 contributed talks, and five invited talks by Brigitte Pientka (McGill University, Canada), Egbert Rijke (University of Ljubljana, Slovenia), Talia Ringer (University of Illinois at Urbana-Champaign, USA), Nicola Gambino (University of Manchester, UK), and Michael Rathjen (University of Leeds, UK). The last two of these constituted a special session in memory of Peter Aczel, who passed away in August 2023. This volume is dedicated to the memory of him, and of Thomas Streicher, who passed away in January 2024.

The conference was a successful event with 109 registered participants. All the details of the conference can be found at `https://types2024.itu.dk/`.

For the post-proceedings, 13 papers were initially submitted, out of which 9 were accepted. We thank all the authors and reviewers for their hard work to make this possible!

Benno van den Berg and Rasmus Ejlers Møgelberg, May 2025.

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Thorsten Altenkirch (7)
University of Nottingham, UK

Felix Cherubini (3)
University of Gothenburg, Sweden;
Chalmers University of Technology,
Gothenburg, Sweden

Thierry Coquand (3)
University of Gothenburg, Sweden;
Chalmers University of Technology,
Gothenburg, Sweden

Tom de Jong (1)
School of Computer Science,
University of Nottingham, UK

Freek Geerligs (3)
University of Gothenburg, Sweden;
Chalmers University of Technology,
Gothenburg, Sweden

Philipp Joram (6)
Department of Software Science,
Tallinn University of Technology, Estonia

Neel Krishnaswami (5)
University of Cambridge, UK

Adrienne Lancelot (4)
Inria & LIX, École Polytechnique, UMR 7161,
Palaiseau, France; Université Paris Cité, CNRS,
IRIF, F-75013, Paris, France

Dominique Larchey-Wendling (2)
Université de Lorraine, CNRS, LORIA,
F-54000 Nancy, France

Meven Lennon-Bertrand (5)
University of Cambridge, UK

Daniel R. Licata (9)
Dept. of Mathematics & Computer Science,
Wesleyan University, Middletown, CT, USA

Hugo Moeneclaey (3)
University of Gothenburg, Sweden;
Chalmers University of Technology,
Gothenburg, Sweden

Jacob Neumann (7)
University of Nottingham, UK

Robert Rose (9)
Dept. of Mathematics & Computer Science,
Wesleyan University, Middletown, CT, USA

Matthew Sirman (5)
University of Cambridge, UK

Shinichiro Tanaka (8)
Institute for Logic, Language and Computation,
University of Amsterdam, The Netherlands

Niccolò Veltri (6)
Department of Software Science,
Tallinn University of Technology, Estonia

Formalizing Equivalences Without Tears

Tom de Jong ✉ 🏠

School of Computer Science, University of Nottingham, UK

Abstract

This expository note describes two convenient techniques in the context of homotopy type theory for proving – and formalizing – that a given map is an equivalence. The first technique decomposes the map as a series of basic equivalences, while the second refines this approach using the 3-for-2 property of equivalences. The techniques are illustrated by proving a basic result in synthetic homotopy theory.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases 3-for-2 property, 2-out-of-3 property, definitional equality, equivalence, formalization of mathematics, synthetic homotopy theory, type theory

Digital Object Identifier 10.4230/LIPIcs.TYPES.2024.1

Funding This work was supported by The Royal Society (grant reference URF\R1\191055). The HIM programme was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813.

Acknowledgements I am grateful to Josh Chen, Stefania Damato, Martín Escardó, Nicolai Kraus, Fredrik Nordvall Forsberg, Stiéphen Pradal and Jon Sterling for comments on this note. I also thank the participants, organizers and support staff of the HIM trimester programme *Prospects of Formal Mathematics* for the opportunity to present the ideas in this note. Finally, I thank the anonymous reviewers for their helpful suggestions.

1 Introduction

A very common problem in *homotopy type theory (HoTT)* [8] is to prove that a given map is an equivalence. The purpose of this short note is to describe convenient techniques for doing this, in particular when one is interested in formalizing the argument in a proof assistant. I claim no originality in the results of this note. Indeed, the technique I wish to highlight already informs much of the AGDA-UNIMATH library developed by Rijke and contributors [5], while I picked up the other (decomposition) technique in this note via the Agda development TypeTopology of Escardó and collaborators [1] as well as Escardó's comprehensive introduction to univalent foundations and its formalization in Agda [2]. Rather, I hope that this note will contribute to a greater awareness of these techniques, especially among junior type theorists.

Outline

The note is outlined as follows. Section 2 explains why directly proving that a map is an equivalence is often cumbersome. Section 3 describes an alternative technique by decomposing the given map into a sequence of (smaller) equivalences, while Section 4 further refines this method using the fact that equivalences satisfy the *3-for-2 property*. These techniques are then illustrated in Section 5 in the context of *synthetic homotopy theory* [8, §8]. The example is self-contained and no prior knowledge of this area is required, although someone familiar with classical homotopy theory may find its simplicity appealing and consider it an invitation to learn more (see e.g. [7]).

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg; Article No. 1; pp. 1:1–1:6
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Terminology

The 3-for-2 property of equivalences states that if any two maps in a commutative triangle are equivalences, then so is the third. This property is perhaps more commonly known as the *2-out-of-3 property*. André Joyal proposed the name 3-for-2 by analogy to how discounts are often advertised; if you prove two maps are equivalences, then the third is for free [4]. Since we are interested in reducing the amount of (formalization) work and wish to get as much as possible for free, the analogy is quite apt and we prefer Joyal's terminology in this note.

Foundations

As mentioned at the very start, this note is concerned with equivalences in homotopy type theory, although the issues and techniques described should carry over to other intensional type theories [3] and classes of maps that satisfy 3-for-2.[1] We mostly adopt the terminology and notation of the HoTT Book [8], e.g. writing $\equiv$ for judgemental (definitional) equality, using $=$ for identity types (sometimes known as propositional equality), and $\sim$ for homotopies (i.e. pointwise identities).

2 A naive approach

Presented with the problem of showing that a map $f : A \to B$ is an equivalence,[2] a direct approach would be to try and construct a map $g : B \to A$ together with identifications $g \circ f \sim \mathrm{id}_A$ and $f \circ g \sim \mathrm{id}_B$. What makes this approach infeasible at times is that the construction of g may be involved, resulting in nontrivial computations (often involving *transport*) when showing that the round trips are homotopic to the identity maps, especially in proof-relevant settings such as HoTT.

In some cases we are lucky and the desired identifications hold definitionally, in which case the proof assistant can simply do the work for us by unfolding definitions. We return to using definitional equalities to our advantage in Section 4.

3 Decomposition into equivalences

Instead of directly arguing that a given map $f : A \to B$ is an equivalence, it is often convenient to instead decompose f as a series of "building block equivalences", general maps that we already know to be equivalences, as depicted below.

$$\begin{tikzcd}
A \arrow[rrrrr, "f"] \arrow[dr, bend right, "\simeq"] & & & & & B \\
& X_1 \arrow[r, "\simeq"] & X_2 \arrow[r, "\simeq"] & \dots \arrow[r, "\simeq"] & X_n \arrow[ur, bend right, "\simeq"'] &
\end{tikzcd} \tag{1}$$

Some quintessential examples of such building block equivalences are as follows.

[1] However, I don't have a good illustration to hand of applying the techniques to a class of maps other than the equivalences. The class of n-equivalences, i.e. those maps whose n-truncation is an equivalence, comes to mind, but there I have found it easier to work with the fact that these maps can be characterized as those maps for which precomposition into n-types is an equivalence [6, Lem. 2.9] (see also [5, `k-equivalences`]). Moreover, while equivalences are invertible, maps satisfying 3-for-2 need not be of course, which means that it may be more difficult to arrange a diagram like in (2).

[2] The precise definition of an equivalence is somewhat subtle, as discussed at length in [8, §4], but it is not too important for our purposes.

Projection from the total space of a contractible family. Given a type X and a dependent type Y over it, if each $Y(x)$ is contractible (i.e. it is equivalent to the unit type), then the projection map $\mathrm{pr}_1 : \sum(x : X)\, Y(x) \to X$ is an equivalence. In fact, this is an equivalence if and only if each $Y(x)$ is contractible.

Contractibility of singletons. For any type X and $x : X$, the type $\sum(y : X)\, x = y$ is contractible and hence the two projection maps from $\sum(x : X) \sum(y : X)\, x = y$ to X are equivalences.

Associativity of dependent sums. For a type A, and dependent types $B(a)$ and $C(a, b)$ over A and $\sum(a : A)\, B(a)$, respectively, the map

$$\begin{array}{rcl} \sum(a : A) \sum(b : B(a))\, C(a, b) & \to & \sum\Big(p : \sum(a : A)\, B(a)\Big)\, C(p) \\ (a, b, c) & \mapsto & ((a, b), c) \end{array}$$

is an equivalence.

Reindexing dependent sums along an equivalence. Given an equivalence between types $f : A \simeq B$ and a dependent type $Y(b)$ over B, the assignment

$$\begin{array}{rcl} \sum(a : A)\, Y(f(a)) & \to & \sum(b : B)\, Y(b) \\ (a, y) & \mapsto & (f(a), y) \end{array}$$

is an equivalence. And of course there is a similar result for dependent products.

Distributivity of $\prod$ over $\sum$. Given a type A and dependent types B and Y over A and $\sum(a : A)\, B(a)$, respectively, the map

$$\begin{array}{rcl} \prod(a : A) \sum(b : B(a))\, Y(a, b) & \to & \sum\Big(f : \prod(a : A)\, B(a)\Big) \prod(a : A)\, Y(a, f(a)) \\ \alpha & \mapsto & (\lambda a.\, \mathrm{pr}_1(\alpha(a)), \lambda a.\, \mathrm{pr}_2(\alpha(a))) \end{array}$$

is an equivalence with inverse $(f, p) \mapsto \lambda a.\, (f(a), p(a))$.

This, and especially its nondependent version

$$\prod(a : A) \sum(b : B)\, Y(a, b) \to \sum(f : A \to B) \prod(a : A)\, Y(a, f(a)),$$

is traditionally called the "type theoretic axiom of choice", but this is a misnomer as there is no choice involved, see also [8, pp. 32 and 104].

Distributivity of $\sum$ over $+$. Given a type A and dependent types X and Y over it, the map

$$\begin{array}{rcl} \sum(a : A)\, (X(a) + Y(a)) & \to & \sum(a : A)\, X(a) + \sum(a : A)\, Y(a) \\ (a, \mathrm{inl}\, x) & \mapsto & \mathrm{inl}(a, x) \\ (a, \mathrm{inr}\, y) & \mapsto & \mathrm{inr}(a, y) \end{array}$$

is an equivalence.

Congruence of type formers. All type formers respect equivalences. For example, if we have $f : A \simeq X$ and $g : B \simeq Y$, then $(A + B) \simeq (X + Y)$ by applying f to the elements on the left and g to the elements on the right.

Composition with a fixed path. Given elements x, y and z of a type X and a path $p_0 : x = y$, the path composition maps

$$\begin{array}{rcl} (z = x) & \to & (z = y) \\ p & \mapsto & p \cdot p_0 \end{array} \qquad \text{and} \qquad \begin{array}{rcl} (x = z) & \to & (y = z) \\ p & \mapsto & p_0 \cdot p \end{array}$$

are equivalences.

This list is not exhaustive, but should give a good impression of the available building blocks. The interested reader may find many more examples in [1, `UF.EquivalenceExamples`].

The idea is to reduce the task of proving that f is an equivalence to identifying suitable building blocks – the equivalences in (1) – *and proving that the diagram* (1) *commutes.* It is the latter point that may pose similar difficulties to those explained in the previous section: the commutativity proof could involve nontrivial computations. We turn to a refinement in the next section to address this.

We should mention that this technique is still very valuable, especially when we are not interested in having a particular equivalence, or when there is a unique such equivalence, e.g. when proving that a type X is contractible.

4 A refinement using 3-for-2

To address the issue identified above, we will make use of the fact that equivalences satisfy the 3-for-2 property:

▶ **Lemma 1** (3-for-2 for equivalences, [8, Thm. 4.7.1])**.** *In a commutative triangle*

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

if two of the maps are equivalences, then so is the third.

Rather than decomposing f, the idea is to simply involve f into *any* commutative diagram where all (other) maps are equivalences, as depicted below.

$$\begin{tikzcd} X_1 \arrow[r, "\simeq"] \arrow[rrrrrrrrr, bend right, "\simeq"'] & X_2 \arrow[r, "\simeq"] & \dots \arrow[r, "\simeq"] & X_i \arrow[r, "\simeq"] & A \arrow[r, "f"] & B \arrow[r, "\simeq"] & X_{i+1} \arrow[r, "\simeq"] & \dots \arrow[r, "\simeq"] & X_n \end{tikzcd} \tag{2}$$

By the 3-for-2 property, we can still conclude that f is an equivalence from (2), but verifying the commutativity may now be easier, especially when the type X_n is simpler than B. Indeed, we can often ensure this in practice as illustrated and commented on at the end of the next section.

5 An example in synthetic homotopy theory

Usually [8, §7.5], a type is said to be *n-connected* if its n-truncation is contractible. For us, the following characterization may serve as a definition.

▶ **Proposition 2** ([8, Cor. 7.5.9])**.** *A type A is n-connected if and only if for every n-type*[3] *B, the constants map*

$$\begin{aligned} B &\to (A \to B) \\ b &\mapsto \lambda a.\, b \end{aligned}$$

is an equivalence.

[3] We recall from [8, §7.1] that the n-types are inductively defined for $n \geq -2$: a -2-type is a contractible type and an $(n+1)$-type is a type whose identity types are n-types.

We will illustrate the above techniques by giving a slick proof of a well-known result (see e.g. [8, Thm. 8.2.1]) in homotopy theory: taking the suspension of a type increases its connectedness by one. The reader may wish to compare the proof below to that given in *op. cit*, or inspects its formalization as part of the AGDA-UNIMATH library [5, `Suspensions increase connectedness`].

For completeness, we recall suspensions in homotopy type theory.

▶ **Definition 3** (Suspension Σ). *The* suspension $\Sigma\, A$ *of a type A is the pushout of the span* $\mathbf{1} \leftarrow A \rightarrow \mathbf{1}$*. Equivalently, it is the higher inductive type generated by two point constructors* $\mathrm{N}, \mathrm{S} : \Sigma\, A$ *(short for* North *and* South*) and a path constructor* $\mathrm{merid} : A \to \mathrm{N} = \mathrm{S}$ *(short for* meridian*).*

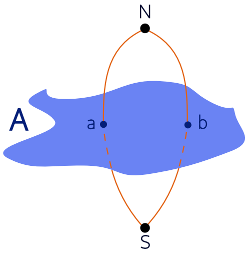

Figure 1 Illustration of the suspension of a type A. The lines from N to S going through the points $a : A$ and $b : A$ represent the paths $\mathrm{merid}(a)$ and $\mathrm{merid}(b)$, respectively.

We shall only need the following basic fact about suspensions which is just the universal property of the suspension as a pushout. (We recall from [8, §2.2] that ap_g denotes the action of g on the identity types.)

▶ **Proposition 4** (Universal property of the suspension, [8, Exer. 6.11]).
The map

$$\begin{aligned} (\Sigma\, A \to B) &\quad\to\quad \sum(b_N : B) \sum(b_S : B)\,(A \to b_N = b_S) \\ g &\quad\mapsto\quad \big(g(\mathrm{N}), g(\mathrm{S}), \lambda a.\, \mathrm{ap}_g(\mathrm{merid}(a))\big) \end{aligned}$$

is an equivalence for all types A and B.

▶ **Theorem 5.** *The suspension of an n-connected type is $(n+1)$-connected.*

Proof. Let A be an n-connected type and B an $(n+1)$-type. By Proposition 2, we need to show that the constants map $\mathrm{consts} : B \to (\Sigma\, A \to B)$ is an equivalence. We first consider the evaluation map

$$\mathrm{eval} : (\Sigma\, A \to B) \to B \quad \text{defined by} \quad \mathrm{eval}(g) :\equiv g(\mathrm{N}),$$

and note that the diagram

$$\begin{tikzcd} B \arrow[r, "\mathrm{consts}"] \arrow[rr, bend right, "\mathrm{id}_B"'] & (\Sigma\, A \to B) \arrow[r, "\mathrm{eval}"] & B \end{tikzcd}$$

commutes definitionally. Hence, by **3-for-2**, it suffices to prove that eval is an equivalence.

We use the **decomposition** technique to do so and consider the following diagram.

$$\begin{tikzcd}
(\Sigma\, A \to B) \arrow[rrr, "\mathrm{eval}"] \arrow[rd, "g \mapsto (g(N), g(S), \dots)", "\simeq"'] & & & B \\
& \sum(b_N : B) \sum(b_S : B)\, (A \to b_N = b_S) \arrow[r, bend right, "\simeq", "(b_N, b_S, \varphi) \mapsto (b_N, b_S, \dots)"'] & \sum(b_N : B) \sum(b_S : B)\, (b_N = b_S) \arrow[ru, "(b_N, b_S, \dots) \mapsto b_N", "\simeq"'] &
\end{tikzcd} \tag{3}$$

Note that because A is n-connected and $b_N = b_S$ is an n-type (since B is an $(n+1)$-type), the constants map $(b_N = b_S) \to (A \to b_N = b_S)$ is an equivalence. The middle equivalence in the diagram (3) is induced by the inverse of this map. It should be stressed that the definitional behaviour of this inverse is completely irrelevant for verifying the commutativity of the diagram (3); we only need to know the middle map's behaviour on the first two components of the Σ-types to see that the diagram commutes definitionally.

Finally, the first map in the decomposition of eval is an equivalence by Proposition 4 and the last map is an equivalence by **contractibility of singletons** (from page 3). ◀

We end this note by pointing out an important **heuristic** that is nicely illustrated by the above proof: *We always try to orient the maps in our diagrams towards the simplest possible type* – which in (3) is clearly B. The reason for this is that checking commutativity should then be the easiest as there is relatively little data in the target type to compare. Similarly, we defined a section of eval, rather than directly defining a map $B \to (\Sigma\, A \to B)$ and proving it's an inverse to eval which would involve the cumbersome task of comparing functions $\Sigma\, A \to B$ for equality.

References

1 Martín H. Escardó and contributors. TypeTopology. `http://www.cs.bham.ac.uk/~mhe/TypeTopology/index.html`. Agda development. URL: `https://github.com/martinescardo/TypeTopology`.

2 Martín Hötzel Escardó. Introduction to univalent foundations of mathematics with Agda, 2019. `arXiv:1911.00580`.

3 Martin Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, University of Edinburgh, 1995. Published in Springer's *Distinguished Dissertations* series in 1997. `doi:10.1007/978-1-4471-0963-1`.

4 André Joyal. Categorical aspects of type theory, 2013. Talk in the *Progress in Higher Categories* session at the *Canadian Mathematical Society (CMS) Summer Meeting* in Halifax. URL: `https://home.sandiego.edu/~shulman/cmshighercategories2013/Joyal.pdf`.

5 Egbert Rijke, Elisabeth Bonnevier, Jonathan Prieto-Cubides, Fredrik Bakke, and others. The agda-unimath library. URL: `https://unimath.github.io/agda-unimath/`.

6 Egbert Rijke, J. Daniel Christensen, Luis Scoccola, and Morgan Opie. Localization in homotopy type theory. *Higher Structures*, 4(1):1–32, 2020. `doi:10.21136/hs.2020.01`.

7 Michael Shulman. Homotopy type theory: The logic of space. In Mathieu Anel and Gabriel Catren, editors, *New Spaces in Mathematics*, pages 322–404. Cambridge University Press, 2021. `doi:10.1017/9781108854429.009`.

8 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

Constructive Substitutes for Kőnig's Lemma

Dominique Larchey-Wendling
Université de Lorraine, CNRS, LORIA, F-54000 Nancy, France

Abstract

We propose weaker but constructively provable variants of the contrapositive of Kőnig's lemma. We derive those from a generalization of the FAN theorem for inductive bars to inductive covers, for which we give a concise proof. We compare the positive, negative and sequential characterizations of covers and bars in classical and constructive contexts, giving precise accounts of the role played by the axioms of excluded middle and dependent choice. As an application, we discuss some examples where the use of Kőnig's lemma can be replaced by one of our weaker variants to obtain fully constructive accounts of results or proofs that could otherwise appear as inherently classical.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Constructive mathematics

Keywords and phrases Kőnig's lemma, FAN theorem, constructive mathematics, inductive covers, inductive bars, almost full relations, inductive type theory, Coq

Digital Object Identifier 10.4230/LIPIcs.TYPES.2024.2

Supplementary Material *Software*: `https://github.com/DmxLarchey/Constructive-Konig` [19]
archived at `swh:1:dir:611c848b0dbc19c9de20744122776440c00e413d`

Funding *Dominique Larchey-Wendling*: partially supported by NARCO (ANR-21-CE48-0011).

1 Introduction

Kőnig's (infinity) lemma, named after Dénes Kőnig, was originally published as a theorem of graph theory [18]. Nowadays, it is usually conflated with the following statement:

> Any infinite tree which is finitely branching has an infinite branch.[1]

The restriction to at most binary trees is of particular importance because it can be stated within lightweight foundations like e.g. RCA_0 [26], and is usually called weak Kőnig's lemma (WKL). Notice that Kőnig's lemma is also used in its contrapositive form:

> Any finitely branching tree with only finite branches must be finite.

Classical mathematicians would not mind switching between the two formulations but herein, we refrain from using excluded middle at will, and we adopt a constructivist point of view. In this context, that contrapositive form is sometimes referred to as "Brouwer's FAN theorem" [5, p. 13]. Although there is no universal agreement on what constitutes constructive mathematics, we use the inductive type theory that is the basis of Coq, free of additional axioms, as our constructive foundations.

Kőnig's lemma plays critical roles in various fields of mathematics like logic, computability, tiling theory, etc. and has been investigated by reverse mathematics, e.g. as WKL_0 in [26], and constructive reverse mathematics [2, 3]. Although some of our investigations might be relevant to the program of reverse mathematicians, we do not follow that approach. We favor a more pragmatic perspective: since the lemma does not belong to the realm of purely constructive mathematics, can we propose *weaker* alternatives that could be used, not as drop-in, but rather as low cost replacements for Kőnig's lemma? Of course, we require that those alternatives are constructively provable.

[1] the original statement rather talks about paths in a graph.

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg; Article No. 2; pp. 2:1–2:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Kőnig's lemma can (in particular) be used to establish the termination of algorithms, and typically has been used for the decision procedure of implicational relevance logic [8, 17]. It is instrumental to show the existence of Harvey Friedman's [11] TREE(n) monster (extremely fast growing) function, in combination with Kruskal's tree theorem, see e.g. [13] where both proofs rely on classical mathematics. These are two example applications of our tools aimed at giving constructive accounts of what could otherwise look inherently classical.

As simple as it sounds, Kőnig's lemma involves the notion of infinite tree. Hence, the trees cannot simply be understood as the inductively defined structure to be found in computer science (these are always finite). Also, the notion of infinite is not as straightforward in the constructive world. In reverse mathematics, where the usage of versatile data structures may be constrained, a tree is often conflated with its set of finite branches (so finite sequences of nodes where the next node is a son of the current node). As such, trees are nonempty, prefix closed, sets of finite sequences, possibly with a computable membership predicate. And infinite branches are sequences for which every finite prefix belongs to the tree, i.e. the upper limit of a growing sequence of finite branches of the tree.

In that context, one can prove Kőnig's lemma using excluded middle and a weak form of the axiom of choice (e.g. dependent choice). If a canonical choice can be made over the sons, typically when there is a total order that can sort the sons at every node of the tree, the infinite construction process in the proof can be determinized (by choosing the least son) and the reliance on the axiom of choice is avoidable in that case. However, excluded middle is more critical, in particular to show that when the union of finitely many sub-trees is infinite, it must be because one of them is infinite. "Being infinite" is not a decidable property so the selection performed by excluded middle cannot be turned into a computable value.

Kleene [16] famously gave a counterexample to a computational interpretation of weak Kőnig's lemma: he builds a computable infinite binary tree, so a decidable set of finite sequences of Booleans,[2] for which there exists no *computable* infinite branch, i.e. no infinite sequence of Booleans of which every finite prefix belongs to the tree. This gives a very strong argument against the constructive acceptability of Kőnig's lemma, at least when one "interprets Bishop's mathematics in a recursive way" [6].[3]

Not only Kőnig's lemma could be rejected from a constructivist point of view, but some of its consequences suffer similar defects. Consider the compactness result for Wang tilings:

> A finite set of tiles can tile the plane if and only if it can tile any finite square.

Similarly to Kleene's result, Hanf [14] and Myers [24] famously gave examples of finite sets of tiles that can tile the whole plane, but only in a nonrecursive way. This invalidates a computational understanding of the compactness result. Hence no constructive account of the proof of the compactness result can be given, otherwise it would entail the existence of a recursive tiling.[4]

So there is no real hope at a drop-in constructive replacement for Kőnig's lemma because some of its consequences might live outside the realm of constructive or computable mathematics. Nevertheless, we argue that it might be used in contexts where weaker alternatives would also fit. And it is our aim here to explore some of those alternatives.

For instance, there is an interpretation of its contrapositive form, i.e. "any finitely branching tree with only finite branches must be finite," where the notion of infinity is replaced by finitary notions. Notice that the referred statement still relies on arbitrary (finite or infinite) trees: when saying "only finite branches," one must consider the possibility that it contains infinite branches otherwise this hypothesis is vacuous:

[2] choices between the left or the right son.

[3] as said earlier, the notion of what is constructively acceptable is not universally agreed on.

[4] Notice that the tileability of a finite square is a decidable property.

- one classical way to understand "only finite branches" is by saying that no infinite sequence can have all its finite prefixes in the tree. Hence even though the statement does not refer to infinity, it is uncovered in this unfolding;
- another way is to understand "only finite branches" is to give an inductive characterization of the well-foundedness of branches with the single rule for the $\mathtt{acc}\,F : \mathtt{rel}_1\,X$ predicate:

$$\frac{\forall y,\, F\,x\,y \to \mathtt{acc}\,F\,y}{\mathtt{acc}\,F\,x}\ [\mathtt{acc\_intro}]$$

  where $F : \mathtt{rel}_2\,X$ is a parameter relation. Intuitively, $F\,x\,y$ means that x is the father of y in the tree (or y is a son of x). If nodes are conflated with finite branches, then $F\,x\,y$ means that y has the shape $x +\!\!+ [\_]$, i.e. x followed by a single choice of a son.

In that later case, finiteness of the tree branching along F and rooted at *root* can be defined by ($\mathtt{acc}\,F\,\mathit{root}$), and thus understood as the unavoidable termination of the nondeterministic process of expending branches by adding sons after sons, starting from the *root*. Intuitively, the proof of ($\mathtt{acc}\,F\,\mathit{root}$) is a well-founded tree where a leaf is decorated with a proof of ($\mathtt{acc}\,F\,x$) such that x is childless (i.e. $\forall y, \neg F\,x\,y$). In that inductive understanding of "only finite branches," the contrapositive of Kőnig's lemma can be established by well founded induction, see e.g. [1, p. 15]. We will derive it as a corollary in Section 5.3.

In intuitionistic frameworks, Brouwer's FAN theorem is a consequence of the Bar theorem, originally designed to grasp the full continuum in an approach to real analysis [5]. Its acceptability from a constructive standpoint is a delicate issue. It has a rich track record: see e.g. Troelstra [27, 28] for the context of intuitionistic analysis, but there are more recent results, related to convexity [4] or to the exhaustibility of the Cantor space [9]. Intuitionists have compared Kőnig's lemma with the FAN theorem in various contexts, e.g. [12, 30].

Remember that our aim is not to study the FAN theorem per se, but to propose workable alternatives to the use of Kőnig's lemma for the conversion of classical proofs to constructive ones. In this context, we may abuse referring to Kőnig's lemma even though, from a purely intuitionistic standpoint, the results we discuss are closer to the FAN theorem.

However, in contrast with the above cited work on the FAN theorem, we differ in our approach to quantification over the branches of trees. We follow Coquand's thesis [6] that bar inductive predicates are the correct expression of universal quantification over choice sequences, be they lawlike or lawless; see our discussion in Section 3.5. Hence, we work directly with inductive bars (on finite sequences), avoiding Brouwer's thesis [29] completely. Actually, we use the more general notion of inductive cover [25] on (transition) relations.

As for our contributions, in Section 3 we show that the notion of inductive cover generalizes both inductive bars and (inductive) accessibility, w.r.t. its definition as well as w.r.t. the results that it entails. We then give a detailed comparison between the constructive and classical strength of three characterizations of covers: positive, negative and sequential. In particular, for the classical part of the comparison, we separate the role played by excluded middle and dependent choice and show the key role played by the intermediate negative characterization. It will also play an important role in a constructive context, as a substitute to the sequential characterization, when used in combination with the FAN theorem.

In Section 4, we give a type theoretic interpretation of the FAN theorem for inductive covers, with a concise proof. The central argument, the stability of upward closed inductive covers under binary union, differs from that of the proof of Fridlender's FAN theorem for inductive bars [10] which relies on the stability of monotone inductive bars under binary intersection. However, we derive the FAN for bars as an instance of the FAN for covers, to make the generalization explicit.

In Section 5, we exploit the FAN for inductive covers, followed by an application of the negative characterization of covers, to give several weaker versions of (the contrapositive of) Kőnig's lemma, showing how relations can be represented by rose trees (hence finitary). This includes an extra covering assumption, or an extra bar assumption, or else an extra almost fullness assumption.

In Section 6, we give two examples where Kőnig's lemma can successfully be replaced with one of these weaker variants to give constructive accounts of results of which the former proofs were using the classical form of the lemma.

Additionally, we contribute a mechanization of all the results of the paper in a Coq script that can of course be type checked for correctness, but was especially designed to be read by humans, not only by computers. The script is mostly self contained, largely commented, with concise proofs: the longest is 25 loc but most of them are shorter than 10 loc. It is accessible under a free software license at

`https://github.com/DmxLarchey/Constructive-Konig`

2 Coq preliminaries

We denote by $\mathbb{P}$ the type of propositions and simply by `Type` the Coq hierarchy of types, as usual with this framework. We write $\bot : \mathbb{P}$ for the empty proposition and use the standard notations for logical connectives. Recall that the logic of Coq is intuitionistic hence the negation is defined by $\neg P \coloneqq P \to \bot$. Following the BHK interpretation, $X \to Y$ more generally denotes the type of maps from X to Y, and we write $\forall x : X,\ P\,x$ for the dependent product, irrelevant of whether $P : X \to \mathbb{P}$ or $P : X \to \texttt{Type}$. Whenever it can be guessed, the type annotation in $x : X$ is simply avoided. The dependent sum has several flavors in Coq: for $P : X \to \mathbb{P}$ we have the proposition $\exists x,\ P\,x : \mathbb{P}$ and the type $\{x \mid P\,x\} : \texttt{Type}$ which behave somewhat similarly but are however fundamentally different because propositions of sort $\mathbb{P}$ cannot systematically be eliminated to build terms of sort `Type`.

The type of Peano natural numbers $\mathbb{N}$ is inductively defined in Coq as $\mathbb{N} : n \coloneqq 0 \mid \mathtt{S}\,n$ and arithmetic in Coq, which we assume, is built on this type. We manipulate finite sequences as lists, polymorphic[5] over the carrier type X, in the inductive type $\texttt{list}\,X :\ l \coloneqq [] \mid x :: l$ where $x : X$. Additionally, list concatenation (resp. membership) is named `app` (resp. `In`), denoted infix by $\cdot +\!\!+ \cdot : \texttt{list}\,X \to \texttt{list}\,X \to \texttt{list}\,X$ (resp. $\cdot \in \cdot : X \to \texttt{list}\,X \to \mathbb{P}$), and defined by a guarded fixpoint. Moreover, we use the reverse $\texttt{rev} : \texttt{list}\,X \to \texttt{list}\,X$ and the length $\lfloor\cdot\rfloor : \texttt{list}\,X \to \mathbb{N}$ functions as well as the permutation relation $\cdot \sim_p \cdot : \texttt{list}\,X \to \texttt{list}\,X \to \mathbb{P}$, as inductively defined in the `Permutation` module of the Coq standard library.

We define finiteness as a property $\texttt{finite}\,P : \mathbb{P}$ of unary relations (viewed as sets):[6]

$$\texttt{finite}\ \{X\}\ (P : X \to \mathbb{P}) \coloneqq \exists l,\ \forall x,\ P\,x \leftrightarrow x \in l$$

i.e. there exists a list spanning the relation P. This characterization of finiteness as listability is equivalent to *Kuratowski finiteness* but easier to manipulate formally.

We manipulate relations as functions outputting propositions, hence we denote by $\texttt{rel}_2\,X\,Y \coloneqq X \to Y \to \mathbb{P}$ the type of heterogeneous binary relations between X and Y. In the homogeneous case, we simply write $\texttt{rel}_2\,X \coloneqq X \to X \to \mathbb{P}$, and $\texttt{rel}_1\,X \coloneqq X \to \mathbb{P}$ in the unary case. We use the letters $P, Q : \texttt{rel}_1\,\_$ to denote unary relations and $R, T : \texttt{rel}_2\,\_\,\_$

[5] operators on lists are parametric in X and this first argument is nearly always left implicit.

[6] Like lists based results, `finite` is parametric in X and the braces around it *specify* an implicit argument.

to denote binary relations. We write $P \subseteq Q$ or $R \subseteq T$ for the inclusion between relations. Except for commonly found notations like $\in$, $\sim_p$ or $\subseteq$, we generally write related pairs with e.g. a letter for the relation name, in prefix order, like in $T\,x\,y$.

For complex inductive predicates, we rather present the constructors using rules with a horizontal line separating the premises from the conclusion. As an example, we below display those of $\texttt{Forall}\,P : \texttt{rel}_1\,(\texttt{list}\,X)$ (denoted $\wedge_1 P$) and $\texttt{Forall2}\,R : \texttt{rel}_2\,(\texttt{list}\,X)\,(\texttt{list}\,Y)$ (denoted $\wedge_2 R$) which are finitary conjunctions defined in the `List` module of the standard library, for $P : \texttt{rel}_1\,X$ and $R : \texttt{rel}_2\,X\,Y$:

$$\frac{}{\wedge_1 P\ []} \qquad \frac{P\,x \qquad \wedge_1 P\ l}{\wedge_1 P\ (x :: l)} \qquad\qquad \frac{}{\wedge_2 R\ []\ []} \qquad \frac{R\,x\,y \qquad \wedge_2 R\ l\ m}{\wedge_2 R\ (x :: l)\ (y :: m)}$$

The free symbols $x, y : X$ and $l, m : \texttt{list}\,X$ can be instantiated by any value in their respective types. In the corresponding Coq constructors, they are universally quantified over.

3 Inductive covers

We recall the notion of inductive cover [25] which subsumes both accessibility and bar inductive predicates; see Sections 3.2 and 3.3. We discuss three characterizations of covers, the positive, the negative and the sequential, from the strongest to the weakest (constructively), but also explain in some details how to get their classical equivalence, separating the roles played by the axioms of excluded middle and dependent choice. We discuss these characterizations in the context Brouwer's intuitionistic understanding of infinite sequences.

Before we switch to covers, we import the standard order theoretic notion of being upward closed, however not requiring partial orders but any binary relation instead.

▶ **Definition 1** (Upward closed). *Given a type X and a binary relation $T : \texttt{rel}_2\,X$, we say that a unary relation $P : \texttt{rel}_1\,X$ is* T-upward closed *if P is stable under direct T-images. We define:* $\texttt{upclosed}\,T\,P \coloneqq \forall x\,y, T\,x\,y \to P\,x \to P\,y$.

For instance, the finitary conjunction $\wedge_1 P$ is upward closed for permutations, formally stated as $\texttt{upclosed}\,(\cdot \sim_p \cdot)\,\wedge_1 P$. Upward closed unary relations will be preserved by covers, and some results about covers (including the FAN theorem) assume upward closed relations.

3.1 Inductive covers definition, basic results

As in [25], we work with the class of singleton inductively generated formal topologies, as opposed to the more general (e.g. indexed) presentation of [7]. They are defined by the notion of inductive cover of a set (i.e. unary relation) along a (transition) binary relation.

▶ **Definition 2** (Inductive cover [25]). *Given a type X, a binary relation $T : \texttt{rel}_2\,X$ and a unary relation $P : \texttt{rel}_1\,X$, we define the* inductive T-cover of P, *denoted* $\texttt{cover}\,T\,P : \texttt{rel}_1\,X$ *by the two following inductive rules:*

$$\frac{P\,x}{\texttt{cover}\,T\,P\,x}\ [\texttt{cover\_stop}] \qquad\qquad \frac{\forall y, T\,x\,y \to \texttt{cover}\,T\,P\,y}{\texttt{cover}\,T\,P\,x}\ [\texttt{cover\_next}]$$

Notice that $P\,x$ (resp. $T\,x\,y$ and $\texttt{cover}\,T\,P\,x$) is denoted by $x \in P$ (resp. $y \in T(x)$ and $x \lhd P$) in [25] but we favor prefix notations. Remark that the transition relation T is hidden in the infix notation $x \lhd P$ used for the cover whereas we keep it in $\texttt{cover}\,T\,P\,x$. Also in [25], the constructor [`cover_stop`] (resp. [`cover_next`]) is called reflexivity (resp. infinity).

The non-dependent induction principle (or eliminator, depending on your preferred terminology) generated for the $\texttt{cover}\,T\,P$ predicate has the following type:

$$\texttt{cover\_ind}\,T\,P: \ \forall Q',\, P \subseteq Q' \to (\forall x,\, T\,x \subseteq Q' \to Q'\,x) \to (\forall x,\, \texttt{cover}\,T\,P\,x \to Q'\,x).$$

Informally, it states that $\texttt{cover}\,T\,P$ is included in any unary relation Q' closed under the constructors/rules [`cover_stop`] and [`cover_next`]. Coq auto-generates a slight variant[7] of `cover_ind` but they are equivalent as non-dependent eliminators. We choose to present the above one because of its direct link with the positive, negative and sequential characterizations of the cover that we discuss in Section 3.4. In our Coq code, we give a straightforward implementation of `cover_ind` as a guarded `Fixpoint`, similar to the auto-generated one.

Using the `cover_ind` induction principle in combination with the constructors, we show how a morphism can be used to transfer covers between different types and relations.

▶ **Proposition 3** (`cover_morphism`). *Let X, Y be two types, $R : \texttt{rel}_2\,X$ and $T : \texttt{rel}_2\,Y$ be binary relations, and $P : \texttt{rel}_1\,X$ and $Q : \texttt{rel}_1\,Y$ be unary relations. We further assume a map $f : Y \to X$ which is supposed to be a morphism w.r.t. P/Q and R/T, i.e. satisfying*

$$\forall y,\, P(f\,y) \to Q\,y \qquad \textit{and} \qquad \forall y_1\,y_2,\, T\,y_1\,y_2 \to R\,(f\,y_1)\,(f\,y_2).$$

Then we have $\forall x\,y,\, x = f\,y \to \texttt{cover}\,R\,P\,x \to \texttt{cover}\,T\,Q\,y$.

As they are made available as Coq code, we generally do not give detailed proofs herein. To illustrate how we reason by induction using `cover_ind`, we exceptionally give a detailed account of the proof for the `cover_morphism` statement.

Proof. We first reorder the hypotheses and the goal becomes $\forall x,\, \texttt{cover}\,R\,P\,x \to \forall y,\, x = f\,y \to \texttt{cover}\,T\,Q\,y$, which we prove by induction on $\texttt{cover}\,R\,P\,x$: we factor out Q' in the goal as $\forall x,\, \texttt{cover}\,R\,P\,x \to Q'\,x$ with $Q' \coloneqq \lambda\,x,\, \forall y,\, x = f\,y \to \texttt{cover}\,T\,Q\,y$. Reasoning backwards, we apply the instance $\texttt{cover\_ind}\ R\,P\,Q'$ to the goal, replacing it with two sub-goals:[8]

- $P \subseteq Q'$: unfolding Q', we assume x s.t. $P\,x$ and y s.t. $x = f\,y$ and we have to show $\texttt{cover}\,T\,Q\,y$. We derive $P(f\,y)$ by substitution and then $Q\,y$ using the left morphism hypothesis. We then derive $\texttt{cover}\,T\,Q\,y$ using the constructor [`cover_stop`];
- $\forall x,\, (R\,x \subseteq Q') \to Q'\,x$: we assume x s.t. $\mathit{IH}_x : \forall z,\, R\,x\,z \to \forall y', z = f\,y' \to \texttt{cover}\,T\,Q\,y'$ (corresponding to the induction hypothesis) and y s.t. $x = f\,y$, and we have to show $\texttt{cover}\,T\,Q\,y$. We substitute x with $f\,y$ in IH_x and get $\mathit{IH}_y : \forall z,\, R\,(f\,y)\,z \to \forall y', z = f\,y' \to \texttt{cover}\,T\,Q\,y'$. Applying the second constructor [`cover_next`], we replace the goal $\texttt{cover}\,T\,Q\,y$ with $\forall z,\, T\,y\,z \to \texttt{cover}\,T\,Q\,z$. Hence we assume z s.t. $H_{yz} : T\,y\,z$ and we now have to show $\texttt{cover}\,T\,Q\,z$. Using the right morphism hypothesis, we derive $H'_{yz} : R\,(f\,y)\,(f\,z)$. We instantiate the induction hypothesis as $\mathit{IH}_y\ (f\,z)\ H'_{yz}\ z\ \texttt{eq\_refl}$ and get $\texttt{cover}\,T\,Q\,z$ as desired. Notice that we use the reflexive identity for $\texttt{eq\_refl} : f\,z = f\,z$.

This concludes the proof. Using standard automation for backward proof-search (the `eauto` tactic), the Coq proof script however consists in only two lines of code, one for reordering the hypotheses in the initial statement, the second telling which hypothesis to perform induction on, and then launching automation after substituting $f\,y$ for x. ◀

[7] namely of type $\forall Q',\, P \subseteq Q' \to (\forall x,\, T\,x \subseteq \texttt{cover}T\,P \to T\,x \subseteq Q' \to Q'\,x) \to (\forall x,\, \texttt{cover}T\,P\,x \to Q'\,x)$.

[8] factoring out Q' and applying `cover_ind` are automated for us by the Coq `induction` tactic.

For the rest of the section, we assume a fixed type X to be used as carrier for binary relations $R, T : \texttt{rel}_2\, X$ and unary relations $P, Q : \texttt{rel}_1\, X$. The monotonicity of `cover` can be obtained as a particular case, using the identity morphism $f \coloneqq \lambda\, x,\, x$. More precisely, `cover` $(\cdot)\,(\cdot)$ is antitonic in its first argument and monotonic in its second argument:

$$\texttt{cover\_mono}\ R\ T\ P\ Q:\ T \subseteq R \to P \subseteq Q \to \texttt{cover}\ R\, P \subseteq \texttt{cover}\ T\, Q.$$

Additionally to be increasing (by [`cover_stop`]) and monotonic (by `cover_mono`), `cover` T is also an idempotent operator making it a closure operator:

$$\texttt{cover\_idempotent}\ T\ P:\ \texttt{cover}\ T\,(\texttt{cover}\ T\ P) \subseteq \texttt{cover}\ T\ P.$$

Proof. Assuming an arbitrary x, the proof of $\texttt{cover}\,T\,(\texttt{cover}\,T\,P)\,x \to \texttt{cover}\,T\,P\,x$ proceeds by induction on $\texttt{cover}\ T\,(\texttt{cover}\ T\ P)\ x$. ◀

Then we get that the `cover` T operator preserves T-upward closed unary relations:

$$\texttt{cover\_upclosed}\ T\ P:\ \texttt{upclosed}\,T\,P \to \texttt{upclosed}\,T\,(\texttt{cover}\ T\ P).$$

Proof. We assume $\texttt{upclosed}\,T\,P$ and an arbitrary x and show $\texttt{cover}\ T\,P\,x \to \forall y,\, T\,x\,y \to \texttt{cover}\ T\,P\,y$ by induction on $\texttt{cover}\ T\,P\,x$. ◀

3.2 Inductive cover and accessibility

In this section, we fix a type X to serve as carrier for relations below. We recall that the `cover` predicate is a generalization of the accessibility predicate, also called R-founded in [25].

▶ **Definition 4** (acc(essibility), R-founded). *Given a binary relation $R : \texttt{rel}_2\ X$, the* `acc`(essibility) predicate[9] *for* R *and the* R-founded predicate[10] *are defined inductively, each with one single rule:*

$$\frac{\forall y,\, R\,x\,y \to \texttt{acc}\ R\ y}{\texttt{acc}\ R\ x}\,[\texttt{acc\_intro}] \qquad \frac{\neg R\,x\,x \qquad \forall y,\, R\,x\,y \to \texttt{founded}\,R\,y}{\texttt{founded}\,R\,x}$$

A simple observation shows that the shape of the constructor [`acc_intro`] is the same as the second constructor [`cover_next`] of the `cover` predicate. Furthermore, the first constructor [`cover_stop`] can be neutralized by setting P as the empty relation $\emptyset \coloneqq \lambda\_,\, \bot$. Hence we immediately derive the equivalence:

▶ **Proposition 5.** *The* `acc`*(essibility) predicate is an instance of the* `cover` *predicate.*

$$\texttt{acc\_iff\_cover\_empty}\ R\ x:\ \texttt{acc}\ R\ x \leftrightarrow \texttt{cover}\ R\ \emptyset\ x.$$

Moreover, accessible elements are necessarily irreflexive. Indeed, we show $\forall x,\, \texttt{acc}\ R\,x \to R\,x\,x \to \bot$ by induction on $\texttt{acc}\ R\,x$. Hence it follows that the left premise $\neg R\,x\,x$ of the constructor of R-founded is superfluous:

▶ **Proposition 6.** *R-founded and accessibility define equivalent notions:*

$$\texttt{founded\_iff\_acc}\ R\ x:\ \texttt{founded}\,R\,x \leftrightarrow \texttt{acc}\ R\,x.$$

As a corollary we get $\texttt{founded}\,R\,x \leftrightarrow \texttt{cover}\ R\,\emptyset\,x$, a result already established in [25, Theorem 3.2] but, seemingly, the authors did not observe that the left premise ($\neg R\,x\,x$ i.e. irreflexivity) of the introduction rule for R-founded was superfluous.

[9] The variant `Acc` as defined in the Coq standard library module `Prelude`, simply uses the reversed relation R^{-1} instead of R for `acc`. So we have $\texttt{Acc}\ R \simeq \texttt{acc}\ R^{-1}$ and $\texttt{Acc}\ R^{-1} \simeq \texttt{acc}\ R$.

[10] R-founded is defined in [25, Definition 3.1].

3.3 Inductive cover and inductive bars

Let X be a carrier type for lists. We consider unary relations on the type $\texttt{list}\,X$ that we use to represent finite sequences. We show that inductive covers, in addition to generalizing accessibility predicates (see Section 3.2), also generalize inductive bar predicates [6, 10].

▶ **Definition 7** (Inductive bar). *Let $P : \texttt{rel}_1\,(\texttt{list}\,X)$ be a unary relation on lists. We define the inductive $\texttt{bar}\,P : \texttt{rel}_1\,(\texttt{list}\,X)$ unary relation with the two following inductive rules:*

$$\frac{P\,l}{\texttt{bar}\,P\,l}\ [\texttt{bar\_stop}] \qquad \frac{\forall x,\ \texttt{bar}\,P\,(x::l)}{\texttt{bar}\,P\,l}\ [\texttt{bar\_next}]$$

Compared to [10, Definition 6], there are two slight differences. First our lists expand from the left, whereas often in the literature [6, 10, 29], finite sequences expand from the right. Hence rule [bar_next] would be written

$$\frac{\forall x,\ \texttt{bar}\,P\,(l \mathbin{++} [x])}{\texttt{bar}\,P\,l}$$

with such a reversed convention. However, this difference can be viewed as just of matter of ordering the display of the arguments of the list constructor ::. Another more important difference compared to [10, Definition 6] or else [29], is the absence of the inductive rule

$$\frac{\texttt{bar}\,P\ l}{\texttt{bar}\,P\,(x::l)}\ [\texttt{bar\_monotone}]$$

in Definition 7. We discard rule [bar_monotone] because it is admissible for monotone unary relations on finite sequences.

▶ **Definition 8** (Monotone unary relation). *A unary relation $P : \texttt{rel}_1\,(\texttt{list}\,X)$ is* monotone *if it satisfies $\texttt{monotone}\,P \coloneqq \forall x\,l,\ P\,l \to P(x::l)$.*

The (discarded) [bar_monotone] rule/constructor would ensure that $\texttt{bar}\,P$ is a monotone predicate even when P is not monotone. However, as an instance of `cover_upclosed`, if P is monotone then so is $\texttt{bar}\,P$; see `bar_monotone` after Proposition 10 below. We observe that monotone unary relations are those which are upward closed under list extension:

▶ **Definition 9** (list extension). *The $\texttt{extends} : \texttt{rel}_2\,(\texttt{list}\,X)$ binary relation on lists is defined by the single inductive rule:* $\dfrac{}{\texttt{extends}\ l\ (x::l)}$

We could have used the first order characterization $\texttt{extends}\ l\ m \leftrightarrow \exists x,\ m = x::l$ to give an alternate definition of `extends`. With this notion, we get the equivalence

$$\texttt{upclosed\_extends\_iff\_monotone}\ P :\ \texttt{upclosed}\,\texttt{extends}\,P \leftrightarrow \texttt{monotone}\,P$$

as an immediate consequence, but the specialization goes further:

▶ **Proposition 10** (bar_iff_cover_extends). *Given a unary relation $P : \texttt{rel}_1\,(\texttt{list}\,X)$ and a list $l : \texttt{list}\,X$, we have the equivalence $\texttt{bar}\,P\ l \leftrightarrow \texttt{cover}\,\texttt{extends}\,P\ l$.*

Thanks to Proposition 10 and `upclosed_extends_iff_monotone`, the two below results are specializations of respectively `cover_upclosed` and `cover_mono`.

$$\begin{array}{ll}\texttt{bar\_monotone}\ P : & \texttt{monotone}\,P \to \texttt{monotone}\,(\texttt{bar}\,P);\\ \texttt{bar\_mono}\ P\ Q : & P \subseteq Q \to \texttt{bar}\,P \subseteq \texttt{bar}\,Q.\end{array}$$

More generally, the analysis that we are going to present for inductive covers in the next section can be specialized to either accessibility predicates and inductive bar predicates.

3.4 Positive, negative and sequential characterizations

We now discuss other characterizations of covers, which are not constructively equivalent to the inductive one, but however are classically equivalent, hence the abusive use of the word "characterization." We present a detailed analysis of those characterizations and which classical axioms their equivalence depends on.

The results of this section that assume classical axioms are not used elsewhere in this paper: these axioms are (propositional) excluded middle (XM), giving us De Morgan laws for logical connectives and quantifiers, and dependent choice (DC):

$$\begin{aligned}&\texttt{xm} : \forall\, A : \mathbb{P},\; A \vee \neg A;\\ &\texttt{dc} : \forall\,(A : \texttt{Type})\,(R : \texttt{rel}_2\; A),\; (\forall a \exists b,\; R\,a\,b) \to \forall a \exists \rho : \mathbb{N} \to A,\; \rho_0 = a \wedge \forall n,\; R\;\rho_n\;\rho_{1+n}.\end{aligned}$$

The names of the results that depend on these added axioms are suffixed with _XM or _DC or both for an unambiguous exposition.

We start with the following definitions of the positive characterization cover_pos, the negative characterization cover_neg, and the sequential characterization cover_seq.

▶ **Definition 11** (Nonequivalent characterizations of cover).

$$\begin{aligned}&\texttt{cover\_pos}\;T\;P\;x \coloneqq \lambda\,Q : \texttt{rel}_1\;X,\; P \subseteq Q \to (\forall y,\; T\,y \subseteq Q \to Q\,y) \to Q\,x;\\ &\texttt{cover\_neg}\;T\;P\;x \coloneqq \lambda\,Q : \texttt{rel}_1\;X,\; Q\,x \to (\forall y,\; Q\,y \to \exists z,\; Q\,z \wedge T\,y\,z) \to \exists y, P\,y \wedge Q\,y;\\ &\texttt{cover\_seq}\;T\;P\;x \coloneqq \lambda\,\rho : \mathbb{N} \to X,\; \rho_0 = x \to (\forall n,\; T\,\rho_n\;\rho_{1+n}) \to \exists n,\; P\,\rho_n.\end{aligned}$$

Although not equivalent, the constructive strength of these characterizations can be compared: they are displayed from the strongest (cover_pos) to the weakest (cover_seq). Beware that both Q and ρ are universally quantified over in the characterizations below.

The positive characterization cover_pos is really just a reordering of the implications in the induction principle cover_ind, so we get the following equivalence purely constructively:

$$\texttt{cover\_iff\_cover\_pos}\;T\;P\;x :\;\; \texttt{cover}\,T\,P\,x \leftrightarrow \forall Q,\; \texttt{cover\_pos}\;T\;P\;x\;Q.$$

The positive characterization is constructively stronger that the negative one:

$$\texttt{cover\_pos\_\_cover\_neg}\;T\;P\;x :\;\; (\forall Q,\; \texttt{cover\_pos}\;T\;P\;x\;Q) \to (\forall Q,\; \texttt{cover\_neg}\;T\;P\;x\;Q).$$

Proof. We use $\forall Q$, cover_pos $T\;P\;x\;Q$ as the formulation of an induction principle. ◀

The negative characterization is constructively stronger than the sequential one. The below proof argument anticipates the intuition behind the definition of the negative characterization.

$$\texttt{cover\_neg\_\_cover\_seq}\;T\;P\;x :\;\; (\forall Q,\; \texttt{cover\_neg}\;T\;P\;x\;Q) \to (\forall \rho,\; \texttt{cover\_seq}\;T\;P\;x\;\rho).$$

Proof. Assuming a T-sequence $\rho : \mathbb{N} \to X$, we instantiate Q with the direct image $\rho(\mathbb{N}) \coloneqq \lambda\,y,\; \exists n, \rho_n = y$. We show cover_neg $T\;P\;x\;\rho(\mathbb{N}) \to$ cover_seq $T\;P\;x\;\rho$ and conclude. ◀

We now explain the intuition behind those definitions by turning to a *classical interpretation* where all those characterizations are equivalent, discussing the precise roles played by XM and DC. The negative characterization cover_neg is central to our analysis and can be understood in two ways, either as deriving from cover_pos or generalizing cover_seq:

- The first understanding of cover_neg is as contrapositive form of cover_pos:

$$\texttt{cover\_pos\_iff\_neg\_XM}\;T\;P\;x\;Q : \texttt{cover\_pos}\;T\;P\;x\;Q \;\leftrightarrow\; \texttt{cover\_neg}\;T\;P\;x\;(\neg Q).$$

The proof involves excluded middle but *first-order De Morgan transformations* are enough to get the equivalence.[11] The converse implication of `cover_pos__cover_neg` above is unlikely to be constructively provable (see Section 3.5), but it is a direct corollary to `cover_pos_iff_cover_neg_XM`,[12] however assuming XM as an added axiom;

- The second way to understand the negative characterization `cover_neg` is to view it as a generalization of the sequential characterization `cover_seq`. Notice that the statement $\forall \rho$, `cover_seq` $T\,P\;x\;\rho$ is the usual intuitive definition of being a T-cover for P:

  Any infinite T-sequence starting at x meets P.[13]

  However this interpretation depends on *what are the inhabitants of the type* $\mathbb{N} \to X$ of which ρ is a member; see Section 3.5. In the proof of `cover_neg__cover_seq`, we used the direct image $\rho(\mathbb{N})$ as a particular instance of Q in `cover_neg`. Q represents a set of values containing x and over which T is a total binary relation, which generalizes T-sequences by removing the requirement of determinism. The quantification over T-sequences $\rho : \mathbb{N} \to X$ is replaced by quantification over Q which is an *T-unstoppable nondeterministic process*: indeed any point in Q has at least one T-image in Q. This property of unstoppability $\forall y,\, Q\,y \to \exists z,\, Q\,z \wedge T\,y\,z$ is shared also by Brouwer's notion of spread.

As a consequence of the above discussion, constructively already, the positive characterization is equivalent to the inductive one, and stronger than the negative one, which is itself stronger than the sequential one. Hence we derive:

`cover_negative` $T\,P\;x$: `cover` $T\,P\;x \to \forall Q,\, Q\,x \to (\forall y,\, Q\,y \to \exists z,\, Q\,z \wedge T\,y\,z) \to \exists y, P\,y \wedge Q\,y$
`cover_sequences` $T\,P\;x$: `cover` $T\,P\;x \to \forall \rho,\, \rho_0 = x \to (\forall n,\, T\,\rho_n\;\rho_{1+n}) \to \exists n,\, P\,\rho_n$

If one is interested in the converse implications, then, on the one hand, XM would be used to prove that $\forall Q$, `cover_neg` $T\,P\;x\;Q$ implies `cover` $T\,P\;x$. On the other hand, to recover $\forall Q$, `cover_neg` $T\,P\;x\;Q$ from $\forall \rho$, `cover_seq` $T\,P\;x\;\rho$, one uses `DC` $\{x \mid Q\,x\}$ which is dependent choice specialized on the Σ-type $\{x \mid Q\,x\}$ where Q : $\mathtt{rel}_1\;X$. Indeed, the statement of `DC` X, i.e. dependent choice specialized on type X is:

$$\mathtt{DC}\;X \coloneqq \forall R : \mathtt{rel}_2\;X,\; (\forall x \exists y,\; R\,x\,y) \to \forall x \exists \rho : \mathbb{N} \to X,\; \rho_0 = x \wedge \forall n,\; R\;\rho_n\;\rho_{1+n}.$$

When Q : $\mathtt{rel}_1\;X$, we reformulate the instance `DC` $\{x \mid Q\,x\}$ as[14]

$$\forall R,\; (\forall x,\; Q\,x \to \exists y,\; Q\,y \wedge R\,x\,y) \to \forall x,\; Q\,x \to \exists \rho,\; \rho_0 = x \wedge \forall n,\; Q\,\rho_n \wedge R\;\rho_n\;\rho_{1+n}$$

which is exactly what is needed to extract a sequence $\rho : \mathbb{N} \to X$ out of the T-unstoppable process Q starting at x.

`cover_seq__cover_neg_DC` $T\,P\;x$: $(\forall \rho,$ `cover_seq` $T\,P\;x\;\rho) \to (\forall Q,$ `cover_neg` $T\,P\;x\;Q)$.

▶ **Theorem 12** (in the spirit of Brouwer's bar theorem). *Assuming* `xm` *and* `dc`*, the inductive and the sequential characterizations of covering are equivalent:*

$$\mathtt{cover}\;T\,P\;x \quad\leftrightarrow\quad \forall \rho, \rho_0 = x \to (\forall n,\; T\,\rho_n\;\rho_{1+n}) \to \exists n,\; P\,\rho_n.$$

[11] In the Coq script, we insist on obtaining that equivalence via De Morgan rewriting and congruence only.
[12] see `cover_neg__cover_pos_XM`.
[13] Such formulation are more commonly found for the "intuitive" (read sequential) definition of "being a bar for P" [6]. See `bar_sequences` in Section 3.5 for the corresponding specialization.
[14] see `DC_sig__DC_Σ` in the Coq code.

Hence under XM+DC, any cover is an inductive cover while Brouwer's "bar theorem" states that "any bar is inductive bar," or, quoting [6], for sequences of natural numbers:

$$\forall P : \texttt{rel}_1\,(\texttt{list}\,\mathbb{N}),\ \texttt{bar}\ P\ [\,] \leftrightarrow \forall \alpha : \mathbb{N} \to \mathbb{N},\ \exists n,\ P\,[\alpha_{n-1}; \ldots ; \alpha_0].$$

The bar theorem statement is an instance of Theorem 12 where $T \coloneqq \texttt{extends}$. Indeed, an `extends`-sequence of lists in $\mathbb{N} \to \texttt{list}\,X$ corresponds to the n-prefixes of a sequence $\mathbb{N} \to X$; see `Brouwer_bar_XM_DC` in the Coq code.

3.5 Discussion

We have explained how the inductive predicates `bar` and `acc` are just specializations of the notion of inductive `cover` so the remarks below also apply to those restricted notions. For instance we get the following specializations for $R : \texttt{rel}_2\ X$ and $P : \texttt{rel}_1\ (\texttt{list}\,X)$:

$$\begin{array}{ll}
\texttt{acc\_negative}\ x : & \texttt{acc}\,R\ x \to \forall Q,\ Q\,x \to (\forall y,\ Q\,y \to \exists z,\ Q\,z \wedge R\,y\,z) \to \bot; \\
\texttt{acc\_sequences}\ x : & \texttt{acc}\,R\ x \to \forall \rho,\ \rho_0 = x \to (\forall n,\ R\ \rho_n\ \rho_{1+n}) \to \bot; \\
\texttt{bar\_negative} : & \texttt{bar}\ P\ [\,] \to \forall Q,\ Q\,[\,] \to \big(\forall l,\ Q\,l \to \exists x,\ Q\,(x :: l)\big) \to \exists l, P\,l \wedge Q\,l; \\
\texttt{bar\_sequences} : & \texttt{bar}\ P\ [\,] \to \forall \alpha, \exists n,\ P\,[\alpha_{n-1}; \ldots ; \alpha_0].
\end{array}$$

The negative characterization is intermediate between the inductive/positive characterization (strongest) and the sequential characterization (weakest). We isolate the role played by XM (in fact De Morgan laws) and DC. While it avoids DC, the negative characterization, using unstoppable nondeterministic processes instead of sequences, still likely requires XM to be equivalent with the positive characterization. Indeed, were the negative characterization be constructively equivalent to positive/inductive characterization, such a result would instantly give us Theorem 12 (and Brouwer's bar theorem) using DC alone, hence avoiding XM.

The discussion on what is nature of (infinite) sequences is central to the sequential characterization of bars, and of course, as the infinite itself, is very much debated in constructive mathematics. Clearly, adjoining XM and DC populates the type $\mathbb{N} \to X$ with enough *lawless sequences*. Brouwer however rejected XM and DC and instead justifies his bar theorem using "Brouwer's thesis" [29] which is not as strong as an axiom as XM+DC. In [6], Coquand criticizes the use of the type $\mathbb{N} \to X$ to cover "all" sequences in the sequential characterization of bars:

> "This example is paradigmatic: by replacing systematically the intuitive notion of bar by the notion of inductive bar, we can now prove Brouwer's fan theorem. More generally, we can think of `bar` P $[\,]$ as the *correct format expression of a universal quantification over all sequences*, not necessarily given by a law." (emphasis added)

To be more specific, absent of extra axioms, the type $\mathbb{N} \to X$ of *lawlike sequence* (on which the sequential characterization is based) cannot account for sequences that do not evolve according to a predetermined law, see e.g. Veldman [29]:

> "the intuitionistic mathematician [...] admits the possibility of sequences $\alpha_0, \alpha_1, \alpha_2, \ldots$ that are created step-by-step and thus, in some sense, are given by a black box. He is very much aware that *he is unable to make any kind of survey of the totality of all infinite sequences* of natural numbers." (emphasis added)

In a way, we follow and extend to covers the program proposed by Coquand [6] to systematically replace the intuitive (understand sequential) notion of cover by the inductive version, avoiding axioms altogether. But we can still use the sequential or negative versions, in a limited way, at the end of a constructive deduction, e.g. following the FAN theorem.

4 The FAN theorem for inductive covers

In this section, we present another interpretation of the FAN theorem in type theory, generalizing the FAN theorem for inductive bars [10] to inductive covers [7, 25] instead. We give a concise proof for this result, which differs significantly from that of [10, Theorem 6]. Hence, as an specialization, we get an alternate proof of that former result as well.

In this section, let us fix a type X and a binary relation $T : \texttt{rel}_2\, X$. We extend the binary relation T to lists (viewed as finite sets), as $T^\dagger : \texttt{rel}_2\, (\texttt{list}\, X)$ using the direct image and this way, we can view FANs over T as $T^\dagger$-sequences over finite sets.

4.1 Lifting a relation to finite sets

We define the finitary image relation on $\texttt{list}\, X$ viewed as finite sets, i.e. permutations and contractions are admissible for lists used in that context.

▶ **Definition 13** (Finitary image). *We define the* finitary image *binary relation on lists, denoted $T^\dagger : \texttt{rel}_2\, (\texttt{list}\, X)$, by $T^\dagger \coloneqq \lambda\, l\, m,\, \forall y,\, y \in m \to \exists x,\, x \in l \land T\, x\, y$, i.e. $T^\dagger\, l\, m$ holds when m is included in the direct image of l.*

The finitary image relation $T^\dagger$ is monotonic in its first argument and antitonic in its second argument, i.e. $l_1 \subseteq l_2 \to m_2 \subseteq m_1 \to T^\dagger\ l_1\ m_1 \to T^\dagger\ l_2\ m_2$ holds.

One critical observation for the proof of the FAN theorem below is how $T^\dagger$ behaves when splitting its first/source argument in two halves. Then there is a corresponding splitting of the second/image argument, but since $T^\dagger$ ignores the order on the elements of lists, this splitting only holds up to a permutation of the image list:

$$\texttt{fimage\_split\_inv}\ l_1\ l_2\ m :\ T^\dagger\ (l_1 +\!\!+\, l_2)\ m \to \exists\, m_1\, m_2,\ m \sim_p m_1 +\!\!+\, m_2 \land T^\dagger\ l_1\ m_1 \land T^\dagger\ l_2\ m_2$$

which we show by induction on m. Additionally, we show that $T^\dagger\ (\cdot)\ k$ is upward closed for permutations for any k, which can be written as $\texttt{upclosed}\, (\cdot \sim_p \cdot)\, (T^\dagger \cdot k)$. And to conclude this section, if P is upward closed for T then the finitary conjunction $\wedge_1 P$ of P (over lists) is upward closed for $T^\dagger$, i.e. $\texttt{upclosed}\, T\, P \to \texttt{upclosed}\, T^\dagger\, \wedge_1 P$.

4.2 Proof of the FAN theorem for inductive covers

We give a proof of the statement of the FAN theorem for inductive covers, using the finitary image relation $T^\dagger$ to represent FANs over the relation T.

▶ **Theorem 14** (FAN for inductive covers). *Let $P : \texttt{rel}_1\, X$ be T-upward closed. If x is in the T-cover of P then the singleton list $[x]$ is in the $T^\dagger$-cover of $\wedge_1 P$, i.e.*

$$\texttt{FAN\_cover} :\ \texttt{upclosed}\, T\, P \to \forall x,\ \texttt{cover}\, T\, P\ x \to \texttt{cover}\, T^\dagger\ \wedge_1 P\ [x].$$

Using a sequential understanding of covers, the statement could be read as: if any T-sequence starting at x meets P then any $T^\dagger$-sequence starting at $[x]$ meets $\wedge_1 P$, hence "any finitary FAN rooted at x meets a monotone P uniformly," which is a commonly found informal statement of the FAN theorem.

While this sequential understanding cannot be established in our constructive framework (for reasons discussed in Section 3.5), we below give a quite compact inductive proof of the positive/inductive understanding of the statement of the FAN theorem for inductive covers.

Proof. Let us assume P with $\texttt{upclosed}\,T\,P$. We first show that $\texttt{cover}\,T^\dagger\ \wedge_1 P$ is upward closed for permutations, stated as $\texttt{upclosed}\,(\cdot \sim_p \cdot)\,(\texttt{cover}\,T^\dagger\ \wedge_1 P)$. For this, we prove $\texttt{cover}\,T^\dagger\ \wedge_1 P\ l \to \forall m,\, l \sim_p m \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ m$ by induction on $\texttt{cover}\,T^\dagger\ \wedge_1 P\ l$.

Now, we establish the *key result* that $\texttt{cover}\,T^\dagger\ \wedge_1 P$ is stable under (binary) union, herein represented by the append operation on lists:

$$\texttt{cover\_fimage\_union}\ l\ m :\ \texttt{cover}\,T^\dagger\ \wedge_1 P\ l \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ m \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ (l \mathbin{+\!\!+} m).$$

The proof proceeds by *nested induction*, first on $\texttt{cover}\,T^\dagger\ \wedge_1 P\ l$ and then on $\texttt{cover}\,T^\dagger\ \wedge_1 P\ m$, with a critical use of `fimage_split_inv` to invert two statements of shape $T^\dagger\ (\cdot \mathbin{+\!\!+} \cdot)\ (\cdot)$ where the first argument of $T^\dagger$ is a union of lists. As a corollary of `cover_fimage_union`, we get the specialization where $l \coloneqq [x]$ is a singleton as

$$\forall\, x\, m,\ \texttt{cover}\,T^\dagger\ \wedge_1 P\ [x] \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ m \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ (x :: m)$$

and then, as a direct consequence

$$\texttt{cover\_fimage\_Forall}\ l :\ \big(\forall x,\, x \in l \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ [x]\big) \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ l$$

for which we proceed by induction on l.

We can conclude with the proof of the FAN theorem for inductive covers. We establish $\texttt{cover}\,T^\dagger\ \wedge_1 P\ [x]$, reasoning by induction on $\texttt{cover}\,T\,P\,x$:

- the base case where $P\,x$ holds is trivially solved by giving a proof of $\wedge_1 P\ [x]$ and then deriving $\texttt{cover}\,T^\dagger\ \wedge_1 P\ [x]$ with an instance of first constructor [`cover_stop`];
- in the recursive case where $\forall y,\, T\,x\,y \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ [y]$ is the induction hypothesis, we show $\forall l,\, T^\dagger\ [x]\ l \to \forall y,\, y \in l \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ [y]$ and then combine this with `cover_fimage_Forall` and an instance of the second constructor [`cover_next`].

This concludes our proof of the FAN theorem for inductive covers. ◀

We can immediately derive $\wedge_1(\texttt{cover}\,T\,P)\ l \to \texttt{cover}\,T^\dagger\ \wedge_1 P\ l$ by induction on l and then the following characterization of covering for the finitary image:

$$\texttt{cover\_fimage\_iff} :\ \texttt{upclosed}\,T\,P \to \forall l,\, \texttt{cover}\,T^\dagger\ \wedge_1 P\ l \leftrightarrow (\forall x,\, x \in l \to \texttt{cover}\,T\,P\,x).$$

i.e. the list l is $T^\dagger$-covered for $\wedge_1 P$ if and only if all the members of l are T-covered for P.

4.3 The FAN theorem for inductive bars

We recall the interpretation of the FAN theorem in type theory [10] and derive an alternate proof of that result as an instance of Theorem 14, which illustrates our claim of generalization. We fix a carrier type X for lists and consider relations over $\texttt{list}\,X$ and $\texttt{list}\,(\texttt{list}\,X)$. For $lc : \texttt{list}\,(\texttt{list}\,X)$, let us first define the

$$\texttt{FAN}\ lc \coloneqq \lambda\, l,\ \wedge_2(\cdot \in \cdot)\ l\ lc$$

i.e. if written as $l = [x_1; \ldots; x_n]$ and $lc = [c_1; \ldots; c_p]$, $\texttt{FAN}\ lc\ l$ means $n = p$ and $x_1 \in c_1, x_2 \in c_2, \ldots, x_n \in c_n$. Stated in plain english, l is a list of one-to-one choices for the choice list lc; see the inductive definition of $\wedge_2 R$ in Section 2. Using generic tools designed for the `finite` abstraction, we can show that $\texttt{FAN}\ lc$ is a finite, i.e.

$$\texttt{FAN\_finite}\ lc :\ \texttt{finite}\,(\texttt{FAN}\ lc).$$

However in [10, page 102], the author gives a *specific* construction of a list which collects the lists of choices l s.t. $\texttt{FAN}\ lc\ l$, that we denote $\texttt{list\_fan}\ lc$ herein, satisfying:

$$\texttt{list\_fan\_spec}\ lc: \ \forall l, \texttt{FAN}\ lc\ l\ \leftrightarrow\ l \in \texttt{list\_fan}\ lc.$$

Thus the dependent pair $(\texttt{list\_fan}\ lc, \texttt{list\_fan\_spec}\ lc)$ is an (explicitly given) proof of the proposition $\texttt{finite}\,(\texttt{FAN}\ lc)$. The value of $\texttt{list\_fan}\ lc$ can be viewed as a generalization of the exponential function to lists, computing the *list of choice sequences for lc.*

The FAN theorem as stated and proved in [10] relies on the particular implementation of the exponential `list_fan` given there, but the result itself only depends on the fact that `list_fan` satisfies `list_fan_spec`. Theorem 6 of [10] also assumes the added rule [`bar_monotone`] in the inductive definition of the `bar` predicate but it is admissible for monotone relations.

▶ **Theorem 15** (reminder of Theorem 6 of [10]). *Let* $P : \texttt{rel}_1\ (\texttt{list}\,X)$ *be unary relation. The following statement holds:* $\texttt{monotone}\,P \to \texttt{bar}\ P\ [] \to \texttt{bar}\ \big(\lambda\,lc,\ \wedge_1 P\,(\texttt{list\_fan}\ lc)\big)\ []$.

Proof. We first reformulate the result as

$$\texttt{FAN\_bar}\ P: \ \texttt{monotone}\,P \to \texttt{bar}\ P\ [] \to \texttt{bar}\ (\lambda\,lc,\ \texttt{FAN}\ lc \subseteq P)\ []$$

which is an equivalent statement thanks to the monotonicity `bar_mono` of the `bar` predicate. Indeed, using `list_fan_spec`, we get the equivalence $\wedge_1 P\,(\texttt{list\_fan}\ lc)\ \leftrightarrow\ \texttt{FAN}\ lc \subseteq P$ for any lc. Now the statement $\texttt{FAN\_bar}\,P$ is independent of the implementation of `list_fan`.

Using the results of Section 3.3, we replace the hypotheses $\texttt{monotone}\,P$ and $\texttt{bar}\ P\ []$ by $\texttt{upclosed}\ \texttt{extends}\,P$ and $\texttt{cover}\ \texttt{extends}\ P\ []$, and the goal $\texttt{bar}\ (\lambda\,lc,\ \texttt{FAN}\ lc \subseteq P)\ []$ becomes $\texttt{cover}\ \texttt{extends}\ (\lambda\,lc,\ \texttt{FAN}\ lc \subseteq P)\ []$. Hence, by Theorem 14 we get $\texttt{cover}\,\texttt{extends}^\dagger\ \wedge_1 P\ [[]]$. Then we transfer the inductive cover using `list_fan` as a morphism (Proposition 3):

$$\texttt{cover}\,\texttt{extends}^\dagger\ \wedge_1 P\ [[]] \to \texttt{cover}\,\texttt{extends}\ (\lambda\,lc,\ \texttt{FAN}\ lc \subseteq P)\ []$$

after having checked that the following statements hold: $[[]] = \texttt{list\_fan}\ []$, $\texttt{extends}\ l\ m \to \texttt{extends}^\dagger\ (\texttt{list\_fan}\,l)\ (\texttt{list\_fan}\,m)$ and $\wedge_1 P\ (\texttt{list\_fan}\ lc) \to \texttt{FAN}\ lc \subseteq P$. ◀

Theorem 6 of [10] (cf. Theorem 15), and its original proof, even though it uses one particular implementation of `list_fan` both in the proved statement and inside the arguments, can be adapted to work for any implementation of `list_fan` as soon as it satisfies `list_fan_spec`. The reason is that we pass through `FAN_bar` which is independent of the actual implementation of `list_fan`. This is how the proof is implemented our Coq code.

Besides the previous remark and the detour via inductive covers, the proof we give differs from that of [10] in an important way. Indeed, the core argument of the later proof is the closure of monotone inductive `bar`s under binary intersection [10, Proposition 3]:

$$\texttt{monotone}\,P \to \texttt{monotone}\,Q \to \texttt{bar}\ P\ l \to \texttt{bar}\ Q\ l \to \texttt{bar}\ (P \cap Q)\ l$$

which is there established by nested inductions on $\texttt{bar}\ P\ l$, and then on $\texttt{bar}\ Q\ l$. On the contrary, the core argument in the proof of Theorem 14 lies in `cover_fimage_union`, i.e. the closure of $\texttt{cover}\ T^\dagger\ \wedge_1 P$ under binary union (the append operator on lists). In a way, it generalizes to upward closed inductive covers the stability under binary unions of finiteness.

5 Weaker variants of the contrapositive of Kőnig's lemma

Recall the contrapositive form of Kőnig's lemma: any finitely branching tree without infinite branches is finite. We introduce (inductive) rose trees, i.e. finite trees with arbitrary (finite) branching at each node, and we give a type theoretic variant which has stronger assumptions (e.g. the covering assumption below), and which replaces the notion of possibly infinite tree that is implicit in formulation "any ... tree without infinite branches" with that of a relation:

> Assume a finitely branching relation $T : \mathtt{rel}_2\, X$ and $P : \mathtt{rel}_1\, X$ which is T-upward closed. If x belongs to the T-cover of P then the finite paths along T starting at x and avoiding P are the branches of a rose tree rooted at x.

Notice that we use equivalence between paths and branches to express that (part of) a relation is "the same" as a rose tree. Because we only view the relation via its paths, the acyclicity assumption, as used when (infinite) trees are viewed as graphs, can be dropped. But before we formalize this statement, we must define paths, rose trees and their branches.

5.1 Path, rose trees and their branches

Let us fix a type X as carrier for relations and indices of rose trees below.

▶ **Definition 16** (Inductive path). *For a relation $T : \mathtt{rel}_2\, X$, the paths in T are described by a ternary relation $\mathtt{path}\ T : X \to \mathtt{list}\, X \to X \to \mathbb{P}$ defined by two inductive rules:*

$$\frac{}{\mathtt{path}\ T\ x\ []\ x} \qquad \frac{T\ x\ y \qquad \mathtt{path}\ T\ y\ p\ z}{\mathtt{path}\ T\ x\ (y :: p)\ z}$$

Intuitively, $\mathtt{path}\ T\, x\, p\, y$ means that p is the sequence of values encountered on a path from x to y, following the relation T, including the endpoint y but excluding starting point x. The existence of a T-path from x to y is equivalent to the reflexive-transitive closure of T (we do not use this characterization however), and hence we have:

$$\mathtt{upclosed\_path}\ T\, P : \mathtt{upclosed}\, T\, P \to \mathtt{upclosed}\, (\lambda\, x\, y,\ \exists p,\ \mathtt{path}\ T\ x\ p\ y)\, P.$$

▶ **Definition 17** (Inductive rose tree). *The type of X-indexed* rose trees *denoted* $\mathtt{tree}\, X : \mathsf{Type}$ *is inductively defined by a single rule:*

$$\frac{x : X \qquad l : \mathtt{list}\,(\mathtt{tree}\, X)}{\langle x|l\rangle : \mathtt{tree}\, X}\ [\mathtt{node}]$$

where we denote $\langle x|l\rangle$ as a shortcut for $(\mathtt{node}\ x\ l)$*. The* root *of $t = \langle x|l\rangle$ is indexed by x and we write $\mathtt{root}\, t = x$, and l is the* list of the sons *of t. We define the* height *of a rose tree, denoted $\mathtt{tree\_ht} : \mathtt{tree}\, X \to \mathbb{N}$, using the fixpoint equation $\mathtt{tree\_ht}\ \langle x|[t_1; \dots; t_n]\rangle = 1 + \mathtt{list\_max}\,[\mathtt{tree\_ht}\, t_1; \dots; \mathtt{tree\_ht}\, t_n]$.*

The branches *of a rose tree (the paths starting at the root) are characterized using a ternary relation $\mathtt{branch} : \mathtt{tree}\, X \to \mathtt{list}\, X \to X \to \mathbb{P}$ inductively defined by two rules:*

$$\frac{}{\mathtt{branch}\ \langle x|l\rangle\ []\ x} \qquad \frac{\langle y|m\rangle \in l \qquad \mathtt{branch}\ \langle y|m\rangle\ p\ z}{\mathtt{branch}\ \langle x|l\rangle\ (y :: p)\ z}$$

Hence a branch is either empty, stopping at the root, or the choice of a son (i.e. sub-tree) and of a branch in that son. The predicate $\mathtt{branch}\ t\ p\ y$ relates a tree t, a list of visited indices p up to the index y of the root of a sub-tree of t.

5.2 Representing binary relations using rose trees

We give a formal definition for the statement "a binary relation is a finite tree." This is required indeed because the type of binary relations and the type of rose trees are very different. We use paths in relations and branches in rose trees as a means to define the notion of *representation* by a rose tree, for the part of a relation $T : \texttt{rel}_2\, X$ rooted at x of which the paths from x satisfy the property $P : \texttt{list}\, X \to X \to \mathbb{P}$.

▶ **Definition 18** (Representation). *Assume a binary relation $T : \texttt{rel}_2\, X$, a property for paths $P : \texttt{list}\, X \to X \to \mathbb{P}$ and a point $x : X$. We say that P* in *T* at *x* is strongly represented by *$t : \texttt{tree}\, X$ and write $\texttt{strongly\_represents}\ T\, P\, x\, t$ if:*

$$\texttt{strongly\_represents}\ T\, P\, x\ t \coloneqq \texttt{root}\, t = x \wedge \forall\, p\, y,\ \texttt{branch}\ t\ p\ y\ \leftrightarrow\ \texttt{path}\, T\ x\ p\ y \wedge P\ p\ y.$$

We say that P in *T* at *$x : X$* is represented by *$t : \texttt{tree}\, X$ and write $\texttt{represents}\ T\, P\, x\ t$ if:*

$$\texttt{represents}\ T\, P\, x\ t \coloneqq \texttt{root}\, t = x \wedge \forall\, p\, y,\ P\, p\, y \to (\texttt{branch}\ t\ p\ y\ \leftrightarrow\ \texttt{path}\ T\ x\ p\ y).$$

The property P for paths is applied only to those originating at x but can depend on the destination as well as on the sequence of visited nodes on the path to the destination.

We observe that $\texttt{strongly\_represents}\ T\, P\, x\, t \to \texttt{represents}\ T\, P\, x\, t$. While the strong notion would be a first/natural choice to formalize the idea that the relation T starting at x and restricted by P "is a tree," this choice can however be questioned in the light of decidability issues. Indeed, when X is equipped with a (propositionally) decidable equality,[15] e.g. when $X = \mathbb{N}$, then both $\texttt{branch}\ t\ p\ y$ and $\texttt{path}\ T\ x\ p\ y$ become decidable predicates. In that case, $\texttt{strongly\_represents}\ T\, P\, x\, t$ implies that P is decidable as well, an assumption we want to avoid for building representations. In the case of $\texttt{represents}$, P does not need to be decidable but the representing tree may contain branches which do not satisfy P.

We assume a fixed $T : \texttt{rel}_2 X$ which is furthermore *finitely branching*, i.e. $\forall x,\ \texttt{finite}\, (T\, x)$. We show that paths of bounded length can be strongly represented.

▶ **Theorem 19.** *When $T : \texttt{rel}_2\, X$ is finitely branching, for any $n : \mathbb{N}$ and any $x : X$, the property $(\lambda\, p\, y,\ \lfloor p \rfloor \le n)$ in T at x has a strong representation.*

Proof. We build the tree t s.t. $\texttt{strongly\_represents}\ T\ (\lambda\, p\, y,\ \lfloor p \rfloor \le n)\ x\ t$ by induction on n, after generalizing on x. ◀

Now we characterize the properties of paths that have representations as those which hold only for small paths.

▶ **Theorem 20.** *When $T : \texttt{rel}_2 X$ is finitely branching, for any property $P : \texttt{list}\, X \to X \to \mathbb{P}$ and any point $x : X$, the two following statements are equivalent:*

- *$\exists t,\ \texttt{represents}\ T\, P\, x\ t$;*
- *$\exists n, \forall\, p\, y,\ \texttt{path}\ T\ x\ p\ y \to P\, p\, y \to \lfloor p \rfloor < n$.*

Proof. In the forward direction, the bound n can be chosen to be the height $\texttt{tree\_ht}\, t$ of the representation of P in T at x. In the reverse direction, given a bound n for the length of paths satisfying P, we first obtain a tree t s.t. $\texttt{strongly\_represents}\, T\, (\lambda\, p\, y,\ \lfloor p \rfloor \le n)\, x\, t$. We then check that this tree t represents P in T at x. ◀

[15] i.e. $\forall\, x\, y : X,\ x = y \vee x \neq y$.

Theorem 20 states that, in the finitely branching case, a relation is represented by a (finite) rose tree if and only if there is a global bound on the length of its paths. In the context of the FAN theorem (the contrapositive of Kőnig's lemma), it relates the finiteness of a tree to the boundedness of its height, the length of its longest branch. It can be compared to the characterization of binary trees[16] which are finite as those for which there is a uniform bound on the length of their branches, see e.g. [15].

5.3 Kőnig's lemma for inductive covers, accessibility and inductive bars

We establish statements of weakened variants of (the contrapositive of) Kőnig's lemma, assuming e.g. the existence of a cover for the root of the "tree."

▶ **Theorem 21** (Kőnig's lemma for inductive covers)**.** *Let us assume a finitely branching binary relation $T : \mathtt{rel}_2\, X$, i.e. $\forall x,\ \mathtt{finite}\,(T\,x)$, a T-upward closed unary relation $P : \mathtt{rel}_1\, X$, a root $x : X$ which is T-covered by P. Then the paths which refute P at their tail are represented in T at x, i.e. $\exists t,\ \mathtt{represents}\ T\ (\lambda\, p\, y,\ \neg P\, y)\ x\ t$.*

Proof. Using the length of paths, we define $\mathtt{circle}\, n$, the circle (centered at x) of radius n, and the collection of $Q : \mathtt{rel}_1\, (\mathtt{list}\, X)$ of finite supports of some circle as

$$\mathtt{circle}\, n \coloneqq \lambda\, y,\ \exists p,\ \mathtt{path}\ T\ x\ p\ y \wedge n = \lfloor p \rfloor \qquad Q\, l \coloneqq \exists n,\ \forall x,\ \mathtt{circle}\, n\, x \leftrightarrow x \in l.$$

Because T has finite direct images, we deduce that circles are finite, by induction on n. Hence we get $\forall n,\ \mathtt{finite}\,(\mathtt{circle}\, n)$.

Let us show that Q meets $\wedge_1 P$. Indeed, as we assume $\mathtt{cover}\ T\, P\, x$, using the FAN Theorem 14 for covers we get $\mathtt{cover}\ T^\dagger\ \wedge_1 P\ [x]$. Then we use `cover_negative` with Q. We only need to show that Q holds at $[x]$ and is $T^\dagger$-unstoppable i.e. $\forall l,\ Q\, l \to \exists m,\ Q\, m \wedge T^\dagger\, l\, m$:

- $Q\,[x]$ holds because $[x]$ is a support for the circle of radius 0;
- Q is $T^\dagger$-unstoppable because the circle of radius $1+n$ is a $T^\dagger$-image of that of radius n.

As Q meets $\wedge_1 P$, then P contains some circle, i.e. there is n such that $\mathtt{circle}\, n \subseteq P$. As a consequence, since T-paths from x of length greater that n cross $\mathtt{circle}\, n$ hence meet P at that crossing point, their tail must belong to P as well, because P is T-upward closed. Hence $\forall\, p\, y,\ \mathtt{path}\ T\ x\ p\ y \to n \le \lfloor p \rfloor \to P\, y$ holds and we conclude using Theorem 20. ◀

This proof uses the FAN Theorem 14 for inductive covers, and then combines it with the `cover_negative` characterization. The finiteness of circles $\forall n,\ \mathtt{finite}\,(\mathtt{circle}\, n)$, which lives $\mathbb{P}$ (and not in `Type`), is not strong enough to be able to define `circle` as a map $\mathbb{N} \to \mathtt{list}\, X$, which would be needed if the `cover_sequences` characterization were to be used instead of `cover_negative`.

As the instance of Theorem 21 where $P \coloneqq \emptyset$, we recover a finitary form of Kőnig's lemma similar to [1, p. 15]. A direct proof by induction on (the proof of) $\mathtt{acc}\, T\, x$ would probably be shorter but we here illustrate the generality of Kőnig's lemma for inductive covers.

▶ **Corollary 22** (Kőnig's lemma for accessibility [1])**.** *Let $T : \mathtt{rel}_2\, X$ be a binary relation s.t. $\forall x,\ \mathtt{finite}\,(T\,x)$ and let $x : X$ be a T-accessible point of X, i.e. $\mathtt{acc}\, T\, x$. Then there is a rose tree $t : \mathtt{tree}\, X$ with root x such that the T-paths from x are exactly the branches of t.*

We present a variant of Kőnig's lemma for inductive bars. It is not exactly an instance of Theorem 21 because the properties of paths are not limited to those of their endpoint.

[16] as sets of finite sequences of Booleans representing their finite branches.

▶ **Theorem 23** (Kőnig's lemma for inductive bars). *Let us assume a finitely branching binary relation* $T : \mathtt{rel}_2\, X$*, a monotone unary relation* $P : \mathtt{rel}_1\,(\mathtt{list}\, X)$*, and a point* $x : X$*. If* $\mathtt{bar}\ P\ []$ *then* $\exists t, \mathtt{represents}\ T\ \big(\lambda\, p\, y, \neg P\,(\mathtt{rev}\, p)\big)\ x\ t$*.*

Proof. The proof is comparable (not identical) to the proof of Theorem 21 and uses `FAN_bar` and `bar_negative` instead as replacements for `FAN_cover` and `cover_negative`. ◀

5.4 Kőnig's lemma for sequences of finite choices

Bar predicates can be specialized using the notion of *good sequence*, i.e. one containing a redundant pair w.r.t. a binary (redundancy) relation. This relation can be the identity, but there are other interesting cases, e.g. multiset inclusion [20]. In this case, bar predicates characterize *inductive almost full relations* [20, 31].

We assume a binary relation $R : \mathtt{rel}_2\, X$ to represent a notion of redundancy, and define two unary relations $\mathtt{good}\, R$ and $\mathtt{irred}\, R$ of type $\mathtt{rel}_1\,(\mathtt{list}\, X)$, $\mathtt{good}\, R$ characterizing lists which contain a good pair, and $\mathtt{irred}\, R$ characterizing lists which are irredundant, i.e. avoiding good pairs:[17]

$$\begin{aligned}
&\mathtt{good}\, R\ p \ \coloneqq \exists\, l\, x\, m\, y\, r,\ p = l +\!\!+ [x] +\!\!+ m +\!\!+ [y] +\!\!+ r \wedge R\, y\, x;\\
&\mathtt{irred}\, R\ p \coloneqq \forall\, l\, x\, m\, y\, r,\ p = l +\!\!+ [x] +\!\!+ m +\!\!+ [y] +\!\!+ r \to R\, x\, y \to \bot.
\end{aligned}$$

It is obvious that $\mathtt{good}\, R$ is monotone. Moreover, we show the correspondence between bad (i.e. not good) lists and irredundant ones:[18]

$$\mathtt{not\_good\_iff\_irred}\ R\ p :\ \neg\big(\mathtt{good}\, R\ (\mathtt{rev}\, p)\big) \leftrightarrow \mathtt{irred}\, R\ p.$$

▶ **Definition 24** (Almost full relation [31]). *For binary relations* $R : \mathtt{rel}_2\, X$*, we define the predicate* $\mathtt{af}\ R : \mathbb{P}$ *using the two inductive rules, where* $R{\uparrow}u \coloneqq \lambda\, x\, y,\ R\, x\, y \vee R\, u\, x$*:*

$$\frac{\forall x\, y,\ R\, x\, y}{\mathtt{af}\ R}\ [\mathtt{af\_full}] \qquad\qquad \frac{\forall u,\ \mathtt{af}\ R{\uparrow}u}{\mathtt{af}\ R}\ [\mathtt{af\_lift}]$$

We recall that $\mathtt{af}\ R$ is another way of stating (i.e. is equivalent to) $\mathtt{bar}\ (\mathtt{good}\, R)\ []$ (see e.g. [20, p. 11] or [21]) but below we just need the implication in this direction:

$$\mathtt{af\_\_bar\_good\_nil}\ R :\ \mathtt{af}\ R \to \mathtt{bar}\ (\mathtt{good}\, R)\ [].$$

Proof. First we establish $\mathtt{bar}\ (\mathtt{good}\, R{\uparrow}u)\ p \to \mathtt{bar}\ (\mathtt{good}\, R)\ (p +\!\!+ [u])$ by induction on $\mathtt{bar}\ (\mathtt{good}\, R{\uparrow}u)\ p$. As an instance where $p \coloneqq []$, we get $\mathtt{bar}\ (\mathtt{good}\, R{\uparrow}u)\ [] \to \mathtt{bar}\ (\mathtt{good}\, R)\ [u]$. Then we can show the implication $\mathtt{af}\ R \to \mathtt{bar}\ (\mathtt{good}\, R)\ []$ by induction on $\mathtt{af}\ R$. ◀

We can deduce usual the sequential characterization of almost full relations, but, as with covers and bars, this characterization is constructively weaker.

$$\mathtt{af\_sequences}\ R :\ \mathtt{af}\ R \to \forall \alpha : \mathbb{N} \to X, \exists\, i\, j,\ i < j \wedge R\, \alpha_i\, \alpha_j.$$

Proof. We obtain n such that $\mathtt{good}\, R\ [\alpha_{n-1}; \ldots ; \alpha_0]$ using `af__bar_good_nil` above, followed by `bar_sequences` from Section 3.5. We conclude by analyzing the identity $l +\!\!+ [x] +\!\!+ m +\!\!+ [y] +\!\!+ r = [\alpha_{n-1}; \ldots ; \alpha_0]$ where $R\, y\, x$ holds. ◀

[17] See [21] for an equivalent inductive characterization of $\mathtt{good}\, R$.

[18] `good` and `irred` use R in opposite ways, hence the use of the `reverse` function.

We finish our tour of weakened forms of Kőnig's lemma with a slightly different form where the outcome is not a representing rose tree but just its height, hence a bound on the length of its branches. In light of Theorem 20, these are equivalent conditions for finitely branching relations. While we insisted so far on getting tree representations in the spirit of Kőnig's lemma, in its applications on e.g. termination, a bound on the height of this tree is often sufficient to conclude.

Given a sequence of relations $P : \mathbb{N} \to \texttt{rel}_1\, X$, we define a predicate $\texttt{choice\_list}\, P$: $\texttt{rel}_1\, (\texttt{list}\, X)$ such that $\texttt{choice\_list}\, P\ [x_0; \ldots ; x_{n-1}] \ \leftrightarrow\ P_0\, x_0 \wedge \cdots \wedge P_{n-1}\, x_{n-1}$, i.e. $\texttt{choice\_list}\, P\ l$ holds exactly when the members of l are successive choices in P_0, P_1, ...

▶ **Theorem 25.** *Given an almost full relation, i.e.* $R : \texttt{rel}_2\, X$ *s.t.* $\texttt{af}\ R$*, and a sequence of finite unary relations, i.e.* $P : \mathbb{N} \to \texttt{rel}_1\, X$ *s.t.* $\forall n,\ \texttt{finite}\, P_n$*. Then the length of irredundant choice lists for* P *is (uniformly) bounded. Formally, we get:*

$$\texttt{af}\ R \to (\forall n,\ \texttt{finite}\, P_n) \to \exists n,\ \forall l,\ \texttt{choice\_list}\, P\ l \to \texttt{irred}\, R\ l \to \lfloor l \rfloor < n.$$

Proof. From `af__bar_good_nil`, we know that $\texttt{bar}\, (\texttt{good}\, R)\, []$ holds and we apply the `FAN_bar` form of Theorem 15 and derive $\texttt{bar}\ (\lambda\, lc,\ \texttt{FAN}\ lc \subseteq \texttt{good}\, R)\ []$.

We define $\texttt{support}\ n\ l \coloneqq \forall x,\ P_n\, x \ \leftrightarrow\ x \in l$, meaning that l is a supporting list for the (finite) unary relation P_n. We use the `bar_negative` characterization of inductive bars applied to $\texttt{bar}\ (\lambda\, lc,\ \texttt{FAN}\ lc \subseteq \texttt{good}\, R)\ []$ with $Q \coloneqq \lambda\, lc,\ \texttt{choice\_list}\ \texttt{support}\ (\texttt{rev}\ lc)$. We get lc such that $\texttt{FAN}\ lc \subseteq \texttt{good}\, R$ and $\texttt{choice\_list}\ \texttt{support}\ (\texttt{rev}\ lc)$.

Then we check that $n \coloneqq \lfloor lc \rfloor$ satisfies the property $\forall l,\ \texttt{choice\_list}\ P\ l \to \lfloor l \rfloor = n \to \texttt{good}\, R\ (\texttt{rev}\ l)$. The same value then bounds the length of irredundant choice lists for P. ◀

Again, we use a combination of a FAN theorem and the negative characterization of inductive bars. The assumption of finiteness $\forall n,\ \texttt{finite}\, P_n : \mathbb{P}$ is not strong enough to be able to build a *sequence* $\mathbb{N} \to \texttt{list}\, X$ that enumerates the respective supports for P_0, P_1, ... but the negative characterization allows us to reason without escaping from the $\mathbb{P}$ sort.

5.5 Type-bounded variants and the FAN functional

To answer a question of one of the reviewers, we briefly discuss how the FAN theorem for covers and its consequences can be lifted to `Type`-bounded variants allowing for the computation of FAN functionals, which output bounds on the length of branches of finite trees. For this, the types of the `cover`, `bar` or `af` predicates are lifted as:

$$\begin{aligned}
&\texttt{cover}_t\ \{X\}\ (T : X \to X \to \texttt{Type})\ (P : \texttt{rel}_1\, X) : X \to \texttt{Type}\\
&\texttt{bar}_t\ \{X\}\ \big(P : \texttt{rel}_1\, (\texttt{list}\, X)\big) : \texttt{list}\, X \to \texttt{Type}\\
&\texttt{af}_t\ \{X\}\ (R : \texttt{rel}_2\, X) : \texttt{Type}
\end{aligned}$$

and, as a consequence, parts of the `List` and `Permutation` library results need to be given `Type`-bounded variants as well. Noticeably, the transition relation argument T in $\texttt{cover}_t$ is lifted, and not only its output sort.

In Coq jargon, we say that those predicates become informative, meaning that their inductive structure can be used to compute values of sort `Type`, typically bounds on the length of branches in $\mathbb{N}$. As an illustration here, these variants include the following FAN functional, lifting Theorem 25 above to an output of sort `Type`:

$$\texttt{af}_t\, R \to (\forall n,\ \texttt{finite}_t\, P_n) \to \{n : \mathbb{N} \mid \forall l,\ \texttt{choice\_list}\, P\ l \to \texttt{irred}\, R\ l \to \lfloor l \rfloor < n\}$$

where $\texttt{finite}_t\ \{X\}\ (P : X \to \mathbb{P}) \coloneqq \{l \mid \forall x,\ P\, x\ \leftrightarrow\ x \in l\}$ is the lifting of finiteness.

We delegate to the distributed Coq code, specifically file `constructive_konig_type.v`, the explanations on how such liftings are performed.

6 Two examples of replacements of Kőnig's lemma

In this section, we discuss two applications of our constructive variants of Kőnig's lemma that allow to transfer some "classical" proofs into the realm of constructive mathematics.

6.1 The decidability from implicational relevance logic

In [20], we use a variant of Kőnig's lemma for almost full relations, corresponding here to Theorem 25, to show the termination of an exhaustive proof search procedure for implicational relevance logic (IR), based on a sequent system designed by Curry [8]. The termination of this system was established by Kripke [17], building on Curry's work, rediscovering Dickson's lemma, and concluding with Kőnig's lemma.

The idea of the proof is the following. Curry's sequent proof system is proved sound and complete for IR. It has three essential properties:

- each sequent rule has finitely many premises, in fact less than two;
- for any conclusion, there are only finitely many rule instances having that conclusion;
- there is a notion of redundancy for sequents such that, if a sequent S_2 is redundant over S_1, then any proof of S_2 can be contracted into a proof of S_1 of lesser height. This property is called *Curry's lemma.*

Kripke proved that the notion of redundancy, derived from the natural inclusion ordering on multisets, forms a well quasi order (WQO), and thus any sequence of sequents contains a redundant pair. Notice that the WQO terminology and Dickson's lemma, the key ingredient in the result, were only popularized later on. Then, using Curry's lemma, Kripke argued that any proof search branch must contain a redundant pair, and by Kőnig's lemma, the proof search tree for irredundant proofs is finite.

Replacing the classical approach to WQOs by inductive almost full relations, in [20] we prove that the notion of redundancy is almost full, the constructive form of Dickson's lemma been derived from Coquand's constructive form of Ramsey's theorem [31]. Then we use the `Type`-bounded variant of Theorem 25 called `Constructive_Koenigs_lemma` and outlined in Section 5.5 to show that the irredundant part of the proof search tree has a computable bound on its height, so the search process can be safely pruned above that height.

6.2 Building Harvey Friedman's $\mathrm{TREE}(n)$ monster

In [23], we build on a Coq constructive proof of Kruskal's tree theorem [22] to implement TREE(n) function (that we specify below), invented and studied by Harvey Friedman [11] in his groundbreaking work on reverse mathematics.

The *(homeomorphic) embedding on rose trees* is a WQO as soon as the comparison between decorations of the nodes is itself a WQO: this is the statement of Kruskal's theorem in a classical setting. In [22], we implement a constructively provable form by replacing WQOs with (inductive) `af` relations (see Definition 24). Notice that this constructive form of Kruskal's theorem has a quite involved proof that we do not discuss here.

Using Kruskal's theorem, the homeomorphic embedding between roses trees decorated with elements of the finite set $\{1, \ldots, n\}$ is `af` and we use this relation as our redundancy relation. This means, using the sequential characterization `af_sequences` of Section 5.4, that any sequence T_1, T_2,... of roses trees contains a redundant pair. Now Friedman bounds the number of possible choice for T_i by requiring that its size (number of nodes) is less than i: we say that T_i *is sized.* Hence, considering the set of all such sized sequences $(T_i)_{0<i}$, they form a finitely branching tree and all infinite branches contain a redundant pair. Following

the argumentation of e.g. [13], by Kőnig's lemma, the irredundant part of that tree is finite and thus sized sequences have maximal length, which is by definition TREE(n).

We circumvent this classical argumentation by applying Theorem 25, hence, according to its proof, first applying the FAN theorem for inductive bars and then the negative characterization of inductive bars. We obtain, constructively, the existence of a uniform bound on the length of irredundant sequences of sized trees $(T_i)_{0<i}$. The exact value of the bound, i.e. TREE(n), can then be computed by unbounded linear search [23].[19]

7 Conclusion

Besides the Coq script that supports the results presented herein, we can summarize our contributions as following. We show that the notion of inductive cover generalizes both accessibility and bar inductive predicates, hence we can discuss concepts and results at the level of covers and they instantiate on these restricted notions as well. We follow Coquand's program [6] and replace characterizations based on sequences with inductive ones, that constructively do not fall short on lawless sequences.

We compare the strength of the positive, negative and sequential characterizations of covers, or (as an instance) of "being a bar," both in constructive and classical contexts. We analyze the precise roles played by the axioms of excluded middle and dependent choice.

The negative characterization is a remarkable intermediate notion: a) it is a De Morgan dual of the positive characterization; b) it expels determinism from the sequential characterization, and shares properties with Brouwer's notion of spread; c) it is relevant in practice, for instance when dealing with `Prop`-bounded Coq definitions.

We give a concise constructive proof of a FAN theorem for inductive covers that generalizes the type theoretic interpretation of the FAN theorem for inductive bars [10]. We notice that the respective core argument of these two proofs differ significantly.

The negative or sequential characterizations of covers (or bars) are weaker than the positive/inductive characterization. They fail when trying to constructively establish important closure properties, such as the FAN theorem. However, they can still be used constructively, after the inductive FAN theorem, to obtain uniform bounds on the length of branches of trees. This is the core argumentation behind several weaker variants of Kőnig's lemma that we derive and present, herein insisting on representations by inductive rose trees.

To conclude, we discuss two applications of those constructive variants of Kőnig's lemma that allow the transport of classical results in the constructive realm.

Almost full relations give a satisfactory constructive account for the notion of well quasi order, i.e., finitary closure properties such as Dickson's lemma, Higman's lemma and Kruskal's tree theorem can be constructively established with this notion. However, as far as we are aware, the stronger notion of better quasi order (BQO) has not yet been given a suitable inductive account, and it would be quite a challenge to lean towards an inductive definition of BQOs, hopefully satisfying additional infinitary closure properties.

References

1 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

2 J. Berger. Brouwer's fan theorem, 2021. `doi:10.48550/arXiv.2001.00064`.

3 J. Berger and H. Ishihara. Brouwer's fan theorem and unique existence in constructive analysis. *Mathematical Logic Quarterly*, 51(4):360–364, 2005. `doi:10.1002/malq.200410038`.

[19] Using a `Type`-bounded variant of Theorem 25, one can use bounded linear search instead of unbounded linear search. However, there is little to gain in obtaining an efficient algorithm for computing TREE(n) since writing TREE(3) in decimal would already exhaust all atoms of the known universe.

4 J. Berger and G. Svindland. Brouwer's fan theorem and convexity. *The Journal of Symbolic Logic*, 83(4):1363–1375, 2018. `doi:10.1017/jsl.2018.49`.

5 C. Cheung. Brouwer's Fan Theorem: An Overview. Master's thesis, Cornell University, 2015. URL: `https://api.semanticscholar.org/CorpusID:203584866`.

6 T. Coquand. About Brouwer's Fan Theorem. *Revue internationale de philosophie*, 230:483–489, September 2004. URL: `http://www.jstor.org/stable/23955601`.

7 T. Coquand, G. Sambin, J. Smith, and S. Valentini. Inductively generated formal topologies. *Annals of Pure and Applied Logic*, 124(1):71–106, 2003. `doi:10.1016/S0168-0072(03)00052-6`.

8 H. B. Curry. *A Theory of Formal Deductibility.* Notre Dame mathematical lectures. University of Notre Dame, 1957.

9 M. Escardo. Infinite sets that admit fast exhaustive search. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 443–452, 2007. `doi:10.1109/LICS.2007.25`.

10 D. Fridlender. An Interpretation of the Fan Theorem in Type Theory. In *Types for Proofs and Programs*, pages 93–105. Springer Berlin Heidelberg, 1999. `doi:10.1007/3-540-48167-2_7`.

11 H. M. Friedman. Internal finite tree embeddings. In *Reflections on the Foundations of Mathematics: Essays in Honor of Solomon Feferman*, Lecture Notes in Logic, pages 60–91. Cambridge University Press, 2002.

12 M. Fujiwara. Kőnig's lemma, weak Kőnig's lemma, and the decidable fan theorem. *Mathematical Logic Quarterly*, 67(2):241–257, 2021. `doi:10.1002/malq.202000020`.

13 J. H. Gallier. What's so special about Kruskal's theorem and the ordinal Γ_0? A survey of some results in proof theory. *Annals of Pure and Applied Logic*, 53(3):199–260, 1991. `doi:10.1016/0168-0072(91)90022-E`.

14 W. Hanf. Nonrecursive Tilings of the Plane. I. *The Journal of Symbolic Logic*, 39(2):283–285, 1974. `doi:10.2307/2272640`.

15 H. Ishihara. Weak Kőnig's Lemma Implies Brouwer's Fan Theorem: A Direct Proof. *Notre Dame Journal of Formal Logic*, 47(2):249–252, 2006. `doi:10.1305/ndjfl/1153858649`.

16 S. C. Kleene and R. E. Vesley. *The Foundations of Intuitionistic Mathematics: Especially in Relation to Recursive Functions.* North-Holland Publishing Co., 1965.

17 S. Kripke. The Problem of Entailment (abstract). *Journal of Symbolic Logic*, 24:324, 1959.

18 D. Kőnig. Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Sci. Math. (Szeged)*, 3(2–3):121–130, 1927.

19 D. Larchey-Wendling. Constructive substitutes for Kőnig's lemma (artifact). Software, NARCO (ANR-21-CE48-0011), swhId: `swh:1:dir:611c848b0dbc19c9de20744122776440c00e413d` (visited on 2025-05-25). URL: `https://github.com/DmxLarchey/Constructive-Konig`, `doi:10.4230/artifacts.23151`.

20 D. Larchey-Wendling. Constructive Decision via Redundancy-Free Proof-Search. *Journal of Automated Reasoning*, 64:1197–1219, 2020. `doi:10.1007/s10817-020-09555-y`.

21 D. Larchey-Wendling. Quasi Morphisms for Almost Full Relations. In *30th International Conference on Types for Proofs and Programs, TYPES*, 2024. URL: `https://easychair.org/publications/preprint/M3Km`.

22 D. Larchey-Wendling. The Coq-Kruskal project, April 2024. URL: `https://github.com/DmxLarchey/Coq-Kruskal`.

23 D. Larchey-Wendling. The Friedman-TREE project, November 2024. URL: `https://github.com/DmxLarchey/Friedman-TREE`.

24 D. Myers. Nonrecursive Tilings of the Plane. II. *The Journal of Symbolic Logic*, 39(2):286–294, 1974. `doi:10.2307/2272641`.

25 C. Sacerdoti Coen and S. Valentini. General Recursion and Formal Topology. In *PAR-10. Partiality and Recursion in Interactive Theorem Provers*, volume 5 of *EPiC Series in Computing*, pages 72–83. EasyChair, 2012. `doi:10.29007/hl75`.

26 S. G. Simpson. *Subsystems of Second Order Arithmetic.* Perspectives in Logic. Cambridge University Press, 2 edition, 2009.

27 A. S. Troelstra. Note on the Fan Theorem. *The Journal of Symbolic Logic*, 39(3):584–596, 1974. doi:10.2307/2272902.

28 A. S. Troelstra. Some Models for Intuitionistic Finite Type Arithmetic with Fan Functional. *The Journal of Symbolic Logic*, 42(2):194–202, 1977. doi:10.2307/2272120.

29 W. Veldman. Brouwer's Real Thesis on Bars. *Philosophia Scientiæ*, pages 21–42, 2006. doi:10.4000/philosophiascientiae.404.

30 W. Veldman. Brouwer's Fan Theorem as an axiom and as a contrast to Kleene's alternative. *Arch. Math. Logic*, 53:621–693, 2014. doi:10.1007/s00153-014-0384-9.

31 D. Vytiniotis, T. Coquand, and D. Wahlstedt. Stop When You Are Almost-Full. In *Interactive Theorem Proving*, pages 250–265. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-32347-8_17.

A Foundation for Synthetic Stone Duality

Felix Cherubini
University of Gothenburg, Sweden
Chalmers University of Technology, Gothenburg, Sweden

Thierry Coquand
University of Gothenburg, Sweden
Chalmers University of Technology, Gothenburg, Sweden

Freek Geerligs
University of Gothenburg, Sweden
Chalmers University of Technology, Gothenburg, Sweden

Hugo Moeneclaey
University of Gothenburg, Sweden
Chalmers University of Technology, Gothenburg, Sweden

Abstract

The language of homotopy type theory has proved to be an appropriate internal language for various higher toposes, for example for the Zariski topos in Synthetic Algebraic Geometry. This paper aims to do the same for the higher topos of light condensed anima of Dustin Clausen and Peter Scholze. This seems to be an appropriate setting for synthetic topology in the style of Martín Escardó.

We use homotopy type theory extended with 4 axioms. We prove Markov's principle, LLPO and the negation of WLPO. Then we define a type of open propositions, inducing a topology on any type such that any map is continuous. We give a synthetic definition of second countable Stone and compact Hausdorff spaces, and show that their induced topologies are as expected. This means that any map from e.g. the unit interval $\mathbb{I}$ to itself is continuous in the usual epsilon-delta sense.

With the usual definition of cohomology in homotopy type theory, we show that $H^1(S,\mathbb{Z}) = 0$ for S Stone and that $H^1(X,\mathbb{Z})$ for X compact Hausdorff can be computed using Čech cohomology. We use this to prove $H^1(\mathbb{I}^1,\mathbb{Z}) = 0$ and $H^1(\mathbb{S}^1,\mathbb{Z}) = \mathbb{Z}$ where $\mathbb{S}^1$ is the set $\mathbb{R}/\mathbb{Z}$. As an application, we give a synthetic proof of Brouwer's fixed-point theorem.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Homotopy Type Theory, Synthetic Topology, Cohomology

Digital Object Identifier 10.4230/LIPIcs.TYPES.2024.3

Funding ForCUTT project, ERC advanced grant 101053291.

Acknowledgements The idea of proving Theorem 4.18 by using the topological characterization of Stone spaces as totally disconnected compact Hausdorff spaces was explained to us by Martín Escardó. We profited a lot from a discussion with Reid Barton and Johann Commelin. The fact that Markov's principle (Corollary 1.16) holds was noticed by David Wärn. We had an interesting discussion with Bas Spitters on the topic of the article at TYPES 2024.

Introduction

The language of homotopy type theory consists of dependent type theory enriched with the univalence axiom and higher inductive types. It has proven exceptionnally well-suited to a synthetic development of homotopy theory [13]. It also provides a framework precise enough to analyze categorical models of type theory [20]. Moreover, arguments in this language can be represented in proof assistants rather directly. In this article we use homotopy type theory to give a synthetic development of topology, analogous to the synthetic development of algebraic geometry in [4].

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg; Article No. 3; pp. 3:1–3:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We introduce four axioms inspired by the light condensed sets introduced in [5]. Interestingly, our axioms have strong connections with constructive mathematics [3], in particular constructive reverse mathematics [11, 7]. Indeed they imply several of Brouwer's principles (e.g. any real function on the unit interval is continuous, the celebrated fan theorem), as well as not intuitionistically valid principles (Markov's Principle, the so-called Lesser Limited Principle of Omniscience).

Our axioms also closely align with the program of Synthetic Topology [9, 12, 19, 10, 21]. Indeed we have a dominance of open propositions, so that any type comes with an induced topology. Using this induced topology, we manage to capture synthetically the notion of second-countable compact Hausdorff spaces. While working on our axioms, we learnt about [2], which provides a similar axiomatisation in extensional type theory. We show that some of their axioms are consequences of ours. For example[1], we can define in our setting the notion of overtly discrete types, which is dual to the notion of compact Hausdorff spaces.

A central theme of homotopy type theory is that the notion of *type* is more general than the notion of *set*. We illustrate this theme in this work. Indeed we can form the types of Stone spaces and of compact Hausdorff spaces, which are not sets but rather a groupoids. Moreover these spaces are closed under Σ-type types, which would be impossible to formulate in the traditional setting. Additionally, we can leverage higher types by using the elegant definition of cohomology groups in homotopy type theory [13]. We then prove a special case of a theorem of Dyckhoff [8] describing the cohomology of compact Hausdorff spaces. As an application, we give a synthetic proof of Brouwer's fixed point theorem, similar to the proof of an approximated form in [16].

We expect our axioms to be validated by the interpretation of homotopy type theory into the higher topos of light condensed anima [17], although checking this rigorously is still work in progress. We even expect this to be valid in a constructive metatheory, using [6]. It is important to stress that our axioms only capture the properties of light condensed anima that are *internally* valid. Since David Wärn [23] has proved that an important property of condensed abelian groups is *not* valid internally, this means that we cannot prove it in our setting. We also conjecture that the present axiom system is *complete* for the properties that are internally valid.

1 Stone duality

1.1 Preliminaries

▶ Remark 1.1. For X any type, a subtype U of X is a family of propositions over X. We write $U \subseteq X$. If X is a set, we call U a subset. Given $x : X$ we sometimes write $x \in U$ instead of $U(x)$. For subtypes $A, B \subseteq X$, we write $A \subseteq B$ for pointwise implication. We will freely switch between a subtype $U \subseteq X$ and the corresponding embedding $\sum_{x:X} U(x) \hookrightarrow X$. In particular, if we write $x : U$ we mean $x : X$ such that $U(x)$.

▶ **Definition 1.2.** *A type is countable if and only if it is merely equal to some decidable subset of* $\mathbb{N}$.

▶ **Definition 1.3.** *For I a set we write $2[I]$ for the free Boolean algebra on I. A Boolean algebra B is countably presented if there exist countable sets I, J with generators $g : I \to B$ and relations $f : J \to 2[I]$ such that g induces an equivalence between $2[I]/(f_j)_{j:J}$ and B.*

[1] We can actually prove all of their axioms, from which their *directed univalence* follows. This will be presented in a following paper.

▶ Remark 1.4. Any countably presented algebra is merely of the form $2[\mathbb{N}]/(r_n)_{n:\mathbb{N}}$.

▶ Remark 1.5. We denote the type of countably presented Boolean algebras by Boole_ω. This type does not depend on a choice of universe. Moreover Boole_ω has a natural category structure.

▶ **Example 1.6.** If both the set of generators and relations are empty, we get the Boolean algebra 2. Its underlying set is $\{0, 1\}$ with $0 \neq 1$. We have that 2 is initial in Boole_ω.

▶ **Definition 1.7.** *For B a countably presented Boolean algebra, we define the spectrum* $\mathrm{Sp}(B)$ *as the set* $\mathrm{Hom}(B, 2)$ *of Boolean morphisms from B to 2. Any type which is merely equivalent to some spectrum is called a Stone space.*

▶ **Example 1.8.**

(i) There is only one Boolean morphism from 2 to 2, thus $\mathrm{Sp}(2)$ is the singleton type $\top$.

(ii) The trivial Boolean algebra is presented as $2/(1)$. We have $0 = 1$ in the trivial Boolean algebra, so there cannot be a map from it into 2 preserving both 0 and 1. Therefore the corresponding Stone space is the empty type $\bot$.

(iii) The type $\mathrm{Sp}(2[\mathbb{N}])$ is called the Cantor space. It is equivalent to the set of binary sequences $2^{\mathbb{N}}$. Given $\alpha : \mathrm{Sp}(2[\mathbb{N}])$ and $n : \mathbb{N}$, we write α_n for $\alpha(g_n)$, the n-th bit of the corresponding binary sequence.

(iv) We denote by B_∞ the Boolean algebra generated by $(g_n)_{n:\mathbb{N}}$ quotiented by the relations $g_m \wedge g_n = 0$ for $n \neq m$. A morphism $B_\infty \to 2$ corresponds to a function $\mathbb{N} \to 2$ that hits 1 at most once. We denote $\mathrm{Sp}(B_\infty)$ by $\mathbb{N}_\infty$. For $\alpha : \mathbb{N}_\infty$ and $n : \mathbb{N}$ we write α_n for $\alpha(g_n)$. For $n : \mathbb{N}$, we define $n : \mathbb{N}_\infty$ as the unique $\alpha : \mathbb{N}_\infty$ such that $\alpha_n = 1$. We define $\infty : \mathbb{N}_\infty$ as the unique $\alpha : \mathbb{N}_\infty$ such that $\alpha_n = 0$ for all $n : \mathbb{N}$.
By conjunctive normal form, any element of B_∞ can be written uniquely as $\bigvee_{i:I} g_n$ or as $\bigwedge_{i:I} \neg g_n$ for some finite $I \subseteq \mathbb{N}$.

▶ **Lemma 1.9.** *Given $\alpha : 2^{\mathbb{N}}$, we have an equivalence of propositions:*

$$(\forall_{n:\mathbb{N}}\ \alpha_n = 0) \leftrightarrow \mathrm{Sp}(2/(\alpha_n)_{n:\mathbb{N}}).$$

Proof. There is only one Boolean morphism $x : 2 \to 2$, and it satisfies $x(\alpha_n) = 0$ for all $n : \mathbb{N}$ if and only if $\alpha_n = 0$ for all $n : \mathbb{N}$. ◀

1.2 Axioms

▶ **Axiom 1** (Stone duality). *For all B : Boole_ω, the evaluation map $B \to 2^{\mathrm{Sp}(B)}$ is an isomorphism.*

▶ **Axiom 2** (Surjections are formal surjections). *For all morphism $g : B \to C$ in Boole_ω, we have that g is injective if and only if $(-) \circ g : \mathrm{Sp}(C) \to \mathrm{Sp}(B)$ is surjective.*

▶ **Axiom 3** (Local choice). *For all B : Boole_ω and type family P over $\mathrm{Sp}(B)$ such that $\Pi_{s:\mathrm{Sp}(B)} \|P(s)\|$, there merely exists some C : Boole_ω and surjection $q : \mathrm{Sp}(C) \to \mathrm{Sp}(B)$ such that $\Pi_{t:\mathrm{Sp}(C)} P(q(t))$.*

▶ **Axiom 4** (Dependent choice). *For all types $(E_n)_{n:\mathbb{N}}$ with surjections $E_{n+1} \twoheadrightarrow E_n$ for all $n : \mathbb{N}$, the projection from the sequential limit $\lim_k E_k$ to E_0 is surjective.*

1.3 Anti-equivalence of Boole_ω and Stone

By Axiom 1, the map Sp is an embedding of Boole_ω into any universe of types. We denote its image by Stone.

▶ Remark 1.10. Stone spaces will take over the role of the affine schemes from [4], so let us repeat some results here. Analogously to Lemma 3.1.2 of [4], for $X : \mathsf{Stone}$, Axiom 1 tells us that $X = \mathrm{Sp}(2^X)$. Proposition 2.2.1 of [4] now says that Sp gives a natural equivalence

$$\mathrm{Hom}(A, B) = (\mathrm{Sp}(B) \to \mathrm{Sp}(A))$$

By the above and Lemma 9.4.5 of [13], the map Sp defines a dual equivalence of categories between Boole_ω and Stone. In particular the spectrum of any colimit in Boole_ω is the limit of the spectrum of the opposite diagram.

▶ Remark 1.11. Axiom 3 can also be formulated as follows: Given $S : \mathsf{Stone}$ with E, F arbitrary types, a map $f : S \to F$ and a surjection $e : E \twoheadrightarrow F$, there exists a Stone space T, a surjective map $T \twoheadrightarrow S$ and an arrow $T \to E$ making the following diagram commute:

$$\begin{tikzcd}
T \arrow[r, dashed] \arrow[d, dashed, two heads] & E \arrow[d, two heads, "e"] \\
S \arrow[r, "f"'] & F
\end{tikzcd}$$

▶ **Lemma 1.12.** *For $B : \mathsf{Boole}_\omega$, we have $0 =_B 1$ if and only if $\neg\,\mathrm{Sp}(B)$.*

Proof. If $0 =_B 1$, there is no map in $B \to 2$ preserving both 0 and 1, thus $\neg\,\mathrm{Sp}(B)$. Conversely, if $\neg\,\mathrm{Sp}(B)$ then $\mathrm{Sp}(B) = \bot$. Since $\bot$ is the spectrum of the trivial Boolean algebra and Sp is an embedding, we conclude that B is the trivial Boolean algebra, hence $0 =_B 1$. ◀

▶ **Corollary 1.13.** *For $S : \mathsf{Stone}$, we have that $\neg\neg S \to \|S\|$*

Proof. Let $B : \mathsf{Boole}_\omega$ and suppose $\neg\neg\,\mathrm{Sp}(B)$. By Lemma 1.12 we have that $0 \neq_B 1$, therefore the morphism $2 \to B$ is injective. By Axiom 2 the map $\mathrm{Sp}(B) \to \mathrm{Sp}(2)$ is surjective, thus $\mathrm{Sp}(B)$ is merely inhabited. ◀

1.4 Principles of omniscience

The so-called principles of omniscience are all weaker than the law of excluded middle (LEM), and help measure how close a logical system is to satisfying LEM [7, 11]. In this section, we will show that two such principles hold (MP and LLPO), and that another one fails (WLPO).

▶ **Theorem 1.14** (The negation of the weak lesser principle of omniscience ($\neg$WLPO)).

$$\neg\forall_{\alpha:2^{\mathbb{N}}}((\forall_{n:\mathbb{N}}\,\alpha_n = 0) \vee \neg(\forall_{n:\mathbb{N}}\,\alpha_n = 0))$$

Proof. We will prove that any decidable property of binary sequences is determined by a finite prefix of fixed length, contradicting $\forall_{n:\mathbb{N}}\,\alpha_n = 0$ being decidable for all α. Indeed assume $f : 2^{\mathbb{N}} \to 2$ such that $f(\alpha) = 0$ if and only if $\forall_{n:\mathbb{N}}\ \alpha_n = 0$. By Axiom 1, there is some $c : 2[\mathbb{N}]$ with $f(\alpha) = 0$ if and only if $\alpha(c) = 0$. There exists $k : \mathbb{N}$ such that c is expressed in terms of the generators $(g_n)_{n \leq k}$. Now consider $\beta, \gamma : 2^{\mathbb{N}}$ given by $\beta(g_n) = 0$ for all $n : \mathbb{N}$ and $\gamma(g_n) = 0$ if and only if $n \leq k$. As β and γ are equal on $(g_n)_{n \leq k}$, we have $\beta(c) = \gamma(c)$. However, $f(\beta) = 0$ and $f(\gamma) = 1$, giving a contradiction. ◀

▶ **Theorem 1.15.** *For all $\alpha : \mathbb{N}_\infty$, we have that*

$$(\neg(\forall_{n:\mathbb{N}}\, \alpha_n = 0)) \to \Sigma_{n:\mathbb{N}}\, \alpha_n = 1$$

Proof. By Lemma 1.9, we have that $\neg(\forall_{n:\mathbb{N}}\, \alpha_n = 0)$ implies that $\mathrm{Sp}(2/(\alpha_n)_{n:\mathbb{N}})$ is empty. Hence $2/(\alpha_n)_{n:\mathbb{N}}$ is trivial by Lemma 1.12. Then there exists $k : \mathbb{N}$ such that $\bigvee_{i\leq k} \alpha_i = 1$. As $\alpha_i = 1$ for at most one $i : \mathbb{N}$, there exists a unique $n : \mathbb{N}$ with $\alpha_n = 1$. ◀

▶ **Corollary 1.16** (Markov's principle (MP))**.** *For all $\alpha : 2^{\mathbb{N}}$, we have that*

$$(\neg(\forall_{n:\mathbb{N}}\, \alpha_n = 0)) \to \Sigma_{n:\mathbb{N}}\, \alpha_n = 1$$

Proof. Given $\alpha : 2^{\mathbb{N}}$, consider the sequence $\alpha' : \mathbb{N}_\infty$ satisfying $\alpha'_n = 1$ if and only if n is minimal with $\alpha_n = 1$. Then apply the above theorem. ◀

▶ **Theorem 1.17** (The lesser limited principle of omniscience (LLPO))**.** *For all $\alpha : \mathbb{N}_\infty$, we have that*

$$(\forall_{k:\mathbb{N}}\, \alpha_{2k} = 0) \vee (\forall_{k:\mathbb{N}}\, \alpha_{2k+1} = 0)$$

Proof. Define $f : B_\infty \to B_\infty \times B_\infty$ on generators as follows

$$f(g_n) = \begin{cases} (g_k, 0) \text{ if } n = 2k \\ (0, g_k) \text{ if } n = 2k+1 \end{cases}$$

Note that f is a well-defined morphism in Boole_ω as $f(g_n) \wedge f(g_m) = 0$ whenever $m \neq n$. We claim that f is injective. If $I \subseteq \mathbb{N}$, write $I_0 = \{k \mid 2k \in I\}, I_1 = \{k \mid 2k+1 \in I\}$. Recall that any $x : B_\infty$ is of the form $\bigvee_{i\in I} g_i$ or $\bigwedge_{i\in I} \neg g_i$ for some finite set I.

- If $x = \bigvee_{i\in I} g_i$, then $f(x) = (\bigvee_{i\in I_0} g_i, \bigvee_{i\in I_1} g_i)$. So if $f(x) = 0$, then $I_0 = I_1 = I = \emptyset$ and $x = 0$.
- Suppose $x = \bigwedge_{i\in I} \neg g_i$. Then $f(x) = (\bigwedge_{i\in I_0} \neg g_i, \bigwedge_{i\in I_1} \neg g_i)$, so $f(x) \neq 0$.

By Axiom 2, we have that f corresponds to a surjection $s : \mathbb{N}_\infty + \mathbb{N}_\infty \to \mathbb{N}_\infty$. Thus for $\alpha : \mathbb{N}_\infty$, there exists some $x : \mathbb{N}_\infty + \mathbb{N}_\infty$ such that $s(x) = \alpha$. If $x = \mathrm{inl}(\beta)$, then for any $k : \mathbb{N}$ we have that

$$\alpha_{2k+1} = s(x)_{2k+1} = x(f(g_{2k+1})) = \mathrm{inl}(\beta)(0, g_k) = \beta(0) = 0.$$

Similarly, if $x = \mathrm{inr}(\beta)$, we have that $\alpha_{2k} = 0$ for all $k : \mathbb{N}$. ◀

The surjection $s : \mathbb{N}_\infty + \mathbb{N}_\infty \to \mathbb{N}_\infty$ above does not have a section. Indeed:

▶ **Lemma 1.18.** *The function f defined above does not have a retraction.*

Proof. Suppose $r : B_\infty \times B_\infty \to B_\infty$ is a retraction of f. Then $r(0, g_k) = g_{2k+1}$ and $r(g_k, 0) = g_{2k}$. Note that $r(0, 1) \geq r(0, g_k) = g_{2k+1}$ for all $k : \mathbb{N}$. As a consequence, $r(0, 1)$ is of the form $\bigwedge_{i\in I} \neg g_i$ for some finite set I. By similar reasoning so is $r(1, 0)$. But then

$$r(0,1) \wedge r(1,0) = r((1,0) \wedge (0,1)) = r(0,0) = 0.$$

This is a contradiction. ◀

1.5 Open and closed propositions

Open (resp. closed) propositions are defined as countable disjunctions (resp. conjunctions) of decidable propositions. In this section we will study their logical properties.

▶ **Definition 1.19.** *A proposition P is open (resp. closed) if there exists some $\alpha : 2^{\mathbb{N}}$ such that $P \leftrightarrow \exists_{n:\mathbb{N}}\alpha_n = 0$ (resp. $P \leftrightarrow \forall_{n:\mathbb{N}}\alpha_n = 0$). We denote by* Open *and* Closed *the types of open and closed propositions.*

▶ Remark 1.20. The negation of an open proposition is closed, and by MP (Corollary 1.16), the negation of a closed proposition is open. Moreover both open and closed propositions are $\neg\neg$-stable. By $\neg$WLPO (Theorem 1.14), not every closed proposition is decidable. Therefore, not every open proposition is decidable. Every decidable proposition is both open and closed.

▶ **Lemma 1.21.** *We have the following:*

- *Closed propositions are stable under countable conjunctions and finite disjunctions.*
- *Open propositions are stable under countable disjunctions and finite conjunctions.*

Proof. All statements but the one about finite disjunctions have similar proofs, so we only present the proof that closed propositions are stable under countable conjunctions. Let $(P_n)_{n:\mathbb{N}}$ be a countable family of closed propositions. By countable choice, for each $n : \mathbb{N}$ we have an $\alpha_n : 2^{\mathbb{N}}$ such that $P_n \leftrightarrow \forall_{m:\mathbb{N}}\,\alpha_{n,m} = 0$. Consider a surjection $s : \mathbb{N} \twoheadrightarrow \mathbb{N} \times \mathbb{N}$, and let $\beta_k = \alpha_{s(k)}$. Note that $\forall_{k:\mathbb{N}}\,\beta_k = 0$ if and only if $\forall_{n:\mathbb{N}}P_n$.

To prove that closed propositions are closed under finite disjunctions, we use the known fact that LLPO (Theorem 1.17) is equivalent to the statement that for P and Q open, we have that $(\neg P \vee \neg Q) \leftrightarrow \neg(P \wedge Q)$. We conclude using that closed propositions are negations of open propositions, and that the conjunction of two open propositions is open. ◀

From now on we will use the above properties silently.

▶ **Corollary 1.22.** *If a proposition is both open and closed, then it is decidable.*

Proof. If P is open and closed, then $P \vee \neg P$ is open. So it is $\neg\neg$-stable and we conclude from $\neg\neg(P \vee \neg P)$. ◀

▶ **Lemma 1.23.** *For $(P_n)_{n:\mathbb{N}}$ a sequence of closed propositions, we have $\neg\forall_{n:\mathbb{N}}P_n \leftrightarrow \exists_{n:\mathbb{N}}\neg P_n$.*

Proof. Both $\neg\forall_{n:\mathbb{N}}P_n$ and $\exists_{n:\mathbb{N}}\neg P_n$ are open, hence $\neg\neg$-stable. The equivalence follows. ◀

▶ **Lemma 1.24.** *If P is open and Q is closed then $P \to Q$ is closed. If P is closed and Q open, then $P \to Q$ is open.*

Proof. Note that $\neg P \vee Q$ is closed. Using $\neg\neg$-stability we conclude that $(P \to Q) \leftrightarrow (\neg P \vee Q)$. The other proof is similar. ◀

1.6 Types as spaces

The subset Open of the set of propositions induces a topology on every type. This is the viewpoint taken in synthetic topology, from which we borrow terminology [9, 12].

▶ **Definition 1.25.** *Let T be a type, and let $A \subseteq T$ be a subtype. We call $A \subseteq T$ open (resp. closed) if $A(t)$ is open (resp. closed) for all $t : T$.*

▶ Remark 1.26. It follows immediately that the pre-image of an open by any map is open, so that any map is continuous. In Theorem 3.11, we will see that the resulting topology is as expected for Stone spaces. In Lemma 4.27, we will see that the same holds for the unit interval.

2 Overtly discrete spaces

▶ **Definition 2.1.** *We call a type overtly discrete if it is a sequential colimit of finite sets.*

▶ Remark 2.2. It follows from Corollary 7.7 of [18] that overtly discrete types are sets, and that the sequential colimit can be defined as in set theory. We write ODisc for the type of overtly discrete types.

Using dependent choice, we have the following results:

▶ **Lemma 2.3.** *A map between overtly discrete sets is a sequential colimit of maps between finite sets.*

▶ **Lemma 2.4.** *For $f : A \to B$ a sequential colimit of maps of finite sets $f_n : A_n \to B_n$, we have that the factorisation $A \twoheadrightarrow Im(f) \hookrightarrow B$ is the sequential colimit of the factorisations $A_n \twoheadrightarrow Im(f_n) \hookrightarrow B_n$.*

▶ **Corollary 2.5.** *An injective (resp. surjective) map between overtly discrete types is a sequential colimit of injective (resp. surjective) maps between finite sets.*

2.1 Closure properties of ODisc

We can get the following result using Lemma 2.3 and dependent choice.

▶ **Lemma 2.6.** *Overtly discrete types are stable under sequential colimits.*

We have that Σ-types, identity types and propositional truncation commute with sequential colimits (Theorem 5.1, Theorem 7.4 and Corollary 7.7 in [18]). Then by closure of finite sets under these constructors, we can get the following:

▶ **Lemma 2.7.** *Overtly discrete types are stable under Σ-types, identity types and propositional truncations.*

2.2 Open and ODisc

▶ **Lemma 2.8.** *A proposition is open if and only if it is overtly discrete.*

Proof. If P is overtly discrete, then $P \leftrightarrow \exists_{n:\mathbb{N}} \|F_n\|$ with F_n finite sets. But a finite set being inhabited is decidable, hence P is a countable disjunction of decidable propositions, so it is open. Suppose $P \leftrightarrow \exists_{n:\mathbb{N}} \alpha_n = 1$. Let $P_n = \exists_{n \leq k}(\alpha_n = 1)$, which is a decidable proposition, hence a finite set. Then the colimit of P_n is P. ◀

▶ **Corollary 2.9.** *Open propositions are stable under Σ-types.*

▶ **Corollary 2.10** (transitivity of openness)**.** *Let T be a type, let $V \subseteq T$ open and let $W \subseteq V$ open. Then $W \subseteq T$ is open as well.*

▶ Remark 2.11. It follows from Proposition 2.25 of [12] that Open is a dominance in the setting of Synthetic Topology.

▶ **Lemma 2.12.** *A type B is overtly discrete if and only if it is the quotient of a countable set by an open equivalence relation.*

Proof. If B : ODisc is the sequential colimit of finite sets B_n, then B is an open quotient of $(\Sigma_{n:\mathbb{N}} B_n)$. Conversely, assume $B = D/R$ with $D \subseteq \mathbb{N}$ decidable and R open. By dependent choice we get $\alpha : D \to D \to 2^{\mathbb{N}}$ such that $R(x, y) \leftrightarrow \exists_{k:\mathbb{N}} \alpha_{x,y}(k) = 1$. Define $D_n = (D \cap \mathbb{N}_{\leq n})$, and define $R_n : D_n \to D_n \to 2$ as the equivalence relation generated by the relation $\exists_{k \leq n}\, \alpha_{x,y}(k) = 1$. Then the $B_n = D_n/R_n$ are finite sets, and their colimit is B. ◀

2.3 Relating ODisc and Boole$_\omega$

▶ **Lemma 2.13.** *Every countably presented Boolean algebra is a sequential colimit of finite Boolean algebras.*

Proof. Consider a countably presented Boolean algebra of the form $B = 2[\mathbb{N}]/(r_n)_{n:\mathbb{N}}$. For each $n : \mathbb{N}$, let G_n be the union of $\{g_i \mid i \leq n\}$ and the finite set of generators occurring in r_i for some $i \leq n$. Denote $B_n = 2[G_n]/(r_i)_{i \leq n}$. Each B_n is a finite Boolean algebra, and there are canonical maps $B_n \to B_{n+1}$. Then B is the colimit of this sequence. ◀

▶ **Corollary 2.14.** *A Boolean algebra B is overtly discrete if and only if it is countably presented.*

Proof. Assume $B : \mathsf{ODisc}$. By Lemma 2.12, we get a surjection $\mathbb{N} \twoheadrightarrow B$ and that B has open equality. Consider the induced surjective morphism $f : 2[\mathbb{N}] \twoheadrightarrow B$. By countable choice, we get for each $b : 2[\mathbb{N}]$ a sequence $\alpha_b : 2^{\mathbb{N}}$ such that $(f(b) = 0) \leftrightarrow \exists_{k:\mathbb{N}}(\alpha_{b,k} = 1)$. Consider $r : 2[\mathbb{N}] \to \mathbb{N} \to 2[\mathbb{N}]$ given by

$$r(b,k) = \begin{cases} b & \text{if } \alpha_b(k) = 1 \\ 0 & \text{if } \alpha_b(k) = 0 \end{cases}$$

Then $B = 2[\mathbb{N}]/(r(b,k))_{b:2[\mathbb{N}],k:\mathbb{N}}$. The converse comes from Lemma 2.13. ◀

▶ Remark 2.15. By Lemma 2.7 and Corollary 2.14, it follows that any $g : B \to C$ in Boole_ω has an overtly discrete kernel. As a consequence, the kernel is enumerable and $B/Ker(g)$ is in Boole_ω. By uniqueness of epi-mono factorizations and Axiom 2, the factorization $B \twoheadrightarrow B/Ker(g) \hookrightarrow C$ corresponds to $\mathrm{Sp}(C) \twoheadrightarrow \mathrm{Sp}(B/Ker(g)) \hookrightarrow \mathrm{Sp}(B)$.

▶ Remark 2.16. Similarly to Lemma 2.3 and Lemma 2.4, a (resp. surjective, injective) morphism in Boole_ω is a sequential colimit of (resp. surjective, injective) morphisms between finite Boolean algebras.

3 Stone spaces

3.1 Stone spaces as profinite sets

Here we present Stone spaces as sequential limits of finite sets. This is the perspective taken in Condensed Mathematics [15, 1, 5]. Some of the results in this section are versions of the axioms used in [2]. A full proof of all these axioms is part of future work.

▶ **Lemma 3.1.** *Any S : Stone is a sequential limit of finite sets.*

Proof. Assume $B : \mathsf{Boole}_\omega$. By Remark 1.10 and Lemma 2.13, we have that $\mathrm{Sp}(B)$ is a sequential limit of spectra of finite Boolean algebras, which are finite sets. ◀

▶ **Lemma 3.2.** *A sequential limit of finite sets is a Stone space.*

Proof. By Remark 1.10 and Lemma 2.6, we have that Stone is closed under sequential limits, and finite sets are Stone. ◀

▶ **Corollary 3.3.** *Stone spaces are stable under finite limits.*

▶ Remark 3.4. By Remark 2.16 and Axiom 2, maps (resp. surjections, injections) of Stone spaces are sequential limits of maps (resp. surjections, injections) of finite sets.

▶ **Lemma 3.5.** *For $(S_n)_{n:\mathbb{N}}$ a sequence of finite types with $S = \lim_n S_n$ and $k : \mathbb{N}$, we have that $\mathrm{Fin}(k)^S$ is the sequential colimit of $\mathrm{Fin}(k)^{S_n}$.*

Proof. By Remark 1.10 we have $\mathrm{Fin}(k)^S = \mathrm{Hom}(2^k, 2^S)$. Since 2^k is finite, we have that $\mathrm{Hom}(2^k, \_)$ commutes with sequential colimits, therefore $\mathrm{Hom}(2^k, 2^S)$ is the sequential colimit of $\mathrm{Hom}(2^k, 2^{S_n})$. By applying Remark 1.10 again, the latter type is $\mathrm{Fin}(k)^{S_n}$. ◀

▶ **Lemma 3.6.** *For S : Stone and $f : S \to \mathbb{N}$, there exists some $k : \mathbb{N}$ such that f factors through $\mathrm{Fin}(k)$.*

Proof. For each $n : \mathbb{N}$, the fiber of f over n is a decidable subset $f_n : S \to 2$. We must have that $\mathrm{Sp}(2^S/(f_n)_{n:\mathbb{N}}) = \bot$, hence there exists some $k : \mathbb{N}$ with $\bigvee_{n \leq k} f_n =_{2^S} 1$. It follows that $f(s) \leq k$ for all $s : S$ as required. ◀

▶ **Corollary 3.7.** *For $(S_n)_{n:\mathbb{N}}$ a sequence of finite types with $S = \lim_n S_n$, we have that $\mathbb{N}^S$ is the sequential colimit of $\mathbb{N}^{S_n}$.*

Proof. By Lemma 3.6 we have that $\mathbb{N}^S$ is the sequential colimit of $\mathrm{Fin}(k)^S$. By Lemma 3.5, $\mathrm{Fin}(k)^S$ is the sequential colimit of the $\mathrm{Fin}(k)^{S_n}$ and we can swap the sequential colimits to conclude. ◀

3.2 Closed and Stone

▶ **Corollary 3.8.** *For all S : Stone, the proposition $\|S\|$ is closed.*

Proof. By Lemma 1.12, $\neg S$ is equivalent to $0 =_{2^S} 1$, which is open by Lemma 2.13 and Lemma 2.12. Hence $\neg\neg S$ is a closed proposition, and by Corollary 1.13, so is $\|S\|$. ◀

▶ **Corollary 3.9.** *A proposition P is closed if and only if it is a Stone space.*

Proof. By the above, if S is both a Stone space and a proposition, it is closed. By Lemma 1.9, any closed proposition is Stone. ◀

▶ **Lemma 3.10.** *For all S : Stone and $s, t : S$, the proposition $s = t$ is closed.*

Proof. Suppose $S = \mathrm{Sp}(B)$ and let G be a countable set of generators for B. Then $s = t$ if and only if $s(g) = t(g)$ for all $g : G$. So $s = t$ is a countable conjunction of decidable propositions, hence closed. ◀

3.3 The topology on Stone spaces

▶ **Theorem 3.11.** *Let $A \subseteq S$ be a subset of a Stone space. The following are equivalent:*

(i) *There exists a map $\alpha : S \to 2^{\mathbb{N}}$ such that $A(x) \leftrightarrow \forall_{n:\mathbb{N}}\, \alpha_{x,n} = 0$ for any $x : S$.*

(ii) *There exists a family $(D_n)_{n:\mathbb{N}}$ of decidable subsets of S such that $A = \bigcap_{n:\mathbb{N}} D_n$.*

(iii) *There exists a Stone space T and some embedding $T \to S$ whose image is A.*

(iv) *There exists a Stone space T and some map $T \to S$ whose image is A.*

(v) *A is closed.*

Proof.

- $(i) \leftrightarrow (ii)$. D_n and α can be defined from each other by $D_n(x) \leftrightarrow (\alpha_{x,n} = 0)$. Then observe that

$$x \in \bigcap_{n:\mathbb{N}} D_n \leftrightarrow \forall_{n:\mathbb{N}}(\alpha_{x,n} = 0).$$

- *(ii)* → *(iii)*. Let $S = \mathrm{Sp}(B)$. By Axiom 1, we have $(d_n)_{n:\mathbb{N}}$ in B such that $D_n = \{x : S \mid x(d_n) = 0\}$. Let $C = B/(d_n)_{n:\mathbb{N}}$. Then $\mathrm{Sp}(C) \to S$ is as desired because:
$$\mathrm{Sp}(C) = \{x : S \mid \forall_{n:\mathbb{N}}\, x(d_n) = 0\} = \bigcap_{n:\mathbb{N}} D_n.$$
- *(iii)* → *(iv)*. Immediate.
- *(iv)* → *(ii)*. Assume $f : T \to S$ corresponds to $g : B \to C$ in Boole_ω. By Remark 2.15, $f(T) = \mathrm{Sp}(B/Ker(g))$ and there exists a surjection $d : \mathbb{N} \to Ker(g)$. For $n : \mathbb{N}$, we denote by D_n the decidable subset of S corresponding to d_n. Then we have that $\mathrm{Sp}(B/Ker(g)) = \bigcap_{n:\mathbb{N}} D_n$.
- *(i)* → *(v)*. By definition.
- *(v)* → *(iv)*. We have a surjection $2^{\mathbb{N}} \twoheadrightarrow \mathsf{Closed}$ defined by $\alpha \mapsto \forall_{n:\mathbb{N}}\, \alpha_n = 0$. Remark 1.11 gives us that there merely exists T, e, β. as follows:
$$\begin{tikzcd} T \arrow[r, "\beta"] \arrow[d, two heads, "e"'] & 2^{\mathbb{N}} \arrow[d, two heads] \\ S \arrow[r, "A"'] & \mathsf{Closed} \end{tikzcd}$$
Define $B(x) \leftrightarrow \forall_{n:\mathbb{N}}\, \beta_{x,n} = 0$. As *(i)* → *(iii)* by the above, B is the image of some Stone space. Note that A is the image of B, thus A is the image of some Stone space. ◀

▶ **Corollary 3.12.** *Closed subtypes of Stone spaces are Stone.*

▶ **Corollary 3.13.** *For S :* Stone *and $A \subseteq S$ closed, we have that $\exists_{x:S} A(x)$ is closed.*

Proof. By Corollary 3.12, we have that $\Sigma_{x:S} A(x)$ is Stone, so its truncation is closed by Corollary 3.8. ◀

▶ **Corollary 3.14.** *Closed propositions are closed under sigma types.*

Proof. Let P : Closed and $Q : P \to$ Closed. Then $\Sigma_{p:P} Q(p) \leftrightarrow \exists_{p:P} Q(p)$. As P is Stone by Corollary 3.9, Corollary 3.13 gives that $\Sigma_{p:P} Q(p)$ is closed. ◀

▶ Remark 3.15. Analogously to Corollary 2.10 and Remark 2.11, it follows that closedness is transitive and Closed forms a dominance.

▶ **Lemma 3.16.** *Assume S :* Stone *with $F, G : S \to$* Closed *such that $F \cap G = \emptyset$. Then there exists a decidable subset $D : S \to 2$ such $F \subseteq D, G \subseteq \neg D$.*

Proof. Assume $S = \mathrm{Sp}(B)$. By Theorem 3.11, for all $n : \mathbb{N}$ there is $f_n, g_n : B$ such that $x \in F$ if and only if $\forall_{n:\mathbb{N}}\, x(f_n) = 0$ and $y \in G$ if and only if $\forall_{n:\mathbb{N}}\, y(g_n) = 0$. Denote by h the sequence defined by $h_{2k} = f_k$ and $h_{2k+1} = g_k$. Then $\mathrm{Sp}(B/(h_k)_{k:\mathbb{N}}) = F \cap G = \emptyset$, so by Lemma 1.12 there exists finite sets $I, J \subseteq \mathbb{N}$ such that $1 =_B ((\bigvee_{i:I} f_i) \vee (\bigvee_{j:J} g_j))$. If $y \in F$, then $y(f_i) = 0$ for all $i : I$, hence $y(\bigvee_{j:J} g_j) = 1$ If $x \in G$, we have $x(\bigvee_{j:J} g_j) = 0$. Thus we can define the required D by $D(x) \leftrightarrow x(\bigvee_{j:J} g_j) = 1$. ◀

4 Compact Hausdorff spaces

▶ **Definition 4.1.** *A type X is called a compact Hausdorff space if its identity types are closed propositions and there exists some S :* Stone *with a surjection $S \twoheadrightarrow X$. We write* CHaus *for the type of compact Hausdorff spaces.*

4.1 Topology on compact Hausdorff spaces

▶ **Lemma 4.2.** *Let $X : \mathsf{CHaus}$, $S : \mathsf{Stone}$ and $q : S \twoheadrightarrow X$ surjective. Then $A \subseteq X$ is closed if and only if it is the image of a closed subset of S by q.*

Proof. As q is surjective, we have $q(q^{-1}(A)) = A$. If A is closed, so is $q^{-1}(A)$ and hence A is the image of a closed subset of S. Conversely, let $B \subseteq S$ be closed. Then $x \in q(B)$ if and only if

$$\exists_{s:S}(B(s) \wedge q(s) = x).$$

Hence by Corollary 3.13, $q(B)$ is closed. ◀

The next two corollaries mean that compact Hausdorff spaces are compact in the sense of Synthetic Topology.

▶ **Corollary 4.3.** *Assume given $X : \mathsf{CHaus}$ with $A \subseteq X$ closed. Then $\exists_{x:X}A(x)$ is closed, and equivalent to $A \neq \emptyset$.*

Proof. From Lemma 4.2 and Theorem 3.11, it follows that $A \subseteq X$ is closed if and only if it is the image of a map $T \to X$ for some $T : \mathsf{Stone}$. Then $\exists_{x:X}A(x)$ if and only $\|T\|$, which is closed by Corollary 3.8. Therefore $\exists_{x:X}A(x)$ is $\neg\neg$-stable and equivalent to $A \neq \emptyset$. ◀

▶ **Corollary 4.4.** *Assume given $X : \mathsf{CHaus}$ with $U \subseteq X$ open. Then $\forall_{x:X}U(x)$ is open.*

The next lemma means that compact Hausdorff spaces are not too far from being compact in the classical sense.

▶ **Lemma 4.5.** *Given $X : \mathsf{CHaus}$ and $C_n : X \to \mathsf{Closed}$ closed subsets such that $\bigcap_{n:\mathbb{N}} C_n = \emptyset$, there is some $k : \mathbb{N}$ with $\bigcap_{n \leq k} C_n = \emptyset$.*

Proof. By Lemma 4.2 it is enough to prove the result when X is Stone, and by Theorem 3.11 we can assume C_n decidable. So assume $X = \mathrm{Sp}(B)$ and $c_n : B$ such that

$$C_n = \{x : B \to 2 \mid x(c_n) = 0\}.$$

Then we have that

$$\mathrm{Sp}(B/(c_n)_{n:\mathbb{N}}) \simeq \bigcap_{n:\mathbb{N}} C_n = \emptyset.$$

Hence $0 = 1$ in $B/(c_n)_{n:\mathbb{N}}$ and there is some $k : \mathbb{N}$ with $\bigvee_{n \leq k} c_n = 1$, which means that

$$\emptyset = \mathrm{Sp}(B/(c_n)_{n \leq k}) \simeq \bigcap_{n \leq k} C_n$$

as required. ◀

▶ **Corollary 4.6.** *Let $X, Y : \mathsf{CHaus}$ and $f : X \to Y$. Suppose $(G_n)_{n:\mathbb{N}}$ is a decreasing sequence of closed subsets of X. Then $f(\bigcap_{n:\mathbb{N}} G_n) = \bigcap_{n:\mathbb{N}} f(G_n)$.*

Proof. It is always the case that $f(\bigcap_{n:\mathbb{N}} G_n) \subseteq \bigcap_{n:\mathbb{N}} f(G_n)$. For the converse direction, suppose that $y \in f(G_n)$ for all $n : \mathbb{N}$. We define $F \subseteq X$ closed by $F = f^{-1}(y)$. Then for all $n : \mathbb{N}$ we have that $F \cap G_n$ is non-empty. By Lemma 4.5 this implies that $\bigcap_{n:\mathbb{N}}(F \cap G_n) \neq \emptyset$. By Corollary 4.3, we have that $\bigcap_{n:\mathbb{N}}(F \cap G_n)$ is merely inhabited. Thus $y \in f(\bigcap_{n:\mathbb{N}} G_n)$ as required. ◀

▶ **Corollary 4.7.** *Let $A \subseteq X$ be a subset of a compact Hausdorff space and $p : S \twoheadrightarrow X$ be a surjective map with $S : \mathsf{Stone}$. Then A is closed (resp. open) if and only if there exists a sequence $(D_n)_{n:\mathbb{N}}$ of decidable subsets of S such that $A = \bigcap_{n:\mathbb{N}} p(D_n)$ (resp. $A = \bigcup_{n:\mathbb{N}} \neg p(D_n)$).*

Proof. The characterization of closed subsets follows from characterization (ii) in Theorem 3.11, Lemma 4.2 and Corollary 4.6. To deduce the characterization of open subsets we use Remark 1.20 and Lemma 1.23. ◀

▶ Remark 4.8. For $S : \mathsf{Stone}$, there is a surjection $\mathbb{N} \twoheadrightarrow 2^S$. It follows that for any $X : \mathsf{CHaus}$ there is a surjection from $\mathbb{N}$ to a basis of X. Classically this means that X is second countable.

The next lemma means that compact Hausdorff spaces are normal.

▶ **Lemma 4.9.** *Assume $X : \mathsf{CHaus}$ and $A, B \subseteq X$ closed such that $A \cap B = \emptyset$. Then there exist $U, V \subseteq X$ open such that $A \subseteq U$, $B \subseteq V$ and $U \cap V = \emptyset$.*

Proof. Let $q : S \twoheadrightarrow X$ be a surjective map with $S : \mathsf{Stone}$. As $q^{-1}(A)$ and $q^{-1}(B)$ are closed, by Lemma 3.16, there is some $D : S \to 2$ such that $q^{-1}(A) \subseteq D$ and $q^{-1}(B) \subseteq \neg D$. Note that $q(D)$ and $q(\neg D)$ are closed by Lemma 4.2. As $q^{-1}(A) \cap \neg D = \emptyset$, we have that $A \subseteq \neg q(\neg D) := U$. Similarly $B \subseteq \neg q(D) := V$. Then U and V are disjoint because $\neg q(D) \cap \neg q(\neg D) = \neg(q(D) \cup q(\neg D)) = \neg X = \emptyset$. ◀

4.2 Compact Hausdorff spaces are stable under sigma types

▶ **Lemma 4.10.** *A type X is Stone if and only if it is merely a closed subset of $2^{\mathbb{N}}$.*

Proof. By Remark 1.4, any $B : \mathsf{Boole}_\omega$ can be written as $2[\mathbb{N}]/(r_n)_{n:\mathbb{N}}$. By Remark 2.15, the quotient map induces an embedding $\mathrm{Sp}(B) \hookrightarrow \mathrm{Sp}(2[\mathbb{N}]) = 2^{\mathbb{N}}$, which is closed by Theorem 3.11. ◀

▶ **Lemma 4.11.** *Compact Hausdorff spaces are stable under Σ-types.*

Proof. Assume $X : \mathsf{CHaus}$ and $Y : X \to \mathsf{CHaus}$. By Corollary 3.14 we have that identity types in $\Sigma_{x:X} Y(x)$ are closed. By Lemma 4.10 we know that for any $x : X$ there merely exists a closed $C \subseteq 2^{\mathbb{N}}$ with a surjection $\Sigma_{\alpha:2^{\mathbb{N}}} C(\alpha) \twoheadrightarrow Y(x)$. By local choice we merely get $S : \mathsf{Stone}$ with a surjection $p : S \twoheadrightarrow X$ such that for all $s : S$ we have $C_s \subseteq 2^{\mathbb{N}}$ closed and a surjection $\Sigma_{2^{\mathbb{N}}} C_s \twoheadrightarrow Y(p(s))$. This gives a surjection $\Sigma_{s:S,\alpha:2^{\mathbb{N}}} C_s(\alpha) \twoheadrightarrow \Sigma_{x:X} Y_x$ and the source is Stone by Remark 3.4 and Corollary 3.12. ◀

4.3 Stone spaces are stable under sigma types

We will show that Stone spaces are precisely totally disconnected compact Hausdorff spaces. We will use this to prove that a sigma type of Stone spaces is Stone.

▶ **Lemma 4.12.** *Assume $X : \mathsf{CHaus}$, then 2^X is countably presented.*

Proof. There is some surjection $q : S \twoheadrightarrow X$ with $S : \mathsf{Stone}$. This induces an injection of Boolean algebras $2^X \hookrightarrow 2^S$. Note that $a : S \to 2$ lies in 2^X if and only if:

$$\forall_{s,t:S}\ q(s) =_X q(t) \to a(s) = a(t).$$

As equality in X is closed and equality in 2 is decidable, the implication is open for every $s, t : S$. By Corollary 4.4, we conclude that 2^X is an open subalgebra of 2^S. Therefore, it is in ODisc by Lemma 2.8 and Lemma 2.7 and in Boole_ω by Corollary 2.14. ◀

▶ **Definition 4.13.** *For all X : CHaus and $x : X$, we define Q_x the connected component of x as the intersection of all $D \subseteq X$ decidable such that $x \in D$.*

▶ **Lemma 4.14.** *For all X : CHaus with $x : X$, we have that Q_x is a countable intersection of decidable subsets of X.*

Proof. By Lemma 4.12, we can enumerate the elements of 2^X, say as $(D_n)_{n:\mathbb{N}}$. For $n : \mathbb{N}$ we define E_n as D_n if $x \in D_n$ and X otherwise. Then $\cap_{n:\mathbb{N}} E_n = Q_x$. ◀

▶ **Lemma 4.15.** *Assume X : CHaus with $x : X$ and suppose $U \subseteq X$ open with $Q_x \subseteq U$. Then we have some decidable $E \subseteq X$ with $x \in E$ and $E \subseteq U$.*

Proof. By Lemma 4.14, we have $Q_x = \bigcap_{n:\mathbb{N}} D_n$ with $D_n \subseteq X$ decidable. If $Q_x \subseteq U$, then

$$Q_x \cap \neg U = \bigcap_{n:\mathbb{N}} (D_n \cap \neg U) = \emptyset.$$

By Lemma 4.5 there is some $k : \mathbb{N}$ with

$$(\bigcap_{n \le k} D_n) \cap \neg U = \bigcap_{n \le k} (D_n \cap \neg U) = \emptyset.$$

Therefore $\bigcap_{n \le k} D_n \subseteq \neg\neg U$. As U is open, $\neg\neg U = U$ and $E := \bigcap_{n \le k} D_n$ is as desired. ◀

▶ **Lemma 4.16.** *Assume X : CHaus with $x : X$. Then any map in $Q_x \to 2$ is constant.*

Proof. Assume $Q_x = A \cup B$ with A, B decidable and disjoint subsets of Q_x. Assume $x \in A$. By Lemma 4.14, $Q_x \subseteq X$ is closed. Using Remark 3.15, it follows that $A, B \subseteq X$ are closed and disjoint. By Lemma 4.9 there exist U, V disjoint open such that $A \subseteq U$ and $B \subseteq V$. By Lemma 4.15 we have a decidable D such that $Q_x \subseteq D \subseteq U \cup V$. Note that $E := D \cap U = D \cap (\neg V)$ is clopen, hence decidable by Corollary 1.22. But $x \in E$, hence $B \subseteq Q_x \subseteq E$ but $B \cap E = \emptyset$, hence $B = \emptyset$. ◀

▶ **Lemma 4.17.** *Let X : CHaus, then X is Stone if and only $\forall_{x:X}\, Q_x = \{x\}$.*

Proof. By Axiom 1, it is clear that for all $x : S$ with S : Stone we have that $Q_x = \{x\}$. Conversely, assume X : CHaus such that $\forall_{x:X}\, Q_x = \{x\}$. We claim that the evaluation map $e : X \to \mathrm{Sp}(2^X)$ is both injective and surjective, hence an equivalence. Let $x, y : X$ be such that $e(x) = e(y)$, i.e. such that $f(x) = f(y)$ for all $f : 2^X$. Then $y \in Q_x$, hence $x = y$ by assumption. Thus e is injective. Let $q : S \twoheadrightarrow X$ be a surjective map. It induces an injection $2^X \hookrightarrow 2^S$, which by Axiom 2 induces a surjection $p : \mathrm{Sp}(2^S) \twoheadrightarrow \mathrm{Sp}(2^X)$. Note that $e \circ q$ is equal to p so e is surjective. ◀

▶ **Theorem 4.18.** *Assume S : Stone and $T : S \to \mathsf{Stone}$. Then $\Sigma_{x:S} T(x)$ is Stone.*

Proof. By Lemma 4.11 we have that $\Sigma_{x:S} T(x)$ is a compact Hausdorff space. By Lemma 4.17 it is enough to show that for all $x : S$ and $y : T(x)$ we have that $Q_{(x,y)}$ is a singleton. Assume $(x', y') \in Q_{(x,y)}$, then for any map $f : S \to 2$ we have that:

$$f(x) = f \circ \pi_1(x, y) = f \circ \pi_1(x', y') = f(x')$$

so that $x' \in Q_x$ and since S is Stone, by Lemma 4.17 we have that $x = x'$. Therefore we have $Q_{(x,y)} \subseteq \{x\} \times T(x)$. Assume $z, z' : Q_{(x,y)}$, then for any map $g : T(x) \to 2$ we have that $g(z) = g(z')$ by Lemma 4.16. Since $T(x)$ is Stone, we conclude $z = z'$ by Lemma 4.17. ◀

4.4 The unit interval as a compact Hausdorff space

Since we have dependent choice, the unit interval $\mathbb{I} = [0, 1]$ can be defined using Cauchy reals or Dedekind reals. We can freely use results from constructive analysis [3]. As we have $\neg$WLPO, MP and LLPO, we can use the results from constructive reverse mathematics that follow from these principles [11, 7].

▶ **Definition 4.19.** *We define for each $n : \mathbb{N}$ the Stone space 2^n of binary sequences of length n. And we define $cs_n : 2^n \to \mathbb{Q}$ by $cs_n(\alpha) = \sum_{i<n} \frac{\alpha(i)}{2^{i+1}}$. Finally we write $\sim_n$ for the binary relation on 2^n given by $\alpha \sim_n \beta \leftrightarrow |cs_n(\alpha) - cs_n(\beta)| \leq \frac{1}{2^n}$.*

▶ Remark 4.20. The inclusion $\text{Fin}(n) \hookrightarrow \mathbb{N}$ induces a restriction $\_|_n : 2^{\mathbb{N}} \to 2^n$ for each $n : \mathbb{N}$.

▶ **Definition 4.21.** *We define $cs : 2^{\mathbb{N}} \to \mathbb{I}$ as $cs(\alpha) = \sum_{i=0}^{\infty} \frac{\alpha(i)}{2^{i+1}}$.*

▶ **Theorem 4.22.** *The type $\mathbb{I}$ is a compact Hausdorff space.*

Proof. By LLPO, we have that cs is surjective. Note that $cs(\alpha) = cs(\beta)$ if and only if for all $n : \mathbb{N}$ we have $\alpha|_n \sim_n \beta|_n$. This is a countable conjunction of decidable propositions, so that equality in $\mathbb{I}$ is closed. ◀

The following is also given by Definitions 2.7 and 2.10 of [3].

▶ **Definition 4.23.** *Assume given $x, y : \mathbb{I}$ and $\alpha, \beta : 2^{\mathbb{N}}$ such that $x = cs(\alpha), y = cs(\beta)$. Then $x < y$ is the proposition $\exists_{n:\mathbb{N}} cs_n(\alpha) + \frac{1}{2^n} <_{\mathbb{Q}} cs_n(\beta)$, which is independent of the choice of α, β.*

▶ Remark 4.24. For all $x, y : \mathbb{I}$, we have that $x < y$ is an open proposition and that $x \neq y$ is equivalent to $(x < y) \vee (y < x)$.

▶ **Lemma 4.25.** *For all $D \subseteq 2^{\mathbb{N}}$ decidable, we have that $cs(D)$ is a finite union of closed intervals.*

Proof. If D is contains precisely the $\alpha : 2^{\mathbb{N}}$ with a fixed initial segment, $cs(D)$ is a closed interval. Any decidable subset of $2^{\mathbb{N}}$ is a finite union of such subsets. ◀

▶ **Lemma 4.26.** *The complement of a finite union of closed intervals is a finite union of open intervals.*

By Corollary 4.7 we can thus conclude:

▶ **Lemma 4.27.** *Every open $U \subseteq \mathbb{I}$ can be written as a countable union of open intervals.*

It follows that the topology of $\mathbb{I}$ is generated by open intervals, which corresponds to the standard topology on $\mathbb{I}$. Hence our notion of continuity agrees with the ϵ, δ-definition of continuity one would expect and we get the following:

▶ **Theorem 4.28.** *Every function $f : \mathbb{I} \to \mathbb{I}$ is continuous in the ϵ, δ-sense.*

5 Cohomology

In this section we compute $H^1(S, \mathbb{Z}) = 0$ for all S Stone, and show that $H^1(X, \mathbb{Z})$ for X compact Hausdorff can be computed using Čech cohomology. We use this to compute $H^1(\mathbb{I}, \mathbb{Z}) = 0$.

▶ Remark 5.1. We only work with the first cohomology group with coefficients in $\mathbb{Z}$ as it is sufficient for the proof of Brouwer's fixed-point theorem, but the results could be extended to $H^n(X, A)$ for A any family of countably presented abelian groups indexed by X.

▶ Remark 5.2. We write Ab for the type of abelian groups and if G : Ab we write BG for the delooping of G [13, 22]. This means that $H^1(X, G)$ is the set truncation of $X \to \text{B}G$.

5.1 Čech cohomology

▶ **Definition 5.3.** *Given a type S, types T_x for $x : S$ and $A : S \to \mathrm{Ab}$, we define $\check{C}(S, T, A)$ as the chain complex*

$$\prod_{x:S} A_x^{T_x} \xrightarrow{d_0} \prod_{x:S} A_x^{T_x^2} \xrightarrow{d_1} \prod_{x:S} A_x^{T_x^3}$$

where the boundary maps are defined as

$$\begin{aligned} d_0(\alpha)_x(u, v) &= \alpha_x(v) - \alpha_x(u) \\ d_1(\beta)_x(u, v, w) &= \beta_x(v, w) - \beta_x(u, w) + \beta_x(u, v) \end{aligned}$$

▶ **Definition 5.4.** *Given a type S, types T_x for $x : S$ and $A : S \to \mathrm{Ab}$, we define its Čech cohomology groups by*

$$\check{H}^0(S, T, A) = \ker(d_0) \qquad \check{H}^1(S, T, A) = \ker(d_1)/\mathrm{im}(d_0)$$

We call elements of $\ker(d_1)$ cocycles and elements of $\mathrm{im}(d_0)$ coboundaries.

This means that $\check{H}^1(S, T, A) = 0$ if and only if $\check{C}(S, T, A)$ is exact at the middle term. Now we give three general lemmas about Čech complexes.

▶ **Lemma 5.5.** *Assume a type S, types T_x for $x : S$ and $A : S \to \mathrm{Ab}$ with $t : \prod_{x:S} T_x$. Then $\check{H}^1(S, T, A) = 0$.*

Proof. Assume given a cocycle, i.e. $\beta : \prod_{x:S} A_x^{T_x^2}$ such that for all $x : S$ and $u, v, w : T_x$ we have that $\beta_x(u, v) + \beta_x(v, w) = \beta_x(u, w)$. We define $\alpha : \prod_{x:S} A_x^{T_x}$ by $\alpha_x(u) = \beta_x(t_x, u)$. Then for all $x : S$ and $u, v : T_x$ we have that $d_0(\alpha)_x(u, v) = \beta_x(t_x, v) - \beta_x(t_x, u) = \beta_x(u, v)$ so that β is a coboundary. ◀

▶ **Lemma 5.6.** *Given a type S, types T_x for $x : S$ and $A : S \to \mathrm{Ab}$, we have that $\check{H}^1(S, T, \lambda x.A_x^{T_x}) = 0$.*

Proof. Assume given a cocycle, i.e. $\beta : \prod_{x:S} A_x^{T_x^3}$ such that for all $x : S$ and $u, v, w, t : T_x$ we have that $\beta_x(u, v, t) + \beta_x(v, w, t) = \beta_x(u, w, t)$. We define $\alpha : \prod_{x:S} A_x^{T_x^2}$ by $\alpha_x(u, t) = \beta_x(t, u, t)$. Then for all $x : S$ and $u, v, t : T_x$ we have that $d_0(\alpha)_x(u, v, t) = \beta_x(t, v, t) - \beta_x(t, u, t) = \beta_x(u, v, t)$ so that β is a coboundary. ◀

▶ **Lemma 5.7.** *Assume a type S and types T_x for $x : S$ such that $\prod_{x:S} \|T_x\|$ and $A : S \to \mathrm{Ab}$ such that $\check{H}^1(S, T, A) = 0$. Then given $\alpha : \prod_{x:S} \mathrm{B}A_x$ with $\beta : \prod_{x:S} (\alpha(x) = *)^{T_x}$, we can conclude $\alpha = *$.*

Proof. We define $g : \prod_{x:S} A_x^{T_x^2}$ by $g_x(u, v) = \beta_x(v) - \beta_x(u)$. It is a cocycle in the Čech complex, so that by exactness there is $f : \prod_{x:S} A_x^{T_x}$ such that for all $x : S$ and $u, v : T_x$ we have that $g_x(u, v) = f_x(v) - f_x(u)$. Then we define $\beta' : \prod_{x:S} (\alpha(x) = *)^{T_x}$ by $\beta'_x(u) = \beta_x(u) - f_x(u)$ so that for all $x : S$ and $u, v : T_x$ we have that $\beta'_x(u) = \beta'_x(v)$ is equivalent to $f_x(v) - f_x(u) = \beta_x(v) - \beta_x(u)$, which holds by definition. So β' is constant on each T_x and therefore gives $\prod_{x:S} (\alpha(x) = *)^{\|T_x\|}$. By $\prod_{x:S} \|T_x\|$ we conclude $\alpha = *$. ◀

5.2 Cohomology of Stone spaces

▶ **Lemma 5.8.** *Assume given $S : \mathsf{Stone}$ and $T : S \to \mathsf{Stone}$ such that $\prod_{x:S} \|T(x)\|$. Then there exists a sequence of finite types $(S_k)_{k:\mathbb{N}}$ with limit S and a compatible sequence of families of finite types T_k over S_k with $\prod_{x:S_k} \|T_k(x)\|$ and $\lim_k \left(\sum_{x:S_k} T_k(x)\right) = \sum_{x:S} T(x)$.*

Proof. By theorem Theorem 4.18 and the usual correspondence between surjections and families of inhabited types, a family of inhabited Stone spaces over S correspond to a Stone space T with a surjection $T \to S$. Then we conclude using Remark 3.4. ◀

▶ **Lemma 5.9.** *Assume given $S : \mathsf{Stone}$ with $T : S \to \mathsf{Stone}$ such that $\prod_{x:S} \|T_x\|$. Then we have that $\check{H}^1(S, T, \mathbb{Z}) = 0$.*

Proof. We apply Lemma 5.8 to get S_k and T_k finite. Then by Corollary 3.7 we have that $\check{C}(S, T, \mathbb{Z})$ is the sequential colimit of the $\check{C}(S_k, T_k, \mathbb{Z})$. By Lemma 5.5 we have that each of the $\check{C}(S_k, T_k, \mathbb{Z})$ is exact, and a sequential colimit of exact sequences is exact. ◀

▶ **Lemma 5.10.** *Given $S : \mathsf{Stone}$, we have that $H^1(S, \mathbb{Z}) = 0$.*

Proof. Assume given a map $\alpha : S \to \mathrm{B}\mathbb{Z}$. We use local choice to get $T : S \to \mathsf{Stone}$ such that $\prod_{x:S} \|T_x\|$ with $\beta : \prod_{x:S} (\alpha(x) = *)^{T_x}$. Then we conclude by Lemma 5.9 and Lemma 5.7. ◀

▶ **Corollary 5.11.** *For any $S : \mathsf{Stone}$ the canonical map $\mathrm{B}(\mathbb{Z}^S) \to (\mathrm{B}\mathbb{Z})^S$ is an equivalence.*

Proof. This map is always an embedding. To show it is surjective it is enough to prove that $(\mathrm{B}\mathbb{Z})^S$ is connected, which is precisely Lemma 5.10. ◀

5.3 Čech cohomology of compact Hausdorff spaces

▶ **Definition 5.12.** *A Čech cover consists of $X : \mathsf{CHaus}$ and $S : X \to \mathsf{Stone}$ such that $\prod_{x:X} \|S_x\|$ and $\sum_{x:X} S_x : \mathsf{Stone}$.*

By definition any compact Hausdorff space X is part of a Čech cover (X, S).

▶ **Lemma 5.13.** *Given a Čech cover (X, S) and $A : X \to \mathrm{Ab}$, we have an isomorphism $H^0(X, A) = \check{H}^0(X, S, A)$ natural in A.*

Proof. By definition an element in $\check{H}^0(X, S, A)$ is a map $f : \prod_{x:X} A_x^{S_x}$ such that for all $u, v : S_x$ we have $f(u) = f(v)$. Since A_x is a set and the S_x are merely inhabited, this is equivalent to $\prod_{x:X} A_x$. Naturality in A is immediate. ◀

▶ **Lemma 5.14.** *Given a Čech cover (X, S) we have an exact sequence*

$$H^0(X, \lambda x.\mathbb{Z}^{S_x}) \to H^0(X, \lambda x.\mathbb{Z}^{S_x}/\mathbb{Z}) \to H^1(X, \mathbb{Z}) \to 0$$

Proof. We use the long exact cohomology sequence associated to

$$0 \to \mathbb{Z} \to \mathbb{Z}^{S_x} \to \mathbb{Z}^{S_x}/\mathbb{Z} \to 0$$

We just need $H^1(X, \lambda x.\mathbb{Z}^{S_x}) = 0$ to conclude. But by Corollary 5.11 we have that $H^1(X, \lambda x.\mathbb{Z}^{S_x}) = H^1\left(\sum_{x:X} S_x, \mathbb{Z}\right)$ which vanishes by Lemma 5.10. ◀

▶ **Lemma 5.15.** *Given a Čech cover (X, S) we have an exact sequence*

$$\check{H}^0(X, S, \lambda x.\mathbb{Z}^{S_x}) \to \check{H}^0(X, S, \lambda x.\mathbb{Z}^{S_x}/\mathbb{Z}) \to \check{H}^1(X, S, \mathbb{Z}) \to 0$$

Proof. For $n = 1, 2, 3$, we have that $\Sigma_{x:X} S_x^n$ is Stone so that $H^1(\Sigma_{x:X} S_x^n, \mathbb{Z}) = 0$ by Lemma 5.10, giving short exact sequences

$$0 \to \Pi_{x:X} \mathbb{Z}^{S_x^n} \to \Pi_{x:X} (\mathbb{Z}^{S_x})^{S_x^n} \to \Pi_{x:X} (\mathbb{Z}^{S_x}/\mathbb{Z})^{S_x^n} \to 0$$

They fit together in a short exact sequence of complexes

$$0 \to \check{C}(X, S, \mathbb{Z}) \to \check{C}(X, S, \lambda x.\mathbb{Z}^{S_x}) \to \check{C}(X, S, \lambda x.\mathbb{Z}^{S_x}/\mathbb{Z}) \to 0$$

But since $\check{H}^1(X, \lambda x.\mathbb{Z}^{S_x}) = 0$ by Lemma 5.6, we conclude using the associated long exact sequence. ◀

▶ **Theorem 5.16.** *Given a Čech cover (X, S), we have that $H^1(X, \mathbb{Z}) = \check{H}^1(X, S, \mathbb{Z})$*

Proof. By applying Lemma 5.13, Lemma 5.14 and Lemma 5.15 we get that $H^1(X, \mathbb{Z})$ and $\check{H}^1(X, S, \mathbb{Z})$ are cokernels of isomorphic maps, so they are isomorphic. ◀

This means that Čech cohomology does not depend on S.

5.4 Cohomology of the interval

▶ Remark 5.17. Recall from Definition 4.19 that there is a binary relation $\sim_n$ on $2^n =: \mathbb{I}_n$ such that $(2^n, \sim_n)$ is equivalent to $(\mathrm{Fin}(2^n), \lambda x, y.\ |x - y| \leq 1)$ and for $\alpha, \beta : 2^{\mathbb{N}}$ we have $(cs(\alpha) = cs(\beta)) \leftrightarrow (\forall_{n:\mathbb{N}} \alpha|_n \sim_n \beta|_n)$.

We define $\mathbb{I}_n^{\sim 2} = \Sigma_{x,y:\mathbb{I}_n} x \sim_n y$ and $\mathbb{I}_n^{\sim 3} = \Sigma_{x,y,z:\mathbb{I}_n} x \sim_n y \wedge y \sim_n z \wedge x \sim_n z$.

▶ **Lemma 5.18.** *For any $n : \mathbb{N}$ we have an exact sequence*

$$0 \to \mathbb{Z} \xrightarrow{d_0} \mathbb{Z}^{\mathbb{I}_n} \xrightarrow{d_1} \mathbb{Z}^{\mathbb{I}_n^{\sim 2}} \xrightarrow{d_2} \mathbb{Z}^{\mathbb{I}_n^{\sim 3}}$$

where $d_0(k) = (\_ \mapsto k)$ and

$$\begin{aligned} d_1(\alpha)(u, v) &= \alpha(v) - \alpha(u) \\ d_2(\beta)(u, v, w) &= \beta(v, w) - \beta(u, w) + \beta(u, v). \end{aligned}$$

Proof. It is clear that the map $\mathbb{Z} \to \mathbb{Z}^{\mathbb{I}_n}$ is injective as $\mathbb{I}_n$ is inhabited, so the sequence is exact at $\mathbb{Z}$. Assume a cocycle $\alpha : \mathbb{Z}^{\mathbb{I}_n}$, meaning that for all $u, v : \mathbb{I}_n$, if $u \sim_n v$ then $\alpha(u) = \alpha(v)$. Then by Remark 5.17 we see that α is constant, so the sequence is exact at $\mathbb{Z}^{\mathbb{I}_n}$.

Assume a cocycle $\beta : \mathbb{Z}^{\mathbb{I}_n^{\sim 2}}$, meaning that for all $u, v, w : \mathbb{I}_n$ such that $u \sim_n v$, $v \sim_n w$ and $u \sim_n w$ we have that $\beta(u, v) + \beta(v, w) = \beta(u, w)$. Using Remark 5.17 to pass along the equivalence between 2^n and $\mathrm{Fin}(2^n)$, we define $\alpha(k) = \beta(0, 1) + \cdots + \beta(k - 1, k)$. We can check that $\beta(k, l) = \alpha(l) - \alpha(k)$, so that β is indeed a coboundary and the sequence is exact at $\mathbb{Z}^{\mathbb{I}_n^{\sim 2}}$. ◀

▶ **Proposition 5.19.** *We have that $H^0(\mathbb{I}, \mathbb{Z}) = \mathbb{Z}$ and $H^1(\mathbb{I}, \mathbb{Z}) = 0$.*

Proof. Consider $cs : 2^{\mathbb{N}} \to \mathbb{I}$ and the associated Čech cover T of $\mathbb{I}$ defined by:

$$T_x = \Sigma_{y:2^{\mathbb{N}}} (x =_{\mathbb{I}} cs(y))$$

Then for $l = 2, 3$ we have that $\lim_n \mathbb{I}_n^{\sim l} = \sum_{x:\mathbb{I}} T_x^l$. By Lemma 5.18 and stability of exactness under sequential colimit, we have an exact sequence

$$0 \to \mathbb{Z} \to \mathrm{colim}_n \mathbb{Z}^{\mathbb{I}_n} \to \mathrm{colim}_n \mathbb{Z}^{\mathbb{I}_n^{\sim 2}} \to \mathrm{colim}_n \mathbb{Z}^{\mathbb{I}_n^{\sim 3}}$$

By Corollary 3.7 this sequence is equivalent to

$$0 \to \mathbb{Z} \to \Pi_{x:\mathbb{I}}\mathbb{Z}^{T_x} \to \Pi_{x:\mathbb{I}}\mathbb{Z}^{T_x^2} \to \Pi_{x:\mathbb{I}}\mathbb{Z}^{T_x^3}$$

So it being exact implies that $\check{H}^0(\mathbb{I}, T, \mathbb{Z}) = \mathbb{Z}$ and $\check{H}^1(\mathbb{I}, T, \mathbb{Z}) = 0$. We conclude by Lemma 5.13 and Theorem 5.16. ◀

▶ Remark 5.20. We could carry a similar computation for $\mathbb{S}^1$, by approximating it with 2^n with $0^n \sim_n 1^n$ added. We would find $H^1(\mathbb{S}^1, \mathbb{Z}) = \mathbb{Z}$. We will give an alternative, more conceptual proof in the next section.

5.5 Brouwer's fixed-point theorem

Here we consider the modality defined by localising at $\mathbb{I}$ as explained in [14]. It is denoted by $L_{\mathbb{I}}$. We say that X is $\mathbb{I}$-local if $L_{\mathbb{I}}(X) = X$ and that it is $\mathbb{I}$-contractible if $L_{\mathbb{I}}(X) = 1$.

▶ **Lemma 5.21.** *$\mathbb{Z}$ and 2 are $\mathbb{I}$-local.*

Proof. By Proposition 5.19, from $H^0(\mathbb{I}, \mathbb{Z}) = \mathbb{Z}$ we get that the map $\mathbb{Z} \to \mathbb{Z}^{\mathbb{I}}$ is an equivalence, so $\mathbb{Z}$ is $\mathbb{I}$-local. We see that 2 is $\mathbb{I}$-local as it is a retract of $\mathbb{Z}$. ◀

▶ Remark 5.22. Since 2 is $\mathbb{I}$-local, we have that any Stone space is $\mathbb{I}$-local.

▶ **Lemma 5.23.** *$\mathrm{B}\mathbb{Z}$ is $\mathbb{I}$-local.*

Proof. Any identity type in $\mathrm{B}\mathbb{Z}$ is a $\mathbb{Z}$-torsor, so it is $\mathbb{I}$-local by Lemma 5.21. So the map $\mathrm{B}\mathbb{Z} \to \mathrm{B}\mathbb{Z}^{\mathbb{I}}$ is an embedding. From $H^1(\mathbb{I}, \mathbb{Z}) = 0$ we get that it is surjective, hence an equivalence. ◀

▶ **Lemma 5.24.** *Assume X a type with $x : X$ such that for all $y : X$ we have $f : \mathbb{I} \to X$ such that $f(0) = x$ and $f(1) = y$. Then X is $\mathbb{I}$-contractible.*

Proof. For all $y : X$ we get a map $g : \mathbb{I} \to L_{\mathbb{I}}(X)$ such that $g(0) = [x]$ and $g(1) = [y]$. Since $L_{\mathbb{I}}(X)$ is $\mathbb{I}$-local this means that $\prod_{y:X}[x] = [y]$. We conclude $\prod_{y:L_{\mathbb{I}}(X)}[x] = y$ by applying the elimination principle for the modality. ◀

▶ **Corollary 5.25.** *We have that $\mathbb{R}$ and $\mathbb{D}^2 = \{(x, y) : \mathbb{R}^2 \mid x^2 + y^2 \leq 1\}$ are $\mathbb{I}$-contractible.*

▶ **Proposition 5.26.** *$L_{\mathbb{I}}(\mathbb{R}/\mathbb{Z}) = \mathrm{B}\mathbb{Z}$.*

Proof. As for any group quotient, the fibers of the map $\mathbb{R} \to \mathbb{R}/\mathbb{Z}$ are $\mathbb{Z}$-torsors, so we have an induced pullback square

$$\begin{tikzcd}
\mathbb{R} \arrow[r] \arrow[d] & 1 \arrow[d] \\
\mathbb{R}/\mathbb{Z} \arrow[r] & \mathrm{B}\mathbb{Z}
\end{tikzcd}$$

Now we check that the bottom map is an $\mathbb{I}$-localisation. Since $\mathrm{B}\mathbb{Z}$ is $\mathbb{I}$-local by Lemma 5.23, it is enough to check that its fibers are $\mathbb{I}$-contractible. Since $\mathrm{B}\mathbb{Z}$ is connected it is enough to check that $\mathbb{R}$ is $\mathbb{I}$-contractible. This is Corollary 5.25. ◀

▶ Remark 5.27. By Lemma 5.23, for any X we have that $H^1(X, \mathbb{Z}) = H^1(L_{\mathbb{I}}(X), \mathbb{Z})$, so that by Proposition 5.26 we have that $H^1(\mathbb{R}/\mathbb{Z}, \mathbb{Z}) = H^1(\mathrm{B}\mathbb{Z}, \mathbb{Z}) = \mathbb{Z}$.

We omit the proof that $\mathbb{S}^1 = \{(x, y) : \mathbb{R}^2 \mid x^2 + y^2 = 1\}$ is equivalent to $\mathbb{R}/\mathbb{Z}$. The equivalence can be constructed using trigonometric functions, which exist by Proposition 4.12 in [3].

▶ **Proposition 5.28.** *The map $\mathbb{S}^1 \to \mathbb{D}^2$ has no retraction.*

Proof. By Corollary 5.25 and Proposition 5.26 we would get a retraction of $\mathrm{B}\mathbb{Z} \to 1$, so $\mathrm{B}\mathbb{Z}$ would be contractible. ◀

▶ **Theorem 5.29** (Intermediate value theorem). *For any $f : \mathbb{I} \to \mathbb{I}$ and $y : \mathbb{I}$ such that $f(0) \leq y$ and $y \leq f(1)$, there exists $x : \mathbb{I}$ such that $f(x) = y$.*

Proof. By Corollary 4.3, the proposition $\exists_{x:\mathbb{I}} f(x) = y$ is closed and therefore $\neg\neg$-stable, so we can proceed with a proof by contradiction. If there is no such $x : \mathbb{I}$, we have $f(x) \neq y$ for all $x : \mathbb{I}$. By Remark 4.24 we have that $a < b$ or $b < a$ for all distinct numbers $a, b : \mathbb{I}$. So the following two sets cover $\mathbb{I}$

$$U_0 := \{x : \mathbb{I} \mid f(x) < y\} \qquad U_1 := \{x : \mathbb{I} \mid y < f(x)\}$$

Since U_0 and U_1 are disjoint, we have $\mathbb{I} = U_0 + U_1$ which allows us to define a non-constant function $\mathbb{I} \to 2$, which contradicts Lemma 5.21. ◀

▶ **Theorem 5.30** (Brouwer's fixed-point theorem). *For all $f : \mathbb{D}^2 \to \mathbb{D}^2$ there exists $x : \mathbb{D}^2$ such that $f(x) = x$.*

Proof. As above, by Corollary 4.3, we can proceed with a proof by contradiction, so we assume $f(x) \neq x$ for all $x : \mathbb{D}^2$. For any $x : \mathbb{D}^2$ we set $d_x = x - f(x)$, so we have that one of the coordinates of d_x is invertible. Let $H_x(t) = f(x) + t \cdot d_x$ be the line through x and $f(x)$. The intersections of H_x and $\partial\mathbb{D}^2 = \mathbb{S}^1$ are given by the solutions of an equation quadratic in t. By invertibility of one of the coordinates of d_x, there is exactly one solution with $t > 0$. We denote this intersection by $r(x)$ and the resulting map $r : \mathbb{D}^2 \to \mathbb{S}^1$ has the property that it preserves $\mathbb{S}^1$. Then r is a retraction from $\mathbb{D}^2$ onto its boundary $\mathbb{S}^1$, which is a contradiction by Proposition 5.28. ◀

▶ Remark 5.31. In constructive reverse mathematics [7], it is known that both the intermediate value theorem and Brouwer's fixed-point theorem are equivalent to LLPO. But LLPO does not hold in real cohesive homotopy type theory, so [16] prove a variant of the statement involving a double negation.

References

1 Dagur Ásgeirsson. The foundations of condensed mathematics. Master's thesis, Université de Paris, 2021. URL: `https://dagur.sites.ku.dk/condensed-foundations/`.

2 Reid Barton and Johan Commelin. Lean-ctt-snapshot. URL: `https://github.com/jcommelin/lean-ctt-snapshot`.

3 Errett Bishop and Douglas Bridges. *Constructive analysis*, volume 279 of *Grundlehren der mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1985. `doi:10.1007/978-3-642-61667-9`.

4 Felix Cherubini, Thierry Coquand, and Matthias Hutzler. A foundation for synthetic algebraic geometry, 2023. `arXiv:2307.00073`.

5 Dustin Clausen and Peter Scholze. Analytic stacks, 2023-2024. Lecture series. URL: `https://www.youtube.com/playlist?list=PLx5f8IelFRgGmu6gmL-Kf_R1_6Mm7juZO`.

6 Thierry Coquand, Fabian Ruch, and Christian Sattler. Constructive sheaf models of type theory. *Math. Struct. Comput. Sci.*, 31(9):979–1002, 2021. doi:10.1017/S0960129521000359.

7 Hannes Diener. Constructive reverse mathematics : Habilitationsschrift, 2018. URL: https://dspace.ub.uni-siegen.de/handle/ubsi/1306.

8 Roy Dyckhoff. Categorical methods in dimension theory. Categor. Topol., Proc. Conf. Mannheim 1975, Lect. Notes Math. 540, 220-242 (1976)., 1976.

9 Martín Escardó. Synthetic topology: of data types and classical spaces. *Electronic Notes in Theoretical Computer Science*, 87:21–156, 2004. Proceedings of the Workshop on Domain Theoretic Methods for Probabilistic Processes. doi:10.1016/j.entcs.2004.09.017.

10 Florian Faissole and Bas Spitters. Synthetic topology in homotopy type theory for probabilistic programming. In *Proc*, page 2017, 2017.

11 Hajime Ishihara. Reverse Mathematics in Bishop's Constructive Mathematics. *Philosophia Scientiae*, 6:43–59, September 2006. doi:10.4000/philosophiascientiae.406.

12 Davorin Lešnik. *Synthetic Topology and Constructive Metric Spaces.* PhD thesis, University of Ljubljana, 2021. arXiv:2104.10399.

13 The Univalent Foundations Program. *Homotopy type theory: Univalent foundations of mathematics.* https://homotopytypetheory.org/book, 2013.

14 Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Logical Methods in Computer Science*, Volume 16, Issue 1, January 2020. doi:10.23638/LMCS-16(1:2)2020.

15 Peter Scholze. Lectures on condensed mathematics, 2019. URL: https://people.mpim-bonn.mpg.de/scholze/Condensed.pdf.

16 Michael Shulman. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *Mathematical Structures in Computer Science*, 28(6):856–941, 2018. doi:10.1017/S0960129517000147.

17 Michael Shulman. All $(\infty, 1)$-toposes have strict univalent universes. *arXiv preprint*, 2019. arXiv:1904.07004.

18 Kristina Sojakova, Floris van Doorn, and Egbert Rijke. Sequential colimits in homotopy type theory. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2020)*, page 14. ACM, New York, [2020] ©2020. doi:10.1145/3373718.3394801.

19 Paul Taylor. Abstract Stone duality, 2011. URL: https://www.paultaylor.eu/ASD/.

20 Niels van der Weide. The internal languages of univalent categories, 2024. doi:10.48550/arXiv.2411.06636.

21 Steven Vickers. Locales and toposes as spaces. In *Handbook of spatial logics*, pages 429–496. Springer, 2007. doi:10.1007/978-1-4020-5587-4_8.

22 David Wärn. Eilenberg-Maclane spaces and stabilisation in homotopy type theory. *J. Homotopy Relat. Struct.*, 18(2-3):357–368, 2023. doi:10.1007/s40062-023-00330-5.

23 David Wärn. On internally projective sheaves of groups, 2024. arXiv:2409.12835.

Separating Terms by Means of Multi Types, Coinductively

Adrienne Lancelot
Inria & LIX, École Polytechnique, UMR 7161, Palaiseau, France
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Abstract

Intersection type systems, as adequate models of the λ-calculus, induce an equational theory on terms, that we refer to as type equivalence. We give a new proof technique to coinductively characterize type equivalence. To do so, we explore a simple setting, namely weak head type equivalence, which is the equational theory induced by a weak head non-idempotent intersection type system.

We prove a folklore result: weak head type equivalence coincides with Sangiorgi's normal form bisimilarity. What is new in our development is that we only rely on coinductive program equivalences, bypassing the need to introduce term approximants, which were used in previous works characterizing type equivalence.

The crucial part of this characterization is to show that type equivalent terms are normal form bisimilar: we do so by constructing *shape typings* that can only type terms of a specific normal form structure. Shape typings are a light form of principal types, a technique often used in intersection types to generate from one or few principal typing all possible typings of a term.

2012 ACM Subject Classification Theory of computation → Lambda calculus

Keywords and phrases lambda calculus, intersection types, program equivalence

Digital Object Identifier 10.4230/LIPIcs.TYPES.2024.4

Acknowledgements I am grateful to the reviewers for their careful read of the paper and helpful comments and to Beniamino Accattoli for providing me feedback on a first draft.

1 Introduction

Intersection types were developed to study the theory of programming languages and are at the intersection between operational and denotational techniques. While only a certain subset of terminating programs are typable in a simple type system, all terminating programs are typable in an *intersection* type system. As they can give a meaning to all terminating terms, intersection type systems may give syntactic presentations of denotational models [21, 12].

Non-idempotent intersection types, here referred to as *multi types*, have been especially successful syntactic presentations of the relational model, both in Head Call-by-Name (CbN) [14] and Call-by-Value (CbV) [17]. In this work we focus on another variant, namely *Weak Head Call-by-Name* (further restricting head reduction to forbid reduction under lambdas).

In this work, we will syntactically characterize the equational theory induced on terms by multi type systems. Such a characterization is already known for CbN: Böhm tree equivalence exactly represents the equational theory induced by the CbN idempotent intersection type system [22] and by the CbN non-idempotent variant [13]. Both developments introduce term approximants to carefully define the syntactic program equivalence that will match the equational theory induced by the type system. Terms approximants are a powerful but heavy tool in the study of program equivalences; in particular, they require to extend the calculus to a calculus of approximants. The aim of this paper is to show that approximants can be avoided and replaced by a simple use of coinduction, namely normal form bisimulations [23, 19].

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg; Article No. 4; pp. 4:1–4:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Syntactic Equality and Normal Form Bisimilarities. Denotational models give rise to mathematical interpretations of the λ-calculus, where different λ-terms may be sent to the same mathematical object. Equivalence classes of λ-terms in these models should always include syntactical equality–that only equates programs that have exactly the same syntax (up to α-equivalence)–and β-equivalence. Indeed, $\mathtt{I} := \lambda x.x$ and $\mathtt{II}$ have different syntax but represent the same identity function. The computational theory $\lambda\beta$, that only quotients by β-equivalence, is still far from reaching most denotational models' theories. Indeed, some terms are not β-equivalent but have the same behavior–the paradigmatic example being fixpoint combinators. To be able to relate those terms, one needs to define syntactical theories based on infinite trees, namely normal form bisimilarities.

Normal form bisimilarity is an intensional equivalence, that compares the structure of the terms' (possibly infinite) normal forms. The normal form bisimilarity of interest here is Sangiorgi's normal form bisimilarity, denoted $\simeq_{\mathsf{wh}}$. It is obtained from the study of bisimulations in the π-calculus and the translation of the λ-calculus into the π-calculus [23]. Later, Lassen related this bisimilarity, there called weak head normal form bisimilarity, to Lévy-Longo tree equivalence [19].

Weak Head Multi Types. Adequate intersection types for weak head call-by-name evaluation were first introduced in an idempotent setting by Dezani-Ciancaglini et al. [16] and then refined by Bucciarelli et al. [15] in a non-idempotent setting, allowing for quantitative information and lighter proofs. We shall use the latter presentation, via multi types defined later in Section 3. The quantitative aspect will play a crucial role, as explained in Section 5.

Type Equivalence. Given an intersection type system, one can define what we call here *type equivalence* $\simeq_{\text{type}}$, which is the equational theory induced by the model syntactically represented by the type system. Two terms are type equivalent if they are typable by exactly the same pairs of typing contexts and types. While it is easier to prove type equivalence than contextual equivalence on certain pairs of terms, it is still quantified universally and, unlike normal form bisimilarities, does not provide a straightforward proof technique. Type equivalence is included in contextual equivalence, but this inclusion is in general strict.

In the case of weak head multi types, type equivalence $\simeq_{\text{type}}$ does not validate well-known examples of equations validated by contextual equivalence $\simeq_{\text{ctx}}$ (that are also examples of incompleteness for Sangiorgi's normal form bisimilarity): for instance, $x\lambda y.xy \simeq_{\text{ctx}} xx$ but $x\lambda y.xy \not\simeq_{\text{type}} xx$ (and $x\lambda y.xy \not\simeq_{\mathsf{wh}} xx$) [1, 23].

This Paper: Coinductively Characterizing Type Equivalence. Our main result is that type equivalence induced by the weak head multi type system coincides with Sangiorgi's normal form bisimilarity. This characterization may be folklore for experts of the topic but we provide here a clean development of the coinductive proof technique.

There are various related works on characterizing syntactically the equational theory of a model. To the best of our knowledge, characterizations regarding intersection types all mention term approximants. Approximants are an inductive way to look at infinitary behaviors and were the core of the original definition of Böhm trees. Coinduction is a more modern way to define infinitary computation and it is particularly successful in defining (Böhm) tree equivalence as normal form bisimilarity. By avoiding approximants, we do not need to complexify theorem statements to include approximants (on top of terms). The proof technique is more easily breakable into sub-lemmas, as it rests on a coinductive argument.

We split the characterization proof in two directions:

$$\begin{tikzcd}\text{NF Bisimilarity} \arrow[r, Rightarrow, shift left, "\text{Typing Transfer}"] & \text{Type Equivalence} \arrow[l, Rightarrow, shift left, "\text{Shape Typings}"]\end{tikzcd}$$

Typing transfer shows that normal form bisimilar terms are typable by the same pairs of typing contexts and types (in fact, sometimes even using the same type derivations). This first direction is the easy one, done by induction, exploiting quantitative arguments of multi types. The other direction uses coinduction and is shown by building shape typings, which explictly specify that type equivalent terms have matching normal forms. Shape typings are reminiscent of principal types, and share some of their properties. They are however lighter and principal types for intersection/multi types are a quite technical topic beyond the scope of this paper.

Going back to finitary normal forms, we give an intersection type variant of Böhm theorem: for any two distinct β-normal forms, there exists a typing context and a type that may only type one of the normal forms and not the other (in the weak head intersection type system).

Monotonicity of Normal Form Bisimilarity, for free. Once Sangiorgi's normal form bisimilarity is proven equivalent to type equivalence, we can deduce the monotonicity of Sangiorgi's normal form bisimilarity without much work. A relation is monotone if it is stable by context closure. Such a property is not trivial for intensional equivalences like Böhm tree equivalence or normal form bisimilarity, as they are not defined compositionally. On the other hand, proving monotonicity for type equivalence is an immediate induction on derivations, as derivations are built compositionally.

While there exists some techniques to show that normal form bisimilarities are monotone, they do not scale up so easily. Lassen's adaptation of Howe's method for normal form bisimilarity [19] is a tedious but efficient process for Böhm tree equivalence and Sangiorgi's normal form bisimilarity. It does not scale up to extensional call-by-value bisimilarities, where Biernacki et al. need to introduce an extension of the proof method [11]. These limits justify looking for alternative proof techniques of monotonicity.

On Related Separation Constructions. The crucial part of the proof relies in the separation construction, where it is shown that type equivalent terms are normal form bisimilar. Our construction of shape typings is not so different than the separation described in the book of Barendregt and Manzonetto [10], which is exactly the proof from Breuvart et al. [13].

Separation theorems are not a new concept, see Barendregt's book about the λ-calculus [9] for an overview. These techniques are also reminiscent of the well-known Böhm out technique [9], where from two terms that are not normal form bisimilar, one builds a context separating them for contextual equivalence. However these techniques are usually presented via contrapositive: from a difference in Böhm trees, one can extract a context/type that can separate them. In our case, we do not explicitly go by contrapositive but specify principal types, closer to Ronchi Della Rocca's development [22] where principal types are carefully defined and used.

A Single Ground Type is Enough. We delve into a more technical aspect of multi types. Intersection types for call-by-name evaluation, whether it is head or weak head, require ground types, that are atom types for variables of a term. In general, it is sufficient to have only one ground type, as properties of multi types are still preserved, hence we sometimes only use one ground type for simplicity [4]. To characterize type equivalence, however, Breuvart et al. deliberately use infinitely many ground types to ease the separation of terms [13]. We follow at first their simplification and clearly outline where these ground types are needed. We are then able to give an alternative construction that only requires a single ground type.

2 Weak Head Reduction

We briefly survey the definition of weak head reduction and its associated contextual equivalence.

Weak Head Reduction. We focus on a specific reduction of the λ-calculus, weak head reduction, the variant of head reduction that does not reduce under lambdas.

$$\textsc{Terms} \quad \Lambda \ni t, u, s \quad ::= \quad x \mid \lambda x.t \mid tu \qquad\qquad \textsc{Weak Head Ctxs} \quad E \quad ::= \quad \langle\cdot\rangle \mid Et$$

Weak head reduction, denoted $\rightarrow_{\mathsf{wh}}$, is the closure of the beta rule $(\lambda x.t)u \mapsto_\beta t\{x{\leftarrow}u\}$ under applicative *weak head* contexts. In essence, all $\rightarrow_{\mathsf{wh}}$ steps are of the shape $(\lambda x.t)u\ s_1 \ldots s_k \rightarrow_{\mathsf{wh}} t\{x{\leftarrow}u\}\ s_1 \ldots s_k$ for $k \geq 0$.

Normal forms with respect to $\rightarrow_{\mathsf{wh}}$ are exactly characterized by the following grammar, separating normal forms into rigid terms and abstractions:

$$\textsc{Rigid Terms} \quad r, r' ::= x \mid rt \qquad\qquad \textsc{Weak Head Normal Forms} \quad w, w' ::= \lambda x.t \mid r$$

We say that a term is $\rightarrow_{\mathsf{wh}}$-normalizing if it reduces to a $\rightarrow_{\mathsf{wh}}$-normal form. Note that by well-known normalization theorems for weak head reduction, the fact that a term β-reduces to a $\rightarrow_{\mathsf{wh}}$-normal form is equivalent to $\rightarrow_{\mathsf{wh}}$-reducing to a $\rightarrow_{\mathsf{wh}}$-normal form. We define the contextual preorder and equivalence based on weak head termination.

▶ **Definition 1** (Contextual Preorder and Equivalence). *The weak head* contextual preorder $\precsim_{\text{ctx}}$ *and* contextual equivalence $\simeq_{\text{ctx}}$ *are defined as follows:*

- $t \precsim_{\text{ctx}} t'$ *if, for all context C such that $C\langle t\rangle$ and $C\langle t'\rangle$ are closed terms, whenever $C\langle t\rangle$ is $\rightarrow_{\mathsf{wh}}$-normalizing, so is $C\langle t'\rangle$.*
- $t \simeq_{\text{ctx}} t'$ *is the equivalence relation induced by $\precsim_C$, that is, $t \simeq_{\text{ctx}} t' \iff t \precsim_{\text{ctx}} t'$ and $t' \precsim_{\text{ctx}} t$.*

Contrarily to the usual call-by-name contextual equivalence (based on head termination), Ω and $\lambda x.\Omega$ are not contextually equivalent. Indeed, in the empty context, the former does not terminate whereas the latter is in weak head normal form.

On Contextual Equivalence and η-equivalence. As a consequence, the η rule does not hold in general: Ω and $\lambda x.\Omega x$ are not contextually equivalent (for $\Omega := (\lambda x.xx)(\lambda x.xx)$, the paradigmatic looping term). In fact, if one is interested in the weak head contextual preorder, $\lambda x.tx$ refines t but t does not always refine $\lambda x.tx$ ($t \precsim_{\text{ctx}} \lambda x.tx$ for all t but $\lambda x.yx \not\precsim_{\text{ctx}} y$ for all variables y). If t converges to an abstraction, there is no difference between both terms, as this η-equivalence is actually included in β equivalence. Otherwise, it is in fact not true that t refines $\lambda x.tx$ as t can be non terminating whereas $\lambda x.tx$ is *always* a weak head normal form.

Issues with η-equivalence and weak head reduction actually go beyond this first observation. There exists η-equivalent terms that are contextually equivalent, the paradigmatic example being $x\lambda y.xy \simeq_{\text{ctx}} xx$ [1, 23]. These two terms are contextually equivalent even though $\lambda y.xy$ and x are not. Contextual *inequivalence* is somehow non-compositional in presence of η.

$$\begin{array}{lllll}
\textsc{Linear Types} & L, L' & ::= & \star_k \mid M \multimap L & k \in \mathbb{N} \\
\textsc{Multi Types} & M, N & ::= & [L_1, \ldots, L_n] & n \geq 0
\end{array}$$

$$\frac{}{x\!:\![L] \vdash x\!:\!L}\ \mathsf{ax} \qquad \frac{}{\emptyset \vdash \lambda x.t\!:\!\star_k}\ \lambda_k \qquad \frac{\Gamma, x\!:\!M \vdash t\!:\!L}{\Gamma \vdash \lambda x.t\!:\!M \multimap L}\ \lambda$$

$$\frac{\Gamma \vdash t\!:\![L_i]_{i\in I} \multimap L' \quad (\Gamma_i \vdash u\!:\!L_i)_{i\in I} \quad I \text{ finite}}{\Gamma \uplus (\uplus_{i\in I}\Gamma_i) \vdash tu\!:\!L'}\ @$$

Figure 1 $\mathcal{WH}$ Multi Type System.

3 Weak Head Multi Types $\mathcal{WH}$

We follow the definitions of multi types for weak head reduction as designed by Bucciarelli et al. [15], recalled in Fig. 1.

Multi Types. Multi types M are finite multisets of linear types L that are either base types $\star_k$ or arrow types $M \multimap L$. In the original presentation of Bucciarelli et al. [15], there are distinct ground types and an abstraction type $\sharp$: we unify both for simplicity. Note that there is a countable number of distinct base types, following Breuvart et al. [13], to simplify separating terms by types.

Typing contexts Γ are finite-support maps from variables to multi types, denoted $x_1 : M_1, \cdots, x_k\!:\!M_k$. Any omitted variable is implicitly typed by the empty type $[\,]$. The empty typing context, where all variables are typed by the empty type, is denoted $\emptyset$.

Typing Rules. The typing rules described in 1 are the rules to infer the type of a term.

There are two axiom rules, one for variables (ax) and one for abstractions (λ_k). The latter is specific to weak head call-by-name, by allowing to type all abstractions.

There is another rule to type abstractions (λ) whose only premise gives a type for the body of the abstraction, given a multi type for the binded variable, that now moves to the typing context.

The application rule ($@$) is the only way to type terms of the form tu. Intuitively, this rule requires the argument u to be typed for every occurrence that will be used in typing the function t. The premices of the rule indeed require to type t with a linear type $[L_1, \cdots, L_k] \multimap L$ and u with many linear types $L_1, \cdots, L_k$, giving a different type derivation for each of the linear types in the argument position of the type of t. In particular, if t is typed with the linear type $[\,] \multimap L$, then one does not have to provide any type derivation for u to apply the $@$-rule.

▶ **Definition 2** (Typing). *A typing of a term t is a pair (Γ, L) of a typing context Γ and a linear type L such that there exists a derivation π with final judgment $\Gamma \vdash t\!:\!L$.*

We write $\pi \triangleright \Gamma \vdash t\!:\!L$ if π is a type derivation with final judgment $\Gamma \vdash t\!:\!L$.

Results about Multi Types. Intersection types are in particular known for the fact that typability may exactly capture normalizable terms. This system is an example of such characterization, as a term is typable in $\mathcal{WH}$ if and only if it is *weak head* normalizable.

▶ **Theorem 3** (Characterization of Termination, [15]). *A term t is typable in $\mathcal{WH}$ if and only if $t \to_{\mathsf{wh}}^* n$ where n is a weak head normal form.*

Proving characterization of termination for idempotent intersection types is generally obtained by reducibility arguments, but this proof technique can be avoided by switching to multi types–where characterization of termination can be shown with combinatorial arguments. The main selling point of multi types, the *non-idempotent* variant of intersection types, is indeed to obtain *quantitative* results from typing judgments. The most famous example is the following statement of *quantitative subject reduction*, that allows to show that any type of a term is stable by reduction, and the derivation of the reduct shall be of a strictly smaller size. The size of a derivation $|\pi|$ is its total number of rules.

▶ **Proposition 4** (Quantitative Subject Reduction, [15]). *For all* (π, Γ, L, t, t') *such that* $\pi \triangleright \Gamma \vdash t : L$ *and* $t \to_{\mathsf{wh}} t'$, *there exists* $\pi' \triangleright \Gamma \vdash t' : L$ *with* $|\pi'| < |\pi|$.

From quantitative subject reduction, it is typical to deduce (without reducibility arguments!) that the intersection type system is correct, *i.e.* all typable terms are normalizing. For the converse implication to hold, thus completing the characterization of termination, one can easily deduce it from subject expansion. Note that the following statement of subject expansion contains quantitative information about the size of derivations but the quantitative argument plays no role in proving that all normalizing terms are typable (the induction is done on the number of steps to normalization).

▶ **Proposition 5** (Quantitative Subject Expansion, [15]). *For all* (π, Γ, L, t, t') *such that* $\pi \triangleright \Gamma \vdash t' : L$ *and* $t \to_{\mathsf{wh}} t'$, *there exists* $\pi' \triangleright \Gamma \vdash t : L$ *with* $|\pi'| > |\pi|$.

Quantitative arguments are also sometimes taken a step further, introducing alternative type systems that exactly measure the length of the reduction sequence to normal forms, see Accattoli et al. [3].

The Equational Theory Induced by the Multi Type System. In this work, we focus on the program equivalence induced by the multi type system of Fig. 1. Indeed, multi type systems induce a model, that we shall not discuss here, and every model induces an equivalence relation on terms (by relating terms with the same interpretation in the model). We introduce the *type preorder*, that exactly rephrases the (in)equational theory induced by the weak head multi type system.

▶ **Definition 6** (Type Preorder). *The type preorder* $\precsim_{\text{type}}$ *and type equivalence* $\simeq_{\text{type}}$ *are relations on terms defined as follows:*

- $t \precsim_{\text{type}} t'$ *if for all* Γ, L *such that there exists* $\pi \triangleright \Gamma \vdash t : L$ *then there exists* $\pi' \triangleright \Gamma \vdash t' : L$.
- *Type equivalence is defined by symmetry:* $t \simeq_{\text{type}} t'$ *iff* $t \precsim_{\text{type}} t'$ *and* $t' \precsim_{\text{type}} t$.

It is easy to check that $\precsim_{\text{type}}$ is indeed a preorder (reflexive and transitive). Furthermore by subject reduction and subject expansion it is invariant by $\to_{\mathsf{wh}}$ and *adequate*. As it is also stable by contexts, the type preorder satisfies the usual definition of an (in)equational theory [9, 10].

We say that t' (type-)improves t if $t \precsim_{\text{type}} t'$, that is if all typings of t are typings of t'. Symmetrically, two terms are type equivalent if that they have the same set of typings.

▶ **Proposition 7.** *The type preorder enjoys the following properties:*

1. wh-invariance: *if* $t \to_{\mathsf{wh}} t'$ *then* $t \precsim_{\text{type}} t'$ *and* $t' \precsim_{\text{type}} t$;
2. Adequacy: *if* $t \precsim_{\text{type}} t'$ *and* t *is* $\to_{\mathsf{wh}}$*-normalizing then* t' *is* $\to_{\mathsf{wh}}$*-normalizing;*
3. Monotonicity: *if* $t \precsim_{\text{type}} t'$ *then* $C\langle t\rangle \precsim_{\text{type}} C\langle t'\rangle$ *for any context* C.

As a direct corollary of Point 2 and 3 of Proposition 7, the type preorder is included in the contextual preorder.

▶ **Corollary 8.** *The type preorder is included in the contextual preorder, i.e.* $\precsim_{\text{type}} \subseteq \precsim_{\text{ctx}}$. *Similarly for equivalences:* $\simeq_{\text{type}} \subseteq \simeq_{\text{ctx}}$.

4 Weak Head Normal Form Bisimilarity

In this section, we recall the definition of Sangiorgi's normal form (bi)similarity and show the correspondence with the type preorder.

First, from a relation $\mathcal{R}$ on terms we define its closure on $\rightarrow_{\mathsf{wh}}$-normal forms $\langle\mathcal{R}\rangle_{nf}$. Formally, $\langle\mathcal{R}\rangle_{nf}$ is defined, by induction, as the smallest relation including the three following rules:

$$\frac{}{x\,\langle\mathcal{R}\rangle_{nf}\,x} \qquad \frac{r\,\langle\mathcal{R}\rangle_{nf}\,r' \quad t\;\mathcal{R}\;u}{rt\,\langle\mathcal{R}\rangle_{nf}\,r'u} \qquad \frac{t\;\mathcal{R}\;u}{\lambda x.t\,\langle\mathcal{R}\rangle_{nf}\,\lambda x.u}$$

▶ **Definition 9** (Weak head normal form simulation). *A relation on terms $\mathcal{R}$ is a weak head normal form (whnf) simulation if for all t, t' such that $t\;\mathcal{R}\;t'$ either t does not $\rightarrow_{\mathsf{wh}}$-terminate, or t $\rightarrow_{\mathsf{wh}}$-reduces to a normal form w and t' $\rightarrow_{\mathsf{wh}}$-reduces to a normal form w' such that $w\,\langle\mathcal{R}\rangle_{nf}\,w'$.*

Whnf similarity, noted $\precsim_{\mathsf{wh}}$, is defined, by coinduction, as the largest whnf simulation.

Note that the definition here is slightly different than Sangiorgi's and Lassen's well-known presentations [23, 19] as we make explicit an inductive definition for rigid terms, via $\langle\mathcal{R}\rangle_{nf}$.

We define an $\mathtt{F}$-operator, $\mathtt{F} : \Lambda\times\Lambda \to \Lambda\times\Lambda$, where $\mathtt{F}(\mathcal{R})$ is the whnf simulation induced by $\mathcal{R}$. Formally:

$$\mathtt{F}(\mathcal{R}) := \left\{ (t,t') \mid \begin{array}{l} \text{either } t \text{ does not } \rightarrow_{\mathsf{wh}}\text{-terminate,} \\ \text{or } t \rightarrow_{\mathsf{wh}}\text{-reduces to a normal form } w \text{ and} \\ t' \rightarrow_{\mathsf{wh}}\text{-reduces to a normal form } w' \text{ such that } w\,\langle\mathcal{R}\rangle_{nf}\,w' \end{array} \right\}$$

We have that $\mathcal{R}$ is a whnf simulation iff $\mathcal{R} \subseteq \mathtt{F}(\mathcal{R})$. To ensure that whnf similarity can be coinductively defined, we prove that the $\mathtt{F}$-operator is monotone.

▶ **Proposition 10** (Monotonicity of whnf simulations). *If $\mathcal{R} \subseteq \mathcal{R}'$ then $\mathtt{F}(\mathcal{R}) \subseteq \mathtt{F}(\mathcal{R}')$*

Proof. Let $(t,t') \in \mathtt{F}(\mathcal{R})$. We have two cases:

- If t does not $\rightarrow_{\mathsf{wh}}$-terminate, then $(t,t') \in \mathtt{F}(\mathcal{R})$.
- Otherwise, t $\rightarrow_{\mathsf{wh}}$-reduces to a normal form w and t' $\rightarrow_{\mathsf{wh}}$-reduces to a normal form w' such that $w\,\langle\mathcal{R}\rangle_{nf}\,w'$. As $\mathcal{R} \subseteq \mathcal{R}'$, we have that $w\,\langle\mathcal{R}'\rangle_{nf}\,w'$ (by an easy induction on $\langle\cdot\rangle_{nf}$). We conclude that $(t,t') \in \mathtt{F}(\mathcal{R}')$ by definition. ◀

Type equivalence exactly matches normal form bisimilarity. In the two following sections, we will exactly characterize the type preorder by whnf similarity, *i.e.* $\precsim_{\mathsf{wh}}=\precsim_{\text{type}}$. We call *soundness* $\precsim_{\mathsf{wh}}\subseteq\precsim_{\text{type}}$, the easy part, and *completeness* $\precsim_{\mathsf{wh}}\supseteq\precsim_{\text{type}}$, the difficult part.

Monotonicity, for free. An interesting corollary of this characterization is an alternative of monotonicity for whnf similarity (*i.e.* that $\precsim_{\mathsf{wh}}$ is stable by contexts). Such a property is often also called *compatibility* and is the main technical point when showing soundness of a similarity with respect to contextual equivalence. The proof of such a property is not always complex, as there is a general method described by Lassen, but can be quite lengthy. Note that in some cases, Lassen's method does not directly apply and requires adjustments [11], thus motivating the need for alternative and lighter proof techniques.

▶ **Proposition 11** (Monotonicity of $\precsim_{\mathsf{wh}}$). *If $t \precsim_{\mathsf{wh}} u$ then for all C, $C\langle t\rangle \precsim_{\mathsf{wh}} C\langle u\rangle$.*

The proof follows from the exact characterization of $\precsim_{\mathsf{wh}}$ by $\precsim_{\text{type}}$ (Theorem 26) and the fact that the latter enjoys monotonicity (Point 3 of Proposition 7).

5 Soundness via Typing Transfer

We shall first show soudness and we do so via the following *typing transfer* proposition. It says that if two terms are whnf similar, then any typing of t transfers to t' (note that if t and t' are normal forms, even part of the structure of the derivation transfers). We show such a result via induction on the size of the derivation. A crucial ingredient is that subject reduction does not increase the size of the derivation, otherwise the induction argument would not work. The quantitative aspect of the subject reduction proof is therefore crucial and justifies the need of *multi* types.

▶ **Proposition 12** (Typing Transfer). *Let $\mathcal{R}$ be a whnf simulation. If $t\ \mathcal{R}\ t'$ and there exists a derivation $\pi \rhd \Gamma \vdash t : L$ then there exists a derivation $\pi' \rhd \Gamma \vdash t' : L$.*

Proof. By induction on the size of the derivation $\pi \rhd \Gamma \vdash t : L$.

The term t is typable by the derivation $\pi \rhd \Gamma \vdash t : L$ therefore it is normalizable by Theorem 3. Hence we have $t \to_{\mathsf{wh}}^{k} w$ and therefore (since $\mathcal{R}$ is a bisimulation) $t' \to_{\mathsf{wh}}^{*} w'$ with $w \, \langle \mathcal{R} \rangle_{nf} \, w'$ with w and w weak head normal forms. By quantitative subject reduction (Proposition 4), there is a derivation $\pi_1 \rhd \Gamma \vdash w : M$ whose size is at most the size of π. Instead of looking for a derivation π' of t', we can look for a derivation π_1' of w' and conclude by (qualitative) subject expansion for wh-reduction (Proposition 5).

Now, from $w \, \langle \mathcal{R} \rangle_{nf} \, w'$ and $\pi_1 \rhd \Gamma \vdash w : L$, we build the derivation $\pi_1' \rhd \Gamma \vdash w' : L$. By case analysis on the last rule of the derivation π_1:

1. *Axiom rule.*

$$\pi_1 : \quad \frac{}{x : [L] \vdash w = x : L}\ \mathsf{ax}$$

Then by $w = x \, \langle \mathcal{R} \rangle_{nf} \, w'$, $w' = x$ and $\pi_1' := \pi_1$ types w' accordingly.

2. *Abstraction-$\star$ rule.*

$$\pi_1 : \quad \frac{}{\vdash w = \lambda x.t : \star_i}\ \lambda$$

Then by $w = \lambda x.t \, \langle \mathcal{R} \rangle_{nf} \, w'$, $w' = \lambda x.u$ and $\pi_1' := \pi_1$ types w' accordingly.

3. *Abstraction rule.*

$$\pi_1 : \quad \frac{\Gamma, x : M \vdash u : L'}{\Gamma \vdash w = \lambda x.u : L = M \multimap L'}\ \lambda$$

Then by $w = \lambda x.u \, \langle \mathcal{R} \rangle_{nf} \, w'$, $w' = \lambda x.u'$ with $u\ \mathcal{R}\ u'$.
The derivation $\pi_2 \rhd \Gamma, x : M \vdash u : L'$ is of a strictly smaller size than π. By induction, since $u\ \mathcal{R}\ u'$, there is a derivation $\pi_2' \rhd \Gamma, x : M \vdash u' : L'$.
We build the derivation π_1' by applying the λ typing rule to π_2'.

4. *Application rule.*

$$\pi_1 : \quad \frac{\Gamma \vdash r : [L_i]_{i \in I} \multimap L \quad (\Delta_i \vdash t : L_i)_{i \in I}}{\Gamma \uplus (\uplus_{i \in I} \Delta_i) \vdash w = rt : L}\ @$$

Then by $w = rt \, \langle \mathcal{R} \rangle_{nf} \, w'$, $w' = r'u$ with $r \, \langle \mathcal{R} \rangle_{nf} \, r'$ and $t\ \mathcal{R}\ u$. By induction on the sub-derivations, we get the appropriate derivation for w'. ◀

From the fact that typings transfer for any whnf simulation, we deduce easily soundness ($\precsim_{\mathsf{wh}} \subseteq \precsim_{\text{type}}$) as typings transfer, in particular, for the largest whnf simulation that is whnf similarity.

▶ **Theorem 13.** *For all terms t, u, if $t \precsim_{\mathsf{wh}} u$ then $t \precsim_{\text{type}} u$.*

Proof. Let t, u terms such that $t \precsim_{\mathsf{wh}} u$. Let $\Gamma \vdash t : L$ a typing for t. As $\precsim_{\mathsf{wh}}$ is a whnf bisimulation, by Proposition 12, we have that $\Gamma \vdash u : L$. Hence $t \precsim_{\text{type}} u$. ◀

6 Completeness via Shape Typings

It remains now to show that if two terms are typable by the same intersection types, then they have to be syntactically similar, specifically that they have to be weak head normal form bisimilar.

Coinductive Argument. To prove completeness, *i.e.* that $\precsim_{\text{type}} \subseteq \precsim_{\mathsf{wh}}$, we shall use coinduction. We first show that $\precsim_{\text{type}}$ is a whnf simulation, which implies it is included in the largest whnf simulation, namely whnf similarity $\precsim_{\mathsf{wh}}$.

Building Shape Typings, Principally. To prove that the type preorder is a whnf simulation, we build specific *shape typings* that specify the shape of the normal form of a term. As an example, there exists Γ_x, L_x that ensure that for any term t typable by this typing, *i.e.* $\Gamma_x \vdash t\!:\!L_x$, we have that t weak head normalizes to x. In this specific case, $\Gamma_x \coloneqq x\!:\![\star_0]$ and $L_x \coloneqq \star_0$.

These shape typings specify the structure of normal form of any term they type. We choose them with some flexibility in mind so that they may be used compositionally. In that sense, they are *principal typings* in that they generate a number of other possible typings for terms that they may type. Indeed, another choice for Γ_x, L_x could be to replace $\star_0$ by any linear type. As an example, $\Gamma_x \coloneqq x\!:\![[\star_0] \multimap \star_1]$ and $L_x \coloneqq [\star_0] \multimap \star_1$ is also a correct typing for x but it is not separating, since it also types $\lambda y.xy$.

A first definition of shape typings can be seen with the following *(sub-)type system* (only typing weak head normal forms):

$$\frac{}{x\!:\![\star_k] \vdash_{\text{shape}} x\!:\!\star_k}\ \textsf{shape-ax} \qquad\qquad \frac{}{\emptyset \vdash_{\text{shape}} \lambda x.t\!:\!\star_k}\ \text{shape-}\lambda_k$$

$$\frac{\Gamma \vdash_{\text{shape}} r\!:\!\star_k \qquad \star_k \text{ appears only once in } \Gamma}{\Gamma\{\star_k \leftarrow [\,] \multimap \star_{k'}\} \vdash_{\text{shape}} rt\!:\!\star_{k'}}\ \text{shape-@}$$

where $\Gamma\{\star_k \leftarrow L\}$ denotes the substitution of a type variable (ground type) by a linear type L.

Dry typing systems follow similar definitions (see [2] where dry typings are defined in a more complex setting than weak head reduction). Coming up with these first shape typings resembles finding a typing only inhabited by one or few terms (up to β-conversion). Arrial's implementation of the inhabitation algorithm [7] was helpful to check ideas of shape typings (algorithm developed and proven correct in [8]).

This system is an *outer shape typing system* in the following sense:

1. Any normal form can be typed in the $\vdash_{\text{shape}}$ system;
2. The $\vdash_{\text{shape}}$ system is sound for the weak head multi type system $\vdash$: if $\Gamma \vdash_{\text{shape}} t\!:\!L$ then $\Gamma \vdash t\!:\!L$;
3. These shape typings allow us to distinguish terms with different outer shape normal forms.

Point 1 is immediate given the definition of $\vdash_{\text{shape}}$. Point 2–which is the fact that this transformation is sound with respect to the original weak head type system–is formalized with Lemma 16 (note that the lemma is more general, somehow allowing to substitute a linear type for another one). Point 3 is the starting point of our separation construction, formally specified in Proposition 17.

Proposition 17 (and the shape typing system $\vdash_{\text{shape}}$) is not enough to complete the proof that type equivalent terms generate the same (infinitary) Lévy-Longo tree (*i.e.* are normal form bisimilar), as some terms have the same outer shape but differ on a deeper syntax level (a difference between t and t' generates a difference between $\lambda x.t$ and $\lambda x.t'$, and between xt and xt'). We therefore need to generalize our shape typing technique to be able to detect deep differences (formally proven in the so-called *Normal Inversion Lemma* 21).

In a nutshell, the proof technique of this section will be decomposed in two steps:

1. Look for a possible outer difference with $\vdash_{\text{shape}}$;
2. If the outer syntax matches, go deeper with the inversion lemma.

We'll (coinductively) repeat the process to show that, if t type-improves t' then t is whnf similar to t'.

Building Shape Typings, Formally. Now, we formally prove completeness, by exhibiting our shape typings. We first specify a family of types, *erasing types*, that will be helpful to discriminate between terms and act as shape types for rigid normal forms.

▶ **Definition 14** (Erasing Type). *The* (k, i) *erasing type, written* E_i^k*, represents a computation that erases* k *arguments and returns the* i*th ground type. Precisely,* $E_i^k := ([\,] \multimap)^k \star_i$.

Erasing types can be used to type any rigid term $x\, t_1 \cdots t_k$, by assigning a (k, i) erasing type $([\,] \multimap)^k \star_i$ to the head variable x, resulting in the rigid term to be typed with the $(0, i)$ erasing type $\star_i$ (k is the number of arguments applied to the head variable).

To formally prove this, with the following lemma, we strengthen the inductive hypothesis by showing that assigning a (n, i) erasing type to the head variable x results in the rigid term to be typed with the $(n - k, i)$ erasing type.

▶ **Lemma 15** (Shape Typings for Rigid Terms). *Let* r *be a rigid term of head variable* x *and such that the head variable is applied to* k *arguments. Then, for all* $n \geq 0$ *and* i*,* $x : [E_i^{k+n}] \vdash r : E_i^n$.

In particular, for any i*,* $x : [E_i^k] \vdash r : \star_i$ *is an* (outer) shape typing *for the rigid term* r.[1]

Proof. By induction on rigid terms(/the number k).

- *Variable i.e.* $r = x$. Then the head variable of r is x and it is applied to 0 arguments. We conclude as $x : [E_i^n] \vdash x : E_i^n$ is a correct derivation for all n consisting only of the axiom rule.
- *Application i.e.* $r = r't$. We have that r' has the same head variable as r and that head variable is applied to $k - 1$ arguments in r'. Let n be a natural number.
  By induction, $x : [E_i^{k+n}] \vdash r' : E_i^{n+1}$. We conclude by building the following derivation:

$$\frac{x : [E_i^{k+n}] \vdash r' : E_i^{n+1}}{x : [E_i^{k+n}] \vdash r't : E_i^n}\ @$$

◀

The second property we shall use about shape typings is the following: all typings of rigid terms can be seen as extensions of shape typings, as there exists (1) a type for the head variable, (2) which determines the type of the rigid term.

[1] Note that this is the same shape typing as the one provided in the shape type system $\vdash_{\text{shape}}$, but written in a direct way.

▶ **Lemma 16** (Typings of Rigid Terms follow Shape Typings)**.** *Let r be a rigid term with head variable x and such that the head variable is applied to k arguments.*

For any typing $\Gamma \vdash r : L$, there exist k multi types $(M_i)_{1 \leq i \leq k}$ such that:

1. $[M_1 \multimap \cdots \multimap M_k \multimap L] \subseteq \Gamma(x)$*;*
2. *For any linear type L', we have a new typing $\Gamma' \uplus x : [M_1 \multimap \cdots \multimap M_k \multimap L'] \vdash r : L'$, with $\Gamma' \coloneqq \Gamma \setminus (x : [M_1 \multimap \cdots \multimap M_k \multimap L])^2$.*

Proof. By induction on r.

- *Variable i.e. $r = x$.* The last rule of any derivation must be the axiom rule, which ensures that $\Gamma(x) = [L]$. For the second point, it suffices to see that $x : [L'] \vdash x : L'$ is valid for any L'.
- *Application i.e. $r = r't$.* The last rule of any derivation must be the application rule.

$$\frac{\Delta \vdash r' : [L_i]_{i \in I} \multimap L \quad (\Pi_i \vdash t : L_i)_{i \in I}}{\Gamma = \Delta \uplus \uplus_{i \in I} \Pi_i \vdash r : L} \ @$$

By induction there exists $k-1$ multi types $(M_i)_{1 \leq i \leq k}$ such that $[M_1 \multimap \ldots M_{k-1} \multimap ([L_i]_{i \in I} \multimap L)] \subseteq \Delta(x)$. Set $M_k \coloneqq [L_i]_{i \in I}$ then $[M_1 \multimap \ldots M_k \multimap L)] \subseteq \Gamma(x)$ as $\Delta \subseteq \Gamma$. For the second point, by induction, for all linear type L', we have that $x : [M_1 \multimap \ldots M_{k-1} \multimap ([L_i]_{i \in I} \multimap L')], \Delta' \vdash r : ([L_i]_{i \in I} \multimap L')$, hence we conclude by applying the application rule. ◀

A first step towards showing that the type preorder is a whnf simulation is to show that the structure of the normal forms are preserved (the outermost syntax), which is specified in the following proposition. In the proof, we use shape typings for rigid terms, as well as a shape typing for abstractions $(\emptyset, \star_0)$.

▶ **Proposition 17** (Type preorder preserves the shape of normal forms)**.** *Let w be a weak normal form, t a term, and r a rigid term.*

1. Variable Preservation: *if $x \precsim_{\text{type}} w$, then $w = x$.*
2. Abstraction Preservation: *if $\lambda x.t \precsim_{\text{type}} w$, then $w = \lambda x.u$ for some term u.*
3. Rigid Preservation: *if $rt \precsim_{\text{type}} w$, then $w = r'u$ for some rigid r' and some term u.*

Proof.

1. Suppose $w \neq x$. Cases of w:
   - *Different Variable:* if $w = y$, then $x : [\star_0] \vdash x : \star_0$, but y is not typable by $(x : [\star_0], \star_0)$.
   - *Abstraction:* if $w = \lambda y.u$, then $x : [\star_0] \vdash x : \star_0$, but $\lambda y.u$ is not typable by $(x : [\star_0], \star_0)$, since typing by $\star_0$ requires an empty context for abstractions.
   - *Rigid Term:* if $w = r'u$, then $x : [\star_0] \vdash x : \star_0$, but $x : [\star_0] \not\vdash r'u : \star_0$, as a non variable rigid term needs an arrow type in the context for its head variable (per Lemma 16).
2. We know that $\emptyset \vdash \lambda x.t : \star_0$ however $\emptyset \not\vdash r : \star_0$ for all rigid term r (at least one variable has to appear with a non-empty type in the context by Lemma 16). Thus, w is an abstraction of the form $\lambda x.u$ for some term u.
3. There exists a derivation $x : [E_0^k] \vdash rt : \star_0$ where $k \geq 1$ by Lemma 15 but $x : [E_0^k] \not\vdash y : \star_0$ for all y variables and $x : [E_0^k] \not\vdash \lambda y.s : \star_0$ for all abstraction $\lambda y.s$. Hence w is a non-variable rigid term. ◀

[2] The notation $\Gamma' \coloneqq \Gamma \setminus \Delta$ means that Γ' is such that $\Gamma = \Gamma' \uplus \Delta$. Note that this definition does not constrain the domains of Γ' and Δ to be disjoint.

The main properties of shape typings (Lemma 16) are used to prove the two following lemmas, which shall be key to be able to know the structure of typing derivation.

▶ **Lemma 18.** *Let r be a rigid term of head variable x such that $\Gamma \vdash r : L$. If $x : [([\,] \multimap)^k L] \subseteq \Gamma$ for some $k \geq 0$ and L does not appear in other types in Γ, then $\Gamma = x : [([\,] \multimap)^k L]$.*

Proof. By induction on rigid terms:

- $r = x$. Then the only derivation possible for r is $x : [L] \vdash x : L$, for which the statement holds.
- $r = r't$. Then the derivation $\Gamma \vdash r : L$ must start with a typing rule @:

  $$\frac{\Gamma_0 \vdash r' : [L_i]_{i \in I} \multimap L \quad (\Gamma_i \vdash t' : L_i)_{i \in I}}{\Gamma \vdash r : L}$$

  with $\Gamma = \Gamma_0 \uplus (\uplus_{i \in I} \Gamma_i)$.
  By Lemma 16, $x : [([\,] \multimap)^k L]$ is included in Γ_0 (as it is the only occurrence of the L type).
  By induction hypothesis, $\Gamma_0 = x : [([\,] \multimap)^k L]$. Then by Lemma 16, I must be empty (otherwise it would reflect in the type of x). Hence $\Gamma = x : [([\,] \multimap)^k L]$. ◀

▶ **Lemma 19.** *Let r be a rigid term of head variable x such that $\Gamma \vdash r : M_i \multimap \cdots \multimap M_k \multimap L$ for some $i \leq k$. If $x : [N_1 \multimap \cdots \multimap N_k \multimap L] \subseteq \Gamma$ for some $k \geq 0$ and L does not appear in other types in Γ, then $M_j = N_j$ for $i \leq j \leq k$.*

Proof. By induction on rigid terms:

- $r = x$. Then the only derivation possible for r is $x : [M_1 \multimap \cdots \multimap M_k \multimap L] \vdash x : M_1 \multimap \cdots \multimap M_k \multimap L$, for which the statement holds.
- $r = r't$. Then the derivation $\Gamma \vdash r : M_i \multimap \cdots \multimap M_k \multimap L$ starts with a typing rule @:

  $$\frac{\Gamma_0 \vdash r' : [L_i]_{i \in I} \multimap M_i \multimap \cdots \multimap M_k \multimap L \quad (\Gamma_i \vdash t' : L_i)_{i \in I}}{\Gamma \vdash r : M_i \multimap \cdots \multimap M_k \multimap L}$$

  with $\Gamma = \Gamma_0 \uplus (\uplus_{i \in I} \Gamma_i)$.
  By Lemma 16, $x : [N_1 \multimap \cdots \multimap N_k \multimap L]$ is included in Γ_0 (as it is the only occurrence of the L type).
  By induction hypothesis, $N_{i-1} = [L_i]_{i \in I}$ and $M_j = N_j$ for $i \leq j \leq k$, which concludes the proof. ◀

The following proposition is the only point where we use the fact that there are countably many distinct ground types: in all other proofs, we either rely on this proposition or only use the first ground type $\star_0$. This restriction is actually not needed but eases the proof. In fact, it is enough to have only one ground type and be able to specify large enough types. We show after the proof how to choose the right typings if there is only a single ground type.

The proposition states that normal forms related by the type preorder may be broken apart into subterms that are still related by the type preorder. In this sense, the type preorder is *decompositional* on normal forms.

▶ **Proposition 20** (Decompositionality of $\precsim_{\text{type}}$). *Let t, t' be terms and r, r' be rigid terms.*

1. Left preservation: *if $rt \precsim_{\text{type}} r't'$ then $r \precsim_{\text{type}} r'$.*
2. Right preservation: *if $rt \precsim_{\text{type}} r't'$ and $r \precsim_{\text{type}} r'$ then $t \precsim_{\text{type}} t'$.*

Proof.

1. Let Γ, L such that $\Gamma \vdash r : L$. We shall show that $\Gamma \vdash r' : L$ as well.
   Let i be the maximum index of $\star_i$ that appears in Γ, L (if it does not appear, $i = 0$). By Lemma 16, we can set $L' := [\,] \multimap \star_{i+1}$ and $\Delta := \Gamma', x : [M_1 \multimap \cdots \multimap M_k \multimap L']$ (where $\Gamma = \Gamma', x : [M_1 \multimap \cdots \multimap M_k \multimap L]$) and we still have that $\Delta \vdash r : L'$. It is clear that $(\Delta, \star_{i+1})$ types rt:

   $$\frac{\Delta \vdash r : [\,] \multimap \star_{i+1}}{\Delta \vdash rt : \star_{i+1}}$$

   By hypothesis, we have that $r't'$ is typable by $(\Delta, \star_{i+1})$. The derivation starts as:

   $$\frac{\Gamma_0 \vdash r' : [L_i]_{i \in I} \multimap \star_{i+1} \qquad (\Gamma_i \vdash t' : L_i)_{i \in I}}{\Delta \vdash r't' : \star_{i+1}}$$

   By Lemma 16, $x : [M_1 \multimap \cdots \multimap M_k \multimap L']$ is included in Γ_0 (as L' is the only type where $\star_{i+1}$ occurs in Γ). By Lemma 19 applied on $\Gamma_0 \vdash r' : [L_i]_{i \in I} \multimap \star_{i+1}$, we know $[L_i]_{i \in I} = [\,]$. Hence I is empty and $\Gamma_0 = \Delta$.
   Hence, we have a derivation of final judgment $\Delta \vdash r' : L'$. By Lemma 16 and as $\star_{i+1}$ only appears in one linear type in Δ, we have that $\Gamma \vdash r' : L$ as well.
2. Let Γ, L such that $\Gamma \vdash t : L$. We shall show that $\Gamma \vdash t' : L$.
   Let i be the maximum index of $\star_i$ that appears in Γ, L (if it does not appear, $i = 0$). Let $\Delta := x : [([\,] \multimap)^k \multimap [L] \multimap \star_{i+1}]$ such that $\Delta \vdash r : [L] \multimap \star_{i+1}$ (x is then the head variable of r).

   $$\frac{\Delta \vdash r : [L] \multimap \star_{i+1} \qquad \Gamma \vdash t : L}{\Delta \uplus \Gamma \vdash rt : \star_{i+1}}$$

   Then, as $r \precsim_{\text{type}} r'$, we also have that $\Delta \vdash r' : [L] \multimap \star_{i+1}$.
   By hypothesis, we have that $\Delta \uplus \Gamma \vdash r't' : \star_{i+1}$. The derivation starts with the application rule:

   $$\frac{\Gamma_0 \vdash r' : [L_i]_{i \in I} \multimap \star_{i+1} \qquad (\Gamma_i \vdash t' : L_i)_{i \in I}}{\Delta \uplus \Gamma \vdash r't' : \star_{i+1}}$$

   By Lemma 16, Δ must be included in Γ_0 (as it is the only occurrence of the $\star_{i+1}$ type). The only derivation of r' containing Δ in the context must be $\Delta \vdash r' : [L_i]_{i \in I} \multimap \star_{i+1}$ by Lemma 18. Hence I only contains one element and $L_1 = L$ (by Lemma 16, as the type $[L_i]_{i \in I} \multimap \star_{i+1}$ must appear in Δ).
   Hence, we have a derivation of final judgment $\Gamma \vdash t' : L$ which concludes the proof. ◀

A Single Ground Type is Enough. The proof above uses the fact that there is a countable number of distinct ground types in our multi types–but it is not needed. Alternatively, the key point is to set up a notion of size of linear types by counting the number of top-most arrows in linear types (counting only arrows appearing outside of multi types). We shall use types of size i (that is, with i arrows outside of multi types) as replacement for i-th ground types. Then one has to lookup in Γ, L the *largest* linear type ($i :=$ the maximum number of top-most arrows of a type L' that appears in Γ, L–L' can appear anywhere, even inside multi types) and set up a strictly larger linear type (that is, with strictly more arrows) as our $\star_{i+1}$. Then the proof follows by using Lemma 16, Lemma 18 and Lemma 19 similarly (these lemmas are not particular for countable ground types but mention linear types appearing only once).

Back to the Completeness Proof. The following technical lemma is straightforward and used in subsequent proofs of other lemmas.

▶ **Lemma 21.** *If $r \langle \precsim_{\text{type}} \rangle_{nf}\, r'$ then $r \precsim_{\text{type}} r'$.*

Proof. By induction on rigid terms:

- $r = x$, then by $r \langle \precsim_{\text{type}} \rangle_{nf}\, r'$ we have that $r' = x$. It is trivial to show that $r \precsim_{\text{type}} r'$.
- $r = r_1 t_1$, then by $r \langle \precsim_{\text{type}} \rangle_{nf}\, r'$ we have that $r' = r_2 t_2$ such that $r_1 \langle \precsim_{\text{type}} \rangle_{nf}\, r_2$ and $t_1 \precsim_{\text{type}} t_2$. By induction we have that $r_1 \precsim_{\text{type}} r_2$, and hence $r_1 t_1 \precsim_{\text{type}} r_2 t_2$ by compositionality of $\precsim_{\text{type}}$. ◀

Another technical lemma is the reverse implication of Lemma 21, which is slightly less straightforward. We separate them as this second lemma's proof is more intricate, relying on Proposition 20.

▶ **Lemma 22.** *If $r \precsim_{\text{type}} r'$ then $r \langle \precsim_{\text{type}} \rangle_{nf}\, r'$.*

Proof. By induction on r:

- *Variable*: $r = x$. By Point 1 of Proposition 17, $r' = x$, hence $r \langle \precsim_{\text{type}} \rangle_{nf}\, r'$.
- *Applied Rigid Term*: $r = r_1 t_1$. By Point 3 of Proposition 17, $r' = r_2 t_2$.
  By Point 1 and subsequently Point 2 of Proposition 20, both $r_1 \precsim_{\text{type}} r_2$ and $t_1 \precsim_{\text{type}} t_2$.
  By induction, $r_1 \langle \precsim_{\text{type}} \rangle_{nf}\, r_2$. We can now conclude that $r = r_1 t_1 \langle \precsim_{\text{type}} \rangle_{nf}\, r_2 t_2 = r'$. ◀

Proposition 17 states that normal forms related by the type preorder must have the same normal form structure. The next lemma completes this, by stating that normal forms with the same structure and related by the type preorder have inner sub-terms related by the type preorder as well. This lemma is similar to an inversion lemma, but only working on normal forms, hence its name of *normal inversion lemma.*

▶ **Lemma 23** (Normal Inversion Lemma for $\precsim_{type}$)**.** *Let t, t' be terms and r, r' be rigid terms.*

1. Body of Abstractions: *if $\lambda x.t \precsim_{\text{type}} \lambda x.t'$ then $t \precsim_{\text{type}} t'$.*
2. Rigid Head: *if $rt \precsim_{\text{type}} r't'$ then $r \langle \precsim_{\text{type}} \rangle_{nf}\, r'$.*
3. Rigid Arguments: *if $rt \precsim_{\text{type}} r't'$ and $r \langle \precsim_{\text{type}} \rangle_{nf}\, r'$ then $t \precsim_{\text{type}} t'$*

Proof.

1. Let Γ, M, L such that $\Gamma; x\!:\!M \vdash t\!:\!L$. We shall show that $\Gamma; x\!:\!M \vdash t'\!:\!L$. It is easy to see that $\Gamma \vdash \lambda x.t\!:\!M \multimap L$ by applying rule λ. By hypothesis, we have that the same typing works for $\lambda x.t'$, *i.e.* $\Gamma \vdash \lambda x.t'\!:\!M \multimap L$. Then the typing derivation starts by the rule λ and the premise will read $\Gamma; x\!:\!M \vdash t'\!:\!L$, which concludes the proof.
2. By Point 1 of Prop. 20, $r \precsim_{\text{type}} r'$. By Lemma 22, we have that $r \langle \precsim_{\text{type}} \rangle_{nf}\, r'$.
3. By Lemma 21, as $r \langle \precsim_{\text{type}} \rangle_{nf}\, r'$ we have that $r \precsim_{\text{type}} r'$. By Point 2 of Prop. 20, we conclude that $t \precsim_{\text{type}} t'$. ◀

Now, we are able to prove that for all normal forms, if they are related by the type preorder, then they are related by one step of simulation over the type preorder. The proof uses, first, the fact that the outer shape of normal forms match (Proposition 17) and, second, the normal inversion lemma (Lemma 23).

▶ **Theorem 24.** *Let w and w' be two* wh*-normal forms such that $w \precsim_{\text{type}} w'$. Then $w \langle \precsim_{\text{type}} \rangle_{nf} w'$.*

Proof. By case analysis on w:

1. If $w = x$, then by Point 1 of Prop. 17 we have that $w' = x$.
2. If $w = \lambda x.t$. Then by Point 2 of Prop. 17, we have that $w' = \lambda x.u$ for some term u. By Point 1 of Lemma 23, we have that $t \precsim_{\text{type}} u$.
3. If $w = rt$, then by Point 3 of Prop. 17, we have that $w' = r'u$ for some rigid term r' and some term u. By Point 2 of Lemma 23, we have that $r \langle \precsim_{\text{type}} \rangle_{nf} r'$. By Point 3 of Lemma 23, we have that $t \precsim_{\text{type}} u$. ◀

The previous theorem is enough to prove that the type preorder is a whnf simulation, as whnf simulations only compare normal forms. We are then able to prove our final theorem that concludes the syntactic characterization of the type preorder.

▶ **Theorem 25.**

1. *The type preorder $\precsim_{\text{type}}$ is a whnf simulation.*
2. Completeness: $\precsim_{\text{type}} \subseteq \precsim_{\mathsf{wh}}$

Proof.

1. Let t and t' be such that $t \precsim_{\text{type}} t'$. If t does not $\to_{\mathsf{wh}}$-terminate, we have nothing else to check.
   Otherwise, we have that $t \to_{\mathsf{wh}}^* w$ such that w is a weak head normal form and by Point 2 of Proposition 7, we have that $t' \to_{\mathsf{wh}}^* w'$ such that w' is also a weak head normal form. As $t \precsim_{\text{type}} t'$, and by subject expansion and reduction, $w \precsim_{\text{type}} w'$. By Theorem 24, we have that $w \langle \precsim_{\text{type}} \rangle_{nf} w'$, which concludes the proof.
2. We can finally apply the coinductive argument: $\precsim_{\text{type}}$ is a whnf simulation by Point 1 and $\precsim_{\mathsf{wh}}$ is the largest whnf simulation by definition. ◀

By combining soundness (Theorem 13) and completeness (Theorem 25), we get the complete syntactic characterization of the weak head type preorder $\precsim_{\text{type}}$.

▶ **Theorem 26.** *Weak head normal form similarity coincides with the weak type preorder, that is, for all terms t, u, $t \precsim_{\mathsf{wh}} u$ iff $t \precsim_{\text{type}} u$. By symmetry, $\simeq_{\mathsf{wh}} = \simeq_{\text{type}}$ as well.*

Type-Böhm Theorem for Weak Head. A corollary is a separation theorem (akin to Böhm theorem, but adapted for types) for β-normal forms. The standard Böhm theorem gives separating evaluation contexts for distinct $\beta\eta$-normal forms. Here, we give separating *types* and for distinct β-normal forms: our weak head types are able to distinguish η-equivalent terms.

▶ **Corollary 27** (Type-Böhm Theorem). *Let n, n' be two syntactically distinct β-normal forms. There exists a typing context Γ and a type L such that $\Gamma \vdash n : L$ but $\Gamma \nvdash n' : L$.*

Proof. It is clear that $n \not\precsim_{\mathsf{wh}} n'$ for distinct β-normal forms n and n'. Then, one can apply Theorem 25 to deduce a separating typing context and type. ◀

7 Conclusions

We study the equational theory induced by multi types and focus, in particular, on the proof technique to coinductively characterize it. We are able to avoid introducing term approximants, refining previous works by Breuvart et al. [13] and Ronchi Della Rocca [22].

We apply the proof technique to a simple case where one considers weak head multi types, the appropriate multi types for weak head evaluation. We construct shape typings in order to show that terms that are typable by the same types must have the same normal forms. Building these shape typings is done explicitly, and they resemble the types built by Breuvart et al. [13] in the case of head reduction in order to show that different approximants cannot be type equivalent. After carefully examining the proof, we are able to clearly state that a countable number of ground types are not needed, a single ground type being enough. We also go for a more direct proof, without considering the contrapositive statement (syntactically different terms implies that there exists a separating typing), closer to Ronchi Della Rocca's separation [22], which is also based on head reduction.

This work bridges between two programming languages approaches: the study of the untyped λ-calculus with its models and bisimilarity, a technique imported from process calculi. While the main result is folklore for experts of the λ-calculus (Lévy-Longo tree equivalence coincides with type equivalence), the coinductive flavor of bisimilarity appears to be new in that equivalence–and makes for quite independent lemmas and easy proofs.

Future Work. The work presented here focuses on *non-idempotent* intersection types, which clearly plays a critical role in the soundness proof but appears to be less critical in our completeness development. Techniques with term approximants have been successful in the idempotent setting to characterize the equational theory, which raises the question whether or not our coinductive presentation can also work with idempotent intersection type systems.

We are working on re-using the proof technique described in this paper to a trickier setting, namely the call-by-value λ-calculus. It would be interesting to give a syntactical characterization of the equational theory induced by Ehrhard's call-by-value relational semantics [17], for which there are none. A first (wrong) conjecture appeared in [18]. Preliminary results containing only typing transfer are already available on arXiv [5].

Recent work on CbN idempotent intersection types by Polonsky and Statman discuss uniqueness typings to separate terms up to $\beta\eta$ equivalence [20]. It would be interesting to see if their results generalize to normal form bisimilarity.

Recent developments have been able to refine contextual equivalence to a quantitative contextual equivalence, while crucially retaining the fact that this equivalence is invariant by β (therefore, an equational theory) [6]. This work focusses on call-by-name and head reduction, where it is shown that Böhm tree equivalence coincides with this new quantitative equivalence. The authors conjecture that Lévy-Longo tree equivalence could match a *weak head* quantitative contextual equivalence.

As said in the introduction, we do not use a contrapositive statement to show that type equivalence implies normal form bisimilarity. In intermediate lemmas, we may use classical reasoning techniques. It would be interesting to check (possibly via a proof assistant) which steps can be done only with constructive/intuistionistic reasoning. In the case of Böhm out technique, we are not aware of a result that does not use the contrapositive yet the result is deeply constructive (constructing explicitly a separating context).

References

1 S. Abramsky and C. H. L. Ong. Full Abstraction in the Lazy Lambda Calculus. *Information and Computation*, 105(2):159–267, 1993. doi:10.1006/inco.1993.1044.

2 Beniamino Accattoli. Semantic Bounds and Multi Types, Revisited. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:24, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2024.7.

3 Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds. *PACMPL*, 2(ICFP):94:1–94:30, 2018. doi:10.1145/3236789.

4 Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. Types by need. In Luís Caires, editor, *Programming Languages and Systems*, pages 410–439, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-17184-1_15.

5 Beniamino Accattoli, Adrienne Lancelot, and Claudia Faggian. Normal form bisimulations by value, 2023. doi:10.48550/arXiv.2303.08161.

6 Beniamino Accattoli, Adrienne Lancelot, Giulio Manzonetto, and Gabriele Vanoni. Interaction equivalence. *Proc. ACM Program. Lang.*, 9(POPL), January 2025. doi:10.1145/3704891.

7 Victor Arrial, Giulio Guerrieri, and Delia Kesner. An implementation of the quantitative inhabitation for different lambda calculi in a unifying framework, 2023. doi:10.1145/3554339.

8 Victor Arrial, Giulio Guerrieri, and Delia Kesner. Quantitative inhabitation for different lambda calculi in a unifying framework. *Proc. ACM Program. Lang.*, 7(POPL):1483–1513, 2023. doi:10.1145/3571244.

9 Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1984.

10 Henk Barendregt and Giulio Manzonetto. *A Lambda Calculus Satellite*. College Publications, 2022. URL: https://www.collegepublications.co.uk/logic/mlf/?00035.

11 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Proving soundness of extensional normal-form bisimilarities. *Electronic Notes in Theoretical Computer Science*, 336:41–56, 2018. The Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII). doi:10.1016/j.entcs.2018.03.015.

12 Viviana Bono and Mariangiola Dezani-Ciancaglini. A tale of intersection types. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 7–20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394733.

13 Flavien Breuvart, Giulio Manzonetto, and Domenico Ruoppolo. Relational Graph Models at Work. *Logical Methods in Computer Science*, Volume 14, Issue 3, July 2018. doi:10.23638/LMCS-14(3:2)2018.

14 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Not enough points is enough. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic*, pages 298–312, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74915-8_24.

15 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017. doi:10.1093/jigpal/jzx018.

16 Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, and Silvia Likavec. Behavioural inverse limit λ-models. *Theoretical Computer Science*, 316(1):49–74, 2004. Recent Developments in Domain Theory: A collection of papers in honour of Dana S. Scott. doi:10.1016/j.tcs.2004.01.023.

17 Thomas Ehrhard. Collapsing non-idempotent intersection types. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.CSL.2012.259.

18 Axel Kerinec, Giulio Manzonetto, and Michele Pagani. Revisiting call-by-value böhm trees in light of their taylor expansion. *Log. Methods Comput. Sci.*, 16(3), 2020. URL: https://lmcs.episciences.org/6638.

19 Søren B. Lassen. Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context. In Stephen D. Brookes, Achim Jung, Michael W. Mislove, and Andre Scedrov, editors, *Fifteenth Conference on Mathematical Foundations of Progamming Semantics, MFPS 1999, Tulane University, New Orleans, LA, USA, April 28 - May 1, 1999*, volume 20 of *Electronic Notes in Theoretical Computer Science*, pages 346–374. Elsevier, 1999. doi:10.1016/S1571-0661(04)80083-5.

20 Andrew Polonsky and Richard Statman. On sets of terms having a given intersection type. *Logical Methods in Computer Science*, Volume 18, Issue 3, September 2022. `doi:10.46298/lmcs-18(3:35)2022`.

21 Simona Ronchi Della Rocca. Intersection Types and Denotational Semantics: An Extended Abstract. In Silvia Ghilezan, Herman Geuvers, and Jelena Ivetic, editors, *22nd International Conference on Types for Proofs and Programs (TYPES 2016)*, volume 97 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:7, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.TYPES.2016.2`.

22 Simonetta Ronchi Della Rocca. Characterization theorems for a filter lambda model. *Information and Control*, 54(3):201–216, 1982. `doi:10.1016/S0019-9958(82)80022-3`.

23 Davide Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1):120–153, 1994. `doi:10.1006/inco.1994.1042`.

Implementing a Type Theory with Observational Equality, Using Normalisation by Evaluation

Matthew Sirman
University of Cambridge, UK

Meven Lennon-Bertrand
University of Cambridge, UK

Neel Krishnaswami
University of Cambridge, UK

Abstract

We report on an experimental implementation in Haskell of a dependent type theory featuring an observational equality type, based on Pujet et al.'s $\mathrm{CC}^{\mathrm{obs}}$. We use normalisation by evaluation to produce an efficient normalisation function, which is used to implement a bidirectional type checker. To allow for greater expressivity, we extend the core $\mathrm{CC}^{\mathrm{obs}}$ calculus with quotient types and inductive types. To make the system usable, we explore various proof-assistant features, notably a rudimentary version of a "hole" system similar to Agda's. While rather crude, this experience should inform other, more substantial implementation efforts of observational equality.

2012 ACM Subject Classification Software and its engineering → Functional languages; Theory of computation → Type theory; Theory of computation → Denotational semantics

Keywords and phrases Dependent type theory, Bidirectional typing, Observational equality, Normalisation by evaluation

Digital Object Identifier 10.4230/LIPIcs.TYPES.2024.5

Related Version *Dissertation*: `https://www.cl.cam.ac.uk/~nk480/sirman-dissertation.pdf` [30]

Supplementary Material
Software: `https://github.com/matthew-sirman/observational-type-theory` [31]
archived at `swh:1:dir:a55c73294775921cf98631f7da69b9a9a3dabf82`

1 Introduction

Since the inception of logics based on (dependent) type theories, *propositional equality*, i.e. the one manipulated in proofs, has been a thorn in the side of users. In recent years, an alternative has been proposed by Altenkirch et al. [5, 6], and developed by Pujet and Tabareau [27, 26, 28]: *observational equality.* It has two main characteristics. First, it is *definitionally proof-irrelevant*: any two proofs of equality are identified in the type theory, drastically simplifying its behaviour. Second, rather than being defined uniformly, observational equality has a specific behaviour at each type. Equality between functions is pointwise equality (*function extensionality*), equality between propositions is logical equivalence (*propositional extensionality*), and equality at quotient types is the relation by which the quotient was taken. Together, these aspects make equality closer to what mathematicians are used to, and allow seamless support for quotient types. Lean's mathematics library [32], a leading effort in formalized mathematics, relies on such a definitionally irrelevant equality with function extensionality and quotient, although their approach is type-theoretically somewhat ill-behaved compared to observational equality.

Pujet and Tabareau's work comes with extensive meta-theory, but no implementation. We attack this unexplored aspect with an experimental implementation of $\mathrm{CC}^{\mathrm{obs}}$, based on *normalisation by evaluation* (NbE) [1]. NbE is a modern technique to decide *definitional*

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg; Article No. 5; pp. 5:1–5:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

equality, the equations that the type-checker is able to automatically enforce – in contrast with propositional equality, which require explicit proofs. To decide definitional equality, NbE follows the naïve strategy of computing normal forms for the terms/types under scrutiny. NbE shines, in that it very efficiently computes these normal forms, by instrumenting the evaluation mechanism of the host language.[1]

Contributions. In our implementation, we extend standard NbE techniques, as presented by e.g. Abel [1] and in Kovács' `elaboration-zoo` [19], to an extension of Pujet and Tabareau's $\mathrm{CC}^{\mathrm{obs}}$ [26]. In addition to the constructions already handled by Abel and Kovács, our type theory features a sort of definitionally irrelevant (strict) propositions Ω [13], an observational equality valued in that sort, and quotient types by Ω-valued relations. We also explore inductive types, as first-class constructs equipped with a Mendler-style recursion [23].

Experimental implementations of $\mathrm{CC}^{\mathrm{obs}}$ and of NbE for strict propositions already exist [7, 9], but to the best of our knowledge we are the first to describe one in print. The latter was also theoretically studied [2], but not implemented, and we observe (in Sec. 3.3) that their approach is actually problematic. Our dependently-typed adaptation of Mendler-style induction is also novel, although it would deserve a theoretical investigation we lack.

A last contribution is in some sense a non-contribution: an experience report that, apart from subtleties around strict propositions, NbE mostly *just worked.* This is not our achievement, but we believe it is nonetheless important to stress. The same can also be said of other techniques, such as bidirectional typing and pattern unification, which readily adapted to our setting.

In Section 2 we present the type theory we implement. As our base NbE algorithm is very close to Abel's [1], we refer the reader to that work for background. In Section 3, we tackle the core of our implementation, NbE and type-checking for $\mathrm{CC}^{\mathrm{obs}}$. Section 4 presents extensions: quotient and inductive types, and a lightweight feature similar to Agda's holes.

The Haskell code for the implementation is freely available on GitHub [31]. This article is based on the first author's Part III dissertation [30].

2 Background

Our type theory is an extension of $\mathrm{CC}^{\mathrm{obs}}$ [26, 27, 28], itself based on Martin-Löf Type Theory (MLTT) [21], the staple dependently-typed theory. In this section, we first quickly sum up the additions made by $\mathrm{CC}^{\mathrm{obs}}$ compared to MLTT. Our version of $\mathrm{CC}^{\mathrm{obs}}$ is very close to that of Pujet [26], to which we refer for an extensive discussion. We then present the main point where we depart from it: the addition of inductive type as a first-class construct in the language, featuring Mendler-style recursion.

2.1 Observational type theory

Martin-Löf Type Theory. MLTT is a dependent type theory presented by five mutually defined judgements, characterizing well-formed context $\vdash \Gamma$, types $\Gamma \vdash A$ and terms $\Gamma \vdash t : A$, and asserting that two terms (resp. types) are *convertible* or *definitionally equal* $\Gamma \vdash A \equiv A'$ (resp. $\Gamma \vdash t \equiv t' : A$). During type-checking we need to compare (dependent) types, which can contain terms. Thus, equations between the latter can appear when comparing the former, meaning we have to decide conversion between arbitrary terms.

[1] We somewhat depart from this by implementing defunctionalized NbE [1], where closures are used instead of meta-level functions.

MLTT is a language with binders, represented with names in the text for readability. In the implementation, the front language has names, but internally we use de Bruijn indices for terms and de Bruijn levels for semantic values, as is standard for NbE [1].

We use a single universe $\mathcal{U}$ as the type of all types, with $\mathcal{U} : \mathcal{U}$. This is known to break termination of the system [14], making the type theory undecidable and inconsistent. Yet, as this is orthogonal to our focus, we go with the simple albeit inconsistent approach in our prototype. Apart from this, our MLTT is standard, featuring dependent function (Π) and pair (Σ) types with their η-laws, natural numbers with large elimination, and a unit type.

Proof irrelevance. The first extension of $\mathrm{CC}^{\mathrm{obs}}$ compared to MLTT are *proof-irrelevant propositions*, given by a universe Ω with the following conversion rule:

$$\frac{\Gamma \vdash P : \Omega \qquad \Gamma \vdash t : P \qquad \Gamma \vdash u : P}{\Gamma \vdash t \equiv u : P}$$

We use the symbol $\mathfrak{s}$ for an arbitrary sort, $\mathcal{U}$ or Ω. Propositions include the false and true propositions $\bot$ and $\top$, and existential quantification $\exists(x :_{\mathfrak{s}} A).\ B$. A Π type with a propositional codomain is again a proposition, representing universal quantification if the domain is relevant, or (dependent) implication if it is not.

Observational equality. Observational equality is a family of types, representing identifications between two inhabitants of the same type.

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \sim_A u : \Omega}$$

Equality is proven by reflexivity $\mathbf{refl}(t) : t \sim_A t$, and can be used in two different ways: transport $\mathbf{transp}(t, x\ p.\ C, u, t', e)$ lets us use the proof $e : t \sim_A t'$ to turn the proof $u : C[t/x, \mathbf{refl}\ t/p] : \Omega$ into a proof of $C[t'/x, e/p]$;[2] with cast $\mathbf{cast}(A, B, e, t)$, we can use a proof $e : A \sim_U B$ to construct an inhabitant of B one of A. The difference is that the latter applies to *relevant* types, while the former proves a proposition. Thus, only **cast** needs to be endowed with computational content, as all propositions are convertible.

Beyond **refl**, we add constants for symmetry (**sym**) and transitivity (**trans**). These are provable using **transp** and **refl**, but some of our computation rules need them, so it is easier and cleaner to add them as primitives. This is a benefit of strict propositions: since there is no computation in the irrelevant layer, we are free to add propositional constants without needing to endow them with computational content.

Contrarily to MLTT, where equality is a type constructor and cast computes on reflexivity, in $\mathrm{CC}^{\mathrm{obs}}$ both the equality type and cast compute on the types. Yet, we still retain the following conversion, a generalisation of β reduction of **cast** in MLTT – its special case when e is $\mathbf{refl}_A$. This should be seen as an extensionality rule, similar to η-rules.

$$\frac{\Gamma \vdash e : A \sim_{\mathcal{U}} A' \qquad \Gamma \vdash t : A \qquad \Gamma \vdash A \equiv A' : \mathcal{U}}{\Gamma \vdash \mathbf{cast}(A, A', e, t) \equiv t : A'}$$

[2] Thanks to proof irrelevance, our version of transport where the motive C depends on the proof of equality is inter-derivable with one without, so we include the stronger primitive for ease of use, while Pujet has the weaker primitive to simplify meta-theory.

Quotient types. Observational equality gives us a synthetic language to talk about *setoids* [17, 4], types equipped with an equivalence relation. We can exploit this to integrate quotient types A/R: observational equality at such a quotient type simply boils down to the equivalence relation R. Quotients come with a projection map $\pi : A \to A/R$, and their elimination captures the idea that a function out of a quotient must respect the relation, i.e. map related inputs to equal outputs.

2.2 Inductive types

We extend $\mathrm{CC}^{\mathrm{obs}}$ with a form of first-class indexed inductive types, and explore their interaction with observational equality. Our approach is exploratory, and we do not pretend to have a fully fleshed-out design, especially since we do not carry any meta-theoretic study.

First-class indexed inductive types. Since our system does not distinguish between local and global context, our inductive types are *first-class* [11, 8]: we can bind them to variables, and generally treat them as any other value. They take the following form:

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \{\Gamma, F : A \to \mathcal{U} \vdash B_i : \mathcal{U}\}_i \qquad \{\Gamma, F : A \to \mathcal{U}, x_i : B_i[F] \vdash a_i : A\}_i}{\Gamma \vdash \mu F : A \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : B_i) \to F\ a_i}] : A \to \mathcal{U}}$$

The variable F is bound by μ. A represents the type of indices. We have a finite list of constructors, each with a name C_i, an argument of type B_i, and an index a_i, which might depend on the argument. The F at the end of each constructor is mere syntax indicating the type being constructed; it is not a free variable. In what follows, we use the following shortcut: $\mu F \triangleq \mu F : A \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : B_i) \to F\ a_i}]$, i.e. μF stands for a generic inductive.

We restrict inductive types to have exactly one index, and constructors to have exactly one argument. This loses no expressivity since we can pack arguments or indices together with Σ types, and simplifies the implementation. For further simplification, we also do not implement a positivity checker, so the typing rule allows non-strictly positive inductive types. As for universes, this is mainly orthogonal to our main concerns.

Since inductive types are first class, parameters can be handled by λ-abstraction, as illustrated by the standard example of vectors:

$$Vec \triangleq \lambda A : \mathcal{U}.\ \mu F : \mathbb{N} \to \mathcal{U}.[Nil : \mathbb{1} \to F\ 0\ ;\ Cons : (x : \Sigma(n : \mathbb{N}).\ A \times F\ n) \to F(S(\mathbf{fst}\ x))]$$

Constructors. Values of inductive types are created by the constructors. We use *fording* [22, p. 65]: we allow all constructors to build a value of type $(\mu F)\ a$ for *any* index a if they provide a proof that $a_i \sim_A a$, where a_i is the index computed from the argument of the constructor. In the MLTT presentation of inductive types, this constraint is instead enforced definitionally.

$$\frac{\Gamma \vdash t : B_i[\mu F/F] \qquad \Gamma \vdash e : a_i[\mu F/F, t/x_i] \sim_A a}{\Gamma \vdash C_i(t, e) : (\mu F : A \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : B_i) \to F\ a_i}])\ a}$$

Pattern-matching. Elimination of inductive types is single-level, total pattern matching:

$$\frac{\begin{array}{c}\Gamma \vdash t : (\mu F : A \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : B_i) \to F\ a_i}])\ a \qquad \Gamma, x : (\mu F)\ a \vdash C : \mathfrak{s} \\ \{\Gamma, x_i : B_i[\mu F/F], e_i : a_i[\mu F, x_i] \sim_A a \vdash t_i : C[(C_i\ (x_i, e_i))/x]\}_i\end{array}}{\Gamma \vdash \mathbf{match}\ t\ \mathbf{as}\ x\ \mathbf{return}\ C\ \mathbf{with}\ \{C_i\ (x_i, e_i) \to t_i\} : C[t/x]}$$

This rule might appear strange, as the motive C does *not* abstract over the index. Suppose Vec as above, and we match on $v : \mathit{Vec}\ 0$, so $x : \mathit{Vec}\ 0 \vdash C : \mathfrak{s}$. In the *Cons* branch, we substitute *Cons* into C. However, the index of *Cons* is $S\ n$, this looks ill-typed! But thanks to fording we *can* have $\mathit{Cons}(x, e) : \mathit{Vec}\ 0$, given a proof of $S\ m \sim 0$, which we do have in this branch. Moreover, since $S\ m \sim 0 \equiv \bot$, we can use this to witness that this branch is in fact unreachable. The β rule is straightforward, but for handling of the forded equality.

$$\frac{}{\Gamma \vdash \textbf{match}\ C_i\ (t, e)\ \textbf{as}\ x\ \textbf{return}\ C\ \textbf{with}\ \{C_i(x_i, e_i) \to t_i\} \equiv t_i[t/x_i, e/e_i] : C[C_i(t, e)/x]}$$

Observational equality for inductive types. Observational equality between inductive types does *not* equate inductive types structurally: types with propositionally equal but definitionally different index and constructor types are not deemed equal.

$$\frac{}{\Gamma \vdash (\mu F)\ a \sim_{\mathcal{U}} (\mu F)\ a' \equiv a \sim_A a' : \Omega}$$

The goal is twofold. First, this simplifies the implementation, as it means we avoid having to compare telescopes of parameters with numerous casts. Second, a purely structural equality of inductive types would equate all "boolean" inductive types (those with two argumentless constructors), a severe case of boolean blindness [16].

When the two definitions are not convertible, observational equality is simply stuck,[3] i.e. it does not reduce further. This covers the case of definitely different inductive types (where we could be more eager and reduce to $\bot$), but also of parameterized inductive types. Indeed, for two different variables A and A', $\mathit{Vec}\ A\ n \sim_{\mathcal{U}} \mathit{Vec}\ A'\ n$ is stuck, and will compute further only if A and A' are substituted by convertible types. This equality therefore does not imply $A \sim_{\mathcal{U}} A'$. We thus do not implement a conversion rule like the following:

$$\textbf{cast}(\mathit{Vec}\ A\ 1, \mathit{Vec}\ A'\ 1, \ldots, \mathit{Cons}(0, a, \mathit{Nil}(\ldots))) \equiv \mathit{Cons}(0, \textbf{cast}(A, A', \ldots, a), \mathit{Nil}(\ldots))$$

Deriving these definitional equalities roughly amounts to deriving a general *map* operation for inductive types, a non-trivial enterprise which we did not attempt. Since the design of our prototype, Pujet and Tabareau [29] have explored the question further, and proposed a solution, which we did not try to reproduce.

Equality of general inductives behaves like that of natural numbers: when comparing equal constructors, the proposition steps to equality of the contents, and when they are different, it steps to $\bot$. These respectively reflect injectivity and no-confusion of constructors.

$$\frac{}{\Gamma \vdash C_i\ (t, e) \sim_{(\mu F) a} C_i\ (u, e') \equiv t \sim_{B_i[\mu F]} u : \Omega} \qquad \frac{C_i \neq C_j}{\Gamma \vdash C_i\ (t, e) \sim_{(\mu F) a} C_j\ (u, e') \equiv \bot : \Omega}$$

For casts on constructors, as explained above we do not need to – and indeed cannot – propagate them deep in the structure. Instead, we merely need to handle indices, which amounts to composing equality proofs. Note the use of the **trans** primitive.

$$\frac{}{\Gamma \vdash \textbf{cast}((\mu F)\ a, (\mu F)\ a', e, C_i\ (t, e')) \equiv C_i\ (t, \textbf{trans}_{a_i[\mu F/F, t/x_i], a, a'}\ e'\ e) : (\mu F)\ a'}$$

[3] As noted by Pujet [26, 5.2.2], there is a lot of room in how much definitional equalities are imposed on observational equality, the sole constraint – coming from canonicity – being that casts between convertible closed types must compute.

Mendler induction. Pattern matching as eliminator for inductive types does not let us handle their recursive structure. That is, we have no notion of *induction*, as we can look only one level at a time. To correct this, we introduce **fix** in a style extending Mendler recursion [23] to dependent induction.

To express it, we first need an operation **lift** which applies a functor F to a type X.

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \{\Gamma, F : A \to \mathcal{U} \vdash B_i : \mathcal{U}\}_i \qquad \{\Gamma, F : A \to \mathcal{U}, x_i : B_i[F] \vdash a_i : A\}_i}{\Gamma \vdash \mathbf{lift}[\mu F : A \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : B_i) \to F\ a_i}]] : (A \to \mathcal{U}) \to \mathcal{U}}$$

This operation is defined by the following equation:

$$\Gamma \vdash \mathbf{lift}[\mu F]\ B \equiv \mu F : A \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : B_i[B/F]) \to F\ a_i[B/F]}] : \mathcal{U}$$

In essence, $\mathbf{lift}[\mu F]\ B$ is an inductive type *which is mute in the variable F*, that is, non-recursive. It represents adding a single “layer” of F-structure over the family B.

Using this primitive, we can express the type of Mendler-style induction. As above, we abbreviate the inductive type to μF. The type C is the one we want to inhabit by induction.

$$\frac{\begin{array}{c}\Gamma, G : A \to \mathcal{U}, p : A, x : G\,p \vdash C : \mathfrak{s} \\ \Gamma, G : A \to \mathcal{U}, f : \Pi(p : A)(x : G\,p).C[G, p, x], p : A, x : \mathbf{lift}[\mu F]\,G\,p \vdash t : C[\mathbf{lift}[\mu F]\,G, p, x]\end{array}}{\Gamma \vdash \mathbf{fix}\ [\mu F\ \mathbf{as}\ G]\ f\ p\ x : C = t : \Pi(p : A).\ \Pi(x : (\mu F)\ p).\ C[\mu F, p, x]}$$

To type the body of the fixed point, Mendler induction operates by introducing a generic type family, G and an inhabitant x of $\mathbf{lift}[\mu F]\ G\ p$, and demands that we construct an inhabitant of C at x, while having access to “recursive calls” only on inhabitants of G via the function f. Intuitively, we can pattern-match on x, recovering values of G corresponding to “subterms”, that can be used for recursive calls. Since G is abstract, this is the only way to call f, and so the fixed point we obtain is structurally decreasing.

The **fix** operation computes when applied to a constructor, as per the following rule. The argument *must* be a constructor in order to prevent infinite unfolding.

$$\frac{\mathbf{fix}_f \triangleq \mathbf{fix}\ [\mu F\ \mathbf{as}\ G]\ f\ p\ x : C = t}{\Gamma \vdash \mathbf{fix}_f\ u\ (C_i\ (v, e)) \equiv t[\mu F/G, \mathbf{fix}_f/f, u/p, (C_i\ (v, e))/x]}$$

Views and paramorphisms. Mendler induction ensures **fix** is well-founded by restricting recursive appeals to the induction hypothesis. However, in this process, we lose information: we only have access to recursive calls, but not to the current value of the inductive type. In categorical parlance, fixed-points and pattern matching implement *catamorphisms*: generalised folds over tree-like structures.

What we want instead are *paramorphisms*, which provide primitive recursion by giving access to the current value. We thus introduce an extra function $\iota : \Pi(p : A).\ G\ p \to (\mu F)\ p$ to *view* the opaque G as the inductive type it represents. Once the knot of the fixed point is tied and the variable G is substituted for the inductive type, this becomes the identity.

To type the fixed point’s body we need to lift $\iota : \Pi(p : A).\ G\ p \to (\mu F)\ p$ into $\iota' : \Pi(p : A).\ \mathbf{lift}[\mu F]\ G\ p \to (\mu F)\ p$. Hence, we need again the action of the functor μF on values, i.e. the associated *map*. As explained above, we do not provide infrastructure to derive this operation. We instead allow an inductive definition to come with a user-provided functor action.[4] Note that in the premise we use an inductive type *without* a functor.

[4] We do not check this indeed is the functor action, since generating the equalities a correct *map* operation satisfies is basically the same as deriving that operation itself.

$$\frac{\begin{array}{c}\Gamma \vdash A : \mathcal{U} \qquad \{\Gamma, F : A \to \mathcal{U} \vdash B_i : \mathcal{U}\}_i \qquad \{\Gamma, F : A \to \mathcal{U}, x_i : B_i[F] \vdash a_i : A\}_i \\ \Gamma, X : A \to \mathcal{U}, Y : A \to \mathcal{U}, f : \Pi(p : A).X\, p \to Y\, p, p : A, x : \mathbf{lift}[\mu F]\ X\ p \vdash t : \mathbf{lift}[\mu F]\ Y\ p\end{array}}{\Gamma \vdash \mu F : A \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : B_i) \to F\ a_i}]\ \mathbf{functor}\ X\ Y\ f\ p\ x = t : A \to \mathcal{U}}$$

To be able to use this definition, we introduce the term **fmap** for projecting the functorial action from an inductive type and **in**, witnessing the isomorphism between μF and $\mathbf{lift}[\mu F]\ (\mu F)$, which is the identity function on constructors: $\mathbf{in}\ (C_i\ (t, e)) \equiv C_i\ (t, e)$.

$$\frac{\Gamma \vdash \mu F : \mathcal{U}}{\begin{array}{c}\Gamma \vdash \mathbf{fmap}\ [\mu F] : \Pi(X : A \to \mathcal{U}).\ \Pi(Y : A \to \mathcal{U}). \\ (\Pi(p : A).\ X\ p \to Y\ p) \to \Pi(p : A).\ \mathbf{lift}[\mu F]\ X\ p \to \mathbf{lift}[\mu F]\ Y\ p\end{array}} \qquad \frac{\Gamma \vdash t : \mathbf{lift}[\mu F]\ F\ a}{\Gamma \vdash \mathbf{in}\ t : (\mu F)\ a}$$

With this infrastructure, we extend the typing rule for fixed-points.

$$\frac{\begin{array}{c}\iota' \triangleq \lambda p.\ \lambda x.\ \mathbf{in}\ (\mathbf{fmap}\ [\mu F]\ G\ \mu F\ \iota\ p\ x) \qquad \mathrm{id} \triangleq \lambda p.\ \lambda x.\ x \\ \Gamma, G : A \to \mathcal{U}, \iota : \Pi(p : A).\ G\ p \to (\mu F)\ p, p : A, x : G\ p \vdash C : \mathfrak{s} \\ \Gamma, G : A \to \mathcal{U}, \iota : \Pi(p : A).\ G\ p \to (\mu F)\ p, f : \Pi(p : A).\ \Pi(x : G\ p).\ C[G, \iota, p, x], \\ \qquad\qquad p : A, x : F[G]\ p \vdash t : C[F[G], \iota', p, x]\end{array}}{\Gamma \vdash \mathbf{fix}\ [\mu F\ \mathbf{as}\ G\ \mathbf{view}\ \iota]\ f\ p\ x : C = t : \Pi(p : A).\ \Pi(x : (\mu F)\ p).\ C[\mu F, \mathrm{id}, p, x]}$$

The variable ι is now accessible in both the motive C and the body t of the fixed-point, facilitating primitive recursion. At the top level, ι is substituted by the identity function. By functoriality, this means ι' is $\mathbf{in} \circ \mathrm{id}$, which also behaves as the identity.

In summary, if the user provides a *map* operation, we leverage that to upgrade our Mendler-style catamorphisms to paramorphisms. Of course, in a full-fledged implementation we would derive that operation automatically, and simply give access to the paramorphism.

3 Core Implementation

Let us start with the core features of $\mathrm{CC}^{\mathrm{obs}}$: strict propositions and inductive equality. The implementation is written in Haskell, and makes extensive use of structural data types and declarative style to keep the code as close as possible to the theory. Figure 1 gives an overview of the implementation's structure. We use $\multimap$ for the meta-level (i.e. Haskell's) function type, and a sans font for its types and functions.

3.1 Syntax

Syntax. We present $\mathrm{CC}^{\mathrm{obs}}$ in Russell style [21, 25], so we have a common grammar for both terms and types, given in Figure 2. This is roughly the one given in Pujet [26], with some additions: let bindings and type annotations, which are important with bidirectional typing, and, as explained in Section 2.1, constants for symmetry and transitivity.

We use a unit type $\mathbb{1}$, with the term-directed η-conversions $t \equiv\ !$ and $! \equiv t$. These break transitivity of a term-directed algorithmic conversion: in the context $x, y : \mathbb{1}$, we have $x \equiv\ ! \equiv y$ but $x \not\equiv y$. This is a trade-off: since we impose inductive types to have exactly one index and constructors exactly one argument, this unit type is handy to pad these positions with definitionally irrelevant content. In a more mature implementation, this would not be necessary, and we could drop our ill-behaved $\mathbb{1}$, or adopt the more complex – and satisfactory – solution proposed by Kovács [20].

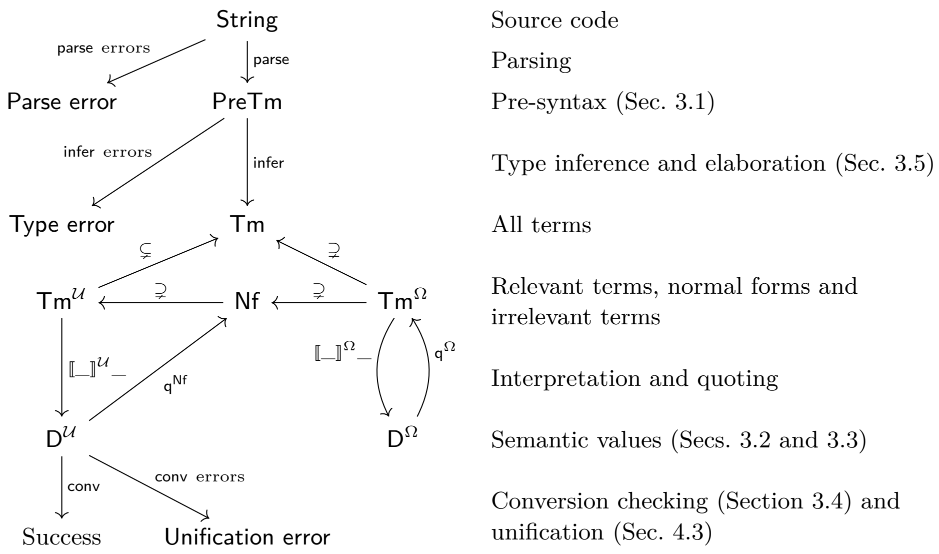
String
parse errors
parse
Parse error
PreTm
infer errors
infer
Type error
Tm
⊊
⊋
$\mathsf{Tm}^{\mathcal{U}}$
⊋
Nf
⊋
Tm^{Ω}
$⟦\_⟧^{\mathcal{U}}\_$
q^{Nf}
$⟦\_⟧^{\Omega}\_$
q^{Ω}
$\mathsf{D}^{\mathcal{U}}$
D^{Ω}
conv
conv errors
Success
Unification error
Source code
Parsing
Pre-syntax (Sec. 3.1)
Type inference and elaboration (Sec. 3.5)
All terms
Relevant terms, normal forms and irrelevant terms
Interpretation and quoting
Semantic values (Secs. 3.2 and 3.3)
Conversion checking (Section 3.4) and unification (Sec. 4.3)

Figure 1 A high-level overview of the interaction of the components in the type-checker.

$\mathfrak{s}$	$::= \mathcal{U} \mid \Omega$	Universe sorts
A, B, C, t, u, e	$::= x$	Variable
	$\mid \mathfrak{s}$	Universe
	$\mid \lambda x_{\mathfrak{s}}.\ t \mid t \mathbin{@^{\mathfrak{s}}} u \mid \Pi(x :_{\mathfrak{s}} A).\ B$	Dependent functions
	$\mid 0 \mid S\,t \mid \mathbf{rec}[z.C](t, 0 \to t_0; (S\,x)\,y \to t_S) \mid \mathbb{N}$	Natural numbers
	$\mid (t, u)_{\mathfrak{s}} \mid \mathbf{fst}\ t \mid \mathbf{snd}\ t \mid \Sigma x : A.B \mid \exists x : A.B$	Dependent pairs
	$\mid \mathbf{abort}_A\ t \mid \bot \mid * \mid \top \mid \mathbb{1} \mid !$	False, true and unit types
	$\mid \Box A \mid \diamond t \mid \Box\textbf{-elim}\ t$	Box types
	$\mid t \sim_A u \mid \mathbf{refl}\ t \mid \mathbf{sym}_{t,u}\ e \mid \mathbf{trans}_{t,u,v}\ e\ e'$	Observational equality
	$\mid \mathbf{transp}(t, x\ y.\ C, u, t', e) \mid \mathbf{cast}(A, B, e, t)$	Transport and casting
	$\mid \mathbf{let}\ x :_{\mathfrak{s}} A = t\ \mathbf{in}\ u$	Let binding
	$\mid (t : A)$	Type annotation

Figure 2 Basic syntax for $\mathrm{CC}^{\mathrm{obs}}$.

Box types $\Box A$ turn a type into a proposition – what is alternatively called squashing, truncation, or `inhabited` – see Pujet [26] for details.

Application $t \mathbin{@^{\mathfrak{s}}} u$ is tagged with the sort $\mathfrak{s}$ of the argument u, and λ-terms and pairs with their domain sort. This is necessary for evaluation, but is always inferred during type-checking; users need not give these annotations in the source syntax.

This grammar inductively defines the type PreTm of *pre-terms*: syntactically well-formed, but untyped. We define another type Tm of well-typed terms. Every well-typed term has a (unique) *sort* – $\mathcal{U}$ or Ω –, the type of its type. We let $\mathsf{Tm}^{\mathcal{U}}$ and Tm^{Ω} be the set of terms with sort $\mathcal{U}$ and Ω respectively. In the code we have separate types for PreTm and Tm, the former using named variables and the latter de Bruijn indices. We cannot enforce in Haskell that Tm contains only well-typed terms, but maintain this as an invariant.

In what follows, we do not always cover all cases, focusing on the most interesting or illustrative ones. The code can be consulted for the complete picture.

Normal forms. Normals and neutrals are defined mutually as predicates on $\mathsf{Tm}^{\mathcal{U}}$. In contrast to Pujet [27], they are deep, so subterms are also required to be normal.

$$\begin{array}{llcl}
\mathsf{Nf} \ni & v, w, V, W & ::= & n \mid \mathfrak{s} \mid \lambda x.\ v \mid \Pi(x :_{\mathfrak{s}} V).\ W \mid 0 \mid S\ v \mid \mathbb{N} \mid (v, v)_{\mathcal{U}} \mid \Sigma(x : V).\ W \\
& & \mid & (v, v)_{\Omega} \mid \exists(x : V).\ W \mid \bot \mid \top \mid\ ! \mid \mathbb{1} \\
\mathsf{Ne} \ni & n, N & ::= & x_i \mid n \mathbin{@^{\mathcal{U}}} v \mid n \mathbin{@^{\Omega}} t \mid \mathbf{rec}[z.V](n, 0 \to v; (S\ x)\ y \to w) \mid \mathbf{abort}_V\ t \\
& & \mid & v \sim_n w \mid n \sim_{\mathbb{N}} v \mid v \sim_{\mathbb{N}} n \mid n \sim_{\mathcal{U}} v \mid v \sim_{\mathcal{U}} n \\
& & \mid & \mathbf{cast}(\mathbb{N}, \mathbb{N}, e, n) \mid \mathbf{cast}(N, V, e, v) \mid \mathbf{cast}(V, N, e, v)
\end{array}$$

We only need to characterize normal forms in $\mathsf{Tm}^{\mathcal{U}}$, i.e. relevant terms. Indeed, since proof-irrelevant terms have no notion of evaluation, they *all* are "normal forms". This means that normal forms are only unique up to η-equality and proof irrelevance. Luckily, thanks to typing constraints, irrelevant subterms of a relevant normal form can only appear as arguments to **abort**, to **cast**, or an application tagged with Ω, which lets us decide which subterms (not) to compare purely syntactically, without typing information.

Normal forms, as standard, correspond to constructors – observational equality and cast are viewed as destructors, and so are not normal forms. Neutral forms are somewhat more involved. The observational equality type can be blocked in *three* places – the type, or either of the terms – although there are no neutral forms when the equality is at a Π or Σ type, as such equalities always reduce. Similarly, casts can also block in three positions: either of the types, or the term being cast. Again, casts between universes, Π or Σ types always reduce, and so cannot be blocked by the argument. To avoid quadratic blow-up in the presentation, some cases overlap, for example $n \sim_{\mathbb{N}} v$ and $v \sim_{\mathbb{N}} n$ both cover $x \sim_{\mathbb{N}} y$.

Pujet [26, 5.2.2] discusses the definitional equalities observational equality should satisfy, for instance whether $\Pi(x : A).\ B \sim_{\mathcal{U}} \Sigma(x : A').\ B'$ should evaluate to $\bot$ or be stuck. He remarks we can replace the reduction of $\Sigma(x : A).\ B \sim_{\mathcal{U}} \Sigma(x : A').\ B'$ by two constants corresponding to the two projections out of the would-be reduct:

$$\mathbf{eq\text{-}fst} : (\Sigma x : A.\ B) \sim_{\mathcal{U}} (\Sigma x : A'.\ B') \to A \sim_{\mathcal{U}} A' \qquad \mathbf{eq\text{-}snd} : \ldots$$

We chose to make conversion compute as much as possible as a minimal form of automation, although it is not entirely clear whether this makes using the proof assistant really easier.

3.2 Evaluation in the relevant layer

The overall picture for NbE is similar to the standard case [1]: we give a semantic interpretation into a domain of values, and a quoting function to reconstruct normal forms. The main difficulty is to account for complex reduction rules for propositions and casts, and proof-irrelevant propositions. We defer the latter to Section 3.3. Note that we implement untyped rather than type-directed NbE. Although the latter would have avoided issues around the unit type (see Sec. 3.1), we chose it to avoid an extra layer of complexity, and check that even a complex type theory like $\mathrm{CC}^{\mathrm{obs}}$ can still be implemented this way.

Semantic domain. First, we construct the various domains data-structures in Figure 3. The domains $\mathsf{D}^{\mathcal{U}}$ and D^{ne} represent respectively normal and neutral forms. The domain D^{Ω} represents propositional values and is explained in Section 3.3. D is the union of $\mathsf{D}^{\mathcal{U}}$ and D^{Ω}, and we often omit the injections V and P.

These are defined mutually with closures $\mathsf{Clos}_k^{\mathfrak{s}}$ indexed by their return sort $\mathfrak{s}$ and their arity k, as not all of them await a single argument. This is a landmark of defunctionalised NbE, which replaces semantic functions by such closures, making the various domains first-order strictly positive datatypes. Abel [1] needs a single kind of closure, consisting of an environment and a term, representing a meta-level function of type $\mathsf{Tm} \rightharpoonup \mathsf{Tm}$. Here, we need to defunctionalise various evaluation functions, and thus have multiple forms.

$$
\begin{array}{llll}
\mathsf{D}^{\mathcal{U}} & \ni & a, b, A, B & ::= \mathsf{U}\ \mathfrak{s} \mid \mathsf{Lam}\ \mathfrak{s}\ \mathcal{F}_1 \mid \mathsf{Pi}\ \mathfrak{s}\ A\ \mathcal{B}_1 \mid \mathsf{Z} \mid \mathsf{S}\ a \mid \mathsf{Nat} \mid \uparrow e \\
 & & & \quad \mid \mathsf{Exists}\ A\ \mathcal{B}_1 \mid \mathsf{Empty} \mid \mathsf{Unit} \\
\mathsf{D}^{\mathsf{ne}} & \ni & e & ::= \mathsf{Var}_l \mid \mathsf{App}_{\mathfrak{s}}\ e\ d \mid \mathsf{Rec}\ \mathcal{A}_1\ e\ a\ \mathcal{F}_2 \mid \mathsf{Eq}\ a\ A\ a' \mid \mathsf{Cast}\ A\ B\ p\ a \\
\mathsf{D}^{\Omega} & \ni & p, q, P, Q & ::= \cdots \qquad\qquad\qquad\qquad \text{(see Sec. 3.3)} \\
\mathsf{D} & \ni & d, f, g, D & ::= \mathsf{V}\ a \mid \mathsf{P}\ p \\
\mathsf{Env} & \ni & \rho & ::= () \mid (\rho, d) \\
\mathsf{Clos}^{\mathfrak{s}}_k & \ni & \mathcal{C}_k & ::= (\underline{\lambda} t)\rho \qquad\qquad\qquad\quad : \mathsf{Clos}^{\mathfrak{s}}_k \\
 & & & \quad\ \mid \mathsf{Lift}\ d \qquad\qquad\qquad\quad\ : \mathsf{Clos}^{\mathfrak{s}}_k \\
 & & & \quad\ \mid \mathsf{EqFun}\ \mathfrak{s}\ f\ \mathcal{C}_1\ g \qquad\quad : \mathsf{Clos}^{\mathfrak{s}}_1 \\
 & & & \quad\ \mid \mathsf{EqPi}\ \mathfrak{s}\ D\ D'\ \mathcal{C}_1\ \mathcal{C}'_1 \qquad : \mathsf{Clos}^{\mathfrak{s}}_1 \\
 & & & \quad\ \mid \mathsf{EqPi}'\ \mathfrak{s}\ D\ D'\ \mathcal{C}_1\ \mathcal{C}'_1\ p \quad : \mathsf{Clos}^{\mathfrak{s}}_1 \\
\mathsf{Clos}^{\mathcal{U}}_k & \ni & \mathcal{A}_k, \mathcal{B}_k, \mathcal{F}_k & \quad\ \mid \mathsf{CastPi}\ \mathfrak{s}\ D\ D'\ \mathcal{C}_1\ \mathcal{C}'_1\ p\ f \ : \mathsf{Clos}^{\mathfrak{s}}_1 \\
\mathsf{Clos}^{\Omega}_k & \ni & \mathcal{P}_k, \mathcal{Q}_k &
\end{array}
$$

Figure 3 Semantic domains, environments and closures.

Neutral terms for observational equality and casting are simplified: we do not keep track of which of the three arguments to Eq and Cast is blocking. This is a trade-off between a precise semantic domain and a simpler interpretation function. As the goal here is to implement the system, not prove its correctness, we choose the simpler representation.

Environments contain values from *both* sorts. An environment interprets a context, where each variable has a sort. It is a program invariant that the domain sort in each position in the environment matches the typing context.

$$
\begin{array}{rcl}
\llbracket x_0 \rrbracket^{\mathcal{U}} (\rho, a) & = & a \\
\llbracket x_{i+1} \rrbracket^{\mathcal{U}} (\rho, d) & = & \llbracket x_i \rrbracket^{\mathcal{U}} \rho \\
\llbracket \mathfrak{s} \rrbracket^{\mathcal{U}} \rho & = & \mathsf{U}\ \mathfrak{s} \\
\llbracket 0 \rrbracket^{\mathcal{U}} \rho & = & \mathsf{Z} \\
\llbracket S\ t \rrbracket^{\mathcal{U}} \rho & = & \mathsf{S}\ (\llbracket t \rrbracket^{\mathcal{U}} \rho) \\
\llbracket \mathbb{N} \rrbracket^{\mathcal{U}} \rho & = & \mathsf{Nat} \\
\llbracket t\ u^{\mathfrak{s}} \rrbracket^{\mathcal{U}} \rho & = & (\llbracket t \rrbracket^{\mathcal{U}} \rho) \cdot_{\mathfrak{s}} (\llbracket u \rrbracket^{\mathfrak{s}} \rho)
\end{array}
\qquad
\begin{array}{rcl}
\llbracket \Pi (x :_{\mathfrak{s}} A).\ B \rrbracket^{\mathcal{U}} \rho & = & \mathsf{Pi}\ \mathfrak{s}\ (\llbracket A \rrbracket^{\mathcal{U}} \rho)\ (\underline{\lambda} B)\rho \\
\llbracket \lambda x_{\mathfrak{s}}.\ t \rrbracket^{\mathcal{U}} \rho & = & \mathsf{Lam}\ \mathfrak{s}\ (\underline{\lambda} t)\rho \\
\llbracket t \sim_A u \rrbracket^{\mathcal{U}} \rho & = & \underline{\mathsf{eq}}(\llbracket t \rrbracket^{\mathcal{U}} \rho, \llbracket A \rrbracket^{\mathcal{U}} \rho, \llbracket u \rrbracket^{\mathcal{U}} \rho) \\
\llbracket \mathbf{cast}(A, B, e, t) \rrbracket^{\mathcal{U}} \rho & = & \underline{\mathsf{cast}}(\llbracket A \rrbracket^{\mathcal{U}} \rho, \llbracket B \rrbracket^{\mathcal{U}} \rho, \llbracket e \rrbracket^{\Omega} \rho, \llbracket t \rrbracket^{\mathcal{U}} \rho) \\
\llbracket \mathbf{let}\ x :_{\mathfrak{s}} A = t\ \mathbf{in}\ u \rrbracket^{\mathcal{U}} \rho & = & \llbracket u \rrbracket^{\mathcal{U}} (\rho, \llbracket t \rrbracket^{\mathfrak{s}} \rho) \\
\llbracket (t : A) \rrbracket^{\mathcal{U}} \rho & = & \llbracket t \rrbracket^{\mathcal{U}} \rho
\end{array}
$$

$$
\llbracket \mathbf{rec}[z.C](t, 0 \to t_0; (S\ x)\ y \to t_S) \rrbracket^{\mathcal{U}} \rho \quad = \quad \underline{\mathsf{rec}}((\underline{\lambda} C)\rho, \llbracket t \rrbracket^{\mathcal{U}} \rho, \llbracket t_0 \rrbracket^{\mathcal{U}} \rho, (\underline{\lambda} t_S)\rho)
$$

Figure 4 Semantic interpretation for relevant terms into the semantic domain $\mathsf{D}^{\mathcal{U}}$.

Interpretation. Inputs to relevant interpretation are well-typed terms of sort $\mathcal{U}$; that is, the set $\mathsf{Tm}^{\mathcal{U}}$. Therefore, we construct the function

$$\llbracket \_ \rrbracket^{\mathcal{U}}\_ : \mathsf{Tm}^{\mathcal{U}} \to \mathsf{Env} \to \mathsf{D}^{\mathcal{U}}$$

The interpretation is given in Figure 4. The underlined functions and $\cdot_{\mathfrak{s}}$ are semantic counterpart to eliminators, and are defined mutually with evaluation, see below.

We also need a function for applying a closure to k values of either sort:

$$\mathsf{app}^{\mathcal{U}} : \mathsf{Clos}^{\mathcal{U}}_k \to \underbrace{\mathsf{D} \to \cdots \to \mathsf{D}}_{k} \to \mathsf{D}^{\mathcal{U}}$$

Applying to argument of the correct sort is a code invariant but not statically type-checked.[5] We use the shorthand notation $\mathcal{C}[d_1, \ldots, d_k]$ for $\mathsf{app}^{\mathcal{U}}\ \mathcal{C}\ d_1\ \ldots\ d_k$.

[5] This could be enforced with closures indexed a type-level list of sorts.

As closures are used to defunctionalise specific operations, we define those alongside the corresponding case for app. To begin with, we have the primary cases of a continuation, $(\underline{\lambda} t)\rho$, corresponding to substitution, and Lift, for a constant closure ignoring its argument.

$$\mathsf{app}^{\mathcal{U}}\ (\underline{\lambda} t)\rho\ d_1\ \dots\ d_k = [\![t]\!]^{\mathcal{U}}(\rho, d_1, \cdots, d_k) \qquad\qquad \mathsf{app}^{\mathcal{U}}\ (\mathsf{Lift}\ a)\ d_1\ \dots\ d_k = a$$

Semantic application $\_ \cdot_{\mathfrak{s}} \_ : \mathsf{D}^{\mathcal{U}} \twoheadrightarrow \mathsf{D}^{\mathfrak{s}} \twoheadrightarrow \mathsf{D}^{\mathcal{U}}$ directly uses generic closure application.

$$(\mathsf{Lam}\ \mathfrak{s}\ \mathcal{F}) \cdot_{\mathfrak{s}} d = \mathcal{F}[d] \qquad\qquad (\uparrow e) \cdot_{\mathfrak{s}} d = \uparrow(\mathsf{App}_{\mathfrak{s}}\ e\ d)$$

$$\begin{aligned}
\underline{\mathsf{eq}}(f, \mathsf{Pi}\ \mathfrak{s}\ A\ \mathcal{B}, g) &= \mathsf{Pi}\ A\ (\mathsf{EqFun}\ \mathfrak{s}\ f\ \mathcal{B}\ g)\\
\underline{\mathsf{eq}}(A, \mathsf{U}\ \Omega, B) &= \mathsf{Exists}\ (\mathsf{Pi}\ A\ (\mathsf{Lift}\ B))\ (\mathsf{Lift}\ (\mathsf{Pi}\ B\ (\mathsf{Lift} A)))\\
\underline{\mathsf{eq}}(\mathsf{Nat}, \mathsf{U}\ \mathcal{U}, \mathsf{Nat}) &= \underline{\mathsf{eq}}(\mathsf{U}\ \mathfrak{s}, \mathsf{U}\ \mathcal{U}, \mathsf{U}\ \mathfrak{s}) = \mathsf{Unit}\\
\underline{\mathsf{eq}}(A, \mathsf{U}\ \mathcal{U}, B) &= \mathsf{Empty} \quad \text{when } A \text{ and } B \text{ have different head constructors}\\
\underline{\mathsf{eq}}(\mathsf{Pi}\ \mathfrak{s}\ A\ \mathcal{B}, \mathsf{U}\ \mathcal{U}, \mathsf{Pi}\ \mathfrak{s}\ A'\ \mathcal{B}') &= \mathsf{Exists}\ (\underline{\mathsf{eq}}(A', \mathsf{U}\ \mathfrak{s}, A))\ (\mathsf{EqPi}\ \mathfrak{s}\ A\ A'\ \mathcal{B}\ \mathcal{B}')\\
\underline{\mathsf{eq}}(\mathsf{Z}, \mathsf{Nat}, \mathsf{Z}) &= \mathsf{Unit}\\
\underline{\mathsf{eq}}(\mathsf{S}\ a, \mathsf{Nat}, \mathsf{S}\ b) &= \underline{\mathsf{eq}}(a, \mathsf{Nat}, b)\\
\underline{\mathsf{eq}}(\mathsf{S}\ a, \mathsf{Nat}, \mathsf{Z}) &= \underline{\mathsf{eq}}(\mathsf{Z}, \mathsf{Nat}, \mathsf{S}\ a) = \mathsf{Empty}\\
\underline{\mathsf{eq}}(a, A, a') &= \uparrow(\mathsf{Eq}\ a\ A\ a') \quad \text{otherwise (one of } A, a \text{ or } a' \text{ is neutral)}
\end{aligned}$$

$$\begin{aligned}
\mathsf{app}^{\mathcal{U}}\ (\mathsf{EqFun}\ \mathfrak{s}\ f\ \mathcal{B}\ g)\ d &= \underline{\mathsf{eq}}(f \cdot_{\mathfrak{s}} d, \mathcal{B}[d], g \cdot_{\mathfrak{s}} d)\\
\mathsf{app}^{\mathcal{U}}\ (\mathsf{EqPi}\ \mathfrak{s}\ A\ A'\ \mathcal{B}\ \mathcal{B}')\ p &= \mathsf{Pi}\ \mathfrak{s}\ A'\ (\mathsf{EqPi}'\ \mathfrak{s}\ A\ A'\ \mathcal{B}\ \mathcal{B}'\ p)\\
\mathsf{app}^{\mathcal{U}}\ (\mathsf{EqPi}'\ \mathcal{U}\ A\ A'\ \mathcal{B}\ \mathcal{B}'\ p)\ a' &= \underline{\mathsf{eq}}(\mathcal{B}[a], \mathsf{U}\ \mathcal{U}, \mathcal{B}'[a']) \quad \text{where } a \triangleq \underline{\mathsf{cast}}(A', A, p, a')\\
\mathsf{app}^{\mathcal{U}}\ (\mathsf{EqPi}'\ \Omega\ A\ A'\ \mathcal{B}\ \mathcal{B}'\ p)\ q' &= \underline{\mathsf{eq}}(\mathcal{B}[q], \mathsf{U}\ \mathcal{U}, \mathcal{B}'[q']) \quad \text{where } q \triangleq \mathsf{PCast}\ \phi(A')\ \phi(A)\ p\ q'
\end{aligned}$$

Figure 5 Semantic observational equality function, and the associated closures.

Semantic observational equality. To complete the semantic interpretation, we need to define $\underline{\mathsf{eq}}$ and $\underline{\mathsf{cast}}$, the semantic counterpart to the observational equality type and to cast, which implement their reduction behaviour. The former is given in Figure 5.

Semantic equality straightforwardly implements the computational behaviour of observational equality. Note the use of the *two* defunctionalising closures EqPi and EqPi': EqPi forwards the equality proof into a second closure EqPi' which performs the computation. This is necessary because we have two positions requiring arity-one closures, so they cannot be combined. When the Π types have an irrelevant domain, we need a propositional witness of type A, so we use PCast and the *freeze* function $\phi : \mathsf{D} \hookrightarrow \mathsf{D}^{\Omega}$ for embedding relevant values into the irrelevant domain, both of which will be introduced in Section 3.3.

Semantic cast. The final component to complete evaluation is the semantic $\underline{\mathsf{cast}}$ function, given in Figure 6, again straightforwardly implementing reduction rules for casts. Note the proof manipulation implemented by propositional application PApp and projections PFst and PSnd, and again the embedding $\phi : \mathsf{D} \hookrightarrow \mathsf{D}^{\Omega}$.

$$
\begin{aligned}
\underline{\mathsf{cast}}(\mathsf{Nat}, \mathsf{Nat}, p, \mathsf{Z}) &= \mathsf{Z} \\
\underline{\mathsf{cast}}(\mathsf{Nat}, \mathsf{Nat}, p, \mathsf{S}\ a) &= \mathsf{S}\ (\underline{\mathsf{cast}}(\mathsf{Nat}, \mathsf{Nat}, p, a)) \\
\underline{\mathsf{cast}}(\mathsf{U}\ \mathfrak{s}, \mathsf{U}\ \mathfrak{s}, p, A) &= A \\
\underline{\mathsf{cast}}(\mathsf{Pi}\ \mathfrak{s}\ A\ \mathcal{B}, \mathsf{Pi}\ \mathfrak{s}\ A'\ \mathcal{B}', p, f) &= \mathsf{Lam}\ \mathfrak{s}\ (\mathsf{CastPi}\ \mathfrak{s}\ A\ A'\ \mathcal{B}\ \mathcal{B}'\ p\ f) \\
\underline{\mathsf{cast}}(A, B, p, a) &= {\uparrow}(\mathsf{Cast}\ A\ B\ p\ a) \\
\mathsf{app}^{\mathcal{U}}\ (\mathsf{CastPi}\ \mathcal{U}\ A\ A'\ \mathcal{B}\ \mathcal{B}'\ p\ f)\ a' &= \underline{\mathsf{cast}}(\mathcal{B}[a], \mathcal{B}'[a'], \mathsf{PApp}_{\Omega}\ (\mathsf{PSnd}\ p)\ \phi(a'), f \cdot_{\mathcal{U}} a) \\
&\qquad \text{where } a \triangleq \underline{\mathsf{cast}}(A', A, \mathsf{PFst}\ p, a') \\
\mathsf{app}^{\mathcal{U}}\ (\mathsf{CastPi}\ \Omega\ A\ A'\ \mathcal{B}\ \mathcal{B}'\ p\ f)\ q' &= \underline{\mathsf{cast}}(\mathcal{B}[q], \mathcal{B}'[q'], \mathsf{PApp}_{\Omega}\ (\mathsf{PSnd}\ p)\ q', f \cdot_{\Omega} q) \\
&\qquad \text{where } q \triangleq \mathsf{PCast}\ A'\ A\ (\mathsf{PFst}\ p)\ q'
\end{aligned}
$$

Figure 6 Semantic cast function, and the associated closure.

$$
\begin{aligned}
\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(\uparrow e) &= \mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}}(e) \\
\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(\mathsf{U}\ \mathfrak{s}) &= \mathfrak{s} \\
\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(\mathsf{Z}) &= 0 \\
\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(\mathsf{S}\ a) &= S\ (\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(a)) \\
\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(\mathsf{Nat}) &= \mathbb{N} \\
\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(\mathsf{Lam}\ \mathfrak{s}\ \mathcal{F}) &= \lambda_{\mathfrak{s}}.\ \mathsf{q}^{\mathsf{Nf}}_{\mathsf{n+1}}(\mathsf{vs}^{\mathcal{U}}_{\mathsf{n}}(\mathcal{F})) \\
\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(\mathsf{Pi}\ \mathfrak{s}\ A\ \mathcal{B}) &= \Pi\ \mathfrak{s}\ \big(\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(A)\big)\ . \\
&\qquad \big(\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n+1}}(\mathsf{vs}^{\mathcal{U}}_{\mathsf{n}}(\mathcal{B}))\big)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}}(\mathsf{Var}_l) &= x_{n-l-1} \\
\mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}}(\mathsf{App}_{\mathfrak{s}}\ e\ a) &= (\mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}}(e))\ (\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(a))^{\mathfrak{s}} \\
\mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}}(\mathsf{Rec}\ \mathcal{A}\ e\ a_0\ \mathcal{F}_S) &= \mathbf{rec}[\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n+1}}(\mathsf{vs}^{\mathcal{U}}_{\mathsf{n}}(\mathcal{A}))] \\
&\qquad (\mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}}(e), \mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(a_0); \mathsf{q}^{\mathsf{Nf}}_{\mathsf{n+2}}(\mathsf{vs}^{\mathcal{U}}_{\mathsf{n}}(\mathcal{F}_S))) \\
\mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}}(\mathsf{Eq}\ a\ A\ a') &= \mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(a) \sim_{\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(A)} \mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(a') \\
\mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}}(\mathsf{Cast}\ A\ B\ p\ a) &= \mathbf{cast}(\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(A), \mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(B), q^{\Omega}_{n}(p), \mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}}(a))
\end{aligned}
$$

Figure 7 Implementation of quoting for $\mathrm{CC}^{\mathrm{obs}}$: $\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}} : \mathsf{D}^{\mathcal{U}} \rightarrow \mathsf{Nf}$ and $\mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}} : \mathsf{D}^{\mathsf{ne}} \rightarrow \mathsf{Ne}$.

Quoting. The mutually recursive functions $\mathsf{q}^{\mathsf{Nf}}_{\mathsf{n}} : \mathsf{D}^{\mathcal{U}} \rightarrow \mathsf{Nf}$ and $\mathsf{q}^{\mathsf{Ne}}_{\mathsf{n}} : \mathsf{D}^{\mathsf{ne}} \rightarrow \mathsf{Ne}$ let us quote domain values back into normal and neutral forms, at de Bruijn level n. We rely on a helper function $\mathsf{vs}^{\mathcal{U}}_{\mathsf{n}} : \mathsf{Clos}_k \rightarrow \mathsf{D}^{\mathcal{U}}$ to fully apply a closure to fresh variables.

$$\mathsf{vs}^{\mathcal{U}}_{\mathsf{n}}(\mathcal{A}) = \mathcal{A}[\uparrow \mathsf{Var}_n, \ldots, \uparrow \mathsf{Var}_{n+k-1}]$$

Quoting is given in Figure 7, and follows the expected pattern. The cases for equality and casting do not ensure statically that they produce neutral forms. However, it is a program invariant that one position will be a blocking neutral, so the term is a neutral form in Ne.

3.3 Semantic propositions

Proofs, i.e. inhabitants of propositions, are all equal. We should thus never evaluate such irrelevant terms, and this should be reflected in the design of the domain of semantic propositions D^{Ω}. We explored three different approaches, the first two of which (proof erasure and syntactic propositions) we found unsatisfactory, until we reached the final design we are happy with: implementing D^{Ω} as a semantic domain.

Proof erasure. The simplest strategy to handle proof-irrelevance is to rather brutally *erase* irrelevant terms at evaluation, as suggested in Abel et al. [2]. This amounts to having a single semantic proposition, i.e. defining $\mathsf{D}^{\Omega} ::= \mathsf{Witness}$. This vastly simplifies the implementation, by removing intricate proof manipulations.

Although this approach is correct in terms of deciding the equational theory, it has a major drawback: normal forms cannot be quoted back to terms. Indeed, since proofs are erased, there is no information left to quote! This has nasty consequences. First, we must deal with this missing information when printing terms back to users, for instance in error messages. Quoting is also using during unification, to provide term solutions for unification variables – see Sec. 4.3.

As a mitigation, Abel et al. propose to extend the language by a constant O which proves every provable proposition. In practice, this would amount to indicate in quoted terms where proofs exist, but without giving a precise witness. However, one must be careful to exclude O from the source language, as it makes type-checking wildly undecidable by making it depend on inhabitation. Decidability could be recovered by instead having O prove *all* propositions, at the cost of making the type theory inconsistent, also far from ideal. Thus, in this approach we have to maintain a careful and clumsy distinction between "user-issued" terms and "kernel-generated" ones, with annoying consequences in the interaction with users: for instance, it means they would not be able to copy code from messages. We thus chose to reject this solution and look for another one.

Syntactic propositions. A natural alternative is to retain *syntactic* witnesses for proof terms during evaluation: since we never need to evaluate propositions, there seems to be no point in translating them to a domain such as $\mathsf{D}^{\mathcal{U}}$, which is designed for efficient evaluation. This amounts to setting $\mathsf{D}^{\Omega} ::= \mathsf{Prop}\ t\ \rho$ – a proof witness t and an environment ρ interpreting its free variables. Indeed, while they do not reduce, propositions still admit *substitutions* of their variables, which happens when evaluating relevant terms, and must be accounted for. This is the point of the stored environment, used to interpret free variables during quoting.

More challenging, though, is the proof manipulation which occurs in casting reduction rules. This requires inserting a proof-relevant value into the proof witness and shifting propositions to be well-typed in a different context. In this approach, evaluation and quoting become mutually defined, and great care must be taken to transform witnesses correctly. This entanglement of evaluation and quoting is both unsatisfying and error-prone.

Semantic propositions. We therefore introduce a third design: using a semantic domain similar to $\mathsf{D}^{\mathcal{U}}$. The idea is that values in this domain never reduce, they only admit *substitution*, which is handled by closures, but they use de Bruijn levels, making it unnecessary to shift or quote terms during evaluation. As relevant data might appear as subterms of propositions, D^{Ω} also embeds relevant terms. However, these do not reduce: they are "frozen".

The domain of semantic propositions has a structure similar to terms, except for the use of closures for bound variables, and de Bruijn levels. Even let bindings and type annotations are represented, unevaluated. Calligraphic letters represent closures, which are the same as in Section 3.2, but have values from D^{Ω} in place of $\mathsf{D}^{\mathcal{U}}$.

$$\begin{aligned}
\mathsf{D}^{\Omega} \ni P, Q, p, q ::= {} & \mathsf{PVar}_l \mid \mathsf{PU}\ \mathfrak{s} \mid \mathsf{PLet}\ P\ p\ q \mid \mathsf{PAnn}\ p\ P \mid \mathsf{PAbort}\ P\ p \mid \mathsf{PEmpty} \\
& \mid \mathsf{PLam}\ \mathfrak{s}\ \mathcal{P}_1 \mid \mathsf{PApp}_{\mathfrak{s}}\ p\ q \mid \mathsf{PPi}\ \mathfrak{s}\ P\ \mathcal{Q}_1 \mid \mathsf{PZ} \mid \mathsf{PS}\ p \mid \mathsf{PRec}\ \mathcal{Q}_1\ p\ q\ \mathcal{P}_2 \mid \mathsf{PNat} \\
& \mid \mathsf{POne} \mid \mathsf{PUnit} \mid \mathsf{PPair}\ p\ q \mid \mathsf{PFst}\ p \mid \mathsf{PSnd}\ p \mid \mathsf{PExists}\ P\ \mathcal{Q} \\
& \mid \mathsf{PTransp}\ p\ \mathcal{Q}_2\ q\ p'\ e \mid \mathsf{PEq}\ p\ P\ p' \mid \mathsf{PCast}\ P\ Q\ p\ q \mid \mathsf{PRefl}\ p
\end{aligned}$$

Next, we introduce *freezing*, the mapping $\phi : \mathsf{D}^{\mathcal{U}} \rightarrowtriangle \mathsf{D}^{\Omega}$ of semantic relevant values into semantic propositions, so that they may be used in proof terms. It is defined mutually with $\Phi_k : \mathsf{Clos}_k^{\mathcal{U}} \rightarrowtriangle \mathsf{Clos}_k^{\Omega}$ which embeds closures. Representative cases are given as follows:

$$\begin{array}{rcl@{\quad}rcl}
\phi(\uparrow e) & = & \phi^{\mathsf{Ne}}(e) & \phi^{\mathsf{Ne}}(\mathsf{Var}_l) & = & \mathsf{PVar}_l \\
\phi(\mathsf{U}\ \mathfrak{s}) & = & \mathsf{PU}\ \mathfrak{s} & \phi^{\mathsf{Ne}}(\mathsf{App}_{\mathfrak{s}}\ e\ a) & = & \mathsf{PApp}_{\mathfrak{s}}\ (\phi^{\mathsf{ne}}(e))\ (\phi(a)) \\
\phi(\mathsf{Lam}\ \mathfrak{s}\ \mathcal{F}) & = & \mathsf{PLam}\ \mathfrak{s}\ (\Phi_1(\mathcal{F})) & \phi^{\mathsf{Ne}}(\mathsf{App}_{\mathfrak{s}}\ e\ p) & = & \mathsf{PApp}_{\mathfrak{s}}\ (\phi^{\mathsf{ne}}(e))\ p \\
\phi(\mathsf{Prop}\ p) & = & p & & \dots &
\end{array}$$

Freezing has to traverse the whole term, which is rather inefficient, especially if we repeatedly freeze the same term. In retrospect, it might be possible to replace it by a mere constructor which freely embeds $\mathsf{D}^{\mathcal{U}}$ into D^{Ω}.

Semantic interpretation for propositions, $[\![\_]\!]^{\Omega}\_ : \mathsf{Tm} \rightarrowtriangle \mathsf{Env} \rightarrowtriangle \mathsf{D}^{\Omega}$, is particularly easy as there are no reductions. We give only a few cases, others are entirely similar.

$$\begin{array}{rcl@{\quad}rcl}
[\![x_0]\!]^{\Omega}(\rho, \mathsf{P}\ p) & = & p & [\![\lambda x_{\mathfrak{s}}.\ t]\!]^{\Omega}\rho & = & \mathsf{PLam}\ \mathfrak{s}\ (\underline{\lambda} t)\rho \\
[\![x_0]\!]^{\Omega}(\rho, \mathsf{V}\ a) & = & \phi(a) & [\![t\ u^{\mathfrak{s}}]\!]^{\Omega}\rho & = & \mathsf{PApp}_{\mathfrak{s}}\ ([\![t]\!]^{\Omega}\rho)\ ([\![u]\!]^{\Omega}\rho) \\
[\![x_{i+1}]\!]^{\Omega}(\rho, d) & = & [\![x_i]\!]^{\Omega}\rho & & \dots &
\end{array}$$

Projections from the environment are like in relevant interpretation, although when the entry is relevant, we freeze it with $\phi : \mathsf{D}^{\mathcal{U}} \rightarrowtriangle \mathsf{D}^{\Omega}$. This handles substitution – syntactic variables x_i are substituted by values from the environment. λ-expressions introduce a closure which can be entered, but this never happens due to application, only during quoting. Interpretation of application always produces a PApp, even when the interpretation of t is PLam.

We also define a function $\mathsf{app}^{\Omega} : \mathsf{Clos}_k^{\Omega} \rightarrowtriangle \mathsf{D} \rightarrowtriangle \cdots \rightarrowtriangle \mathsf{D} \rightarrowtriangle \mathsf{D}^{\Omega}$ for applying closures. This works similarly to $\mathsf{app}^{\mathcal{U}}$, only no reduction occurs: compared to Figure 5, we replace each semantic operator by a constructor. For example,

$$\mathsf{app}^{\Omega}\ (\mathsf{EqFun}\ \mathfrak{s}\ p\ \mathcal{Q}\ p')\ q = \mathsf{PEq}\ (\mathsf{PApp}_{\mathfrak{s}}\ p\ q)\ \mathcal{Q}[q]\ (\mathsf{PApp}_{\mathfrak{s}}\ p'\ q)$$

Finally, we also have quoting for semantic propositions $\mathsf{q}_{\mathsf{n}}^{\Omega}$, which computes in the same way as quoting for $\mathsf{D}^{\mathcal{U}}$, fully applying closures to fresh variables.

3.4 Conversion checking

Conversion checking is used to decide the conversion relation $\equiv$, by operating on *semantic values*. This relation is non-trivial because normalisation, due to being untyped, does not handle extensionality rules. Yet, we still want to check equality up to η and irrelevance, so this is implemented when comparing semantic values. Conversion checking is *untyped* and *term-directed*, so η-expansion is triggered when one side of the conversion equation is a constructor, and the other is neutral.[6] For Π types, for instance, we get

$$\frac{\Gamma, x \vdash \mathcal{F}[\mathsf{Var}^{\mathfrak{s}}_{|\Gamma|}] \equiv t' \cdot_{\mathfrak{s}} \mathsf{Var}^{\mathfrak{s}}_{|\Gamma|}}{\Gamma \vdash \mathsf{Lam}\ \mathfrak{s}\ \mathcal{F} \equiv t'}$$

In practice, Γ is replaced by an integer representing its length.

Conversion checking between irrelevant terms always succeeds. In fact, we only define conversion checking between values in $\mathsf{D}^{\mathcal{U}}$, and proof irrelevance is implemented by *not recursively comparing irrelevant terms.* Consider for instance applications:

$$\frac{\Gamma \vdash \uparrow e \equiv \uparrow e' \qquad \Gamma \vdash a \equiv a'}{\Gamma \vdash \uparrow(\mathsf{App}_{\mathcal{U}}\ e\ a) \equiv \uparrow(\mathsf{App}_{\mathcal{U}}\ e'\ a')} \qquad \frac{\Gamma \vdash \uparrow e \equiv \uparrow e'}{\Gamma \vdash \uparrow(\mathsf{App}_{\Omega}\ e\ p) \equiv \uparrow(\mathsf{App}_{\Omega}\ e'\ p')}$$

[6] This strategy is due to Coquand [10], and implemented in Coq and in Agda for functions.

When arguments are irrelevant, $p \equiv p'$ is automatic and thus not checked. This explains the annotation of applications with the sort of their argument, to decide which rule to choose.

On top of η-conversion, we also implement the extra $\mathbf{cast}(A, A, e, t) \equiv t$ equation. Treating this rule as a reduction breaks determinism, as there are multiple paths to evaluate some cast expressions. More worrying, it makes reduction and conversion checking mutually recursive. Thus, it is easier to implement it during conversion checking, although this necessitates backtracking. The implementation uses the following rules.

$$\textsc{Cast-Eq-Left}\ \frac{\begin{array}{c}\Gamma \vdash A \equiv B \\ \Gamma \vdash a \equiv a'\end{array}}{\Gamma \vdash \mathsf{Cast}\ A\ B\ e\ a \equiv a'} \qquad \textsc{Cast-Eq-Right}\ \frac{\begin{array}{c}\Gamma \vdash A' \equiv B' \\ \Gamma \vdash a \equiv a'\end{array}}{\Gamma \vdash a \equiv \mathsf{Cast}\ A'\ B'\ e'\ a'} \qquad \textsc{Cast-Conv}\ \frac{\begin{array}{c}\Gamma \vdash A \equiv A' \qquad \Gamma \vdash B \equiv B' \\ \Gamma \vdash a \equiv a'\end{array}}{\Gamma \vdash \mathsf{Cast}\ A\ B\ e\ a \equiv \mathsf{Cast}\ A'\ B'\ e'\ a'}$$

When the left-hand side of the equality is a cast between A and B, we first try Cast-Eq-Left. If this fails, we proceed with Cast-Eq-Right, and if that fails too, Cast-Conv. Backtracking is necessary as it is impossible to know a priori which strategy succeeds. Fortunately, eagerly applying Cast-Eq-Left and Cast-Eq-Right is complete, i.e. it does not lose solutions. Indeed, if Cast-Eq-Left and Cast-Conv both apply to derive $\mathsf{Cast}\ A\ B\ e\ a \equiv \mathsf{Cast}\ A'\ B'\ e'\ a'$, then all four types A, B, A' and B' are convertible, and so Cast-Conv can be replaced by Cast-Eq-Left and Cast-Eq-Right.

3.5 Bidirectional type checking

Our approach to type-checking, based on bidirectional typing, is close to e.g. Gratzer et al. [15], or what Agda implements [24]. We mutually implement the *two* judgements

$$\Gamma; \rho \vdash t \Rightarrow A \text{ (infer)} \qquad\qquad \Gamma; \rho \vdash t \Leftarrow A \text{ (check)}$$

Besides the context Γ, we have an environment ρ interpreting it, which is necessary to handle let-binders, as explained below. Typing uses *semantic* types, $A \in \mathsf{D}^{\mathcal{U}}$. Although we do not include it on paper, type-checking actually *elaborates* terms: while checking a pre-term in PreTm, we generate a term in Tm, for instance annotating applications with a sort.

In this approach, constructors are checked, and destructors are inferred, and thus only normal forms are typable. To allow for more flexibility, we add two mechanisms. Type annotations let us locally record a type in a term to get back to type-checking when necessary, as per the following rule:

$$\frac{\Gamma; \rho \vdash A \Rightarrow \mathsf{U}\ \mathfrak{s} \qquad \Gamma; \rho \vdash t \Leftarrow \llbracket A \rrbracket^{\mathcal{U}} \rho}{\Gamma; \rho \vdash (t : A) \Rightarrow \llbracket A \rrbracket^{\mathcal{U}} \rho}$$

Let-bindings, on the other hand, can be checked or inferred, and the current mode is forwarded to the body u, while the bound term t is always checked against the (evaluation of the) type A.

$$\frac{\begin{array}{c}\Gamma; \rho \vdash A \Leftarrow \mathsf{U}\ \mathfrak{s} \qquad \Gamma; \rho \vdash u \Leftarrow \llbracket A \rrbracket^{\mathcal{U}} \rho \\ \Gamma, x :_{\mathfrak{s}} \llbracket A \rrbracket^{\mathcal{U}} \rho; (\rho, \llbracket u \rrbracket^{\mathfrak{s}} \rho) \vdash t \Rightarrow B\end{array}}{\Gamma; \rho \vdash \mathbf{let}\ x :_{\mathfrak{s}} A = u\ \mathbf{in}\ t \Rightarrow B} \quad \frac{\begin{array}{c}\Gamma; \rho \vdash A \Leftarrow \mathsf{U}\ \mathfrak{s} \qquad \Gamma; \rho \vdash u \Leftarrow \llbracket A \rrbracket^{\mathcal{U}} \rho \\ \Gamma, x :_{\mathfrak{s}} \llbracket A \rrbracket^{\mathcal{U}} \rho; (\rho, \llbracket u \rrbracket^{\mathfrak{s}} \rho) \vdash t \Leftarrow B\end{array}}{\Gamma; \rho \vdash \mathbf{let}\ x :_{\mathfrak{s}} A = u\ \mathbf{in}\ t \Leftarrow B}$$

The environment is extended with the *value* of the variable x, so that it can be used transparently while typing u. We also get sharing, because a let-bound t is only evaluated once at its binding site. Compare with a β-redex, where the environment is extended by a variable:

$$\frac{\Gamma;\rho\vdash A\Rightarrow \mathsf{U}\ \mathfrak{s} \qquad \Gamma, x :_{\mathfrak{s}} A;(\rho,\mathsf{Var}^{\mathfrak{s}}_{|\Gamma|})\vdash B\Rightarrow \mathsf{U}\ \mathfrak{s} \\ \Gamma, x :_{\mathfrak{s}} A;(\rho,\mathsf{Var}^{\mathfrak{s}}_{|\Gamma|})\vdash t\Leftarrow \mathcal{B}[\mathsf{Var}^{\mathfrak{s}}_{|\Gamma|}] \qquad \Gamma;\rho\vdash u\Leftarrow A}{\Gamma;\rho\vdash(\lambda x.\ t:\Pi x:A.B)\ u\Rightarrow\mathcal{B}[u]}$$

Finally, while in vanilla MLTT **refl** and the equality type are constructors, this is not true in CC$^{\text{obs}}$. Since equality types reduce, it is unfeasible, given a semantic type, to guess which equality $a \sim_A a'$ it represents. For instance, many equalities reduce to $\bot$. The reasonable solution is to have reflexivity infer, in line with viewing it as a destructor on the universe:

$$\frac{\Gamma;\rho\vdash t\Rightarrow A}{\Gamma;\rho\vdash \mathbf{refl}\ t\Rightarrow \underline{\mathsf{eq}}(\llbracket t\rrbracket^{\mathcal{U}}\rho, A, \llbracket t\rrbracket^{\mathcal{U}}\rho)}$$

Note the use of the semantic equality type $\underline{\mathsf{eq}}$ as the inferred type. So, for example, **refl** 0 will infer $\top$, as we immediately reduce the type $0 \sim_{\mathbb{N}} 0$. Also note that this issue could have been avoided by having an equality type which does not compute, which is in hindsight a rather compelling argument for this approach.

4 Extensions

4.1 Quotient types

The ability to easily handle *quotient types* is one of the landmarks of observational type theory, it is thus natural to consider them as our first extension. The adaptation of NbE is actually relatively straightforward, and follows the same patterns as the core implementation. We first extend normal and neutral forms:

$$\begin{aligned}\mathsf{Nf}\ni v,w,V,W &::= \cdots \mid V/(x\ y.\ W, x.\ R_r, x\ y\ xRy.\ R_s, x\ y\ z\ xRy\ yRz.\ R_t)\mid \pi\ v\\ \mathsf{Ne}\ni n,N &::= \cdots \mid \mathbf{Q\text{-}elim}[z.\ V](x.\ v_\pi, x\ y\ xRy.\ t_\sim, n)\end{aligned}$$

Note that the proofs that the relation is an equivalence – in quotient formation – and that the quotient is respected – in the eliminator – are arbitrary, as they are propositions.

We extend the semantic domains accordingly, introducing three new closures to defunctionalise the equality type reduction between quotient types and its three binders.

$$\begin{array}{ll}\mathsf{D}^{\mathcal{U}} & ::= \cdots \mid \mathsf{Quotient}\ A\ \mathcal{B}_2\ \mathcal{P}^r_1\ \mathcal{P}^s_3\ \mathcal{P}^t_5 \mid \mathsf{Qproj}\ a\\ \mathsf{D}^{\mathsf{ne}} & ::= \cdots \mid \mathsf{Qelim}\ \mathcal{B}_1\ \mathcal{F}_1\ \mathcal{Q}_3\ e\\ \mathsf{D}^{\Omega} & ::= \cdots \mid \mathsf{PQuotient}\ P\ \mathcal{Q}_2\ \mathcal{P}^r_1\ \mathcal{P}^s_3\ \mathcal{P}^t_5 \mid \mathsf{PQproj}\ p \mid \mathsf{PQelim}\ \mathcal{P}_1\ \mathcal{P}'_1\ \mathcal{Q}_3\ p\\ \mathsf{Clos}^{\mathfrak{s}}_1 & ::= \cdots \mid \mathsf{EqQuotientY}\,p\ a\ D\ D'\ \mathcal{C}_2\ \mathcal{C}_2 \mid \mathsf{EqQuotientX}\,p\ D\ D'\ \mathcal{C}_2\ \mathcal{C}_2 \mid \mathsf{EqQuotient}\,D\ D'\ \mathcal{C}_2\ \mathcal{C}_2\end{array}$$

Relevant interpretation for quotients follows the same pattern as the core NbE algorithm. Quotient types are interpreted into the Quotient constructor, with the appropriate closures. Projections are interpreted into the Qproj constructor. Elimination is defined by

$$\llbracket \mathbf{Q\text{-}elim}[z.B](t_\pi, t_\sim, u)\rrbracket^{\mathcal{U}}(\rho) = \underline{\mathsf{Qelim}}((\underline{\lambda}B)\rho, (\underline{\lambda}t_\pi)\rho, (\underline{\lambda}t_\sim)\rho, \llbracket u\rrbracket^{\mathcal{U}}\rho)$$

where $\underline{\mathsf{Qelim}}$ reduces quotient eliminators applied to projections.

$$\begin{aligned}\underline{\mathsf{Qelim}}(\mathcal{B},\mathcal{F}_\pi,\mathcal{P}_\sim,\mathsf{Qproj}\ b) &= \mathcal{F}_\pi[b]\\ \underline{\mathsf{Qelim}}(\mathcal{B},\mathcal{F}_\pi,\mathcal{P}_\sim,\uparrow e) &= \uparrow(\mathsf{Qelim}\ \mathcal{B}\ \mathcal{F}_\pi\ \mathcal{P}_\sim\ e)\end{aligned}$$

Irrelevant interpretation, $\llbracket\_\rrbracket^{\Omega}\_$, is trivially extended: once again, there is no reduction when an eliminator is applied to a projection. Quoting also takes the same form as before.

4.2 Inductive types and Mendler-style induction

Our next extension is to add inductive types, following the theory laid down in Section 2.2. The normal forms are as follows – we omit the cases of inductive types without functor instances and fixed-points without views; they follow the same shape as those presented.

$$
\begin{array}{lll}
\mathsf{Nf} & ::= & \cdots \mid \mu F : V \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : W_i) \to F\ v_i}]\ \textbf{functor}\ X\ Y\ f\ p\ x = v \\
 & \mid & (\mu F : V \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : W_i) \to F\ v_i}]\ \textbf{functor}\ X\ Y\ f\ p\ x = v)\ w \\
 & \mid & C_i\ (v, e) \mid \textbf{fix}\ [V\ \textbf{as}\ G\ \textbf{view}\ \iota]\ f\ p\ x : W = v \\
 & \mid & (\textbf{fix}\ [V\ \textbf{as}\ G\ \textbf{view}\ \iota]\ f\ p\ x : W = v)\ w \\
\mathsf{Ne} & ::= & \cdots \mid \textbf{match}\ n\ \textbf{as}\ x\ \textbf{return}\ V\ \textbf{with}\ \{C_i\ (x_i, e_i) \to v_i\}_i \\
 & \mid & (\textbf{fix}\ [V\ \textbf{as}\ G]\ f\ p\ x : W = v)\ w\ n \\
 & \mid & \textbf{in}\ n \mid \textbf{lift}[N]\ V \mid \textbf{fmap}[N]
\end{array}
$$

Both unapplied inductive type (families) and fixed points are new normal forms. Inductive types applied to their indices are normal forms at the universe, and similarly for fixed-points. Finally, a fixed point applied to an index is neutral when its second argument, on which the fixed point computes, is itself neutral. The **lift** and **fmap** terms block when their inductive type is unknown – we cannot lift type families and functions without the definition at hand –, while **in** blocks when applied to a neutral.

Semantic domains. As usual, the first step is to extend the domains, driven by the structure of the normal and neutral forms. Like before, we omit values for inductive types without functor definitions (in the implementation, we have optional value for the functor definition).

$$
\begin{array}{lll}
\mathsf{D} & ::= & \cdots \mid \mathsf{Mu}\ A\ (\mathsf{List}\ (\mathcal{B}_1, \mathcal{A}_2))\ \mathcal{F}_5\ (\mathsf{Maybe}\ a) \mid \mathsf{Cons}\ \mathsf{Name}\ a\ p \mid \mathsf{Fix}\ A\ \mathcal{B}_4\ \mathcal{F}_5\ (\mathsf{Maybe}\ a) \\
\mathsf{D}^{\mathsf{ne}} & ::= & \cdots \mid \mathsf{Match}\ e\ \mathcal{B}_1\ (\mathsf{List}\ \mathcal{A}_2) \mid \mathsf{FixApp}\ A\ \mathcal{B}_4\ \mathcal{F}_5\ a\ e \mid \mathsf{In}\ e \mid \mathsf{Lift}\ E\ A \mid \mathsf{Fmap}\ E\ A\ B\ a\ b\ c
\end{array}
$$

The semantic value of inductive types, Mu, contains the index type, followed by a list of pairs representing constructor types and indices. We have an optional value representing the presence or absence of an index. Fixed-points have a similar structure. The neutral form FixApp represents a fixed-point blocked by a neutral argument. Defunctionalizing closures and semantic propositions are entirely unsurprising, and we omit them here.

Interpretation. First, we give the semantics of the newly added terms.

$$
\begin{aligned}
[\![\mu F : A \to \mathcal{U}.\ [\overrightarrow{C_i : (x_i : B_i) \to F\ a_i}]\ \textbf{functor}\ X\ Y\ f\ p\ x = t]\!]^{\mathcal{U}}\rho &= \\
\mathsf{Mu}\ ([\![A]\!]^{\mathcal{U}}\rho)\ \ [(\underline{\lambda}B_i)\rho, (\underline{\lambda}a_i)\rho]_i\ \ (\underline{\lambda}t)\rho\ \mathsf{Nothing} & \\
[\![C_i\ (t, e)]\!]^{\mathcal{U}}\rho &= \mathsf{Cons}\ C_i\ ([\![t]\!]^{\mathcal{U}}\rho)\ ([\![e]\!]^{\Omega}\rho) \\
[\![\textbf{match}\, t\, \textbf{as}\, x\, \textbf{return}\, C\, \textbf{with}\, \{t_i\}_i]\!]^{\mathcal{U}}\rho &= \underline{\mathsf{match}}([\![t]\!]^{\mathcal{U}}\rho, (\underline{\lambda}C)\rho, [(\underline{\lambda}a_i)\rho]_i) \\
[\![\textbf{fix}[M\, \textbf{as}\, G\, \textbf{view}\, \iota]\ f\ p\ x : C = t]\!]^{\mathcal{U}}\rho &= \mathsf{Fix}\ ([\![M]\!]^{\mathcal{U}}\rho)\ (\underline{\lambda}C)\rho\ (\underline{\lambda}t)\rho\ \mathsf{Nothing} \\
[\![\textbf{in}\ t]\!]^{\mathcal{U}}\rho &= \underline{\mathsf{in}}([\![t]\!]^{\mathcal{U}}\rho) \\
[\![\textbf{lift}[M]\ A]\!]^{\mathcal{U}}\rho &= \underline{\mathsf{lift}}([\![M]\!]^{\mathcal{U}}\rho, [\![A]\!]^{\mathcal{U}}\rho) \\
[\![\textbf{fmap}[M](A, B, f, u, t)]\!]^{\mathcal{U}}\rho &= \underline{\mathsf{fmap}}([\![M]\!]^{\mathcal{U}}\rho, [\![A]\!]^{\mathcal{U}}\rho, [\![B]\!]^{\mathcal{U}}\rho, [\![f]\!]^{\mathcal{U}}\rho, [\![u]\!]^{\mathcal{U}}\rho, [\![t]\!]^{\mathcal{U}}\rho)
\end{aligned}
$$

The implementations of $\underline{\mathsf{match}}$, $\underline{\mathsf{lift}}$, $\underline{\mathsf{fmap}}$ and $\underline{\mathsf{in}}$ all branch on their argument being either a constructor, in which case the relevant branch is taken, or a neutral, which leads to a neutral. They are found in the code.

Reduction of fixed points is handled by application, and we thus extend $\_ \cdot_{\mathfrak{s}} \_$ as follows:

$$\begin{aligned}(\mathsf{Mu}\ A\ \mathsf{cs}\ \mathcal{F}\ \mathsf{Nothing}) \cdot_{\mathcal{U}} a &= \mathsf{Mu}\ A\ cs\ \mathcal{F}\ (\mathsf{Just}\ a)\\ (\mathsf{Fix}\ A\ \mathcal{C}\ \mathcal{F}\ \mathsf{Nothing}) \cdot_{\mathcal{U}} a &= \mathsf{Fix}\ A\ \mathcal{C}\ \mathcal{F}\ (\mathsf{Just}\ a)\\ (\mathsf{Fix}\ A\ \mathcal{C}\ \mathcal{F}\ (\mathsf{Just}\ a)) \cdot_{\mathcal{U}} (\mathsf{Cons}\ C_i\ b\ p) &= \mathcal{F}[A, \underline{\mathrm{id}}, \mathsf{Fix}\ A\ \mathcal{C}\ \mathcal{F}\ \mathsf{Nothing}, a, \mathsf{Cons}\ C_i\ b\ p]\\ (\mathsf{Fix}\ A\ \mathcal{C}\ \mathcal{F}\ (\mathsf{Just}\ a)) \cdot_{\mathcal{U}} (\uparrow e) &= \uparrow(\mathsf{FixApp}\ A\ \mathcal{C}\ \mathcal{F}\ a\ e)\end{aligned}$$

When a semantic inductive type or fixed-point have not been applied to their index – indicated by their final component being Nothing –, we move this index into the value. When a fixed-point with an index is applied to a constructor, we invoke the closure $\mathcal{F}$, i.e. the body of the fixed point. Note how we pass the semantic identity function $\underline{\mathrm{id}} \triangleq [\![\lambda p\ x.x]\!]^{\mathcal{U}}$ for the view.

4.3 Pattern unification

Pattern unification [3, 18] does not extend the theory, but makes writing terms more practical. Our implementation combines the contextual metavariables of Abel and Pientka [1] and NbE. Another interesting aspect, as we already mentioned, is that unification partly motivates the structure we chose for semantic propositions in Section 3.3, as we will shortly explain. Contrarily to other work, we do not insist on computing only most general solution, which would require further modifications to the structure of neutral forms.

Storing solved metavariables. Pattern unification relies on metavariables, written $?m$ and recorded in a metavariable context Σ, which contains metavariables already solved through unification, and yet unsolved ones, as follows.

$$\Sigma ::= \cdot \mid \Sigma, ?m_i \mid \Sigma, ?m_i := t \mid \Sigma, ?s_i \mid \Sigma, ?s_i := \mathfrak{s}$$

Note that we have metavariables $?s$ for sorts, too. But what should t be? That is, should metavariables stand for terms or for semantic values? To answer this, we must understand what operations we want to perform on them.

Metavariables are introduced during typing. As explained in Section 3.5, typing really is an elaboration procedure, taking in a preterm and constructing a term. In the presence of metavariables, it becomes *stateful*, as the metavariable context can be updated. The interesting rule is that which handles a hole in the surface syntax, creating two new metavariables representing the term and its type, and adding both to the metavariable context.

$$\frac{?m_i, ?m_j \text{ fresh in } \Sigma}{\Sigma; \Gamma \vdash \_ \rightsquigarrow ?m_i \Rightarrow ?m_j;\ \Sigma, ?m_j, ?m_i}$$

As any other term, we need to be able to evaluate metavariables into the semantic domains. If a metavariable is unsolved, it behaves like a variable and blocks the term, creating a new form of neutrals. We extend the semantic domain accordingly:

$$\mathsf{D}^{\mathsf{ne}} \quad \ni \quad e \quad ::= \quad \cdots \mid \mathsf{Meta}\ ?m_i\ \rho$$

If a metavariable is defined, though, evaluation should reflect it. We achieve this as follows:

$$\begin{aligned}&[\![?m_i]\!]^{\mathcal{U}}\rho = [\![t]\!]^{\mathcal{U}}\rho && \text{when } (?m_i := t) \in \Sigma\\ &[\![?m_i]\!]^{\mathcal{U}}\rho = \mathsf{Meta}\ ?m_i\ \rho && \text{when } ?m_i \in \Sigma\end{aligned}$$

Indeed, we want solved metavariables to admit a substitution operation, so that we can use them in a context different from the one they were created in. Evaluation of terms is tailored to handle an environment representing a substitution, leading to the definition above. Semantic values, on the contrary, do not easily admit substitution. This leads to the choice of taking $t \in \mathsf{Tm}$ above, and to use evaluation to handle the environment.

Solving metavariables. Metavariables are solved during conversion checking, from equations of the form $\Delta \vdash \mathsf{Meta}\ ?m_i\ \rho \equiv a$. To solve such an equation, we create a partial function $\mathsf{invert} : \mathsf{Env} \rightharpoonup \mathsf{Renaming}$ which succeeds when the environment ρ satisfies the linearity conditions necessary to invert it. Partial renamings are lists of variables (with de Bruijn levels) and undefined values, $\Uparrow$, i.e. they are given by the following datastructure

$$\mathsf{Renaming} \quad \ni \quad \theta \quad ::= \quad () \mid (\theta, \mathsf{Var}_l) \mid (\theta, \Uparrow)$$

We equivalently view them as partial functions on variables when this view is more useful.

In conversion, we always compare *values*. However, metavariable solutions are terms. Therefore, when applying the renaming ρ^{-1} to the value a, we simultaneously update free variables and quote the semantic value into a term. This gives us an operation $\mathsf{rename}^{n}_{?m_i} : \mathsf{D}^{\mathcal{U}} \rightharpoonup \mathsf{Renaming} \rightharpoonup \mathsf{Tm}^{\mathcal{U}}$. Like with quoting, the level n representing the depth we rename at. We also have access to the metavariable $?m_i$ for the occurs check.

$$\begin{aligned}
\mathsf{rename}^{n}_{?m_i}(\mathsf{Var}_k)\ \theta &= x_{n-l+1} && \text{when } \theta(\mathsf{Var}_k) = \mathsf{Var}_l \\
\mathsf{rename}^{n}_{?m_i}(\mathsf{Meta}\ ?m_j)\ \theta &=?m_j && \text{when } ?m_i \neq ?m_j \\
\mathsf{rename}^{n}_{?m_i}(\uparrow (\mathsf{App}\ n\ a))\ \theta &= (\mathsf{rename}^{n}_{?m_i}(\uparrow n)\theta)\ (\mathsf{rename}^{n}_{?m_i}(a)\theta) \\
\mathsf{rename}^{n}_{?m_i}(\uparrow (\mathsf{Lam}\ \mathfrak{s}\ \mathcal{F}))\ \theta &= \lambda_{\mathfrak{s}}.\ (\mathsf{rename}^{n+1}_{?m_i}(\mathcal{F}[\mathsf{Var}_n])(\theta, \mathsf{Var}_{n+1}))
\end{aligned}$$

The first rule ensures the variable Var_k is not escaping, by checking it is defined in θ. The second rule implements the occurs check by ensuring the metavariable $?m_i$ does not appear in its own solution. This way, we isolate it from linearity, which depends only on the environment ρ. When going under a binder, we apply the closure to a fresh variable and extend the renaming in the natural way. A similar renaming operation is defined for semantic propositions in D^{Ω}. Since we want to obtain proper terms after renaming, we need D^{Ω} to contain enough information, explaining why the proof erasure semantics is not well-suited.

The final step is to piece these parts together to solve metavariables in conversion checking, then update the metavariable context. Both invert and rename might (validly) fail, so we take $\doteq$ to mean that the result is defined, and assigned to the variable on the left.

$$\frac{\rho^{-1} \doteq \mathsf{invert}(\rho) \qquad t \doteq \mathsf{rename}^{|\Delta|}_{?m_i}(a)\rho^{-1}}{(\Sigma, ?m_i, \Sigma'); \Delta \vdash ?m_i[\rho] \equiv a; (\Sigma, ?m_i := t, \Sigma')}$$

4.4 A lightweight mechanism for goals

To be able to write complex terms, it is handy to build them incrementally. For this, we need a form of interactivity, where the proof assistant informs us as to what we need to provide to complete the proof: *proof goals*. In full-fledged proof assistants, goals are handled by the IDE and displayed using a secondary window. Implementing this is heavy, but proof goals are very necessary to be able to design even moderately-sized examples.

Instead, we implement a lightweight mechanism, where goals are inserted into the source code to throw an error at the specific location we want to investigate. The error message indicates the expected type at that point, if known. Additionally, goals optionally contain lists of terms whose types are reported to the user. We write `?{t1, t2, ..., tn}` for a goal term and a list of terms `ti`. For example, consider the following code, in which we insert a proof goal. Running the checker on the code on the left, we get the message on the right.

```
let f : ℕ → ℕ =
  λx. S x
in
let x : ℕ =
  S (S (S 0))
in
f ?{f, x}
```

```
[error]: Found proof goal.
     ┌─▶ <test-file>@8:7-8:14
   8 │     f ?{f, x}
     •        ┬──────
     •        ╰╴ Expected type [ℕ] at goal.
     •
     •           List of relevant terms and their types:
     •           f : ℕ → ℕ
     •           x : ℕ
─────┘
```

Using this and pattern unification, we were able to build sizable examples. The first implements booleans as a quotient on the natural numbers by the relation which relates all numbers $n \geq 1$, leaving exactly two elements: zero and “everything else”. The second reimplements Fiore’s NbE for the simply typed lambda calculus [12]. This proof demonstrates the expressivity of the system, and makes extensive use of inductive types. We do not reproduce these examples here for lack of space, but the code is available in the repository [31] and dissertation [30].

5 Conclusion

We described an NbE implementation of a significant dependently-typed language, in particular featuring observational equality and Mendler-style induction. Despite requiring some care and new ideas in the design of the implementation, especially around the semantic representation of strict propositions, the standard concepts of NbE largely apply, witnessing the robustness and power of the technique.

More generally, guiding principles drawn from the literature around unification, normalisation by evaluation, and dependent type theory in general were reliable and very useful in the design of our system. Although still very much a prototype, critically missing a proper universe system and a positivity checker, our language is usable: we have been able to implement NbE for the simply typed λ-calculus in the style of Fiore [12] in it.

That we were able to build such a system without major blockers and in a relatively straightforward manner is a testament to the solidity of the state of the art in the implementation of dependent type theories. On the other hand, many of the experimental implementations we learned a lot from are relatively under-documented, and often unpublished. We are surely not the only ones who would have benefitted from a more visible literature on the subject, and we can only encourage the authors of these experimental implementations to describe their valuable work in publications or talks!

References

1 Andreas Abel. *Normalization by Evaluation Dependent Types and Impredicativity*. PhD thesis, Institut für Informatik Ludwig-Maximilians-Universität München, München, May 2013. URL: `https://www.cse.chalmers.se/~abela/habil.pdf`.

2 Andreas Abel, Thierry Coquand, and Miguel Pagano. A modular type-checking algorithm for type theory with singleton types and proof irrelevance. In Pierre-Louis Curien, editor, *Typed Lambda Calculi and Applications*, pages 5–19, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. `doi:10.1007/978-3-642-02273-9_3`.

3 Andreas Abel and Brigitte Pientka. Higher-Order Dynamic Pattern Unification for Dependent Types and Records. In Luke Ong, editor, *Typed Lambda Calculi and Applications*, volume 6690, pages 10–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. `doi:10.1007/978-3-642-21691-6_5`.

4 Thorsten Altenkirch. Extensional equality in intensional type theory. In *Proceedings - Symposium on Logic in Computer Science*, pages 412–420, February 1999. doi:10.1109/LICS.1999.782636.

5 Thorsten Altenkirch and Conor McBride. Towards Observational Type Theory, 2006.

6 Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification*, pages 57–68, Freiburg Germany, October 2007. ACM. doi:10.1145/1292597.1292608.

7 Bob Atkey. Simplified observational type theory, 2017. URL: https://github.com/bobatkey/sott.

8 Henning Basold and Herman Geuvers. Type Theory based on Dependent Inductive and Coinductive Types. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 327–336, July 2016. doi:10.1145/2933575.2934514.

9 Rafael Bocquet. obstt, 2023. URL: https://gitlab.com/RafaelBocquet/obstt.

10 Thierry Coquand. An algorithm for testing conversion in type theory. *Logical frameworks*, 1:255–279, 1991.

11 Thierry Coquand and Christine Paulin. Inductively defined types. In G. Goos, J. Hartmanis, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, Per Martin-Löf, and Grigori Mints, editors, *COLOG-88*, volume 417, pages 50–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990. doi:10.1007/3-540-52335-9_47.

12 Marcelo Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. *Mathematical Structures in Computer Science*, 32(8):1028–1065, September 2022. doi:10.1017/S0960129522000263.

13 Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without k. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–28, January 2019. doi:10.1145/329031610.1145/3290316.

14 Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

15 Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. Implementing a modal dependent type theory. *Proc. ACM Program. Lang.*, 3(ICFP), July 2019. doi:10.1145/3341711.

16 Robert Harper. Boolean Blindness, 2011. URL: https://existentialtype.wordpress.com/2011/03/15/boolean-blindness/.

17 Martin Hofmann. *Extensional Concepts in Intensional Type Theory.* PhD thesis, University of Edinburgh, 1995. URL: https://www.lfcs.inf.ed.ac.uk/reports/95/ECS-LFCS-95-327/.

18 András Kovács. Elaboration with first-class implicit function types. *Proceedings of the ACM on Programming Languages*, 4(ICFP):1–29, August 2020. doi:10.1145/3408983.

19 András Kovács. Elaboration-zoo, May 2023. URL: https://github.com/AndrasKovacs/elaboration-zoo.

20 András Kovács. Efficient evaluation with controlled definition unfolding. In *3rd Workshop on the Implementation of Type Systems*, 2025.

21 Per Martin-Löf. Intuitionistic type theory. In *Studies in Proof Theory*, 1984.

22 Conor McBride. *Dependently typed functional programs and their proofs.* PhD thesis, University of Edinburgh, 1999.

23 Nax Paul Mendler. Inductive types and type constraints in the second-order lambda calculus. *Annals of Pure and Applied Logic*, 51(1-2):159–172, March 1991. doi:10.1016/0168-0072(91)90069-X.

24 Ulf Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, September 2007.

25 Erik Palmgren. *On universes in type theory*, pages 191—-204. Oxford University Press, 1998. doi:10.1093/oso/9780198501275.003.0012.

26 Loïc Pujet. *Computing with Extensionality Principles in Dependent Type Theory*. PhD thesis, Nantes Université, December 2022.

27 Loïc Pujet and Nicolas Tabareau. Observational equality: Now for good. *Proceedings of the ACM on Programming Languages*, 6(POPL):1–27, January 2022. doi:10.1145/3498693.

28 Loïc Pujet and Nicolas Tabareau. Impredicative observational equality. *Proc. ACM Program. Lang.*, 7(POPL), January 2023. doi:10.1145/3571739.

29 Loïc Pujet and Nicolas Tabareau. Observational equality meets cic. In Stephanie Weirich, editor, *Programming Languages and Systems*, pages 275–301. Springer Nature Switzerland, 2024. doi:10.1007/978-3-031-57262-3_12.

30 Matthew Sirman. A normalisation by evaluation implementation of a type theory with observational equality. Bachelor's thesis, University of Cambridge, 2023.

31 Matthew Sirman. observational-type-theory, May 2023. URL: https://github.com/matthew-sirman/observational-type-theory.

32 The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 367–381, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3372885.3373824.

Data Types with Symmetries via Action Containers

Philipp Joram

Department of Software Science, Tallinn University of Technology, Estonia

Niccolò Veltri

Department of Software Science, Tallinn University of Technology, Estonia

Abstract

We study two kinds of containers for data types with symmetries in homotopy type theory, and clarify their relationship by introducing the intermediate notion of action containers. Quotient containers are set-valued containers with groups of permissible permutations of positions, interpreted as (possibly non-finitary) analytic functors on the category of sets. Symmetric containers encode symmetries in a groupoid of shapes, and are interpreted accordingly as polynomial functors on the 2-category of groupoids.

Action containers are endowed with groups that act on their positions, with morphisms preserving the actions. We show that, as a category, action containers are equivalent to the free coproduct completion of a category of group actions. We derive that they model non-inductive single-variable strictly positive types in the sense of Abbott et al.: The category of action containers is closed under arbitrary (co)products and exponentiation with constants. We equip this category with the structure of a locally groupoidal 2-category, and prove that it locally embeds into the 2-category of symmetric containers. This follows from the embedding of a 2-category of groups into the 2-category of groupoids, extending the delooping construction.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Categorical semantics

Keywords and phrases Containers, Homotopy Type Theory, Agda, 2-categories

Digital Object Identifier 10.4230/LIPIcs.TYPES.2024.6

Supplementary Material *Software (Agda)*: `https://github.com/phijor/cubical-containers`
archived at `swh:1:dir:a77c8b642661074b9804670b14cf97cc844aff16`

Funding This work was supported by the Estonian Research Council grant PSG749.
Philipp Joram: Participation at TYPES'24 was supported by the COST Action CA20111 – European Research Network on Formal Proofs (EuroProofNet)

1 Introduction

Containers are a representation of strictly positive data types, introduced by Abbott et al. [1]. A container consists of a type of *shapes* and a type of *positions* associated to each shape. For example, the container of ordered lists consists of natural numbers $n : \mathbb{N}$ as shapes and finite ordinals $\mathsf{Fin}(n)$ as positions, the idea being that each list has a length n and $\mathsf{Fin}(n)$-many locations that can hold data. Containers can be interpreted as polynomial endofunctors, the latter being the polymorphic data types that the containers represent. For example, the interpretation of the list container is the List functor.

Containers form a category. Its morphisms represent polymorphic functions between data types, which are interpreted as natural transformations between the corresponding polynomial endofunctors. This category is rich in structure; among other things it is cocartesian closed, is closed under the construction of initial algebras and terminal coalgebras, and admits a notion of derivative.

Traditionally, the theory of containers is studied in Set-like categories. When interpreted in such categories, the data of containers may however be too restrictive to encode certain data types of interest. This is especially the case if one wants to account for symmetries, i.e. identify configurations of positions when one can turn into the other via the action of certain permutations. For example, one can represent ordered lists, but it is not possible to represent cyclic lists or finite multisets as a container.

Some efforts have been made to enhance the expressivity of containers to represent data types with symmetries. Abbott et al. [3] introduced quotient containers, which are containers in which the assignment of values to positions is invariant under a collection of permutations on positions. This is realized by requiring the presence of a subgroup of the symmetric group on positions, corresponding to the collection of admissible permutations. Interpretation of quotient containers embeds them as a subcategory of set-endofunctors, targeting certain quotients of polynomial endofunctors. For quotient containers with finitary positions, these correspond to Joyal's analytic functors [15].

Symmetric containers, introduced by Gylterud [10], consist of a groupoid of shapes, with positions being a set-valued functor over this groupoid. This means that the symmetries are encoded directly in the isomorphisms of the shape groupoid. From the perspective of (homotopy) type theory, symmetric containers correspond to set-bundles over homotopy-groupoids. They form a locally univalent 2-category, and are interpreted as polynomial endofunctors on the 2-category of groupoids.

Quotient and symmetric containers are two different ways to extend the expressivity of ordinary containers to include symmetries between positions. Our interest lies in understanding how these two approaches are related. To this end, we introduce an intermediate notion: *action containers*. An action container consists of a set of shapes and, for each shape s, a set of positions P_s and a group G_s acting on P_s. On one side, such containers generalize quotient containers, as the allowed permutations are determined by the action of an arbitrary group, and are not restricted to subgroups of the symmetric groups. On the other, they are a special case of symmetric containers: a G_s-action is a functor from G_s (as a 1-object groupoid) to Set, and summation of these functors over all shapes s yields a symmetric container.

We describe a notion of morphisms of action containers, inspired by (pre)morphisms of quotient containers. Differently from the latter, morphisms of action containers have to explicitly preserve the structure of the groups acting on positions. Action containers form a category, and we show that this category is the free coproduct completion of a category of group actions in the form of the Fam-construction. From this we derive closure under arbitrary products and coproducts, as well as exponentials with constant domain.

To compare action containers with the 2-category of symmetric containers, we define a notion of invertible 2-cell between these morphisms, enriching them to a 2-category. Crucially, we observe that this 2-category can again be modularly constructed starting from a 2-category of groups, group homomorphisms and *conjugators* [12] using the techniques of displayed bicategories [6]: we first define a 2-category of group actions, displayed over this 2-category of groups, then repeat a 2-categorical version of the Fam-construction, presenting the 2-category of action containers as that of families of group actions.

We construct a 2-functor between the 2-categories of action containers and symmetric containers; again in multiple steps. First we notice that the delooping of a group extends to a 2-functor $\mathbf{B} : \mathsf{Group} \to \mathsf{hGpd}$ between the 2-categories of groups and groupoids, and that this is locally a weak equivalence. We show, again using the displayed machinery, how to lift this to a local weak equivalence $\bar{\mathbf{B}} : \mathsf{Action} \to \mathsf{SetBundle}$ between the 2-categories of group actions and set bundles.

The Fam-construction yields a 2-functor $\mathrm{Fam}(\bar{\mathbf{B}}) : \mathrm{Fam}(\mathsf{Action}) \to \mathrm{Fam}(\mathsf{SetBundle})$ between the 2-categories of families of group actions, i.e. action containers, and families of set bundles. We show that the action of Fam preserves local fully-faithfullness, but that preservation of local essential surjectivity requires an application of the axiom of choice. Finally, we describe a 2-functor $\Sigma : \mathrm{Fam}(\mathsf{SetBundle}) \to \mathsf{SetBundle}$ performing *summations* of family of set bundles, implicitly employing the universal property of the Fam-construction as free coproduct completion. The latter does not seem to be fully faithful, however its restriction to set bundles with connected bases is. This implies that the composite 2-functor $\Sigma \circ \mathrm{Fam}(\bar{\mathbf{B}}) : \mathrm{Fam}(\mathsf{Action}) \to \mathsf{SetBundle}$ is locally fully faithful. This means that, not only do action containers correspond to certain symmetric containers, but so do their morphisms: conjugators between action container morphisms represent exactly identifications of symmetric container morphisms.

We implement our results using the Agda proof assistant, building on top of the `agda/cubical` [19] and `cubical-categorical-logic` [17] libraries. Our code is freely available at `https://github.com/phijor/cubical-containers`. The repository contains a `README` linking results of this paper to the corresponding formalization in the code.

1.1 Notation and Background

Our work takes place in homotopy type theory, which is a well-suited foundational framework for our investigation. This is mostly due to the fact that we can work with syntethic groupoids, i.e. h-groupoids, in place of categorical groupoids, which considerably simplify the described constructions. We take full advantage of higher inductive types (HITs) to define mere existence via propositional truncation, set quotients and the delooping of groups. In this short section, we recall some basic terminology and fix the notation we use. More details can be found in the HoTT book [20].

We write $\prod_{a:A} B(a)$ for dependent products, $A \to B$ for their non-dependent variant, and $\sum_{a:A} B(a)$ for dependent sums. Two-argument function application is written $f(a, b)$ or, to reduce visual clutter, $f_a(b)$. Most of our constructions are universe-polymorphic, but for the sake of readability in the paper we use only the lowest universe of types, denoted $\mathcal{U}$. The type of h-sets is hSet, the type of h-groupoids is hGpd. We will often simply talk about sets and groupoids instead of h-sets and h-groupoids. We suppress proofs of truncation level when they are routine, e.g. for nested Σ- and Π-types. The type of natural numbers is $\mathbb{N}$ and the finite ordinals $\mathsf{Fin} : \mathbb{N} \to \mathsf{hSet}$. For $x, y : A$, we denote their type of identifications by $x = y$, and call $p : x = y$ either an identification or a path. Given a family B over A and terms $x' : B(x)$, $y' : B(y)$, we write $x' =_{B(p)} y'$ for the type of dependent paths, or $x' =_p y'$ when B can be inferred. Defining equalities are denoted $x := y$; for judgmentally equal x and y, we write $x \doteq y$. For functions into Σ-types, we use binders to name the projections: given $f : X \to \sum_{a:A} B(a)$, we write $\lambda x.\,(a(x), b(x))$ or $\lambda x.\,(a_x, b_x)$ for $a := \mathsf{fst} \circ f$ and $b := \mathsf{snd} \circ f$.

The propositional truncation of a type X is $\|X\|_{\text{-}1}$, the set truncation of X is $\|X\|_0$. Notice that there are two competing conventions for indexing truncation levels: (-2)-based (HoTT-book style) and 0-based (Voevodsky-style). Our formalization, done in Cubical Agda, is 0-based, yet this paper, which is written in HoTT-book style, starts indexing at -2. Whenever possible however, we will explicitly use the words “proposition”, “set” and “groupoid” to avoid confusion. Mere existential quantification is defined as $\exists_{a:A} B(a) := \|\sum_{a:A} B(a)\|_{\text{-}1}$. The set quotient of a type A by a (possibly non-propositional) relation R is A/R. The circle S^1 is a HIT with constructors $\bullet : S^1$ and $\mathsf{loop} : \bullet = \bullet$. Propositional and set truncations, as well as set quotients and the circle, are defined as higher inductive types. The main HIT employed in this paper is the delooping of a group, introduced in Section 2.3.

Given a pointed type (X, x_0), its loop space is $\Omega(X, x_0) := (x_0 = x_0)$, and its fundamental group is its set truncation $\pi_1(X, x_0) := \|\Omega(X, x_0)\|_0$. The connected components of a type X are collected in its set truncation $\pi_0(X) := \|X\|_0$. We say that X is connected if $\|X\|_0$ is contractible.

For groups G and H we denote the type of group homomorphisms by $G \dot{\to} H$. We denote the type of subgroup inclusions by $G \leq H := \sum(\iota : G \dot{\to} H).\, \mathsf{isInjective}(\iota)$. The symmetric group of a set X is $\mathfrak{S}(X) := \Omega(\mathsf{hSet}, X)$. The unit of this group is reflexivity refl, multiplication is composition of paths $p \cdot q$, inverse is path reversal. Recall that, by univalence, $\mathfrak{S}(X)$ is equivalent to the type of equivalences $X \simeq X$. We abbreviate $\mathfrak{S}(n) := \mathfrak{S}(\mathsf{Fin}(n))$. An action of G on a set X is a group homomorphism $\sigma : G \dot{\to} \mathfrak{S}(X)$. For $g : G$, we denote $\mathsf{transport}(\sigma(g)) : X \to X$ simply by $\sigma(g)$, and apply it to some $x : X$ either as $\sigma(g, x)$ or $\sigma_g(x)$. The action is said to be faithful if σ is injective. The set of σ-orbits is denoted X/G, and defined as the set quotient of X by the orbit relation $x \sim y := \exists g : G.\, x = \sigma_g(y)$.

1.2 2-Categories

In this paper, we make use of higher categories in the form of (2,1)-categories. We follow the definitions of bicategorical concepts of [13], and adapt them to the setting of homotopy type theory: a 2-category C consist of a type of objects $x, y : \mathsf{C}_0$, 1-cells $f, g : \mathsf{C}_1(x, y)$, and 2-cells $r, s : \mathsf{C}_2(f, g)$, with horizontal composition of 1- and 2-cells, and vertical composition of 2-cells, subject to suitable axioms. In particular, all types of 2-cells $\mathsf{C}_2(f, g)$ are assumed to be *h*-sets. Composition of 1-cells is unital and associative up to a chosen identification, not just a 2-cell. All instances of 2-categories considered here are either locally strict (i.e. 1-cells form sets) or locally univalent; such 2-categories always admit a unique coherently strict structure.

If C is understood from context, we write $f, g : x \to y$ for $f, g : \mathsf{C}_1(x, y)$, and $r : f \Rightarrow g$ for $r : \mathsf{C}_2(f, g)$. We compose cells in *diagrammatic* order. Juxtaposition denotes horizontal composition, whereas vertical composition of 2-cells is denoted $r \bullet s$.

Let $f, g : x \to y$. Under vertical composition, 2-cells $\mathsf{C}_2(f, g)$ form the morphisms of an (ordinary) category, called the *local category* at x and y, denoted by its type of objects $\mathsf{C}_1(x, y)$. If a proposition P holds for all local categories of a 2-category, we say that it is *locally* P. A *(2,1)-category* is thus defined to be a locally groupoidal 2-category, that is, one for which 2-cells in each local category are invertible. A 2-category is locally *thin* if $\mathsf{C}_2(f, g)$ is a proposition for each pair of 1-cells $f, g : \mathsf{C}_1(x, y)$, i.e. there is at most one 2-cell from f to g. Any ordinary category $\mathcal{C}$ forms a locally thin 2-category: 2-cells are homotopies of 1-cells, $\mathcal{C}_2(f, g) := (f = g)$.

We use the machinery of *displayed bicategories* [6] to define complex 2-categories from modular gadgets. A *displayed 2-category* D over a base 2-category C consists of a family of objects $\mathsf{D}_0 : \mathsf{C}_0 \to \mathcal{U}$, a family of 1-cells $\mathsf{D}_1 : \mathsf{C}_1(x, y) \to \mathsf{D}_0(x) \to \mathsf{D}_0(y) \to \mathcal{U}$, and a family of 2-cells $\mathsf{D}_2 : \mathsf{C}_2(f, g) \to \mathsf{D}_1(f; \bar{x}, \bar{y}) \to \mathsf{D}_1(g; \bar{x}, \bar{y}) \to \mathcal{U}$, satisfying dependent analogues of the 2-category axioms. If unambiguous, we write $\bar{f} : \bar{x} \to_f \bar{y}$ for $\bar{f} : \mathsf{D}_1(f; \bar{x}, \bar{y})$, as well as $\bar{r} : \bar{f} \Rightarrow_r \bar{g}$ for $\bar{r} : \mathsf{D}_2(r; \bar{f}, \bar{g})$. The *total 2-category* of D over C is denoted $\int \mathsf{D}$, and is a 2-categorical analouge of $\sum$-types: objects are pairs of objects $\int_0 \mathsf{D} := \sum_{x : \mathsf{C}_0} \mathsf{D}_0(x)$, 1-cells are $\int_1 \mathsf{D}\big((x, \bar{x}), (y, \bar{y})\big) := \sum_{f : \mathsf{C}_1(x,y)} \mathsf{D}_1(f; \bar{y}, \bar{x})$, and 2-cells $\int_2 \mathsf{D}\big((f, \bar{f}), (g, \bar{g})\big) := \sum_{r : \mathsf{C}_2(f,g)} \mathsf{D}_2(r; \bar{f}, \bar{g})$. To highlight the dependency on C, we sometimes write $\int_{\mathsf{C}} \mathsf{D}$ or even $\int_{x : \mathsf{C}} \mathsf{D}(x)$.

We go between 2-categories via 2-functors. For a 2-functor $F : \mathsf{C} \to \mathsf{D}$ we denote its action on objects, 1-, and 2-cells by F_0, F_1 and F_2 respectively. They are assumed to strictly preserve composition and units of 1-cells, that is up to an identification of 1-cells in the codomain. The actions on 1- and 2-cells define functors of local categories, and we call F *locally* P if all local functors satisfy a proposition P.

We define a notion of *displayed 2-functor* $\bar{F} : \bar{\mathsf{C}} \to_F \bar{\mathsf{D}}$ between 2-categories displayed over C and D and a (base) 2-functor $F : \mathsf{C} \to \mathsf{D}$. To cells in $\bar{\mathsf{C}}$ it assigns cells in $\bar{\mathsf{D}}$ displayed over the image of F: it consists of assignments of objects $\bar{F}_0 : \bar{\mathsf{C}}_0(x) \to \bar{\mathsf{D}}_0(F_0(x))$, 1-cells

$\bar{F}_1 : \bar{\mathsf{C}}_1(f;\bar{x},\bar{y}) \to \bar{\mathsf{D}}_1(F_1(f);\bar{F}_0(\bar{x}),\bar{F}_0(\bar{y}))$ and 2-cells $\bar{F}_2 : \bar{\mathsf{C}}_2(r;\bar{f},\bar{g}) \to \bar{\mathsf{D}}_2(F_2 r;\bar{F}_1\bar{f},\bar{F}_1\bar{g})$, satisfying dependent analogues of the 2-functor axioms. A displayed 2-functor induces a 2-functor between total 2-categories, denoted $\int\bar{F} : \int \mathsf{C} \to \int \mathsf{D}$.

The 2-category of h-groupoids is denoted by hGpd. Its 1-cells are functions between the underlying types and 2-cells are homotopies between functions.

2 Quotient and Symmetric Containers

In this section we recall the notions of quotient and symmetric containers, as well as their morphisms.

2.1 Quotient Containers

Quotient containers were introduced by Abbott et al. [3] as a way to add symmetries to containers in an extensional type theory with quotient types. Here we state their definition in HoTT.

▶ **Definition 1.** *A* quotient container $(S \triangleright P/G)$ *consists of a set of* shapes S*, a family of* positions $P : S \to \mathsf{hSet}$ *and* symmetry groups $\iota_s : G_s \leq \mathfrak{S}(P_s)$ *for each* $s : S$*.*

Each group element $g : G_s$ defines a path $\iota_s(g) : P_s = P_s$. By transport, this induces a map $P_s \to P_s$; in the remainder, we will identify g and this map.

▶ **Example 2.** The quotient container of *unordered n-tuples* $\mathsf{U}_n := (1 \triangleright \mathsf{Fin}(n)/\mathfrak{S}(n))$ has a single shape, and over it positions $\mathsf{Fin}(n)$. On these n positions, the full group of permutations $\mathfrak{S}(\mathsf{Fin}(n))$ acts as its symmetry group. We call U_1 the *identity container*; it has a single shape, on which the trivial group acts.

Like an ordinary container, a quotient container defines an endofunctor on the category of sets, called its *extension*. Whereas for ordinary containers this is a polynomial functor, for quotient containers it is a sum of quotients of representables:

▶ **Definition 3** (extension of quotient containers)**.** *The extension of* $(S \triangleright P/G)$ *is the map* $[\![S \triangleright P/G]\!]_/ : \mathsf{hSet} \to \mathsf{hSet}$ *given by*

$$[\![S \triangleright P/G]\!]_/(X) := \sum_{s:S} \frac{P_s \to X}{\sim_s}, \qquad\qquad v \sim_s w := \exists_{g:G_s}\, v = w \circ g \qquad ⌟$$

When positions are finite sets, the extension is an *analytic functor* in the sense of Joyal [15].

▶ **Example 4** (unordered n-tuples)**.** The extension of U_n is the type of unordered n-tuples:

$$[\![\mathsf{U}_n]\!]_/(X) := \sum_{\_:1} (\mathsf{Fin}(n) \to X)/\!\sim\; = X^n/\!\sim_{\mathfrak{S}(n)}$$

where $x \sim_{\mathfrak{S}(n)} y$ if and only if $x_i = y_{\sigma(i)}$ for some permutation $\sigma : \mathfrak{S}(n)$. When $n = 1$, we obtain the identity function $[\![\mathsf{U}_1]\!]_/ X = X$.

▶ **Definition 5.** *A* premorphism *of quotient containers,* $(u \triangleright f) : (S \triangleright P/G) \rightharpoonup (T \triangleright Q/H)$ *consists of a map of shapes* $u : S \to T$*, a map of positions* $f : \prod_{s:S} Q_{us} \to P_s$*, and a proof that* f preserves symmetries*,* $\prod_{s:S}\prod_{g:G_s}\exists_{h:H_{us}}\, g \circ f_s = f_s \circ h$*.*

Morphisms of quotient containers are defined up to permutation of positions, i.e. as equivalence classes of premorphisms.

▶ **Definition 6.** *The type of* morphisms $F \to G$ *of quotient containers is the set quotient of the type of premorphisms $F \rightharpoonup G$ by the relation (defined by path induction)*

$$(u \triangleright f) \approx (u' \triangleright f') := \sum_{p:u=u'} f \approx'_p f', \qquad f \approx'_{\mathsf{refl}_u} f' := \prod_{s:S} \exists_{h:H_{us}} f_s = f'_s \circ h \qquad ⌟$$

Whenever $u \doteq u'$, this relation posits the mere existence of a triangle filler

$$\begin{tikzcd} & Q_{us} \arrow[rr, dashed, "\exists(h:H_{us})"] \arrow[dr, "f_s"'] & & Q_{us} \arrow[dl, "f'_s"] \\ (u \triangleright f) \approx (u \triangleright f') \quad \simeq & & P_s & \end{tikzcd}$$

▶ **Definition 7.** *Quotient containers and their morphisms form a category* $\mathsf{QuotCont}$.

Extension of quotient containers is a functor $[\![-]\!]_/ : \mathsf{QuotCont} \to \mathsf{Endo}(\mathsf{hSet})$, which is full and faithful. Each premorphism $(u \triangleright f) : F \rightharpoonup G$ defines a natural transformation $[\![u \triangleright f]\!]_/ : [\![F]\!]_/ \Rightarrow [\![G]\!]_/$, with component at $X : \mathsf{hSet}$ a map

$$[\![u \triangleright f]\!]_/^X : \sum_{s:S}(P_s \to X/{\sim_s}) \to \sum_{t:T}(Q_t \to X/{\sim_t})$$

defined by induction on set quotients as $[\![u \triangleright f]\!]_/^X(s, [\ell]) := (us, [\ell \circ f])$. This is well-defined on morphisms of quotient containers: If $(u \triangleright f) \approx (u' \triangleright f')$ then $[\![u \triangleright f]\!]_/^X = [\![u' \triangleright f']\!]_/^X$.

2.2 Symmetric Containers

Symmetric containers were introduced by Gylterud [10] as a way of defining polynomial functors between categorical groupoids. In this section we reformulate their definition in the language of HoTT using homotopy groupoids instead.

▶ **Definition 8.** *A* symmetric container $(S \triangleleft P)$ *consists of shapes $S : \mathsf{hGpd}$ and a family of positions $P : S \to \mathsf{hSet}$.*

▶ **Definition 9.** *A morphism of symmetric containers $(u \triangleleft f) : (S \triangleleft P) \to (T \triangleleft Q)$ consists of a map of shapes $u : S \to T$ and a family $f : \prod_{s:S} Q_{us} \to P_s$ of maps of positions.*

In a (homotopy) type-theoretic setting, the types of morphisms $\mathsf{C}(x, y)$ of a category C are understood to be h-sets. Morphisms of symmetric containers, however, form h-groupoids.[1]

▶ **Definition 10.** *The 2-category* $\mathsf{SymmCont}$ *has as objects symmetric containers, as 1-cells morphisms of symmetric containers, and as 2-cells identifications of such morphisms.*

▶ **Definition 11.** *The extension of a symmetric container $(S \triangleleft P)$ is a function of h-groupoids, $[\![S \triangleleft P]\!] : \mathsf{hGpd} \to \mathsf{hGpd}$, defined as $[\![S \triangleleft P]\!](X) := \sum_{s:S} P_s \to X$.*

Extension of symmetric containers defines a 2-functor $[\![-]\!] : \mathsf{Symm} \to \mathsf{Endo}(\mathsf{hGpd})$. For any morphism $(u \triangleleft f) : (S \triangleleft P) \to (T \triangleleft Q)$, there is a pseudonatural transformation $[\![u \triangleleft f]\!] : [\![S \triangleleft P]\!] \Rightarrow [\![T \triangleleft Q]\!]$ with components given by precomposition

$$[\![u \triangleleft f]\!]_X : \sum_{s:S}(P_s \to X) \to \sum_{t:T}(Q_t \to X) \qquad [\![u \triangleleft f]\!]_X(s, v) := (us, Q_{us} \xrightarrow{f_s} P_s \xrightarrow{v} X)$$

[1] Observe that $(S \triangleleft 0) \to (T \triangleleft 0) \simeq (S \to T)$, which is an h-groupoid since T is.

This 2-functor is locally an equivalence of 2-categories [10, Theorem 2.2.1], thus embeds symmetric containers into endofunctors of groupoids.

One advantage of internalizing symmetric containers as h-groupoids is that we are free to define groupoids of shapes as higher inductive types, encoding the desired symmetries directly in their constructors. For example, cyclic lists can be described using the symmetries of the circle, S^1:

▶ **Example 12.** The symmetric container of *cyclic lists*, Cyc, is defined as follows:

- Shapes are pairs $\mathbb{N} \times S^1$. The first component contains the length of the list. The second has a single point, $\mathsf{base} : S^1$, but its loops $\mathsf{base} = \mathsf{base}$ are going to induce cyclic shifts on positions.
- Positions $\mathsf{Sh} : \mathbb{N} \times S^1 \to \mathsf{hSet}$ are defined by induction on the circle S^1. Over the point, we have n distinct positions, $\mathsf{Sh}(n, \mathsf{base}) := \mathsf{Fin}(n)$. On the non-trivial path, Sh identifies positions by a cyclic shift, $\mathsf{Sh}(\mathsf{refl}_n, \mathsf{loop}) := \mathsf{shift}$. Here, $\mathsf{shift} : \mathsf{Fin}(n) = \mathsf{Fin}(n)$ is the path obtained by univalence from the *successor* equivalence

$$\begin{aligned} \mathsf{suc} &: \mathsf{Fin}(n) \simeq \mathsf{Fin}(n) \\ \mathsf{suc}(k) &:= (k+1) \mod n \end{aligned}$$

Since S^1 is freely generated by a single non-trivial path, Sh identifies positions by arbitrary cyclic shifts. Let $v := (x, y, z)$, $w := (y, z, x)$ terms of type $\mathsf{Fin}(3) \to X$. We are going to exhibit a path $\big((3, \mathsf{base}), v\big) = \big((3, \mathsf{base}), w\big)$ in $[\![\mathsf{Cyc}]\!](X)$. Since $[\![\mathsf{Cyc}]\!](X)$ is an iterated Σ-type, such a path is given by a triple of paths $p : 3 = 3$, $q : \mathsf{base} = \mathsf{base}$, and $r : v =_q w$. Set $p := \mathsf{refl}$ and $q := \mathsf{loop} \cdot \mathsf{loop}$. The path r is dependent over q, and it suffices to give a path $v = w \circ \mathsf{shift} \circ \mathsf{shift}$. But we see that the right side computes to (x, y, z), which is v.

2.3 Lifting Quotient Containers

The data of both quotient and symmetric containers define semantics for datatypes with symmetries. As we will see, it is possible to see any quotient container as a symmetric one. However, this analogy does not extend to (polymorphic) functions between such types, since it is generally not possible to lift a morphism of quotient containers to one of symmetric containers, as the former truncate evidence on how symmetries are preserved.

In order to create a symmetric container from a quotient container, we have to come up with a *groupoid* of shapes that encodes the symmetries present in the quotient container. We borrow an idea from algebraic topology: any group G gives rise to a unique pointed, connected groupoid $(\mathbf{B}G, \bullet)$ such that $\Omega(\mathbf{B}G, \bullet) \simeq G$, called its *delooping*. This type is an instance of an Eilenberg–MacLane space, i.e. a type with a single non-trivial homotopy group. Eilenberg–MacLane spaces have been studied in HoTT [16], and our presentation of $\mathbf{B}G$ coincides with $K(G, 1)$ there. We define $\mathbf{B}G$ as a *higher inductive type* with constructors

$$\frac{}{\bullet : \mathbf{B}G} \qquad \frac{g : G}{\mathsf{loop}\, g : \bullet = \bullet} \qquad \frac{g, h : G}{\mathsf{loop\text{-}comp}(g, h) : \mathsf{loop}\, g \cdot \mathsf{loop}\, h = \mathsf{loop}\, gh}$$

plus one constructor asserting that $\mathbf{B}G$ is an h-groupoid. Its *recursion principle* states that, to define a map $\mathbf{B}G \to X$ for some groupoid X, it suffices to give a point $x_0 : X$ and a group homomorphism $\varphi : G \xrightarrow{\cdot} \Omega\,(X, x_0)$: a map $f : \mathbf{B}G \to X$ is then determined by $f(\bullet) := x_0$ and $f(\mathsf{loop}\, g) := \varphi(g)$. The recursor characterizes functions out of $\mathbf{B}G$, in the following sense:

▶ **Proposition 13.** *The recursor is an equivalence* $(\sum_{x_0 : X} G \xrightarrow{\cdot} \Omega\,(X, x_0)) \simeq (\mathbf{B}G \to X)$*, for* X *a groupoid. When* X *is a set, we have* $(\sum_{x_0 : X} G \to x_0 = x_0) \simeq (\mathbf{B}G \to X)$*.*

The *dependent eliminator* lets us define sections $\prod_{x:\mathbf{B}G} B(x)$ of families $B : \mathbf{B}G \to \mathsf{hGpd}$ by providing a point $b_0 : B(\bullet)$, dependent paths $\varphi : \prod_{g:G}(b_0 =_{B(\mathsf{loop}\, g)} b_0)$, and a coherence condition for composition of dependent paths: for all $g, h : G$, there needs to be a path from $\varphi(g) \cdot \varphi(h)$ to $\varphi(gh)$, dependent over $\mathsf{loop\text{-}comp}(g, h)$.

Note that $\mathsf{loop\text{-}comp} : G \to \Omega(\mathbf{B}G, \bullet)$ preserves the product of G, hence is a morphism of groups. This lets us derive other expected coherences, such as $\mathsf{loop}\, 1_G = \mathsf{refl}$ and $\mathsf{loop}\,(g^{-1}) = (\mathsf{loop}\, g)^{-1}$.

Delooping acts on group homomorphisms:

▶ **Definition 14.** *Any group homomorphism $\varphi : G \to H$ induces a map of groupoids $\mathbf{B}\varphi : \mathbf{B}G \to \mathbf{B}H$, defined by induction:*

$$\mathbf{B}\varphi(\bullet) := \bullet, \qquad \mathbf{B}\varphi(\mathsf{loop}\, g) := \mathsf{loop}\, \varphi(g)$$

A G-action is a particular homomorphism, so the above defines a type family $\mathbf{B}G \to \mathsf{hSet}$. Let us spell this out:

▶ **Definition 15** (associated bundle). *Let G act on a set X via $\sigma : G \dashrightarrow \mathfrak{S}(X)$. Its* associated bundle *$\bar{\mathbf{B}}\sigma : \mathbf{B}G \to \mathsf{hSet}$ is defined by recursion on $\mathbf{B}G$ as*

$$\bar{\mathbf{B}}\sigma(\bullet) := X, \qquad \bar{\mathbf{B}}\sigma(\mathsf{loop}\, g) := \sigma(g),$$

Note that for each $g : G$, $\sigma(g)$ is a path $X = X$.

In the context of quotient containers, we are dealing with *faithful* group actions, that is actions of G on X such that $\sigma : G \to \mathfrak{S}(X)$ is an embedding. In this case, the associated bundle is an embedding on its path spaces, i.e. a set-truncated function [20, 7.6.1]:

▶ **Proposition 16.** *If $\sigma : G \hookrightarrow \mathfrak{S}(X)$ acts* faithfully*, the fibers of $\bar{\mathbf{B}}\sigma : \mathbf{B}G \to \mathsf{hSet}$ are sets.*

Proof. By [20, Lemma 7.6.2], $\bar{\mathbf{B}}\sigma$ has set-valued fibers iff $\mathsf{cong}\, \bar{\mathbf{B}}\sigma : x = y \to \bar{\mathbf{B}}\sigma(x) = \bar{\mathbf{B}}\sigma(y)$ is an embedding for all $x, y : \mathbf{B}G$. This is a proposition, therefore it suffices to show this at $x \doteq y \doteq \bullet$. There, we have $\mathsf{cong}(\bar{\mathbf{B}}\sigma) \circ \mathsf{loop} \doteq \sigma$. But $\mathsf{loop} : G \to \bullet = \bullet$ is an equivalence and σ an embedding, hence $\mathsf{cong}\, \bar{\mathbf{B}}\sigma$ is an embedding as well. ◀

For any quotient container we define a groupoid that is the collection of its delooped symmetry groups:

▶ **Definition 17.** *The* delooping *of a quotient container $(S \triangleright P/G)$ is the symmetric container $\mathbf{B}(S \triangleright P/G) := (\mathbf{S} \triangleleft \mathbf{P})$ consisting of*

- *shapes $\mathbf{S} := \sum_{s:S} \mathbf{B}G_s$, and*
- *positions $\mathbf{P} : \sum_{s:S} \mathbf{B}G_s \to \mathsf{hSet}$, $\mathbf{P}(s, -) := \bar{\mathbf{B}}(\iota_s)$,*

where ι_s is the inclusion of symmetry groups $G_s \hookrightarrow \mathfrak{S}(X)$.

We think of the shapes $\mathbf{S}$ as consisting of the points of S, with loops given by elements in G_s freely added. Indeed, if we compute its connected components, we see that

$$\pi_0(\mathbf{S}) \simeq \|\textstyle\sum_{s:S} \mathbf{B}G_s\|_0 \simeq \sum_{s:S} \|\mathbf{B}G_s\|_0 \simeq S$$

where the last step follows from connectivity of $\mathbf{B}G_s$. Similarly, we compute its first fundamental group: S is a set and $s = s$ is contractible, thus

$$\pi_1(\mathbf{S}, (s, x)) \simeq \textstyle\sum_{p:s=s}(x =_p x) \simeq (x = x) \simeq \pi_1(\mathbf{B}G_s, x) \simeq G_s$$

That each G_s acts faithfully is reflected in the action of the groupoid $\mathbf{S}$, in the sense that the bundle $\mathbf{P}$ embeds paths of $\mathbf{S}$:

▶ **Proposition 18.** *The family* $\mathbf{P} : \mathbf{S} \to \mathsf{hSet}$ *is a set-truncated function.*

Proof. For each $X : \mathsf{hSet}$, we have $\mathsf{fiber}_{\mathbf{P}}\, X \simeq \sum_{s:S} \mathsf{fiber}_{\bar{\mathbf{B}}\iota_s}\, X$, which is a set: S is a set, and since we assume ι_s to be faithful, so are the fibers of $\bar{\mathbf{B}}\iota_s$ by Proposition 16. ◀

This way of obtaining a symmetric container is in some sense conservative: when comparing the associated extensions (and set-truncating that of the symmetric container), we see that they are the same function of sets:

▶ **Theorem 19.** *For a quotient container* Q *and* $X : \mathsf{hSet}$*, there is an equivalence of sets*

$$\| \llbracket \mathbf{B}Q \rrbracket(X) \|_0 \simeq \llbracket Q \rrbracket_/(X)$$

Proof. Let us unfold the definitions of $\llbracket - \rrbracket$ and $\mathbf{B}$,

$$\| \llbracket \mathbf{B}Q \rrbracket(X) \|_0 \simeq \left\| \textstyle\sum_{s:S} \sum_{x:\mathbf{B}G_s} \bar{\mathbf{B}}\iota_s(x) \to X \right\|_0 \tag{1}$$

and, as S is a set, move the truncation under the sum

$$\simeq \textstyle\sum_{s:S} \left\| \sum_{x:\mathbf{B}G_s} \bar{\mathbf{B}}\iota_s(x) \to X \right\|_0 \tag{2}$$

Notice that G_s acts on the function type $P_s \to X$ via precomposition, and that its associated bundle is $(\bar{\mathbf{B}}\iota_s(-) \to X) : \mathbf{B}G_s \to \mathsf{hSet}$. By Lemma 20 below, the connected components of this bundle correspond to orbits of the action,

$$\simeq \sum_{s:S} (P_s \to X)/G_s \tag{3}$$

which is exactly how extension of a quotient container is defined:

$$\simeq \sum_{s:S} (P_s \to X)/\!\sim_s \simeq \llbracket Q \rrbracket_/(X)$$ ◀

▶ **Lemma 20.** *Let* σ *a* G*-action on* X*. The connected components of the total space of its associated bundle and* σ*-orbits are in bijection, that is* $\| \sum_{x:\mathbf{BG}} \bar{\mathbf{B}}\sigma(x) \|_0 \simeq X/G$*.*

Proof. Let us define the left-to-right direction. Since the codomain is a set, it suffices to give $f : \prod_{x:\mathbf{BG}} \bar{\mathbf{B}}\sigma(x) \to X/G$ by induction on $\mathbf{B}G$. Let $f(\bullet) := [-] : X \to X/G$ the surjection onto the quotient. It remains to show that this is well-defined on loops, which reduces to $\prod_{g:G} \prod_{x:X} [x] = [\sigma_g(x)]$. This holds since $x \sim \sigma_g(x)$ by definition of the orbit relation. The inverse is defined by recursion on the set quotient X/G and maps $x : X$ to $(\bullet, x) : \sum_{x:\mathbf{BG}} \bar{\mathbf{B}}\sigma(x)$. From $p : x = \sigma_g(y)$, one constructs a path $(\bullet, x) = (\bullet, y)$: the first component is given by $\mathsf{loop}\, g$, the second as dependent path from p. ◀

Theorem 19 states that, as functions between types, the diagram

$$\begin{tikzcd}
\mathsf{SymmCont} \arrow[r, "\llbracket - \rrbracket"] & (\mathsf{hGpd} \to \mathsf{hGpd}) \arrow[d, "\lambda F.\|F(-)\|_0"] \\
\mathsf{QuotCont} \arrow[u, "\mathbf{B}"] \arrow[r, "\llbracket - \rrbracket_/"'] & (\mathsf{hSet} \to \mathsf{hSet})
\end{tikzcd}$$

commutes. We are interested to see whether this generalizes to a natural isomorphism of functors. To do so, we would have to suitably extend $\mathbf{B}$ to a functor. Unfortunately, it is not clear how to define its action on morphisms of quotient containers. Given a premorphism $(u \triangleright f) : (S \triangleright P/G) \rightharpoonup (T \triangleright Q/H)$, we would have to provide a morphism of shapes

$$\begin{aligned} &\textstyle\sum_{s:S} \mathbf{B}G_s \to \sum_{t:T} \mathbf{B}H_t \\ &\qquad (s, x) \mapsto (us, ?) \end{aligned}$$

To define $? : \mathbf{B}G_s \to \mathbf{B}H_{us}$, it would suffice to provide a morphism of groups $G_s \mathrel{\dot{\to}} H_{us}$. However, we are not given this information: we know that f preserves symmetries, but this only tells us that, for each $g : G_s$, there is *merely* some $h : H_{us}$. Even if we were given an explicit function $G_s \to H_{us}$, it would not have to be a group homomorphism. In fact, it is easy to construct counterexamples:

▶ **Example 21.** Consider $(\mathsf{id} \triangleright !) : \mathsf{U}_1 \rightharpoonup \mathsf{U}_2$. The terminal map $! : 2 \to 1$ trivially preserves symmetries: the diagram

$$\begin{tikzcd} 2 \arrow[r, "!"] \arrow[d, "\varphi_g"'] & 1 \arrow[d, "g"] \\ 2 \arrow[r, "!"'] & 1 \end{tikzcd}$$

commutes for any choice of φ_g, in particular for $\varphi : \mathfrak{S}(1) \to \mathfrak{S}(2), \varphi_{\_} := \mathsf{swap}$, which is *not* a group homomorphism.

Since morphisms of quotient containers are equivalence classes, it might be possible to find another premorphism in the same class for which this assignment is a morphism of groups. In fact, in the above example one could pick $\varphi_g := \mathsf{id}$, which clearly is. Doing so however for *arbitrary* symmetry groups seems constructively impossible, without invoking some form of choice principle.

Instead, we will be looking to enhance the definition of quotient containers to include the necessary information, and investigate their relation to symmetric containers more closely.

3 Action Containers

In this section we define *action containers* and assemble them into a 1-category. Morphisms in this category are akin to premorphisms of quotient containers. In particular, they are not quotiented by a relation on positions. Later, in Section 4, we enrich this category to obtain a (2,1)-category whose 2-cells capture this relation.

Different from quotient containers, the symmetries of action containers are not limited to subgroups of permutations of positions. Instead, an action container has, for each shape, a chosen group *acting* on the set of positions. This lets us flexibly introduce symmetries, e.g. by letting the integers under addition act on a finite set, instead of having to identify the image of this action, see the forthcoming Example 23.

The category of action containers admits a number of limits and colimits, and we will derive the usual container algebra of products and coproducts from a presentation of this category as a category of families in Section 3.1.

▶ **Definition 22.** *An* action container $(S \triangleright P \triangleleft^{\sigma} G)$ *consists of a set of* shapes S*, a family of* positions $P : S \to \mathsf{hSet}$ *and* group actions $\sigma_s : G_s \to \mathfrak{S}(P_s)$ *for each* $s : S$*.*

In the following, the word "container" refers to action containers; other kinds of containers are qualified explicitly. In Example 12, we defined cyclic lists as a symmetric container in which loops of the circle act by cyclic shifts. Since $\Omega(S^1) = \mathbb{Z}$, we are inspired to define a container of cyclic lists by means of a $\mathbb{Z}$-action:

▶ **Example 23** (cyclic lists as an action container)**.** The container $\mathsf{Cyc} := (\mathbb{N} \triangleright \mathsf{Fin} \triangleleft^{\sigma} \mathbb{Z})$ has $\mathbb{Z}$ acting on $\mathsf{Fin}(n)$ as follows: for each n, let $\sigma_n : \mathbb{Z} \to \mathfrak{S}(n)$, $\sigma_n(k) := \lambda\ell.\,(\ell + k) \mod n$. Note that this action is *not* faithful: the kernel of σ_n consists of the integers $n\mathbb{Z} = \{\, nk \mid k \in \mathbb{Z} \,\}$.

In general, it is easy to define $\mathbb{Z}$-actions: $\mathbb{Z}$ is the free group on one generator, thus it suffices to define the action of $1 : \mathbb{Z}$. In the example of cyclic lists, it suffices to define the cyclic shift by one position, $\sigma_n(1) := \lambda\ell.\,(\ell + 1) \mod n$. This is impossible for quotient containers with finitely many positions: $\mathbb{Z}$ is simply never a subgroup of finite symmetry groups.

Unlike premorphisms of quotient containers, morphism of action containers are required to preserve the full structure of containers, including their symmetries:

▶ **Definition 24.** *A* morphism *of action containers* $(u \triangleright f \triangleleft \varphi) : (S \triangleright P \triangleleft^{\sigma} G) \to (T \triangleright Q \triangleleft^{\tau} H)$ *consists of a map of shapes* $u : S \to T$*, a map of positions* $f : \prod_{s:S} Q_{us} \to P_s$*, a family of group homomorphisms* $\varphi : \prod_{s:S} G_s \mathrel{\dot\to} H_{us}$*, and a proof that* f *is* equivariant*: for all* $s : S$ *and* $g : G_s$ *a commutative square*

$$\begin{tikzcd} Q_{us} \arrow[r, "f_s"] \arrow[d, "\tau_{us}(\varphi_s(g))"'] & P_s \arrow[d, "\sigma_s(g)"] \\ Q_{us} \arrow[r, "f_s"'] & P_s \end{tikzcd}$$

Calling f equivariant is justified: each f_s is a morphism between G_s-sets (P_s, σ_s) and $(Q_{us}, \tau_{us} \circ \varphi_s)$ [18, Definition 1.2]. Theorem 27 will explain how this notion of morphism arises naturally from a category of group actions and equivariant maps between them.

▶ **Definition 25.** *Action containers and their morphisms form a category* $\mathsf{ActCont}$.

3.1 Algebra of Action Containers

Like other categories of containers, action containers are closed under a number of constructions. In particular, $\mathsf{ActCont}$ has all products and coproducts. To show this, we could define each of them by hand. In that process, we would have to carefully track the variance of parts of a container. For example, the binary product of containers is a product of shapes and symmetry groups, but a (pointwise) coproduct of families of positions.

Instead, we opt to present $\mathsf{ActCont}$ as a category of *families of group actions*, from which (co)limits are easy to read off. First, we define a category of group actions. It is a version of the category of G-sets (for a fixed group G), in which equivariant maps are permitted to go between sets with actions of *different groups*.

▶ **Definition 26.** *We denote by* Action *the category of group actions and equivariant maps. It is obtained as the total category of the following category displayed over* $\mathsf{Group} \times \mathsf{hSet}^{\mathsf{op}}$*:*

- *Given objects* $G : \mathsf{Group}_0$ *and* $X : \mathsf{hSet}_0^{\mathsf{op}}$*, displayed objects are* G*-actions on* X*, i.e. group homomorphisms* $\mathsf{Action}_0(G, X) := (G \mathrel{\dot\to} \mathfrak{S}(X))$*.*
- *Displayed morphisms between actions* $\sigma : \mathsf{Action}_0(G, X)$ *and* $\tau : \mathsf{Action}_0(H, Y)$ *over* $(\varphi : G \mathrel{\dot\to} H, f : X \leftarrow Y)$ *are proofs of the proposition "*f *is equivariant over* φ*": Let* $\mathsf{Action}_1((\varphi, f); \sigma, \tau) := \mathsf{isEquivariant}_{\varphi,f}(\sigma, \tau) := \prod_{g:G} \sigma(g) \circ f = f \circ \tau(\varphi g)$*.*

Diagrammatically, equivariant maps compose by horizontal pasting of proofs of equivariance:

$$\begin{tikzcd}
X \arrow[d, "\sigma(g)" description] & Y \arrow[l, "f"'] \arrow[d, "\tau(\varphi g)" description] & {} \arrow[d, phantom, ";"] & Y \arrow[d, "\tau(h)" description] & Z \arrow[l, "e"'] \arrow[d, "\upsilon(\psi h)" description] & {} \arrow[d, phantom, ":="] & X \arrow[d, "\sigma(g)" description] & Z \arrow[l, "fe"'] \arrow[d, "\upsilon(\psi\varphi g)" description] \\
X & Y \arrow[l, "f"] & {} & Y & Z \arrow[l, "e"] & {} & X & Z \arrow[l, "fe"]
\end{tikzcd}$$

Observe how over each shape, the data of a container $(S \triangleright P \triangleleft^{\sigma} G)$ is exactly that of a group action: for any $s : S$, G_s acts on P_s via σ_s. Thus, on objects, the category of action containers consists of "families" of group actions. Let us ensure that this analogy extends to morphisms of this category.

Recall that for any category $\mathcal{C}$, its free coproduct completion is the category of families $\mathrm{Fam}(\mathcal{C})$ [4, §2]. Its objects are families $\sum_{I:\mathsf{hSet}} I \to \mathcal{C}_0$, morphisms are families of maps between them.

▶ **Theorem 27.** *The category of action containers is equivalent to families of group actions. In particular, the functor* $\mathsf{F} : \mathsf{ActCont} \to \mathrm{Fam}(\mathsf{Action})$ *with action on objects given by* $\mathsf{F}(S \triangleright P \triangleleft^{\sigma} G) := (S, \lambda s.\, (G_s, P_s, \sigma_s))$ *is an equivalence of categories.*

▶ Remark (univalence of ActCont). The above implies that ActCont is a univalent category: Fam preserves univalence, and Action is a univalent displayed category by [7, Proposition 7.3].

From this presentation of ActCont, we read off closure under sums and products:

▶ **Proposition 28.** Action *has K-indexed products for all sets K. In particular, the trivial group acting on the singleton set is an initial object.*

▶ **Corollary 29.** *Action containers are closed under products and coproducts.*

Proof. $\mathrm{Fam}(\mathcal{C})$ is the free coproduct completion of any category $\mathcal{C}$, thus ActCont is closed under coproducts. Similarly, Action is closed under products (Proposition 28), and Theorem 2.11 of [4] implies that families over it are as well. ◀

Like ordinary containers [2, Proposition 3.9], constant action containers are exponentiable:

▶ **Proposition 30** (constant containers are exponentiable)**.** *The* constant container *of a set K is* $\mathbf{k}K := (K \triangleright 0 \triangleleft^{\mathrm{id}} 1)$*. Given a container* $F = (S \triangleright P \triangleleft^{\sigma} G)$*, the exponential container* $F^K := (S^* \triangleright P^* \triangleleft^{\sigma^*} G^*)$ *is defined to have*

- *shapes* $S^* := K \to S$*,*
- *positions* $P^*_f := \sum_{k:K} P_{fk}$*,*
- *symmetries* $G^*_f := \prod_{k:K} G_{fk}$*, and*
- *actions* $\sigma^*_f : G^*_f \to \mathfrak{S}(P^*_f)$ *given by action of σ on the second component of P^*_f: for* $g : \prod_k G_{fk}$*, let* $\sigma^*_f(g) := \lambda(k,p).\,(k, \sigma_{fk}(g))$

Let $f : K \to S$ and $k : K$. The evaluation morphism $\mathrm{ev} : F^K \times \mathbf{k}K \to F$ *is given by function application $fk : S$ on shapes, pairing* $P_{fk} \to 0 + \sum_k P_{fk}$ *on positions, and the projection homomorphisms* $1 \times \prod_{k'} G_{fk'} \dashrightarrow G_{fk}$ *on symmetries.*

We believe that the above is an instance of constant exponentials in families: Let C have K-fold products for any set K; in particular an initial object 1_{C} and binary products. It should be possible to show that the constant family $(K, \lambda k.\, 1_{\mathsf{C}})$ is exponentiable.

▶ Remark (composition of action containers). When seen as endofunctors, composite data types are constructed as compositions of functors. For ordinary containers [2, 3.6] and symmetric containers [10, 2.2.5], this construction is reflected along $[\![-]\!]$: Given containers F and G, there is a container $F[G]$ such that $[\![F[G]]\!] \cong [\![F]\!] \circ [\![G]\!]$, upgrading $[\![-]\!]$ to a strong monoidal functor. In a set-theoretic setting, Hasegawa [11, Corollary 1.18] gives a construction of composition for analytic functors in terms of wreath products of symmetry groups. Analytic functors are presented by quotient containers with *finitary* positions, and such a restriction of positions to a subuniverse of sets seems necessary when porting the result to action containers: positions and symmetries of the composite can otherwise not be defined coherently by induction on set quotients. This indicates to us that the definition of a composition-monoidal structure of action containers is not entirely straightforward, and we postpone its study for now.

4 The 2-Category of Action Containers

In Section 2.3 we observed that quotient containers lift to symmetric containers, but that the same does not apply to their morphisms. We defined the category of action containers to include the missing data, and are now ready to define an appropriate lifting:

▶ **Proposition 31.** *The* delooping *of a container* $(S \triangleright P \triangleleft^{\sigma} G)$ *is the symmetric container* $\mathbf{B}(S \triangleright P \triangleleft^{\sigma} G) := (\sum_{s:S} \mathbf{B}G_s \triangleleft \bar{\mathbf{B}}\sigma_s)$*. Each morphism* $(u \triangleright f \triangleleft \varphi) : (S \triangleright P \triangleleft^{\sigma} G) \to (T \triangleright Q \triangleleft^{\tau} H)$ *defines a morphism between deloopings,* $(\bar{\varphi} \triangleleft \bar{f}) : \mathbf{B}(S \triangleright P \triangleleft^{\sigma} G) \to \mathbf{B}(T \triangleright Q \triangleleft^{\tau} H)$*.*

Proof. The family φ yields a map of shapes of type $\bar{\varphi} : \sum_{s:S} \mathbf{B}G_s \to \sum_{t:T} \mathbf{B}H_t$, defined as $\bar{\varphi}(s, x) := (us, \mathbf{B}\varphi_s(x))$. The map on positions has (uncurried) type

$$\bar{f} : \prod_{s:S} \prod_{x:\mathbf{B}G_s} \bar{\mathbf{B}}\tau_{us}(\mathbf{B}\varphi(x)) \to \bar{\mathbf{B}}\sigma_s(x)$$

and is defined by induction on $x : \mathbf{B}G_s$. On the point, let $\bar{f}(s, \bullet) := f_s : Q_{us} \to P_s$. It remains to show that $\bar{f}$ is well-defined on loops in $\mathbf{B}G_s$. For all g, we have to provide a dependent path $f_s =_{F(\mathsf{loop}\, g)} f_s$ where $F(x) = \bar{\mathbf{B}}\tau_{us}(\mathbf{B}\varphi(x)) \to \bar{\mathbf{B}}\sigma_s(x)$. By [20, Lemma 2.9.6], this is equivalent to giving paths $\prod_{q:Q_{us}} \sigma_s(f_s(q)) = f_s((\tau_{us}\varphi_s\, g)\, q)$, which we obtain from the proof that f_s is equivariant. ◀

As noted in Section 2.2, symmetric containers do not form an ordinary category, but a 2-category. Thus, in order to show that the above construction is functorial, we must first enrich action containers by a type of 2-cells, defining a 2-category. We do so by pulling back 2-cells of symmetric containers, and will see that this corresponds to the quotiented sets of quotient container morphisms.

We split the construction of the 2-functor taking action containers into symmetric containers into smaller steps. As in Section 3, we first seek to understand the problem for a single action, before considering entire families of such. We observe that symmetric containers, from a homotopical viewpoint, are *set-bundles over groupoids*: both consist of some *base* B : hGpd together with a family of *fibers* $F : B \to$ hSet. Previously, we have seen that each action defines such a bundle, namely its *associated bundle* (Definition 15). We define 2-categories of actions (Action, Definition 37) and set bundles (Definition 38), and show that taking the associated bundle is a weak equivalence of their local categories (Theorem 42). This means that maps of actions are in 1-to-1 correspondence with functions on their associated bundles. By changing our point of view, and seeing actions as single-shape containers and bundles as symmetric containers, this fully classifies morphisms of (single-shape) action containers in terms of morphisms of symmetric containers.

To understand the case of many-shape containers, we give an analogue of the Fam-construction in 2-categories, and define the 2-category of action containers as $\mathsf{ActCont} :=$ Fam(Action). The objects and 1-cells of this category are exactly as in Definition 26. We unfold the type of 2-cells induced by this construction (Proposition 47) and show that it is closely related to the quotient on premorphisms of quotient containers (Definition 6). We observe that set-bundles are, in a suitable sense, closed under Σ-types, and we lift the functor $\bar{\mathbf{B}} : \mathsf{Action} \to \mathsf{SetBundle}$ to

$$\mathsf{ActCont} \xrightarrow{=} \mathrm{Fam}(\mathsf{Action}) \xrightarrow{\mathrm{Fam}(\bar{\mathbf{B}})} \mathrm{Fam}(\mathsf{SetBundle}) \xrightarrow{\Sigma} \mathsf{SetBundle} \xrightarrow{=} \mathsf{SymmCont}$$

finally establishing the connection between action containers and symmetric containers.

4.1 A 2-Category of Groups

We think of the type of h-groupoids, hGpd, as an internal notion of categorical groupoids: The 2-category hGpd has as objects h-groupoids, as morphisms functions of such, and as 2-cells homotopies between them. Our goal is to extend the delooping to a 2-functor taking groups into hGpd in a way that characterizes 2-cells in hGpd. We thus equip the 1-category of groups with the structure of a (2,1)-category [12]:

▶ **Definition 32.** *The category* Group *of groups and group homomorphisms forms a (2,1)-category if equipped with the following 2-cells: Let* $\varphi, \psi : \mathsf{Group}_1(G, H)$. *We say that* $r : H$ *is a* conjugator *of* φ *and* ψ *if*

$$\mathsf{isConjugator}_{\varphi,\psi}(r) := \prod_{g:G} \varphi(g) \cdot r = r \cdot \psi(g)$$

The 2-cells $\mathsf{Group}_2(\varphi, \psi) := \sum_{r:H} \mathsf{isConjugator}_{\varphi,\psi}(r)$ *compose vertically by multiplication in* H. *The horizontal composites of* $r : \mathsf{Group}_2(\varphi, \psi)$ *and* $s : \mathsf{Group}_2(\varphi', \psi')$ *is* $s \cdot \psi'(r)$.

Note that Group is not locally univalent: the identity type of group homomorphisms, $\varphi = \psi$, is a proposition, but the type of conjugators $\mathsf{Group}_2(\varphi, \psi)$ is a set.

▶ **Lemma 33.** *Delooping extends to a 2-functor* $\mathbf{B} : \mathsf{Group} \to \mathsf{hGpd}$.

Proof. A 1-cell $\varphi : \mathsf{Group}_1(G, H)$ is sent to $\mathbf{B}\varphi : \mathbf{B}G \to \mathbf{B}H$, as in Definition 14. On 2-cells, let $r : \mathsf{Group}_2(\varphi, \psi)$ a conjugator of homomorphisms. Delooping assigns a 2-cell $\mathbf{B}\varphi = \mathbf{B}\psi$ as follows: By function extensionality, it suffices to give $\mathbf{B}\varphi(x) = \mathbf{B}\psi(x)$ for any $x : \mathbf{B}G$. By induction on x, we are left to give some $q : \bullet = \bullet$ in $\mathbf{B}H$ such that for all $g : G$, $\mathsf{loop}\,\varphi(g) \cdot q = q \cdot \mathsf{loop}\,\psi(g)$. Choose $q := \mathsf{loop}\,r$, and compute

$$\mathsf{loop}\,\varphi(g) \cdot \mathsf{loop}\,r = \mathsf{loop}\,(\varphi(g)r) \overset{(*)}{=} \mathsf{loop}\,(r\psi(g)) = \mathsf{loop}\,r \cdot \mathsf{loop}\,\psi(g),$$

where $(*)$ uses that r is a conjugator of φ and ψ. By a similar argument, one shows that these assignments preserve composition and identities. ◀

Defined this way, we see that $\mathbf{B}$ preserves the local structure of Group:

▶ **Theorem 34.** *The functor* $\mathbf{B} : \mathsf{Group} \to \mathsf{hGpd}$ *is locally a weak equivalence of categories, i.e. for all groups* G, H, *there merely exists an inverse functor* $\mathsf{hGpd}_1(\mathbf{B}G, \mathbf{B}H) \to \mathsf{Group}_1(G, H)$.

Note that this cannot be strengthened to a full equivalence with explicit inverse: equivalence of categories preserves properties, but hGpd is by definition locally univalent, whereas Group is not.

We prove Theorem 34 by showing that locally, $\mathbf{B}$ is fully faithful and essentially surjective.

▶ **Proposition 35.** *Delooping is a locally fully faithful functor: For groups G, H, the local functor $\mathbf{B} : \mathsf{Group}_1(G, H) \to \mathsf{hGpd}_1(\mathbf{B}G, \mathbf{B}H)$ is an equivalence of categories.*

Proof. Let $\varphi, \psi : \mathsf{Group}_1(G, H)$. We establish a chain of equivalences between the sets of 2-cells $\mathsf{Group}_2(\varphi, \psi)$ and $\mathbf{B}\varphi = \mathbf{B}\psi$. Starting from the definition,

$$\mathsf{Group}_2(\varphi, \psi) \simeq \sum_{h:H} \prod_{g:G} \varphi(g)h = h\psi(g)$$

we apply the equivalence of groups, $\mathsf{loop} : H \simeq \Omega(\mathbf{B}H)$ twice

$$\simeq \sum_{h:H} \prod_{g:G} \mathsf{loop}\,\varphi(g) \cdot \mathsf{loop}\,h = \mathsf{loop}\,h \cdot \mathsf{loop}\,\psi(g)$$

$$\simeq \sum_{\ell:\Omega(\mathbf{B}H)} \prod_{g:G} \mathsf{loop}\,\varphi(g) \cdot \ell = \ell \cdot \mathsf{loop}\,\psi(g)$$

By the recursion principle, this is exactly a type of dependent functions out of $\mathbf{B}G$, namely

$$\simeq \prod_{x:\mathbf{B}G} \mathbf{B}\varphi(x) = \mathbf{B}\psi(x)$$

which, by function extensionality, is equivalent to

$$\simeq \mathbf{B}\varphi = \mathbf{B}\psi$$

One verifies that the map underlying this chain is that of Lemma 33. ◀

▶ **Proposition 36.** *Delooping is a locally essentially surjective functor.*

Proof. Let G, H groups, and $f : \mathbf{B}G \to \mathbf{B}H$ a morphism of groupoids. We show the *mere* existence of some $\varphi : \mathsf{Group}_1(G, H)$ together with an isomorphism $\mathbf{B}\varphi \cong f$ in the local category $\mathsf{hGpd}(\mathbf{B}G, \mathbf{B}H)$. By definition, morphisms in this category are homotopies, and it suffices to exhibit some $h : \prod_{x:\mathbf{B}G} \mathbf{B}\varphi(x) = f(x)$. Since $\mathbf{B}H$ is a *connected* groupoid, there merely exists a path $p : f(\bullet) = \bullet$, and conjugation by p induces an equivalence of groups,

$$\begin{aligned} \mathsf{conj}(p) &: \Omega(\mathbf{B}H, f(\bullet)) \to \Omega(\mathbf{B}H, \bullet) \\ &(q : f(\bullet) = f(\bullet)) \mapsto p^{-1} \cdot q \cdot p \end{aligned}$$

We define φ to be the composite

$$G \xrightarrow{\mathsf{loop}} \Omega(\mathbf{B}G, \bullet) \xrightarrow{\mathsf{cong}(f)} \Omega(\mathbf{B}H, f(\bullet)) \xrightarrow{\mathsf{conj}(p)} \Omega(\mathbf{B}H, \bullet) \xrightarrow{\mathsf{loop}^{-1}} H$$

By induction on $x : \mathbf{B}G$, we show that $\prod_{x:\mathbf{B}G} \mathbf{B}\varphi(x) = f(x)$. On the point, this is given by $p^{-1} : \bullet = f(\bullet)$. On loops, we construct $\prod_{g:G} \mathbf{B}\varphi(\mathsf{loop}\,g) \cdot p^{-1} = p^{-1} \cdot f(\mathsf{loop}\,g)$ as follows: let $g : G$, then $\mathbf{B}\varphi(\mathsf{loop}\,g) \cdot p^{-1} = \mathsf{loop}\,(\mathsf{loop}^{-1}\,(p^{-1} \cdot f(\mathsf{loop}\,g) \cdot p)) \cdot p^{-1} = p^{-1} \cdot f(\mathsf{loop}\,g)$ ◀

4.2 A 2-Category of Group Actions

Any G-action σ comes with an associated bundle, $\bar{\mathbf{B}}\sigma : \mathsf{loop}\, G \to \mathsf{hSet}$ (Definition 15). Let us define 2-categories of actions and of set bundles, and show that "taking the associated bundle" is a functorial construction.

▶ **Definition 37** (2-category of actions). *The 2-category* Action *of group actions displayed over* Group *consists of the following data:*

- *For each group G, objects are G-actions* $\mathsf{Action}_0(G) := \sum_{X:\mathsf{hSet}} G \dashrightarrow \mathfrak{S}(X)$.
- *1-cells between actions $(X, \sigma) : \mathsf{Action}_0(G)$ and $(Y, \tau) : \mathsf{Action}_0(H)$ over $\varphi : \mathsf{Group}_1(G, H)$ are equivariant maps* $\mathsf{Action}_1(\varphi; (G, \sigma), (H, \tau)) := \sum_{f:X \leftarrow Y} \mathsf{isEquivariant}_{\varphi,f}(\sigma, \tau)$, *where* isEquivariant *is as in Definition 26.*
- *The type of 2-cells between $f : \mathsf{Action}_1(\varphi; (X, \sigma), (Y, \tau))$ and $g : \mathsf{Action}_1(\psi; (X, \sigma), (Y, \tau))$ over a conjugator $r : \mathsf{Group}_2(\varphi, \psi)$ is $\mathsf{Action}_2(r; f, g) := (f = g \circ \tau(r))$.* ⌟

The type of 2-cells says that 1-cells agree up to a permutation of their domain. Since it is a proposition, verifying the axioms of a displayed 2-category reduces to defining *some* identity- and composite 2-cells. The vertical composite $p \bullet q : f \Rightarrow_{rs} h$ of 2-cells $p : f \Rightarrow_r g$ and $q : g \Rightarrow_s h$, is some identification $f = h \circ \tau(rs)$. Since τ is an action, we define the composite as $p \bullet q := \big(f \overset{p}{=} g \circ \tau(r) \overset{q}{=} h \circ \tau(s) \circ \tau(r) = h \circ \tau(rs)\big)$. Similarly, horizontal composition depends on 2-cells of Group being conjugators of group homomorphisms.

▶ **Definition 38** (set bundles). *The 2-category of* set bundles, *displayed over* hGpd, *consists of the following data:*

- *Given $G : \mathsf{hGpd}_0$,* set bundles *on G are families* $\mathsf{SetBundle}_0(G) := G \to \mathsf{hSet}$.
- *Over $\varphi : \mathsf{hGpd}_1(G, H)$, morphisms from $X : \mathsf{SetBundle}(G)$ to $Y : \mathsf{SetBundle}(H)$ are dependent functions,* $\mathsf{SetBundle}_1(\varphi; X, Y) := \prod_{g:G} Y(\varphi g) \to X(g)$
- *Displayed 2-cells between $f : \mathsf{SetBundle}_1(\varphi; X, Y)$ and $g : \mathsf{SetBundle}_1(\psi; X, Y)$ over a 2-cell $p : \varphi = \psi$ are dependent identifications,* $\mathsf{SetBundle}_2(p; f, g) := f =_p g$. ⌟

For an object $(G, F) : \mathsf{SetBundle}_0$ in the total category, we call G the *base* of the bundle, and F its fibers.

We are now ready to show that taking the bundle associated to an action is a well-behaved functorial operation. In particular, each equivariant map of actions induces a morphism between associated bundles:

▶ **Definition 39.** *Let $\sigma : \mathsf{Action}(G, X)$, $\tau : \mathsf{Action}(H, Y)$, and $\varphi : \mathsf{Group}_1(G, H)$. Let $f : Y \leftarrow X$ and $p : \mathsf{isEquivariant}_{\varphi,f}(\sigma, \tau)$. The bundle morphism associated to f has type $\bar{\mathbf{B}}f : \prod_{x:\mathbf{B}G} \bar{\mathbf{B}}\tau(\mathbf{B}\varphi x) \to \bar{\mathbf{B}}\sigma(x)$, and is defined using the induction principle of $\mathbf{B}G$. On the point it has type $\bar{\mathbf{B}}f(\bullet) : Y \to X$ and is given by f. On a loop, we need to prove that $\bar{\mathbf{B}}\sigma(\mathsf{loop}\, g) \circ f = f \circ \bar{\mathbf{B}}\tau(\mathbf{B}\varphi(\mathsf{loop}\, g))$ for all $g : G$. This reduces to $\sigma(g) \circ f = f \circ \tau(\varphi(g))$, which is given by p.*

Both Action and SetBundle are total categories, and $\mathbf{B} : \mathsf{Group} \to \mathsf{hGpd}$ is a 2-functor between their bases. We thus define a 2-functor only on the displayed parts:

▶ **Definition 40.** *Taking associated bundles is a displayed 2-functor $\bar{\mathbf{B}} : \mathsf{Action} \to_{\mathbf{B}} \mathsf{SetBundle}$, consisting of the following data:*

- *On objects, it sends a G-action σ to its associated bundle $\bar{\mathbf{B}}\sigma : \mathbf{B}G \to \mathsf{hSet}$.*
- *On 1-cells, it associates to an equivariant map $f : \mathsf{Action}_1(\varphi; \sigma, \tau)$ its morphism of bundles $\bar{\mathbf{B}}f : \mathsf{SetBundle}_1(\mathbf{B}\varphi; \bar{\mathbf{B}}\sigma, \bar{\mathbf{B}}\tau)$.*
- *Over a 2-cell $r : \mathsf{Group}_2(G, H)$, a proof $p : \mathsf{Action}_2(r; f, g) \doteq (f = g\tau(r))$ is sent to a homotopy of bundle maps using the induction principle of $\mathbf{B}G$.*

Both actions on 1- and 2-cells are defined by induction, and thus are equivalences in the sense of Proposition 13:

▶ **Lemma 41.** *The action on 1-cells* $\bar{\mathbf{B}}_1 : \mathsf{Action}_1(\varphi; \sigma, \tau) \to \mathsf{SetBundle}_1(\mathbf{B}\varphi; \bar{\mathbf{B}}\sigma, \bar{\mathbf{B}}\tau)$ *and 2-cells* $\bar{\mathbf{B}}_2 : \mathsf{Action}_2(r; f, g) \to \mathsf{SetBundle}_2(\mathbf{B}r; \bar{\mathbf{B}}f, \bar{\mathbf{B}}g)$ *are equivalences of types.*

▶ **Theorem 42.** *Taking associated bundles* $\int \bar{\mathbf{B}} : \mathsf{Action} \to \mathsf{SetBundle}$ *is locally a weak equivalence.*

Proof. The total functor $\int \bar{\mathbf{B}}$ is locally fully faithful if $\int_2 \bar{\mathbf{B}}$ is an equivalence, but this is a map on Σ-types built from $\mathbf{B}_2$ and $\bar{\mathbf{B}}_2$, which are both equivalences by Theorem 34 and Lemma 41. Local essential surjectivity is proved similarly to Proposition 36, and uses that $\bar{\mathbf{B}}_1$ is an equivalence of types. ◀

The above theorem implies that the local category $\mathsf{SetBundle}(\mathbf{B}G, \mathbf{B}H)$ is the Rezk completion of $\mathsf{Action}(G, H)$. As such the 2-category $\mathsf{SetBundle}$ should be the local univalent completion of Action in the sense of [6, Conjecture 5.6].

4.3 A 2-Categorical Fam-Construction

In the previous section we have seen how containers with a single action relate to set bundles. To lift this relationship to families of actions, we introduce a 2-categorical Fam-construction, again employing displayed machinery.

▶ **Definition 43** (2-category of families). *Let C be a 2-category. The 2-category* $\mathrm{Fam}(C)$ *displayed over* hSet *consists of the following data:*

- *For* $J : \mathsf{hSet}_0$*, the displayed objects are families of C-objects,* $\mathrm{Fam}_0(J) :\equiv J \to C_0$.
- *Let* $J, K : \mathsf{hSet}_0$ *and families* $X : \mathrm{Fam}_0(J)$*,* $Y : \mathrm{Fam}_0(K)$*. The type of 1-cells displayed over some* $\varphi : \mathsf{hSet}_1(J, K)$ *is* $\mathrm{Fam}_1(\varphi; X, Y) :\equiv \prod_{j:J} C_1(X_j, Y_{\varphi j})$.
- *Displayed 2-cells are a family* $\mathrm{Fam}_2 : (\varphi = \psi) \to \mathrm{Fam}_1(\varphi, X, Y) \to \mathrm{Fam}_1(\psi, X, Y) \to \mathcal{U}$ *defined by path-induction on 2-cells in* hSet*:* $\mathrm{Fam}_2(\mathsf{refl}_\varphi; f, g) :\equiv \prod_{j:J} C_2(f_j, g_j)$

▶ **Definition 44.** *Any 2-functor* $F : \mathsf{C} \to \mathsf{D}$ *lifts to a functor* $\mathrm{Fam}(F) : \mathrm{Fam}(\mathsf{C}) \to \mathrm{Fam}(\mathsf{D})$*. This lifting is defined as a total functor* $\mathrm{Fam}(F) : \int_{J:\mathsf{hSet}} \mathrm{Fam}(\mathsf{C})(J) \to \int_{J:\mathsf{hSet}} \mathrm{Fam}(\mathsf{D})(J)$ *over the identity 2-functor on the base* hSet*.*

▶ **Proposition 45.** *Lifting* $F : \mathsf{C} \to \mathsf{D}$ *to a 2-functor of families inherits the following properties:*

1. *If F is locally fully-faithful, so is* $\mathrm{Fam}(F)$*.*
2. *If F is locally split-essentially surjective, so is* $\mathrm{Fam}(F)$*.*
3. *Assuming the axiom of choice for h-sets and that* C *is locally strict, if F is locally essentially surjective, so is* $\mathrm{Fam}(F)$*.*

Proof. Local fully-faithfulness follows from the pointwise definition of Fam_2: if $\mathsf{C}_2(f, g)$ is an equivalence, then so is $\mathrm{Fam}_2(\mathsf{refl}; -, -) \doteq \lambda f, g. \prod_j \mathsf{C}_2(f_j, g_j)$. Local split essential surjectivity follows from a similar pointwise argument.

In the non-split case, fix $x, y : \mathrm{Fam}_0(\mathsf{C})$, and a 1-cell $(\psi, g) : \mathrm{Fam}_0(x) \to \mathrm{Fam}_0(y)$. It suffices to provide merely a family of sections, $\|\prod_{j:J} \sum_f F_1(f) \cong g_j\|_{-1}$; the conclusion follows using the induction principle of the truncation. The assumption that F is locally eso yields $\prod_{j:J} \|\sum_{f:\mathsf{C}_1(x_j, y_{\psi j})} F_1(f) \cong g_j\|_{-1}$, and we use choice to move the truncation outward: J is a set, and so are $\mathsf{C}_1(x_j, y_{\psi j})$ (by local strictness of C) and local isomorphisms $F_1(f) \cong g_j$. ◀

4.4 Action Containers as a 2-Category of Families

As promised, we define the 2-category of action containers as a 2-category of families:

▶ **Definition 46** (2-category of action containers). *The 2-category of action containers is that of families of actions,* $\mathsf{ActCont} := \mathrm{Fam}(\mathsf{Action})$.

By algebra of Σ- and Π-types, we see that the objects and 1-cells coincide with those of the 1-category of action containers (cf. Definition 25). In particular, we have for objects

$$\mathsf{ActCont}_0 \simeq \sum_{S:\mathsf{hSet}} \sum_{P:S\to\mathsf{hSet}} \sum_{G:S\to\mathsf{Group}} \prod_{s:S} \mathsf{Action}(G_s, P_s),$$

and for 1-cells,

$$\begin{aligned}&\mathsf{ActCont}_1((S, \lambda s.\,(P_s, G_s, \sigma_s)), (T, \lambda t.\,(Q_t, H_t, \tau_t))) \\ &\quad\simeq \sum_{u:S\to T} \prod_{s:S} \sum_{\varphi:G_s\to H_{us}} \sum_{f:P_s\leftarrow Q_{us}} \mathsf{isEquivariant}_{\varphi,f}(\sigma_s, \tau_{us})\end{aligned}$$

Unfolding the newly added type of 2-cells, we recover a familiar condition:

▶ **Proposition 47.** *Let* $E, F : \mathsf{ActCont}_0$ *and denote* $E \doteq (S, \lambda s.\,(P_s, G_s, \sigma_s))$ *and* $F \doteq (T, \lambda t.\,(Q_t, H_t, \tau_t))$. *Let* $u : S \to T$ *and* $f, g : \mathsf{ActCont}_1(u; E, F)$. *The type of 2-cells* $\mathsf{ActCont}_2((u, f), (u, g))$ *is equivalent to*

$$\prod_{s:S} \sum_{r:H_{us}} \mathsf{isConjugator}_{\varphi_s,\psi_s}(r) \times f'_s = g'_s \circ \tau_{us}(r),$$

in which $\varphi, \psi : \prod_s G_s \dashrightarrow H_{us}$ *and* $f', g' : \prod_s Q_{us} \to P_s$ *are the maps of symmetries and positions of* f *and* g*, respectively.*

Note the occurrence of the proposition $f'_s = g'_s \circ \tau_{us}(r)$: it has already appeared in the definition of quotient container morphisms as a quotient of premorphisms (Definition 6). In [3, Definition 4.1] it is explained as a necessary condition for labellings of container maps to be defined upto quotient. In our case it is necessary to characterize homotopies between induced bundle maps $\bar{\mathbf{B}}_1(f'_s)$ and $\bar{\mathbf{B}}_1(g'_s)$ (Lemma 41).

Lifting the 2-functor $\bar{\mathbf{B}} : \mathsf{Action} \to \mathsf{SetBundle}$ to families, we immediately obtain the following characterization of 1-cells of action containers. Substituting $\mathsf{ActCont} \doteq \mathrm{Fam}(\mathsf{SetBundle})$ and $\mathsf{SymmCont} \doteq \mathsf{SetBundle}$, we see:

▶ **Corollary 48.** *The lifting* $\mathrm{Fam}(\bar{\mathbf{B}}) : \mathsf{ActCont} \to \mathrm{Fam}(\mathsf{SymmCont})$ *is:*

1. *locally fully faithful*
2. *assuming the axiom of choice, locally a weak equivalence*

Proof. By application of Proposition 45: The 2-category Action is locally strict, and $\bar{\mathbf{B}}$ locally a weak equivalence (Theorem 42). ◀

This says that constructively, morphisms of action containers correspond to a subcategory of morphisms of (families of) symmetric containers. By using the axiom of choice, one sees that this construction indeed covers all such morphism.

To connect action containers to symmetric ones, and not just families of the latter, note the following: Any family of set bundles (hence symmetric containers) can be combined into a single bundle: given $(J, (\lambda j.\,(B_j, F_j))) : \mathrm{Fam}(\mathsf{SetBundle})$, we can consider the bundle of fibers over the sum of bases, $\sum_{j:J} B_j$. This construction defines a 2-functor:

▶ **Definition 49** (summation of set bundles). *Summation of set bundles is a 2-functor* $\Sigma : \mathrm{Fam}(\mathsf{SetBundle}) \to \mathsf{SetBundle}$*, with the following data*

1. $\Sigma_0(J, \lambda j.\,(B_j, F_j))$ *consists of the base* $\sum_{j:J} B_j$*, and fibers* $\lambda(j,b).\,F_j(b)$*.*
2. $\Sigma_1(u, \lambda j.\,(\varphi_j, f_j))$ *is a pair of a reindexing function* $(u, \varphi_{\text{-}}) : \sum_J B \to \sum_K C$ *and a map of fibers,* $f_{\text{-}} : \prod_{(j,b)} G_{uj}(\varphi_j(b)) \to F_j(b)$*.*
3. *On 2-cells, it takes a family of identities of bundle maps to an identity of their sums via function extensionality:* $\Sigma_2(\mathsf{refl}_u, \lambda j.\,(r_j, \bar{r}_j)) :\equiv \mathsf{funext}\,\lambda(j,b).\,\mathsf{cong}_{uj,\text{-}b}(r_j), \bar{r}_j(b)$*.* ⌟

The construction turning action containers into symmetric ones now factors as follows:

$$\mathsf{ActCont} \xrightarrow{=} \mathrm{Fam}(\mathsf{Action}) \xrightarrow{\mathrm{Fam}(\bar{\mathbf{B}})} \mathrm{Fam}(\mathsf{SetBundle}) \xrightarrow{\Sigma} \mathsf{SetBundle} \xrightarrow{=} \mathsf{SymmCont}$$

In general, we do not know whether Σ is locally fully faithful or not. We can however consider its restriction to objects in the image of $\mathrm{Fam}(\bar{\mathbf{B}})$, and deduce fully-faithfulness for some of its local functors:

▶ **Lemma 50.** *Let* $X, Y : \mathrm{Fam}_0(\mathsf{SetBundle})$*. If all bundles in* X *have connected bases, then the local functor* $\Sigma_1 : \mathrm{Fam}_1(X, Y) \to \mathsf{SetBundle}(\Sigma_0(X), \Sigma_0(Y))$ *is fully faithful.*

Proof. The proof proceeds by showing that there is an equivalence of 2-cells. The assumption on connectedness is used as follows: Recall that $X \doteq (J, \lambda j.\,(B_j, F_j))$ has connected bases if all B_j are connected groupoids. The base of the bundle $\Sigma_0(X)$ is $\sum_j B_j$, and maps between such bases are typed $\sum_j B_j \to \sum_k B'_k$. To characterize identifications of these maps, it is necessary show, given morphisms $u, v : J \to K$, that the function $u(j) = v(j) \to (B_j \to u(j) = v(j))$ constant in B_j is an equivalence. But this follows from connectedness of B_j and the fact that $u(j) = v(j)$ is a proposition [20, 7.5.9]. ◀

▶ **Theorem 51.** *The composite* $\Sigma \circ \mathrm{Fam}(\bar{\mathbf{B}}) : \mathsf{ActCont} \to \mathsf{SymmCont}$ *is locally fully faithful.*

5 Conclusions

We introduced action containers for studying data types with symmetries. This class of containers is inspired by quotient containers, but are different from the latter in two key aspects: First, symmetries are encoded by arbitrary groups acting on positions, allowing for more freedom in presenting permutations of positions. Second, morphisms are required to respect symmetries in a coherent way, and are additionally not equivalence classes of an ad-hoc relation. Instead, the category of action containers is presented universally as a free coproduct completion of a category of groups and actions, from which limits and colimits of action containers are easy to read off. We reintroduce the relation between morphisms in terms of a 2-categorical structure, and show that the 2-category of action containers embeds into that of symmetric containers.

A missing piece in our analysis is the relationship between quotient and action containers. The latter subsume the former, but morphisms of action containers are more constrained than those of quotient containers. Finding a functorial connection is not straightforward: Each action container $(S \triangleright P \triangleleft^{\sigma} G)$ can be mapped to a quotient container with the same set of shapes and positions, but for each $s : S$ changing the group to $\mathsf{Im}(\sigma_s)$, which is a subgroup of $\mathfrak{S}(P_s)$. Unfortunately this operation does not act on morphism $(u \triangleright f \triangleleft \varphi) : (S \triangleright P \triangleleft^{\sigma} G) \to (T \triangleright Q \triangleleft^{\tau} H)$, since group homomorphisms $\varphi_s : G_s \dot{\to} H_{us}$ do not generally restrict to the image of the actions $\mathsf{Im}(\sigma_s) \dot{\to} \mathsf{Im}(\tau_{us})$. When restricted to a 1-subcategory of action containers with faithful actions, this construction at least yields an isomorphism-on-objects functor

$\mathsf{ActCont_{faith}} \to \mathsf{QuotCont}$. In the opposite direction, one could search for a functor between the category $\mathsf{QuotCont}$ and the homotopy category of $\mathsf{ActCont}$, i.e. the category with the same objects but with sets of morphisms obtained by set quotienting 1-cells by 2-cells. We have a candidate if we modify Definition 5 of premorphism by turning the existential quantifier in the preservation of symmetries into a dependent sum: send a quotient container $(S \triangleright P/G)$ to the action container with the same set of shapes and positions, but for each $s : S$ changing the group to the free group generated by G_s, which also acts on P_s, though not in a faithful way. This modification shows at least the existence of an action of morphisms. We postpone a deeper investigation in this direction to future work.

In Section 3.1 we analyzed some properties of the category of action containers. Ordinary containers enjoy further properties that make them suitable as a model of data types: Altenkirch et al. [8] show that the category of ordinary containers is cartesian closed, and Abbott et al. [3] construct initial algebras and final coalgebras of containers in one parameter. In the future, we plan to investigate whether action containers also enjoy these properties as well. From previous investigations [14], we know that direct construction of final coalgebras for quotient containers fails constructively. In [5] however, Ahrens et al. show that for any $\mathcal{U}$-valued container (with no restriction on truncation level of shapes or positions), its extension as a polynomial in $\mathcal{U}$ admits a final coalgebra. Since extensions of symmetric containers are $\mathcal{U}$-polynomials as well, we would like to internalize this construction: first, find a symmetric container representing this coalgebra, then investigate if this restricts to the inclusion of action containers. This in particular requires studying the composition-monoidal structure of action containers. Similarly, it should be possible to lift the closure properties of Section 3.1 from the underlying 1-category to proper 2-(co)limits in Action.

In another direction we are interested to see if the heavy machinery of 2-categories can be avoided, or at least be postponed. This is guided by the following insight: The image on objects of the 2-functor $\mathbf{B}$ is not only groupoids, but *pointed, connected groupoids.* The 2-category of such groupoids and pointed maps, displayed over hGpd is, surprisingly, locally thin [9, Lemma 4.4.12]. In other words, pointed, connected groupoids and pointed maps form a 1-category hGroup. A slight modification of the proof of Proposition 36 shows that the 1-category of ordinary groups is equivalent to hGroup, and that this equivalence extends to categories of actions. The same argument shows that the 2-category of *skeletal groupoids*[2] and skeleton-preserving maps is locally thin. We believe that this extends to an equivalence between the 1-categories of action containers (without additional relation between morphisms) and skeletal symmetric containers, i.e. those whose shapes are skeletal groupoids, and whose morphisms preserve such skeleta. This would identify a structure on symmetric containers such that equality of morphisms becomes propositional, giving rise to a convenient 1-category of containers with symmetries.

References

1 Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Categories of Containers. In Andrew D. Gordon, editor, *Proc. of 6th Int. Conf. on Foundations of Software Science and Computation Structures, FoSSaCS'03*, volume 2620 of *Lecture Notes in Computer Science*, pages 23–38. Springer Berlin Heidelberg, 2003. doi:10.1007/3-540-36576-1_2.

2 Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers: Constructing strictly positive types. *Theoretical Computer Science*, 342(1):3–27, 2005. doi:10.1016/j.tcs.2005.06.002.

[2] Those that come with a choice of basepoint $\mathsf{pt} : \prod_{x:\|B\|_0} \mathsf{fiber}_{|-|_0}(x)$ for each connected component.

3 Michael Abbott, Thorsten Altenkirch, Neil Ghani, and Conor McBride. Constructing Polymorphic Programs with Quotient Types. In Dexter Kozen and Carron Shankland, editors, *Proc. of 7th Int. Conf. on Mathematics of Program Construction, MPC'04*, volume 3125 of *Lecture Notes in Computer Science*, pages 2–15. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-27764-4_2.

4 Jiří Adámek and Jiří Rosický. How nice are free completions of categories? *Topology and its Applications*, 273:106972, 2020. doi:10.1016/j.topol.2019.106972.

5 Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. Non-Wellfounded Trees in Homotopy Type Theory. In Thorsten Altenkirch, editor, *Proc. of 13th Int. Conf. on Typed Lambda Calculi and Applications, TLCA'15*, volume 38 of *LIPIcs*, pages 17–30. Schloss Dagstuhl, 2015. doi:10.4230/LIPICS.TLCA.2015.17.

6 Benedikt Ahrens, Dan Frumin, Marco Maggesi, Niccolò Veltri, and Niels van der Weide. Bicategories in univalent foundations. *Mathematical Structures in Computer Science*, 31(10):1232–1269, 2021. doi:10.1017/s0960129522000032.

7 Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed Categories. *Logical Methods in Computer Science*, 15(1), March 2019. doi:10.23638/lmcs-15(1:20)2019.

8 Thorsten Altenkirch, Paul Blain Levy, and Sam Staton. Higher-Order Containers. In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *6th Conf. on Computability in Europe, CiE'10*, volume 6158 of *LNCS*, pages 11–20. Springer, 2010. doi:10.1007/978-3-642-13962-8_2.

9 Marc Bezem, Ulrik Buchholtz, Pierre Cagne, Bjørn Ian Dundas, and Daniel R. Grayson. Symmetry. https://github.com/UniMath/SymmetryBook. Commit: 8546527.

10 Håkon Robbestad Gylterud. Symmetric Containers. Master's thesis, Department of Mathematics, Faculty of Mathematics and Natural Sciences, University of Oslo, 2011. URL: https://hdl.handle.net/10852/10740.

11 Ryu Hasegawa. Two applications of analytic functors. *Theoretical Computer Science*, 272(1–2):113–175, February 2002. doi:10.1016/s0304-3975(00)00349-2.

12 Pieter Hofstra and Martti Karvonen. Inner automorphisms as 2-cells. *Theory and Applications of Categories*, 42(2):19–40, 2024. URL: http://www.tac.mta.ca/tac/volumes/42/2/42-02abs.html.

13 Niles Johnson and Donald Yau. *2-Dimensional Categories*. Oxford University Press, January 2021. doi:10.1093/oso/9780198871378.001.0001.

14 Philipp Joram and Niccolò Veltri. Constructive Final Semantics of Finite Bags. In Adam Naumowicz and René Thiemann, editors, *Proc. of 14th Int. Conf. on Interactive Theorem Proving, ITP'23*, volume 268 of *LIPIcs*, pages 12:1–12:19. Schloss Dagstuhl, 2023. doi:10.4230/LIPICS.ITP.2023.20.

15 André Joyal. Foncteurs analytiques et espèces de structures. In Gilbert Labelle and Pierre Leroux, editors, *Combinatoire énumérative*, volume 1234 of *Lecture Notes in Mathematics*, pages 126–159. Universite du Quebec a Montreal, Springer Berlin Heidelberg, May 1986. doi:10.1007/bfb0072514.

16 Daniel R. Licata and Eric Finster. Eilenberg-MacLane spaces in homotopy type theory. In Thomas A. Henzinger and Dale Miller, editors, *Proc. of the Joint Meeting of the 23rd EACSL Ann. Conf. on Computer Science Logic (CSL) and the 29th Ann. ACM/IEEE Symp. on Logic in Computer Science (LICS), CSL-LICS'14*, pages 66:1–66:9. ACM, 2014. doi:10.1145/2603088.2603153.

17 Max New, Steven Shaefer, and contributors. cubical-categorical-logic, 2024. URL: https://github.com/maxsnew/cubical-categorical-logic.

18 Andrew M. Pitts. *Nominal sets*. Number 57 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2013.

19 The agda/cubical development team. The agda/cubical library, 2018. URL: https://github.com/agda/cubical/.

20 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

Synthetic 1-Categories in Directed Type Theory

Jacob Neumann
University of Nottingham, UK

Thorsten Altenkirch
University of Nottingham, UK

Abstract

The field of *directed type theory* seeks to design type theories capable of reasoning synthetically about (higher) categories, by generalizing the symmetric *identity* types of Martin-Löf Type Theory to asymmetric *hom-types*. We articulate the directed type theory of the *category model*, with appropriate modalities for keeping track of variances and a powerful directed-J rule capable of proving results about arbitrary terms of hom-types; we put this rule to use in making several constructions in synthetic 1-category theory. Because this theory is expressed entirely in terms of *generalized algebraic theories*, we know automatically that this directed type theory admits a syntax model.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Semantics, directed type theory, homotopy type theory, category theory, generalized algebraic theories

Digital Object Identifier 10.4230/LIPIcs.TYPES.2024.7

Related Version *Full Version*: `https://arxiv.org/abs/2410.19520` [26]

Funding *Jacob Neumann*: This publication is based upon work from COST Action EuroProofNet, supported by COST (European Cooperation in Science and Technology, www.cost.eu).

1 Introduction

A key motivation behind the emergent field of *homotopy type theory (HoTT)* [34] is the interpretation that *types are* ∞*-groupoids* [35]. Homotopy type theory can be understood as a *synthetic* theory of ∞-groupoids: all the higher structure is generated by the simple rules for manipulating identity types in Martin-Löf Type Theory [23, 24], permitting efficient reasoning with these complex structures.

Not long after homotopy type theory was established, the search for *directed homotopy type theory* – a synthetic theory of (higher) *categories* – began. In a directed type theory, the identity types of ordinary Martin-Löf Type Theory – which are provably symmetric in the theory, i.e. a witness $p\colon \mathsf{Id}(t,t')$ can be turned into $p^{-1}\colon \mathsf{Id}(t',t)$ – are replaced by asymmetric *hom-types*. However, building a type theory to effectively work with these hom-types is beset by difficulties, in particular the need to carefully track the *variances* of terms. A common feature of many approaches to directed type theory (e.g. [22, 29, 28]) is to track these variances by adopting some kind of modal typing discipline. However, no consensus ever emerged for exactly how to do this. The most developed branch of directed type theory (initiated by Riehl and Shulman [30]) avoids these issues by adopting a more indirect approach inspired by simplicial spaces. The resulting theory provides a synthetic language for ∞-categories, albeit at the cost of a more elaborate, multi-layered theory.

To synthetically study 0- and 1-categories, however, a more modest theory will suffice. This was true in the undirected/groupoidal case: the *setoid model* [15, 2] provides semantics for a synthetic theory of setoids (i.e. 0-groupoids) and the *groupoid model* [17] provides semantics for a synthetic theory of 1-groupoids. Both of these models are an instance of *categories with families (CwFs)* [12], which are the *generalized algebraic theory (GAT)* [9] articulation of the semantics of type theory. Working within the framework of GATs comes with numerous

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg; Article No. 7; pp. 7:1–7:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

advantages: the universal algebraic features of GATs – such as homomorphisms, displayed models, products and coproducts of models, free and cofree models – are well-understood [20]. In particular, every GAT has an initial model, the *syntax model*, which can be constructed as a quotient inductive-inductive type [18]. Adopting CwFs as our notion of 'model of type theory' makes all this machinery available.

The purpose of the present work is, then, to propose a synthetic theory of 1-categories, whose semantics are given as a generalized algebraic theory, i.e. as a CwF with additional structure. Our intended model will be the *category model*, which is directly analogous to the groupoid and setoid models. We closely mirror the kind of metatheoretic arguments made by Hofmann and Streicher [17] and develop a directed analogue of their *universe extensionality*, an early statement of univalence. Like earlier directed type theories, our theory adopts a modal typing discipline to control variance (and prevent the theory from "collapsing" into undirected type theory); our hope is that a GAT-based study of this (relatively simple) modal type theory will help foster a connection between the study of multi-modal type theories [13] and *second-order* GATs [33, 32, 8, 19], though connecting the present work to either of these theories is left to future study.

1.1 Related Work

The present work draws some constructs from the theory of Licata and Harper [22], particularly their treatment of the *opposite category* construction as a modality on contexts and context extension, as well as their treatment of Π-types in the directed setting. Our theory is closest to that of North [28]; in particular, her semantics matches our 'category model' in the interpretation of the basic structural rules of type theory. However, we have different approaches to typing refl within our modal typing discipline: North restricts to groupoid *types* whereas we restrict to groupoid *contexts*. Recent extensions to North's theory [11] *do* include modalities on contexts, suggesting that we may be converging on a common theory.

Our theory, like North's, is "1-dimensional" in the sense of Licata-Harper in that we maintain *judgmental* equality as a symmetric notion, as opposed to "2-dimensional" theories [22, 29, 1] which introduce a theory of *directed reductions*. As with the identity types in Martin-Löf type theory, all our category-theoretic structure is emergent from the hom-type formation rule and all our reasoning about hom-types is done by the directed J-rule; this is in contrast to directed type theories (such as [22] and [27]) which must build in the synthetic category structure "by hand". As mentioned, we adopt a modal typing discipline for handling variances, unlike the theories of [30, 21, 37, 37, 14] and [36], which adopt approaches akin to simplicial and cubical type theories, respectively.

1.2 Contribution and Organization

We present a directed type theory satisfying the following constraints.

(1) It is presented as a generalized algebraic theory – our notion of "model" is a (GAT) extension of the GAT of *categories with families*.

(2) It is 1-dimensional in the sense of Licata-Harper: there are no 'directed reductions' introduced judgmentally.

(3) It is *deeply-polarized*: there is a modal typing discipline to keep track of variances, which operates not just on types but on contexts, substitutions, and context extension.

(4) The directed J-rule (directed path induction) permits reasoning about arbitrary terms of hom-types.

(5) Hom-types can be iterated, expressing synthetic higher categorical structure (though in the present work we only consider 1-category theoretic structure).

To our knowledge, there is no existing type theory satisfying all these criteria. The Licata-Harper theory satisfies (3), but their hom-types are not inductively structured (ruling out (4)), and cannot be iterated (ruling out (5)). North's theory satisfies (2), (4), and (5), but only has what we'll call *shallow polarity* – negation on types, but not contexts and substitutions – and consequently cannot have Π-types. The simplicial type theory of Riehl-Shulman does not satisfy (3) or (4) – they don't adopt a polarity calculus in the sense contemplated here, nor assert hom-types as primitive with judgmental rules – but do satisfy (2) and particularly (5) – it is in this sense that they provide a synthetic theory of ∞-categories. No directed type theory, to our knowledge, explicitly satisfies (1), though some of their model notions may be equivalent to an extension of the GAT of CwFs.

Starting with Section 2, we adopt a semantics-driven approach by investigating a particular model, the *category model* and abstracting its key features into a series of abstract notions of model (Section 3). These notions are all GATs (indeed, CwFs with additional structure), and therefore each give rise to a syntax model. Our main notion is that of a *Directed CwF (DCwF)*, a generalized algebraic theory of directed types with adequate polarity structure to properly track variances.

Achieving (4) while maintaining a modal typing discipline requires a novel approach. In the typing rules of existing directed type theories (including ours), the endpoint terms t and t' of a hom-type $\mathsf{Hom}(t, t')$ are assigned opposite variances: t negative and t' positive. However, this poses a difficulty for typing the identity morphism $\mathsf{refl}_t \colon \mathsf{Hom}(t, t)$ since t must assume both variances. North [28] solves this by restricting t to be a term of a *core type* (interpreted semantically by groupoids), but the consequent J-rule only operates on hom-terms with a core endpoint, not arbitrary ones. Our solution instead uses groupoid *contexts* rather than groupoid *types*.

In Section 4, we show that this is a viable framework for conducting synthetic category theory. In this section, we adopt an informal style reminiscent of [34], showing how this theory can be operated and how the groupoid context can be carefully maintained by a simple syntactic rule. We use our directed J-rule to give several basic constructions in synthetic (1-)category theory.

Finally, we consider the directed universe of sets in the category model, which serves as the category of sets. The existence of a directed universe allows us to make the metatheoretic argument that the syntax of DCwFs cannot prove the symmetry of hom-types (i.e. this is a genuinely *directed* type theory) or the uniqueness of homs (analogous to Hofmann and Streicher's proof that the groupoid model refutes the uniqueness of identity proofs). We conclude by sketching several possible routes for further study.

An expanded version of this work with more detailed calculations and further discussion is available on the arXiv, as [26]. A much more thorough development – including more careful development of the neutral-context method, Σ-types, and synthetic category theory in (1,1)-directed type theory – is given in the first author's forthcoming PhD thesis [25].

1.3 Metatheory and Notation

Throughout, we work in an informal type-theoretic metatheory, using pseudo-Agda notation to specify GATs, make category-theoretic constructions, and define terms in the syntax of Directed CwFs. We use the notations

$$(x \colon X) \to P(x) \qquad \text{and} \qquad (x \colon X) \times P(x)$$

for the dependent function and dependent sum types, respectively. When defining dependent functions, we'll enclose arguments in curly brackets to indicate that they're implicit. Any variables appearing free are also assumed to be implicitly universally quantified. We sometimes use underscores to indicate where the arguments to a function are written.

For equality, we'll use the Agda convention (which, note, is the opposite of the convention used in the HoTT Book [34]): use $=$ to mean *definitional* or *judgmental* equality in our metatheory, whereas $\equiv$ means *propositional* equality (though there's no reason they couldn't coincide, i.e. in an extensional metatheory). We tacitly make use of appropriate extensionality principles (particularly function extensionality) for both notions of equality, and the uniqueness of identity proofs for $\equiv$. We write Prop for the type of h-propositions in our metatheory, i.e. those P such that $p \equiv p'$ for all $p, p' \colon P$.

We assume basic familiarity with category theory. The set of objects of a category Γ is denoted $|\Gamma|$, the set of Γ-morphisms from γ_0 to γ_1 is denoted $\Gamma\,[\gamma_0, \gamma_1]$, and identities are written as id. The *discrete* groupoid/category on a set X is the category whose objects are elements of X and whose morphisms from x_0 to x_1 are inhabitants of the identity type $x_0 \equiv x_1$. The *opposite category* construction is understood to be definitionally involutive, i.e. $|\Gamma^{\mathrm{op}}|$ is defined to be $|\Gamma|$ and $\Gamma^{\mathrm{op}}\,[\gamma_0, \gamma_1]$ is defined to be $\Gamma\,[\gamma_1, \gamma_0]$, and thus $(\Gamma^{\mathrm{op}})^{\mathrm{op}} = \Gamma$. We'll use the notation $\mathcal{C} \Rightarrow \mathcal{D}$ for the type of functors from $\mathcal{C}$ to $\mathcal{D}$.

2 The Category Interpretation of Type Theory

As mentioned, generalized algebraic theories (GATs) are a desirable formalism for expressing models of type theory, particularly when modelling numerous extensions to a 'basic' type theory. When a theory is given as a GAT, all operations and equations are made clear and explicit, making it easier to compare and contrast similar theories. The theory of **Categories with Families (CwFs)** (originally defined by Dybjer [12]) present the fundamental operations of type theory – contexts, variables, terms, types, and substitutions – encoded as a GAT; upon this basic framework, an endless variety of different type theories can be studied.

We assume basic familiarity with CwFs. The main components of a CwF are a category Con of *contexts*, whose morphisms are called *substitutions* (write $\mathsf{Sub}\ \Delta\ \Gamma$ for the set of substitutions from Δ to Γ); a presheaf Ty on Con (whose morphism part we denote $\_[\_]$)[1] and a dependent presheaf Tm over Ty;[2] and a *context extension* operation $\triangleright$ guaranteeing that Tm is *locally representable* (in the sense of [8]). That is, there is an isomorphism (natural in Δ) between the type of pairs (σ, t) with $\sigma \colon \mathsf{Sub}\ \Delta\ \Gamma$ and $t \colon \mathsf{Tm}(\Delta, A[\sigma])$ and the type of substitutions from Δ to $\Gamma \triangleright A$. The left-to-right direction of this isomorphism is denoted $\langle \_, \_ \rangle$ and the opposite direction as $\mathsf{p} \circ \_, \mathsf{v}[\_]$, so

$$\tau \equiv \langle \mathsf{p} \circ \tau, \mathsf{v}[\tau] \rangle \qquad \text{and} \qquad \sigma \equiv \mathsf{p} \circ \langle \sigma, t \rangle \qquad \text{and} \qquad t \equiv \mathsf{v}[\langle \sigma, t \rangle]$$

for any σ, t as above and $\tau \colon \mathsf{Sub}\ \Delta\ (\Gamma \triangleright A)$.

Two paradigm examples of CwFs are the *setoid model* of [15, 2] and the *groupoid model* of [17]. In the former, the contexts are setoids (i.e. sets equipped with equivalence relations), the types are families of setoids (functorially) indexed over their context setoid, and terms are given by the appropriate notion of *section* of their type (see the Coq formalization of [3] for a precise definition). The groupoid model is quite similar: contexts are groupoids, types are families of groupoids functorially indexed over their context groupoid, and terms are the appropriate notion of section. Indeed, we can view the groupoid model as generalizing the setoid model: a setoid can be viewed as a groupoid whose hom-sets are *subsingletons* (or *propositions*, in the terminology of [34]), sets with at most one element. In other words, the groupoid model is what results when the assumption of *proof-irrelevance* is dropped from the setoid model.

[1] I.e. if $\sigma \colon \mathsf{Sub}\ \Delta\ \Gamma$ and $A \colon \mathsf{Ty}\ \Gamma$, then $A[\sigma] \colon \mathsf{Ty}\ \Delta$.

[2] That is, a presheaf on the category of elements of Ty. Its morphism part is also denoted $\_[\_]$, so if $t \colon \mathsf{Tm}(\Gamma, A)$, then $t[\sigma] \colon \mathsf{Tm}(\Delta, A[\sigma])$.

Both these models provide interpretations for numerous type formers, in particular the dependent types, identity types, and universes characteristic of Martin-Löf Type Theory [23, 24]. The difference in these models is reflected in the type theories they interpret: while both models permit arbitrary iteration of the identity type former (expressing identities between identities, and identities between identities between identities, and so on), these iterated identity types become trivial more quickly in the setoid model. More precisely, the setoid model validates the *uniqueness of identity proofs* principle, meaning that any two terms of an identity type, $p, q \colon \mathsf{Tm}(\Gamma, \mathsf{Id}(x, y))$ are themselves identical, $\mathsf{UIP}(p, q) \colon \mathsf{Tm}(\Gamma, \mathsf{Id}(p, q))$. The groupoid model famously violates this principle: in the type theory of the groupoid model, there are types (in particular, the universe of sets) which are not *h-sets*, i.e. they possess terms which are proved identical by multiple, *distinct* identity proofs.

This provides a roadmap for how we might develop a model of directed type theory. Since directed type theory can be described as "dependent type theory, but with *asymmetric* identity types", this leads us to suspect that models of directed type theory will result if we simply drop the assumption of symmetry from the setoid and groupoid models. A setoid without symmetry is a preorder, and a groupoid without symmetry is a category. A close inspection of the definition of the groupoid model reveals that nothing in its interpretation of *just the CwF structure* requires symmetry (i.e. that morphisms are invertible),[3] and thus we can define the preorder model of type theory and the **category model of type theory**. The category model is just a generalization of the groupoid model, obtained by dropping symmetry: contexts are categories (and substitutions are functors); types are families of categories and terms are sections, as written more precisely in Figure 1. We won't focus on the preorder model here, but leave it to future work to develop the directed analogue of setoid-model-specific considerations (e.g. [3]). Instead, we'll highlight those features of the category model which are relevant for modelling directed type theory, before abstracting those features into the notion of a *directed CwF* in the next section.

```
-- A : Ty Γ means A : Γ → Cat
record Ty (Γ : Con) : Set where
  field
    obj : |Γ| → Cat
    map : Γ [ γ0, γ1 ] → Cat [ obj γ0, obj γ1 ]
    fid : map (id_γ) ≡ id_obj(γ)
    fcomp : map (γ12 ∘ γ01) ≡ (map γ12) ∘ (map γ01)

record Tm (Γ : Con) (A : Ty Γ) : Set where
  field
    obj : (γ : |Γ|) → |A(γ)|
    map : (γ01 : Γ [ γ0, γ1 ]) → (A γ1) [A γ01 (obj γ0), obj(γ1)]
    fid : map (id_γ) ≡ id_obj(γ)
    fcomp : map (γ12 ∘ γ01) ≡ (map γ12) ∘ (A γ12 (map γ01))
```

Figure 1 The family structure of the category model.

[3] Indeed, [18, Sect. 7] shows that any GAT gives rise to a CwF of algebras and displayed algebras. The groupoid model and setoid model are not literally instances of this construction, but can be viewed as the CwF of groupoids and displayed groupoids (resp. setoids and displayed setoids) with added fibrancy conditions attached (see [25, Chap. 2]). The same can be done with preorders and categories, yielding the preorder and category models.

While the basic CwF structure of the groupoid model doesn't require symmetry (i.e. that all morphisms are invertible), its interpretations of further type formers certainly do. After all, our hope is that by passing from the groupoid model to the category model, the symmetric identity types of the former will become asymmetric *hom-types* in the latter. Consider the semantics of the identity type former in the groupoid model. Here, and henceforth, we define a type (in this case $\mathsf{Id}(t, t')$) by giving its object- and morphism-parts, which are both written as just $\mathsf{Id}(t, t')$.

$$\begin{aligned}
&\mathsf{Id} : \mathsf{Tm}(\Gamma,\ \mathsf{A}) \to \mathsf{Tm}(\Gamma,\ \mathsf{A}) \to \mathsf{Ty}\ \Gamma \qquad \text{-- } \textit{Taken from } [17,\ \text{Section } 4.10]\\
&\quad (\mathsf{Id(t,t')})\ \gamma = (\mathsf{A}\ \gamma)\ [\mathsf{t}\ \gamma,\ \mathsf{t'}\ \gamma] \qquad \text{-- } \textit{Discrete groupoid}\\
&\\
&\quad (\mathsf{Id(t,t')})\ (\gamma_{01} : \Gamma[\ \gamma_0\ ,\ \gamma_1\ ]) : (\mathsf{A}\ \gamma_0)\ [\mathsf{t}\ \gamma_0,\ \mathsf{t'}\ \gamma_0] \to (\mathsf{A}\ \gamma_1)\ [\mathsf{t}\ \gamma_1,\ \mathsf{t'}\ \gamma_1]\\
&\quad (\mathsf{Id(t,t')})\ \gamma_{01}\ \mathsf{x}_0 = (\mathsf{t'}\ \gamma_{01}) \circ (\mathsf{A}\ \gamma_{01}\ \mathsf{x}_0) \circ (\mathsf{t}\ \gamma_{01})^{-1}
\end{aligned}$$

Here, the fact that $A(\gamma_1)$ is a *groupoid* is used in an essential way (we must take the inverse of $t(\gamma_{01})$), and hence this definition doesn't work in the category model. But notice the following: the term t is in the "negative" position (the domain) and the term t' is in the "positive" position. Fittingly, we only use the *inverse* of $t(\gamma_{01})$ – never $t(\gamma_{01})$ itself – and only use $t'(\gamma_{01})$ but not its inverse. This observation will provide the key to adapting this definition for the category model.

What is needed is for t to be a *contravariant* term of type A, while keeping t' as *covariant*. We can treat this variance in the type theory of the category model, using a fundamental construct from category theory: *opposite categories.* A type $A\colon \mathsf{Ty}\ \Gamma$ in the category model consists of a family of categories $A(\gamma)$ for each object $\gamma\colon\ |\Gamma|$ and a functor $A(\gamma_{01})\colon A(\gamma_0) \Rightarrow A(\gamma_1)$ for each morphism $\gamma_{01}\colon \Gamma[\gamma_0, \gamma_1]$. Given such a family of categories A, we can form a new family A^-, where $A^-(\gamma)$ is defined as the opposite category of $A(\gamma)$. This extends to the morphism part as well, because any functor $f\colon C \Rightarrow D$ can be viewed as a functor on their opposites, $f\colon C^{\mathrm{op}} \Rightarrow D^{\mathrm{op}}$. Alternatively, we could view A as a functor $\Gamma \Rightarrow \mathsf{Cat}$, and define A^- to be the composition of A with the endofunctor $(\_)^{\mathrm{op}}\colon \mathsf{Cat} \Rightarrow \mathsf{Cat}$. We can state generally that the category model validates the following rule:

$$\frac{A\colon \mathsf{Ty}\ \Gamma}{A^-\colon \mathsf{Ty}\ \Gamma.}$$

If $t\colon \mathsf{Tm}(\Gamma, A^-)$, this means that the object part of t will still send objects $\gamma\colon\ |\Gamma|$ to objects of $A(\gamma)$, since $A(\gamma)$ and $A^-(\gamma)$ have the same objects. But observe the type of its morphism part:

$$t\colon (\gamma_{01}\colon \Gamma\ [\gamma_0, \gamma_1]) \to (A\ \gamma_1)\ [t\ \gamma_1, A\ \gamma_{01}\ (t\ \gamma_0)].$$

This is precisely what we need to give the definition of hom-types in the category model: see Figure 2. This definition is almost exactly the same as the semantics of Id in the groupoid model, but with t changed to be a term of A^-, thus eliminating the need for the categories $A(\gamma_i)$ to be groupoids. Here's the hom-type formation, expressed as a rule:

$$\frac{t\colon \mathsf{Tm}(\Gamma, A^-) \quad t'\colon \mathsf{Tm}(\Gamma, A)}{\mathsf{Hom}(t, t')\colon \mathsf{Ty}\ \Gamma.}$$

The type annotation of t as a "negative" term and the implicit annotation of t' as "positive" serve as a kind of *modal typing discipline* for keeping track of the *variances* of terms.

For now, we just state the formation rule for hom-types; introducing and eliminating terms of hom-types will require more machinery. To see what kind of machinery, let's instead consider dependent function types. Like with the formation of hom-types, Π-types involve

```
$\mathsf{Hom} : \mathsf{Tm}(\Gamma,\ \mathsf{A}^-) \to \mathsf{Tm}(\Gamma,\ \mathsf{A}) \to \mathsf{Ty}\ \Gamma$
  $(\mathsf{Hom}(\mathsf{t},\mathsf{t}'))\ \gamma = (\mathsf{A}\ \gamma)\ [\mathsf{t}\ \gamma,\ \mathsf{t}'\ \gamma]$ -- Discrete category

  $(\mathsf{Hom}(\mathsf{t},\mathsf{t}'))\ (\gamma_{01} : \Gamma[\ \gamma_0\ ,\ \gamma_1\ ]) : (\mathsf{A}\ \gamma_0)\ [\mathsf{t}\ \gamma_0,\ \mathsf{t}'\ \gamma_0] \to (\mathsf{A}\ \gamma_1)\ [\mathsf{t}\ \gamma_1,\ \mathsf{t}'\ \gamma_1]$
  $(\mathsf{Hom}(\mathsf{t},\mathsf{t}'))\ \gamma_{01}\ \mathsf{x}_0 = (\mathsf{t}'\ \gamma_{01}) \circ (\mathsf{A}\ \gamma_{01}\ \mathsf{x}_0) \circ (\mathsf{t}\ \gamma_{01})$
```

Figure 2 Semantics of the Hom-type former in the category model.

positive and negative "variance": a function is contravariant in its argument and covariant in its result. Therefore, as we might expect, the interpretation of Π-types in the groupoid model ([17, Section 4.6]) makes essential use of the invertibility of morphisms in a groupoid. Again, it only comes into play when defining the *morphism* part of the interpretation: the object part (reproduced in Figure 4) defines for each $\gamma\colon |\Gamma|$ an auxiliary type B_γ in context $A(\gamma)$, and then specifies the category $\Pi(A, B)\ \gamma$ with terms $\theta\colon \mathsf{Tm}(A(\gamma), B_\gamma)$ as objects. This works fine in the category model. However, defining the morphism part of $\Pi(A, B)$ requires a kind of negative variance *deeper* than the shallow contravariance of A^-: in the type $\mathsf{Hom}(t, t')$ it was a *term* that occurred negatively (t), in the type $\Pi(A, B)$ it's a *type* (A) that occurs negatively.

To make sense of this, we must consider the *opposite category* operation, not just as acting on each $A(\gamma)$ in a family of categories over a context Γ, but as acting on the contexts themselves. In the category model, we have the following rules.

$$\frac{\Gamma\colon \mathsf{Con}}{\Gamma^-\colon \mathsf{Con}} \qquad \frac{\sigma\colon \mathsf{Sub}\ \Delta\ \Gamma}{\sigma^-\colon \mathsf{Sub}\ \Delta^-\ \Gamma^-}$$

That is, we can negate contexts and substitutions as well as types: Γ^- is interpreted as Γ^{op}, and this operation is (*co*variantly) lifted onto functors as before. Now consider the difference in the morphism parts of terms with these different kinds of variance.

```
-- $\mathsf{t} : \mathsf{Tm}(\Gamma,\ \mathsf{A})$ where $\mathsf{A} : \mathsf{Ty}\ \Gamma$
$\mathsf{t}\ \gamma_{01} : (\mathsf{A}\ \gamma_1)\ [\ \mathsf{A}\ \gamma_{01}\ (\mathsf{t}\ \gamma_0),\ \mathsf{t}\ \gamma_1\ ]$
-- $\mathsf{t} : \mathsf{Tm}(\Gamma,\ \mathsf{A}^-)$ where $\mathsf{A} : \mathsf{Ty}\ \Gamma$
$\mathsf{t}\ \gamma_{01} : (\mathsf{A}\ \gamma_1)\ [\ \mathsf{t}\ \gamma_1,\ \mathsf{A}\ \gamma_{01}\ (\mathsf{t}\ \gamma_0)\ ]$
-- $\mathsf{t} : \mathsf{Tm}(\Gamma^-,\ \mathsf{A})$ where $\mathsf{A} : \mathsf{Ty}\ \Gamma^-$
$\mathsf{t}\ \gamma_{01} : (\mathsf{A}\ \gamma_0)\ [\ \mathsf{t}\ \gamma_0,\ \mathsf{A}\ \gamma_{01}\ (\mathsf{t}\ \gamma_1)\ ]$
```

This is why we referred to this as "shallow" and "deep" negation: the difference between the first two is that we've flipped around each $A(\gamma)$, whereas in the third term, the dependence of A on Γ has itself been flipped around (A is now contravariant, so $A\ \gamma_{01}$ takes objects of $A(\gamma_1)$ to objects of $A(\gamma_0)$). It is this latter kind of contravariance that describes A's position in $\Pi(A, B)$.

With this in mind, we might try to state the Π-formation rule as follows

$$\frac{A\colon \mathsf{Ty}(\Gamma^-) \quad B\colon \mathsf{Ty}(\Gamma^- \rhd A)}{\Pi(A, B)\colon \mathsf{Ty}\ \Gamma.}$$

However, this doesn't get the variances quite right. The type A depends negatively on Γ, as desired, but so does B: the morphism part of some $B\colon \mathsf{Ty}(\Gamma^- \rhd A)$ has shape

$$B\colon (\gamma_{01}\colon \Gamma^-\ [\gamma_1, \gamma_0]) \to (x_0\colon (A\ \gamma_0)\ [A\ \gamma_{01}\ a_1, a_0]) \to B(\gamma_1, a_1) \Rightarrow B(\gamma_0, a_0).$$

This is too much contravariance. Semantically, to define the morphism part of $\Pi(A, B)$ – as we do precisely in [26] – we need B to depend *covariantly* on Γ, but A to depend contravariantly on Γ. Syntactically, this matches our understanding of A being in "negative position" and B "positive position" in the type $\Pi(A, B)$.

To achieve the desired arrangement of variances in the (modified) CwF we're building, we must introduce not one, but *two* notions of context extension. The first, or positive context extension, is the usual one: when $A\colon \mathsf{Ty}\ \Gamma$ then Γ may be covariantly extended by A. As explicitly spelled out in Figure 3, the resulting context (which we'll henceforth denote $\Gamma \rhd^{+} A$, and use $\langle \_,_{+} \_\rangle$ for its associated pairing operation and p_{+} for the substitution from $\Gamma \rhd^{+} A$ to Γ), is interpreted in the category model by a Grothendieck construction for the category-valued covariant functor A. But if instead $A\colon \mathsf{Ty}\ \Gamma^{-}$, then, instead of forming the context $\Gamma^{-} \rhd^{+} A$ – which, as discussed above, has the wrong arrangement of variances – we could instead form $\Gamma \rhd^{-} A$, the negative context extension by A. A type B in context $\Gamma \rhd^{-} A$ would have our desired variances for Π-types: B depends covariantly on both Γ, even though A depends contravariantly on Γ.

More concretely, the negative context extension operator satisfies the following "*modal* local representability", a version of the local representability condition required of the context extension operator of a CwF, which incorporates context negation, type negation, and negative context extension:

$$\mathsf{Sub}\ \Delta\ (\Gamma \rhd^{-} A) \cong (\sigma\colon \mathsf{Sub}\ \Delta\ \Gamma) \times (\mathsf{Tm}(\Delta^{-}, A[\sigma^{-}]^{-})) \tag{1}$$

for any $A\colon \mathsf{Ty}\ \Gamma^{-}$. We'll write $\langle \_,_{-} \_\rangle$ for the right-to-left direction of this isomorphism, and write $\mathsf{p}_{-,A}\colon \mathsf{Sub}\ (\Gamma \rhd^{-} A)\ \Gamma$ and $\mathsf{v}_{-,A}\colon \mathsf{Tm}((\Gamma \rhd^{-} A)^{-}, A[\mathsf{p}_{-,A}^{-}]^{-})$ for the data obtained from applying the left-to-right direction to the identity morphism on $\Gamma \rhd^{-} A$.

Notice that the σ on the left is from Δ to Γ (not Γ^{-}), and that a $\Gamma \rhd^{-} A$ morphism from (γ_0, a_0) to (γ_1, a_1) consists of a Γ-morphism from γ_0 to γ_1 (not γ_1 to γ_0). This is what we mean when we say types and terms in $\Gamma \rhd^{-} A$ depend positively on Γ, but A depends negatively on Γ: a substitution into $\Gamma \rhd^{-} A$ must consist of a substitution into Γ – not Γ^{-} – paired with a context- *and* type-negated term. Though perhaps initially daunting, the number of negations on the right-hand side become much less of a problem later on, since for our practical purpose we'll restrict to the case where Γ is a groupoid.

With this, we have everything needed to give the semantics of the Π-type former; this is partly done in Figure 4 – see [26, Appendix A] for full detail. The formation rule is

$$\frac{A\colon \mathsf{Ty}(\Gamma^{-}) \quad B\colon \mathsf{Ty}(\Gamma \rhd^{-} A)}{\Pi(A, B)\colon \mathsf{Ty}\ \Gamma} \tag{2}$$

As expected for Π-types, we have an isomorphism between $\mathsf{Tm}(\Gamma, \Pi(A, B))$ and $\mathsf{Tm}(\Gamma \rhd^{-} A, B)$, the application and lambda-abstraction rules. This is omitted here for reasons of space, but included in the appendix of [26], along with accompanying calculations.

Let's now return to the key feature of directed type theory, hom-types. Above, we gave just the formation rule for hom-types, but said nothing of how to introduce or eliminate terms of this type. Stating the introduction rule, the term refl inhabiting $\mathsf{Hom}(t, t)$ for each term t proves rather subtle. The difficulty stems from the following *mixed-variance problem*: since our formation rule demands the domain term t be of type A^{-} and the codomain term t' to be of type A, it's not immediately clear how to make $\mathsf{Hom}(t, t)$ well-formed. There is, in general, no way to coerce terms of type A into terms of type A^{-} or vice versa, and we have no rule permitting us to use a term in both variances.

$$\begin{array}{l}
\_\triangleright^{+}\_ : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to \mathsf{Con} \\
\quad |\Gamma \triangleright^{+} \mathsf{A}| = (\gamma : |\Gamma|) \times |\mathsf{A}\ \gamma| \\
\quad (\Gamma \triangleright^{+} \mathsf{A})\ [\ (\gamma_0,\ \mathsf{a}_0)\ ,\ (\gamma_1,\ \mathsf{a}_1)\ ] = (\gamma_{01} : \Gamma\ [\ \gamma_0,\ \gamma_1\ ]) \times (\mathsf{A}\ \gamma_1)\ [\ \mathsf{A}\ \gamma_{01}\ \mathsf{a}_0,\ \mathsf{a}_1\ ] \\
\\
\_\triangleright^{-}\_ : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma^{-} \to \mathsf{Con} \\
\quad |\Gamma \triangleright^{-} \mathsf{A}| = (\gamma : |\Gamma|) \times |\mathsf{A}\ \gamma| \\
\quad (\Gamma \triangleright^{-} \mathsf{A})\ [\ (\gamma_0,\ \mathsf{a}_0)\ ,\ (\gamma_1,\ \mathsf{a}_1)\ ] = (\gamma_{01} : \Gamma\ [\ \gamma_0,\ \gamma_1\ ]) \times (\mathsf{A}\ \gamma_1)\ [\ \mathsf{a}_0\ ,\ \mathsf{A}\ \gamma_{01}\ \mathsf{a}_1\ ]
\end{array}$$

Figure 3 Semantics of context extension in the category model.

$$\begin{array}{l}
\Pi : (\mathsf{A} : \mathsf{Ty}\ \Gamma^{-}) \to \mathsf{Ty}(\Gamma \triangleright^{-} \mathsf{A}) \to \mathsf{Ty}\ \Gamma \\
\quad |\Pi(\mathsf{A},\mathsf{B})\ \gamma| = \mathsf{Tm}(\mathsf{A}(\gamma),\ \mathsf{B}_{\gamma}) \\
\qquad \textbf{where} \\
\qquad\quad \mathsf{B}_{\gamma} : \mathsf{Ty}(\mathsf{A}\ \gamma) \\
\qquad\qquad \mathsf{B}_{\gamma}\ \mathsf{a} = \mathsf{B}(\gamma,\mathsf{a}) \\
\qquad\qquad \mathsf{B}_{\gamma}\ (\mathsf{x} : (\mathsf{A}\ \gamma)[\ \mathsf{a}\ ,\ \mathsf{a}'\ ]) = \mathsf{B}(\mathsf{id}_{\gamma},\mathsf{x})
\end{array}$$

Figure 4 Semantics of the Π-type former in the category model.

A solution which avoids this shortcoming is revealed by considering hom-types in the empty context. In the empty context, a type A is the same thing[4] as a category, and a term of type A is the same thing[5] as an object of type A. So then there's no difference between terms of type A and terms of type A^{-}, since a category and its opposite have the same objects. Therefore, in the empty context, there is no mixed-variance problem, and we can state the introduction rule for refl_t simply by coercing t to be positive and negative as needed.

This doesn't extend to arbitrary contexts: as we saw above, terms $t\colon \mathsf{Tm}(\Gamma, A^{-})$ and $t'\colon \mathsf{Tm}(\Gamma, A)$ have different morphism parts. But here's the key observation: if Γ is a *groupoid*, then we can still coerce between A and A^{-}: given $t\colon \mathsf{Tm}(\Gamma, A^{-})$, we can obtain $-t\colon \mathsf{Tm}(\Gamma, A)$, and vice-versa. The definition is given in Figure 5; there (and henceforth), we use $\Gamma\colon \mathsf{NeutCon}$ to indicate that Γ is a groupoid, and therefore can invert Γ-morphisms as needed. So, rather than introduce a new type A^0 whose terms can be either positive or negative, we instead have identified those contexts – neutral contexts – where terms of the familiar types A and A^{-} can be inter-converted. Given this, we can introduce refl:

$$\frac{\Gamma\colon \mathsf{NeutCon} \quad A\colon \mathsf{Ty}\ \Gamma \quad t\colon \mathsf{Tm}(\Gamma, A^{-})}{\mathsf{refl}_t\colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, -t)).}$$

We only need to assert refl_t for t of type A^{-}, because the analogous rule for t' of type A can be derived: given $t'\colon \mathsf{Tm}(\Gamma, A)$, we observe that $t' = -(-t')$, so $\mathsf{refl}_{-t'}\colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t', t'))$.

The solution to this problem proposed by North [28] is not groupoid *contexts*, rather to use *(core) groupoid types*. This solution consists of asserting a new type A^0 for each A (interpreted in the category model as the (fiberwise) "core groupoid" of A), equipped with coercions $\mathsf{Tm}(\Gamma, A^0) \to \mathsf{Tm}(\Gamma, A)$ and $\mathsf{Tm}(\Gamma, A^0) \to \mathsf{Tm}(\Gamma, A^{-})$. Then, for a term

[4] Silently coerce along $\mathbb{1} \Rightarrow \mathsf{Cat} \cong \mathsf{Cat}$.

[5] Silently coerce along $\mathbb{1} \Rightarrow A \cong |A|$ for any category A.

$$
\begin{aligned}
&- : \{\Gamma : \mathsf{NeutCon}\}\{\mathsf{A} : \mathsf{Ty}\ \Gamma\} \to \mathsf{Tm}(\Gamma,\mathsf{A}) \to \mathsf{Tm}(\Gamma,\mathsf{A}^-)\\
&\quad -\mathsf{t}'\ \gamma = \mathsf{t}'\ \gamma\\
&\quad -\mathsf{t}'\ \gamma_{01} = \mathsf{A}\ \gamma_{01}\ (\mathsf{t}'({\gamma_{01}}^{-1}))\\
&- : \{\Gamma : \mathsf{NeutCon}\}\{\mathsf{A} : \mathsf{Ty}\ \Gamma\} \to \mathsf{Tm}(\Gamma,\mathsf{A}^-) \to \mathsf{Tm}(\Gamma,\mathsf{A})\\
&\quad -\mathsf{t}\ \gamma = \mathsf{t}\ \gamma\\
&\quad -\mathsf{t}\ \gamma_{01} = \mathsf{A}\ \gamma_{01}\ (\mathsf{t}({\gamma_{01}}^{-1}))
\end{aligned}
$$

Figure 5 Semantics of neutral-context coercion in the category model.

$t\colon \mathsf{Tm}(\Gamma, A^0)$, it makes sense to write $\mathsf{Hom}(t,t)$, as t can be coerced to both the positive and negative modality, in order to fit the Hom formation rule. From there, a directed J-rule[6] can be stated for eliminating hom terms. The issue with this solution is that it forces homs to have core endpoints: the directed J-rule can *only* be used to prove claims about homs anchored at a term of type A^0 – the only terms t for which refl_t can even be formed. *A priori*, it's not clear that proofs about arbitrary homs can be made with such a J-rule (recall that this was a key desideratum, number 4, for the present theory).

North's solution has the advantage of working in an arbitrary, non-groupoid context. But we find that the restriction to neutral contexts is needed anyway: as developed in [25, Chap. 2], the Π-types of the category model are generally unworkable outside of a groupoid context. Below, when we define the identity function and the composition of functions, notice that we make use of the groupoid-context facilities to do so. Moreover, characterizing the opposites and hom-types of Σ-types, defining the morphism part of functions as synthetic functors, and working with the universe of sets also need a groupoid context to work. So we might as well use this assumption to state refl too. We suspect that, if we work in a groupoid context but *also* have access to North's core types, then our rule of directed path induction (see below) becomes equivalent to North's, so the theories are fundamentally compatible.

Let's conclude this section by giving an eliminator for our hom-type, known as the *directed J-rule* or *directed path induction*. Following [17, Section 4.10], we study directed path induction in the empty context first, which can then be extended to an arbitrary neutral context. Given $A\colon \mathsf{Ty}\ \bullet$ and $t\colon \mathsf{Tm}(\bullet, A^-)$ and some $M\colon \mathsf{Ty}(\bullet \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}_A], \mathsf{v}))$, our goal is to be able to prove $M[\mathsf{id},_+ t',_+ f]$ for arbitrary t' and f, just by supplying a term m of $M[-t, \mathsf{refl}_t]$. Translated into the category model semantics: A is a category, t and t' are objects of A, M is a functor from the coslice category t/A into Cat, f is an A-morphism from t to t', and m is an object of the category $M(t, \mathsf{id}_t)$. The key observation is that f is then *also* a morphism in the coslice category from (t, id) to (t', f).

$$\begin{tikzcd}
 & t \arrow[dl, "\mathsf{id}_t"'] \arrow[dr, "f"] & \\
t \arrow[rr, "f"'] & & t'.
\end{tikzcd}$$

Therefore,

$$M(f)\colon M(t, \mathsf{id}_t) \Rightarrow M(t', f)$$

and so the object part of this functor turns objects of $M(t, \mathsf{id}_t)$ into objects of $M(t', f)$, that is, it turns terms $m\colon \mathsf{Tm}(\bullet, M[\mathsf{id},_+ -t,_+ \mathsf{refl}])$ into terms

$$(\mathsf{J}_{t,M}\ m)\ [\mathsf{id},_+ t',_+ f]\colon \mathsf{Tm}(\bullet, M[\mathsf{id},_+ t',_+ f]).$$

[6] Or, rather, two directed J-rules

And, since $M(\mathsf{id})$ is the identity functor, we have the β law, saying that $\mathsf{J}_{t,M}\ m\,[\mathsf{id},_+ -t,_+ \mathsf{refl}_t] \equiv m$. The general law replaces $\bullet$ with an arbitrary neutral context:

$$\frac{\begin{array}{cc} \Gamma : \mathsf{NeutCon} & A : \mathsf{Ty}\,\Gamma \\ t : \mathsf{Tm}(\Gamma, A^-) & M : \mathsf{Ty}(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}_A], \mathsf{v})) \\ m : \mathsf{Tm}(\Gamma, M[\langle \mathsf{id},_+ -t,_+ \mathsf{refl}_t \rangle]) & \end{array}}{\mathsf{J}_{t,M}\ m : \mathsf{Tm}(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}_A], \mathsf{v}), M)} \tag{3}$$

but the category model interpretation – see Figure 6 – essentially follows this same idea. If M doesn't need to depend on the term of type $\mathsf{Hom}(t, \mathsf{v})$, then we can instead use the simpler rule

$$\frac{\begin{array}{ccc} \Gamma : \mathsf{NeutCon} & A : \mathsf{Ty}\,\Gamma & \\ t : \mathsf{Tm}(\Gamma, A^-) & M : \mathsf{Ty}(\Gamma \rhd^+ A) & m : \mathsf{Tm}(\Gamma, M[\mathsf{id},_+ -t]) \end{array}}{\mathsf{J}_{t,M}\ m : \mathsf{Tm}(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}_A], \mathsf{v}), M[\mathsf{p}_{\mathsf{Hom}(t[\mathsf{p}_A],\mathsf{v})}])} \tag{4}$$

In Section 4 we put this rule to use in synthetic category theory constructions and proofs.

$$\begin{array}{l}
\mathsf{J} : (\mathsf{t} : \mathsf{Tm}(\Gamma,\ \mathsf{A}^-)) \to (\mathsf{M} : \mathsf{Ty}\ (\Gamma \rhd^+ \mathsf{A} \rhd^+ \mathsf{Hom}(\mathsf{t}[\mathsf{p}_A],\mathsf{v}))) \\
\quad \to \mathsf{Tm}(\Gamma,\ \mathsf{M}[\langle \mathsf{id},_+ -\mathsf{t},_+ \mathsf{refl}_\mathsf{t} \rangle]) \to \mathsf{Tm}(\Gamma \rhd^+ \mathsf{A} \rhd^+ \mathsf{Hom}(\mathsf{t}[\mathsf{p}_A],\mathsf{v}),\ \mathsf{M}) \\
\quad (\mathsf{J}_{\mathsf{t,M}}\ \mathsf{m}) : (\gamma : |\Gamma|) \to (\mathsf{a} : |\mathsf{A}\ \gamma|) \to (\mathsf{x} : (\mathsf{A}\ \gamma)\ [\mathsf{t}\ \gamma\ ,\ \mathsf{a}]) \to |\ \mathsf{M}(\gamma,\ \mathsf{a},\ \mathsf{x})\ | \\
\quad (\mathsf{J}_{\mathsf{t,M}}\ \mathsf{m})\ \gamma\ \mathsf{a}\ \mathsf{x} = \mathsf{M}\ (\mathsf{id}_\gamma,\ \mathsf{x},\ \rho)\ (\mathsf{m}\ \gamma) \quad \text{-- } \rho : x \circ A\ id_\gamma\ id_{\mathsf{t}\gamma} \circ t\ id_\gamma \equiv x \\
\\
\quad (\mathsf{J}_{\mathsf{t,M}}\ \mathsf{m}) : (\gamma_{01} : \Gamma\ [\gamma_0,\gamma_1]) \to (\mathsf{a}_{01} : \mathsf{A}\gamma_1\ [\mathsf{A}\ \gamma_{01}\ \mathsf{a}_0,\ \mathsf{a}_1]) \\
\quad \to (\varphi_{01} : \mathsf{a}_{01} \circ (\mathsf{A}\ \gamma_{01}\ \mathsf{x}_0) \circ \mathsf{t}(\gamma_{01}) \equiv \mathsf{x}_1) \\
\quad \to \mathsf{M}(\gamma_1,\ \mathsf{a}_1,\ \mathsf{x}_1)[\ \mathsf{M}(\gamma_{01},\mathsf{a}_{01},\varphi_{01})\ ((\mathsf{J}_{\mathsf{t,M}}\ \mathsf{m})\ \gamma_0\ \mathsf{a}_0\ \mathsf{x}_0),\ ((\mathsf{J}_{\mathsf{t,M}}\ \mathsf{m})\ \gamma_1\ \mathsf{a}_1\ \mathsf{x}_1)\ ] \\
\quad (\mathsf{J}_{\mathsf{t,M}}\ \mathsf{m})\ \gamma_{01}\ \mathsf{a}_{01}\ \rho_{01} = \mathsf{M}(\mathsf{id}_{\gamma_1},\ \mathsf{x}_1,\ \rho_1)\ (\mathsf{m}\ \gamma_{01}) \text{-- } \rho_1 : x_1 \circ A\ id_{\gamma_1}\ id_{\mathsf{t}\gamma_1} \circ t\ id_{\gamma_1} \equiv x_1 \\
\\
\mathsf{J}\beta : (\mathsf{J}_{\mathsf{t,M}}\mathsf{m})\ [\langle \mathsf{id},_+ -\mathsf{t},_+ \mathsf{refl}_\mathsf{t} \rangle] \equiv \mathsf{m}
\end{array}$$

Figure 6 Semantics of directed path induction in the category model.

3 Directed Categories with Families

The aim of the present work is not just to establish the category model as a suitable interpretation of directed type theory, but to abstract the category model to a general, abstract notion of 'model' of directed type theory. Specifically, we wish to present this model notion as a generalized algebraic theory, that is, as a CwF with further structure. We do so in several stages, progressively capturing more of the structure described in the previous section. In addition to making the complex and multifaceted notion of 'directed CwF' more digestible, this approach will also give us several intermediate notions, each of which is worthy of further study in its own right. First, we encapsulate the 'negation' structure.

▶ **Definition 1** (Polarized CwF). *A* ***polarized category with families (PCwF)*** *consists of a CwF $\mathcal{C} = (\mathsf{Con}, \mathsf{Ty}, \mathsf{Tm}, \rhd^+, \ldots)$ equipped with the following operations.*

- *An endofunctor $(\_)^- : \mathsf{Con} \to \mathsf{Con}$ such that $(\Gamma^-)^- \equiv \Gamma$ and $(\sigma^-)^- \equiv \sigma$ for all Γ and σ*
- *A natural transformation $(\_)^- : \mathsf{Ty} \to \mathsf{Ty}$ such that $(A^-)^- \equiv A$ for all A.*

So a PCwF is just a CwF equipped with context-, substitution-, and type-negation involutions. The fact that the type-negation operation is a natural transformation just says that it is stable under substitution, i.e. $A[\sigma]^{-} \equiv A^{-}[\sigma]$. Now, notably absent from this definition is the negative context extension operation $\triangleright^{-}$; by this definition, a PCwF only has the positive one. This is because the negative operation is, in fact, definable: in the category model, the following equation holds for any Γ and any $A \colon \mathsf{Ty}\ \Gamma$:

$$(\Gamma \triangleright^{+} A)^{-} \equiv \Gamma^{-} \triangleright^{-} A^{-}. \tag{5}$$

Here we use the fact that $(\Gamma^{-})^{-} \equiv \Gamma$, and hence $A \colon \mathsf{Ty}\ (\Gamma^{-})^{-}$, making the right-hand side well-formed. Consequently, we can turn this equation around to *define* negative context extension: for $A \colon \mathsf{Ty}(\Gamma^{-})$, let $\Gamma \triangleright^{-} A$ be $(\Gamma^{-} \triangleright^{+} A^{-})^{-}$. The isomorphism characterizing $\triangleright^{-}$ (Equation 1) can then be proved as a consequence of the one for $\triangleright^{+}$.

Also absent from Definition 1 is any mechanism connecting the context/substitution negation endofunctor to the type-negation operation. It's unclear if this ought to be rectified, or if their connection is just a peculiarity of the category model. Not every CwF fits the same mold of "contexts are structures, types are families of structures", so it's not possible to require in general that the type-negation operation is just post-composition with the context-negation functor. There *are* suitably abstract ways of connecting the two – for instance, we can note that the category model is *democratic* in the sense of [10, Defn. 3]: there is an isomorphism K between contexts Γ and closed types; this isomorphism is compatible with both negation operations, in that $K(\Gamma^{-}) = K(\Gamma)^{-}$. However, we don't need need such strong assumptions for the results of Section 4, so we omit them from the general definition of PCwFs.

Of course, the category model and the preorder model are both examples of PCwFs, where the negation is the 'opposite' construction. But so are the groupoid and setoid models. Indeed, the groupoid model is a *sub-PCwF* of the category model: a groupoid Γ is a category, and so it makes perfect sense to take the opposite category of Γ, obtaining Γ^{-}, which is also a groupoid. What makes the groupoid model a peculiar instance of a PCwF is that $\Gamma \cong \Gamma^{-}$ for every Γ, and $A(\gamma) \cong A^{-}(\gamma)$ for every A. It is what we'll call a *symmetric PCwF*. The setoid model is also a symmetric PCwF, but strictly so: there, $\Gamma \equiv \Gamma^{-}$. The situation exemplified by the groupoid/category and setoid/preorder models – a symmetric sub-PCwF of another PCwF – is what we capture in our next notion.[7]

▶ **Definition 2** (Neutral-Polarized CwF). *A* ***sub-PCwF*** [8] $\mathcal{D}$ *of a PCwF* $\mathcal{C}$ *consists of predicates*[9] $\mathsf{D}_{\mathsf{Con}} \colon \mathsf{Con} \to \mathsf{Prop}$ *and* $\mathsf{D}_{\mathsf{Ty}} \colon \{\Gamma \colon \mathsf{Con}\} \to \mathsf{Ty}\ \Gamma \to \mathsf{Prop}$ *such that*

- $\mathsf{D}_{\mathsf{Con}}\ \bullet$*;*
- *if* $\mathsf{D}_{\mathsf{Con}}(\Gamma)$*, then* $\mathsf{D}_{\mathsf{Con}}(\Gamma^{-})$*;*
- *if* $\mathsf{D}_{\mathsf{Ty}}(A)$*, then* $\mathsf{D}_{\mathsf{Ty}}(A^{-})$*;*
- *if* $A \colon \mathsf{Ty}\ \Gamma$ *is such that* $\mathsf{D}_{\mathsf{Ty}}(A)$*, then* $\mathsf{D}_{\mathsf{Ty}}(A[\sigma])$ *for any* $\sigma \colon \mathsf{Sub}\ \Delta\ \Gamma$*; and*
- *if* $\mathsf{D}_{\mathsf{Con}}(\Gamma)$ *and* $\mathsf{D}_{\mathsf{Ty}}(A)$*, then* $\mathsf{D}_{\mathsf{Con}}(\Gamma \triangleright A)$*.*

[7] The definition given here is, admittedly, somewhat *ad hoc*. These are just the constructs we'll need in order to make the subsequent definitions and synthetic category theory development proceed. A more detailed explication of this definition is given in [25, Chap. 2].

[8] A more descriptive name might be "full and *locally full* sub-PCwF": it's a *full subcategory* $\mathsf{DCon} \hookrightarrow \mathsf{Con}$ in that $\mathsf{DCon}\ [\Delta, \Gamma] = \mathsf{Sub}\ \Delta\ \Gamma$, but also "locally full" in the sense that $\mathsf{Tm}_{\mathcal{D}}(\Gamma, A) = \mathsf{Tm}_{\mathcal{C}}(\Gamma, A)$ for $\Gamma \colon \mathsf{DCon}$ and $A \colon \mathsf{DTy}(\Gamma)$.

[9] The first part of this definition articulates a notion of "Prop-valued logical predicate" $\mathsf{D} = (\mathsf{D}_{\mathsf{Con}}, \mathsf{D}_{\mathsf{Sub}}, \mathsf{D}_{\mathsf{Ty}}, \mathsf{D}_{\mathsf{Tm}}, \ldots)$ on the PCwF $\mathcal{C}$, except we don't mention the components $\mathsf{D}_{\mathsf{Sub}}$ and D_{Tm} because fullness and local fullness means that $\mathsf{D}_{\mathsf{Sub}}$ and D_{Tm} (respectively) are *always satisfied.*

We indicate a sub-PCwF by $\mathcal{D} = (\mathsf{DCon}, \mathsf{DTy})$ *to indicate that* DCon *is the subcategory of* $\mathsf{D}_{\mathsf{Con}}$*-contexts, and* DTy *is the subpresheaf of* D_{Ty}*-types.*

A ***neutral-polarized category with families (NPCwF)*** *consists of a PCwF* $\mathcal{C}$ *and a sub-PCwF* $\mathcal{N} = (\mathsf{NeutCon}, \mathsf{NeutTy})$ *such that*

- $\mathsf{NeutCon}$ *is symmetric: every* $\Gamma\colon \mathsf{NeutCon}$ *comes equipped with an* $\mathsf{e}\colon \mathsf{Sub}\ \Gamma^-\ \Gamma$ *such that* $\mathsf{e}^-\colon \mathsf{Sub}\ \Gamma\ \Gamma^-$ *is an inverse of* e*; moreover, this isomorphism is natural in the sense that, for* $\Delta, \Gamma\colon \mathsf{NeutCon}$ *and* $\sigma\colon \mathsf{Sub}\ \Delta\ \Gamma$*, we have* $\sigma \equiv \mathsf{e}_\Gamma \circ \sigma^- \circ \mathsf{e}_\Delta^-$*;*
- *if* Γ *is neutral and* $A\colon \mathsf{Ty}\ \Gamma$*, then there is a coercion operation* $-\colon \mathsf{Tm}(\Gamma, A^-) \to \mathsf{Tm}(\Gamma, A)$ *such that* $-(-t) \equiv t$ *for all* t *and such that* $(-t)[\sigma] \equiv -(t[\sigma])$ *for any* $\sigma\colon \mathsf{Sub}\ \Delta\ \Gamma$ *where* Δ *is neutral;*
- *for every* $\Gamma\colon \mathsf{NeutCon}$ *and* $A\colon \mathsf{Ty}\ \Gamma$*,there is an isomorphism*

$$\mathsf{ee}\colon \Gamma \triangleright^- A[\mathsf{e}] \cong \Gamma \triangleright^+ A$$

  such that
  - $\mathsf{p}_{+,A} \circ \mathsf{ee} \equiv \mathsf{p}_{-,A[\mathsf{e}]}$
  - *for every* $\Delta, \Gamma\colon \mathsf{NeutCon}$*,* $\sigma\colon \mathsf{Sub}\ \Delta\ \Gamma$*,* $A\colon \mathsf{Ty}\ \Gamma$ *and every* $s\colon \mathsf{Tm}(\Delta, A[\sigma])$*,*

$$\mathsf{ee} \circ \langle \sigma,_+ s\rangle \equiv \langle \sigma,_- (-s)[\mathsf{e}]\rangle\ . \tag{6}$$

To summarize: an NPCwF is a PCwF with a (locally) full sub-PCwF of "neutral" contexts and types, which come equipped with machinery for overcoming the polarity calculus (coercing between the context and its opposite, and between a type and its opposite). Our goal is to talk about the category model: we want to take the category theoretic statement "Grpd is a full subcategory of Cat" and extend it to a statement about the semantics of polarized type theory: "the groupoid model is a (full and locally full) symmetric sub-PCwF of the category model". This is what we do in Definition 2: an "NPCwF" is PCwF equipped with an appropriate symmetric sub-PCwF.

Let us also note that a common feature in directed type theories (e.g. [29, 28]) is to include *core types*, i.e. an operation of the form $(\_)^0\colon \mathsf{Ty}\ \Gamma \to \mathsf{NeutTy}\ \Gamma$. In the category model, this is interpreted as applying the *core groupoid* construction to each category $A(\gamma)$, producing a family of groupoids indexed over Γ. We might as well have a deep version too, and consider the core groupoid construction as operating on contexts $(\_)^0\colon \mathsf{Con} \to \mathsf{NeutCon}$ too. So the operation on *contexts* is the coreflector, the right adjoint, of the inclusion $\mathsf{NeutCon} \hookrightarrow \mathsf{Con}$; the operation on *types* is a "local coreflector", a dependent right adjoint [7] to $\mathsf{NeutTy} \hookrightarrow \mathsf{Ty}$. What this means in the category model (equipped with core types) is that $\mathsf{Tm}(\Gamma, A) \cong \mathsf{Tm}(\Gamma, A^0)$ for every $\Gamma\colon \mathsf{NeutCon}$ and $A\colon \mathsf{Ty}\ \Gamma$. So this is another coercion, between terms of A^0 and A, in addition to the NPCwF coercion operations $\mathsf{Tm}(\Gamma, A^-) \cong \mathsf{Tm}(\Gamma, A)$. This is why the above-mentioned concern about North's theory – that the directed J-rule is restricted to only core terms – is resolved in a neutral context, because in neutral contexts we can coerce freely between A, A^-, and A^0. We'll only need *neutral* contexts and types for the present work, not *core* contexts and types, but a thorough study of the latter is certainly needed in order to join the present theory to either North or to multi-modal type theory[13].

Recall that CwFs are not a single notion of model for a single type theory, but rather that CwFs encode the basic structural operations of type theory, upon which innumerable different type theories can be specified by defining the desired term- and type-formers. We have arrived at the same point in our development of a semantics for directed type theory: the notion of NPCwF consists solely of structural components, but nothing that actually allows for the construction of interesting types and terms. So let's rectify this by giving the directed analogue of the standard core of undirected type theory: identity types, dependent function types, and universes. We start with the directed analogue of identity types, hom-types.

▶ **Definition 3** (Directed CwF). *A* ***directed CwF (DCwF)*** *is a NPCwF equipped with the following structure:*

- *a type former*

$$\mathsf{Hom}\colon \{\Gamma\colon \mathsf{Con}\}\{A\colon \mathsf{Ty}\,\Gamma\} \to \mathsf{Tm}(\Gamma, A^-) \to \mathsf{Tm}(\Gamma, A) \to \mathsf{Ty}\,\Gamma$$

  which is stable under substitution:

$$\mathsf{Hom}(t, t')[\sigma] \equiv \mathsf{Hom}(t[\sigma], t'[\sigma]);$$

- *in any* $\Gamma\colon \mathsf{NeutCon}$*, a term* $\mathsf{refl}_t\colon \mathsf{Hom}(t, -t)$ *for each term* $t\colon \mathsf{Tm}(\Gamma, A^-)$*, also stable under substitution by* $\sigma\colon \mathsf{Sub}\;\Delta\;\Gamma$ *for* $\Delta\colon \mathsf{NeutCon}$*; and*
- *a term former* J *as given in Equation 3, also appropriately stable under substitution.*

For any $\Gamma\colon \mathsf{Con}$ *and* $A\colon \mathsf{NeutTy}\,\Gamma$*, write* $\mathsf{Id}(t, t')$ *for* $\mathsf{Hom}(t, t')$*.*[10]

The naming of Hom versus Id is suggestive: the types in a DCwF are supposed to function like synthetic categories (with Hom encoding their morphisms), and the neutral types are synthetic groupoids, whose homs are symmetric like an identity type. This point is best illustrated by the following claim.

▶ **Proposition 4.** *Every DCwF has an operation*

$$\begin{aligned}&\mathsf{symm}\colon \{\Gamma\colon \mathsf{NeutCon}\}\{A\colon \mathsf{NeutTy}\}\{t\colon \mathsf{Tm}(\Gamma, A^-)\}\{t'\colon \mathsf{Tm}(\Gamma, A)\}\\ &\qquad \to \mathsf{Tm}(\Gamma, \mathsf{Id}(t, t')) \to \mathsf{Tm}(\Gamma, \mathsf{Id}(-t', -t))\end{aligned}$$

Proof. By the following construction in the DCwF syntax:

```
symm : {Γ : NeutCon}{A : NeutTy Γ}{t : Tm(Γ, A^-)}{t' : Tm(Γ, A)}
   → Tm(Γ, Id(t, t')) → Tm(Γ, Id(−t', −t))
symm f = (J_{t,S} refl_t)[ id ,+ t',+ f ]  where
      S : Ty (Γ ▷+ A)
      S = Id(−v , −t)
```

◀

This proof relies on the neutrality of A in a very subtle, but critical way: in the definition of the type family S, the variable term $\mathsf{v}\colon \mathsf{Tm}(\Gamma \rhd^+ A, A[\mathsf{p}_A])$ is negated, so that it is of type $A[\mathsf{p}_A]^-$ and therefore able to stand as the first argument to Id. But this is only possible if $\Gamma \rhd^+ A\colon \mathsf{NeutCon}$, because term-negation is only defined in neutral contexts. This reasoning will prove important for the style of reasoning we employ in Section 4, so we isolate it as a principle.

▶ **Principle** (Var Neg). For $\Gamma\colon \mathsf{NeutCon}$, the variable term $\mathsf{v}\colon \mathsf{Tm}(\Gamma \rhd^+ A, A[\mathsf{p}_A])$ can only be negated (i.e. forming $-\mathsf{v}$) if $A\colon \mathsf{NeutTy}\,\Gamma$.

In Section 5, we'll argue that there's *no* way to construct this symmetry term (for arbitrary DCwFs[11]) if A is not assumed to be neutral.

Before proceeding, it's worth explaining what is "the DCwF syntax" mentioned in the proof above. This is where it becomes relevant that DCwFs are presented as generalized algebraic theories: as mentioned in the introduction, [18] proves that any GAT has an

[10] In general, we'll usually want to require that, if A is neutral, then so are its identity types. However, in the present theory, we'll require that *all* hom-types are neutral (see below).

[11] with some nontrivial amount of structure.

initial syntax model. Therefore, any construction done in the syntax model (such as the construction of symm above) can be interpreted into *any* DCwF. This is why a syntactic construction was adequate to prove a claim about *all* DCwFs in the foregoing proof. In the next section, our proofs will all be syntactic, and thereby apply to arbitrary DCwFs.

Let us make an important observation about the syntax of DCwFs. An important criterion for our theory is that hom-types can be *iterated*, that is, our syntax allows for the formation of homs between homs, and homs between homs between homs, and so on. The iteration of identity types is, after all, how homotopy type theory is able to serve as a synthetic language for higher groupoids; and since hom types are iterable in the DCwF syntax, it is a synthetic language for higher categories. However, a given model may be *truncated*, in that the higher structure may become trivial after a certain point. This is the case with the groupoid model: while its types do not all obey the *uniqueness of identity proofs (UIP)* principle and are therefore not mere *h-sets*, they do obey "UIP, one level up": in the groupoid model, identity proofs of identity proofs *are* unique.

The same happens in the category model: in general, there may be terms f of type $\mathsf{Hom}(t,t')^-$ and g of type $\mathsf{Hom}(t,t')$ but no term of type $\mathsf{Hom}(f,g)$ – the same way the groupoid model doesn't validate the uniqueness of *identities*, the category model doesn't validate the uniqueness of *homs*. But it does trivialize "one level up": the type $\mathsf{Hom}(f,g)$ may be either inhabited or uninhabited, sure, but $\mathsf{Hom}(f,g)$ is an h-prop, a subsingleton type; it has at most one element. This is to say that the category model doesn't support synthetic *higher* categories: the synthetic category structure of $\mathsf{Hom}(t,t')$ is a synthetic preorder. But there's another sense in which the category model structure trivializes "one level up": all the hom-types are interpreted as discrete categories, which are necessarily *groupoids*. So $\mathsf{Hom}(t,t')$ is not just a synthetic preorder, it's actually a synthetic *setoid*. This is appropriate for doing synthetic 1-category theory: it makes sense that the hom-types are trivial *as categories*: to do 1-category theoretic arguments, we want our hom-sets to be hom-*sets*, that is, types whose only synthetic category-theoretic structure is propositional, symmetric identity types.

We encapsulate DCwFs like this into a definition for further study.

▶ **Definition 5.** *A* $(1,1)$***-truncated DCwF*** *is a DCwF such that*

- $\mathsf{Hom}(t,t')$ *is a neutral type for any terms* t,t'*; and*
- *UIP holds for identities of hom-terms:*

$$\mathsf{UIP}^1 : (\alpha : \mathsf{Tm}(\Gamma, \mathsf{Id}(p,q)^-)) \to (\beta : \mathsf{Tm}(\Gamma, \mathsf{Id}(p,q))) \to \mathsf{Tm}(\Gamma, \mathsf{Id}(\alpha,\beta))$$

The numbering follows the well-known indexing of (n,m)-categories (see e.g. [6, Defn. 8]) to refer to ∞-categories where all parallel k-morphisms are equal when $k > n$ and all k-morphisms are invertible for $k > m$. We could define (n,m)-truncated DCwFs for arbitrary n and m (for instance, the preorder model would be $(0,1)$-truncated, the groupoid model $(1,0)$-truncated, etc.), but that would take us too far afield. For the present work, we will work with $(1,1)$-truncated DCwFs, and develop the theory of synthetic 1-category theory (i.e. synthetic $(1,1)$-category theory) in that language. The practical consequence of working in the syntax of $(1,1)$-truncated DCwFs is that we only have one "layer" of homs, and the type $\mathsf{Hom}(t,t')$ itself is neutral, i.e. its homs are symmetric identity types.

To conclude this section, we return to a key construct from the previous section, namely Π-types, and define what they look like in NPCwFs. This is just the appropriately-polarized analogue of [16, Defn. 3.15], and is approximately the same rule for Π-types in [22].

▶ **Definition 6.** *A PCwF supports* ***polarized*** Π***-types*** *if it comes equipped with a type former*

$$\Pi\colon (A\colon \mathsf{Ty}\,\Gamma^-) \to \mathsf{Ty}(\Gamma \triangleright^- A) \to \mathsf{Ty}\,\Gamma$$

which is stable under substitution, along with a natural isomorphism

$$\mathit{lam}\colon \mathsf{Tm}(\Gamma \triangleright^- A, B) \cong \mathsf{Tm}(\Gamma, \Pi(A, B))\colon \mathit{app}.$$

The β-law is that $\mathsf{app} \circ \mathsf{lam} \equiv \mathsf{id}$ and the η-law the other way around.

Now, the operation of Π-types provide an independent reason to work in neutral contexts: in a non-neutral context, these polarized Π-types are rather difficult to work with. For instance, consider the task of writing the identity function on some type $A\colon \mathsf{Ty}\,\Gamma$. The type $A \to A$ is not even well-formed, as the domain type of a function must be a type in context Γ^- but the codomain in context Γ. If Γ is neutral, we can fix this using the e isomorphism, taking $A[\mathsf{e}] \to A$ to be the type of endofunctions on A. Likewise, it's unclear how to *write* the identity function without using the tools of neutral contexts, as the variable term v is an element of $\mathsf{Tm}(\Gamma \triangleright^+ A, A[\mathsf{p}])$, but lam needs an element of $\mathsf{Tm}(\Gamma \triangleright^- A[\mathsf{e}], A[\mathsf{e}][\mathsf{p}_-])$. Here, the ee isomorphism solves the problem:

$$\mathsf{lam}(\mathsf{v}[\mathsf{ee}])\colon \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to A).$$

Likewise for application: if we define the application operator by $F \mathbin{\$} t = (\mathsf{app}\,F)[\mathsf{id},_-\, t]$, it will have type

$$\_\$\_\colon \mathsf{Tm}(\Gamma, A[e] \to B) \to \mathsf{Tm}(\Gamma^-, A[e]^-) \to \mathsf{Tm}(\Gamma, B).$$

This is a bit unfortunate: a function and its argument need to come from different contexts! Using that $\Gamma\colon \mathsf{NeutCon}$, however, then we can freely substitute terms between Γ and Γ^- using e, and coerce terms between A and A^- using the term-negation operator, alleviating this difficulty and allowing us to, for instance, construct the composition of functions.

$$\begin{aligned}
&\_\circ\_ : \{\Gamma : \mathsf{NeutCon}\}\{\mathsf{A\ B\ C} : \mathsf{Ty}\ \Gamma\} \to \\
&\quad \mathsf{Tm}(\Gamma,\ \mathsf{B[e]} \to \mathsf{C}) \to \mathsf{Tm}(\Gamma,\ \mathsf{A[e]} \to \mathsf{B}) \to \mathsf{Tm}(\Gamma,\ \mathsf{A[e]} \to \mathsf{C}) \\
&\mathsf{G} \circ \mathsf{F} = \mathsf{lam}(\ (\mathsf{app\ G})[\ \mathsf{ee}^{-1}\ ][\ \mathsf{p}_{-,A}\ ,_{+}\ \mathsf{app\ F}\ ]\ )
\end{aligned}$$

There are several other ways which neutral contexts prove necessary – see [25, Chap. 2] for a more thorough exploration. We view the neutral-context method as essential to making the type theory of the category model viable: they provide not just an alternative solution to the issue of refl's divariance, but, as we can see here with Π-types, solve numerous problems caused by the polarity calculus in arbitrary contexts being too strict. Working in a neutral context in the syntax of (1,1)-directed type theory with Π-types, we're able to actually develop synthetic category theory; we turn our attention to that task now.

4 Synthetic Category Theory

In this section, we work in an arbitrary $(1,1)$-truncated DCwF with polarized Π-types by only working in the syntax. In the main body of the text, we'll adopt an informal type theoretic style (inspired by [34]). We assume that we're working in some neutral context Γ, though we don't explicitly reference Γ. We'll write $t\colon A$ to indicate $t\colon \mathsf{Tm}(\Gamma, A)$. In what follows, we'll use the letters $p, q, r, s, t, u, v, w, f, g$ to name terms (of various types) in Γ, whereas the letters x, y, z will be the names of *variables* obtained by extending Γ. We'll have to be careful to abide by the variable negation rule:

▶ **Principle** (Var Neg)**.** An expression e can only be negated if all the variables occurring in it are of neutral types.

We suppress the distinction between Γ and Γ^-, since we can substitute back and forth with e behind the scenes, as needed. Negative context extension will just behave like positive extension by a negative type: recall that $\mathsf{v}_{-,A} \colon \mathsf{Tm}((\Gamma \triangleright^- A)^-, A[\mathsf{p}^-_{-,A}]^-)$ i.e.

$$\mathsf{v}_{-,A} \colon \mathsf{Tm}(\Gamma^- \triangleright^+ A^-, A[\mathsf{p}_{A^-}]^-)$$

so, if we're suppressing the distinction between Γ and Γ^-, then this is just a variable x of A^-. Accordingly, we'll apply and form functions like this:

$$\frac{f \colon \prod_{(x \colon A^-)} B(x) \quad t \colon A^-}{f(t) \colon B(t)} \qquad \frac{x \colon A^- \vdash e \colon B(x)}{(\lambda(x \colon A^-) \to e) \colon \prod_{x \colon A^-} B(x).}$$

Finally, here's our principle of directed path induction:

▶ **Principle** (Directed Path Induction)**.** For every $t \colon A^-$, if $M(x,y)$ is a type family depending on $x \colon A$ and $y \colon \mathsf{Hom}(t,x)$, then, for each $m \colon M(-t, \mathsf{refl}_t)$, we get an $\mathsf{ind}_M(m,x,y) \colon M(x,y)$ for all x, y.

So, for instance, the construction of symmetry above (the proof of Proposition 4) would be expressed informally as follows: given a neutral type A and a term $t \colon A^-$, define a type family over $x \colon A, y \colon \mathsf{Id}(t,x)$ by $S(x,y) = \mathsf{Id}(-x,-t)$. We have not violated (Var Neg) because $x \colon A$ and A is neutral. We have a term of type $S(-t, \mathsf{refl}_t)$, i.e. $\mathsf{Id}(t,-t)$, namely refl_t. So therefore we get $S(x,y)$ for arbitrary x, y. If we have a particular $t' \colon A$ and $p \colon \mathsf{Id}(t,t')$, we can put $\mathsf{symm}\ p = \mathsf{ind}_S(\mathsf{refl}_t, t', p)$. Again, we emphasize that it is (Var Neg) which prevents this argument from working for non-neutral types, as desired. Below, we are more casual with our application of directed path induction (e.g. not defining the type family explicitly) in cases where (Var Neg) is not a concern.

With that, we can proceed to the informal constructions. Along the way, the explicit constructions in the DCwF syntax are carried out in the accompanying figures. For the full details, see the calculations in [26].

4.1 Composition of Homs (Figure 7)

As mentioned, a type A in directed type theory is supposed to be a *synthetic category*. The terms $t : A$ represent objects, and the terms $p \colon \mathsf{Hom}(t,t')$ represent morphisms. For this to truly be category theory, however, we must be able to compose morphisms. We'll write composition in diagrammatic order: given $t, u \colon A^-$ and $v' \colon A$, we should be able to compose $p \colon \mathsf{Hom}(t,-u)$ with $q \colon \mathsf{Hom}(u,v')$ to get $p \cdot q \colon \mathsf{Hom}(t,v')$. We do this by directed path induction on q, by putting $p \cdot \mathsf{refl}_u = p$.

The refl terms serve as the identity morphisms of the category: by the above, we know that $p \cdot \mathsf{refl}_u \equiv p$, and thus $\mathsf{refl}_p \colon \mathsf{Id}(p \cdot \mathsf{refl}_u, p)$. As for the other unit law, we must again use directed path induction: since $\mathsf{refl}_u \cdot \mathsf{refl}_u \equiv \mathsf{refl}_u$, we have that $\mathsf{refl}_{\mathsf{refl}_u} \colon \mathsf{Id}(\mathsf{refl}_u \cdot \mathsf{refl}_u, \mathsf{refl}_u)$, and, by induction we get for each $q \colon \mathsf{Hom}(u,v')$ a term

$$\mathsf{r{-}unit}\ q = \mathsf{ind}(\mathsf{refl}_{\mathsf{refl}_u}, v, q) \colon \mathsf{Id}(\mathsf{refl}_u \cdot q, q).$$

Finally, we get that the composition operation is associative. Given $t, u, v \colon A^-$ and $w' \colon A$ as well as $p \colon \mathsf{Hom}(t,-u)$, $q \colon \mathsf{Hom}(u,-v)$, and $r \colon \mathsf{Hom}(v,w')$, we construct

$$\mathsf{assoc}\ p\ q\ r \colon \mathsf{Id}(p \cdot (q \cdot r), (p \cdot q) \cdot r)$$

by directed path induction on r. If $r = \mathsf{refl}_v$, then $q \cdot r \equiv q$ and $(p \cdot q) \cdot r \equiv p \cdot q$. Thus, we have $\mathsf{refl}_{p \cdot q} \colon \mathsf{Id}(p \cdot (q \cdot \mathsf{refl}_v), (p \cdot q) \cdot \mathsf{refl}_v)$, and then the induction carries through, and we get $\mathsf{assoc}\ p\ q\ r$ as desired.

```
C : {t : Tm(Γ, A⁻)} → Ty (Γ ▷⁺ A ▷⁺ Hom(t'[p_A], v))
C = Hom(t[p_A],v_A)[ p ]

_·_ : {t t' : Tm(Γ, A⁻)}{t'' : Tm(Γ, A)} → Tm(Γ, Hom(t,−t')) → Tm(Γ, Hom(t',t''))
    → Tm(Γ, Hom(t,t''))
f · g = (J_{t',C} f) [ id,₊ t'' ,₊ g ]
```

Figure 7 Composition of Homs.

4.2 Synthetic Functors (Figure 8)

If types A, B are synthetic categories, it should come as no surprise that terms $F \colon A \to B$ are synthetic *functors*. The object part is given by the usual function application, but the variances are somewhat mixed: if $t \colon A^-$, then we can say $f(t) \colon B$. However, we can still apply f to a term $t' \colon A$, we just have to put a minus on t', i.e. $f(-t')$.

Unlike usual ("analytic") category theory, we don't have to explicitly define the morphism part of a functor; any term of type $A \to B$ we can write down will come with a morphism part for free. To obtain this morphism part, again we use directed path induction: given an $F \colon A \to B$ and some $t \colon A^-$, we can define a B-morphism

$$\mathsf{map}\ F\ f \colon \mathsf{Hom}(-F(t), F(-t'))$$

for every $t' \colon A$ and $f \colon \mathsf{Hom}(t, t')$ by defining $\mathsf{map}\ F\ \mathsf{refl}_t$ to be $\mathsf{refl}_{-F(t)}$. By definition ($J\beta$), this operation preserves identities (sending refl to refl), and respects composition: if we have $t, u \colon A^-$ and $f \colon \mathsf{Hom}(t, -u)$, then, since $\mathsf{map}\ F\ \mathsf{refl}_u \equiv \mathsf{refl}_{-F(u)}$ and $f \cdot \mathsf{refl}_u \equiv f$ and $(\mathsf{map}\ F\ f) \cdot \mathsf{refl}_{-F(u)} \equiv \mathsf{map}\ F\ f$, we have

$$\mathsf{refl}_{(\mathsf{map}\ F\ f)} \colon \mathsf{Id}(\mathsf{map}\ F\ (f \cdot \mathsf{refl}_u), (\mathsf{map}\ F\ f) \cdot (\mathsf{map}\ F\ \mathsf{refl}_t)).$$

By induction, we get an identity between $\mathsf{map}\ F\ (f \cdot g)$ and $(\mathsf{map}\ F\ f) \cdot (\mathsf{map}\ F\ g)$ for arbitrary g.

```
_$_ : (Tm(Γ, A[e] → B)) → Tm(Γ,A⁻) → Tm(Γ,B)
F $ t = (app F)[ id ,₋ t[e] ]

map : {Γ : NeutCon}{A}{B}(f : Tm(Γ, A[e] → B)){t : Tm(Γ,A⁻)}{t' : Tm(Γ,A)}
    → (Tm(Γ, Hom(t,t'))) → Tm(Γ, Hom( −(F $ t) , F $ (−t')))
map F f = (J_{t,MAP} refl_{−(F $ t)})[id ,₊ t' ,₊ f ]    where
        MAP : Ty (Γ ▷⁺ A)
        MAP = Hom( (−(F $ t))[p_A], (app F)[ ee⁻¹ ] ) −− See Equation 6
```

Figure 8 Morphism part of Functors.

5 Further observations about the Category Model

As the previous section showed, the syntax of 1-truncated Directed CwFs provides a nice setting for some very basic constructions in synthetic category theory. However, further expansion of the DCwF syntax is needed to be able to capture the full range of constructions in category theory. In this section, we'll observe some constructions that can be made (and some equivalences that hold) in the category model, which require further study to be internalized into the DCwF syntax.

Probably the most significant omission from the synthetic category theory of the previous section is natural transformations. As we discuss in [26, Sect. 5], there is some work to be done to appropriately accommodate natural transformations in our framework. But for the present work, we'll focus on another important feature: *universes*. The category model comes equipped with several type universes, most significantly the universe of sets. More precisely, we can regard the category Set as a type in each context Γ of the category model, interpreted by the category of sets and functions. The operation $\mathsf{El}\colon \mathsf{Tm}(\Gamma, \mathsf{Set}) \to \mathsf{NeutTy}\,\Gamma$ takes a set X and views it as a discrete category. We can then define

$$\mathsf{Hom{-}to{-}func}\colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(-X, Y)) \to \mathsf{Tm}(\bullet, \mathsf{El}(X)[\mathsf{e}] \to \mathsf{El}(Y))$$

by directed path induction: $\mathsf{Hom{-}to{-}func}\ \mathsf{refl}_{-X}$ is the identity function, $\mathsf{lam}\ (\mathsf{v}[\mathsf{ee}])$ of type $\mathsf{Tm}(\Gamma, \mathsf{El}(X)[\mathsf{e}] \to \mathsf{El}(X))$. We can use this to state the following principle.

▶ **Principle** (Directed Univalence)**.** Hom−to−func is a bijection.

Spelling out the category model semantics, we see that every function is sent to itself. Sufficiently internalized, this principle of Directed Univalence serves as the directed analogue of Hofmann and Streicher's *universe extensionality* axiom [17, Section 5.4]. Further work is required to better develop the theory of isomorphisms in the synthetic category theory, and to compare this principle of directed univalence to existing ones (e.g. [21, 14]).

Let us conclude by observing that the existence of a universe allows for metatheoretic reasoning as well, specifically negative proofs about what *cannot* be done in the syntax. We can view the directed universe Set as a source of nontrivial directedness: if we affirm Directed Univalence, then Set cannot possibly be a neutral type. We show that, in DCwFs equipped with a directed univalent universe Set, hom-types must be asymmetric in general. That is, we cannot construct a term symm like in Proposition 4 for non-neutral types in the syntax DCwF+Set (the initial model of the GAT of DCwFs with a universe Set). We do so the same way Hofmann and Streicher [17] proved that ordinary Martin-Löf Type Theory couldn't prove the Uniqueness of Identity Proofs: by countermodel. Hofmann and Streicher's countermodel was the groupoid model, and, of course, ours is the category model.

▶ **Proposition 7.** *There cannot be an operation*

$$\mathsf{symm}'\colon \{\Gamma\}\{A\}\{t\colon \mathsf{Tm}(\Gamma, A^-)\}\{t'\colon \mathsf{Tm}(\Gamma, A)\} \to \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t')) \to \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t', -t))$$

*definable in the syntax of DCwFs+*Set*.*

Proof. If the syntax model of DCwF+Set had such an operation symm′, then, by initiality, so too would every DCwF with Set, in particular the category model. But then for any $X\colon \mathsf{Tm}(\bullet, \mathsf{Set}^-)$ and $Y\colon \mathsf{Tm}(\bullet, \mathsf{Set})$ and $f\colon \mathsf{Tm}(\bullet, \mathsf{Hom}(X, Y))$, we would obtain $\mathsf{symm}'\ f\colon \mathsf{Tm}(\bullet, \mathsf{Hom}(Y, X))$. But this is absurd, because the function $?\colon \emptyset \to \mathbb{1}$ is a term of type $\mathsf{El}(\emptyset) \to \mathsf{El}(\mathbb{1})$ in the category model, and, by Directed Univalence, corresponds to a term of type $\mathsf{Hom}(\emptyset, \mathbb{1})$, but there cannot be any terms of $\mathsf{Hom}(\mathbb{1}, \emptyset)$, because the set of terms of this type is in bijection with the set of functors $\mathsf{El}(\mathbb{1})$ to $\mathsf{El}(\emptyset)$, of which there are none. ◀

Basically the same argument will show the *uniqueness of homs* principle – that for any hom terms $p\colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(t,t')^-)$ and $q\colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(t,t'))$, there is a witness of $\mathsf{Id}(p,q)$ – is violated in the category model (a counterexample being Set-homs from the two-element set to itself), and therefore not provable in the syntax of DCwFs+Set. Indeed, in the same way that Hofmann and Streicher remark that UIP is contradicted by their universe extensionality, our Directed Univalence is incompatible with the uniqueness of homs. So, in sum, we can conclude that the difference between $(1,1)$-truncated DCwFs, $(1,0)$-truncated DCwFs, and $(0,1)$-truncated CwFs is reflected internally in the syntax.

6 Conclusion and Future Work

We have laid the foundation for a generalized algebraic theory of directed types, and began to conduct synthetic category theory in that setting. Our semantics-forward approach was to study the category model first, and extract its key features into a series of abstract definitions – the GATs of polarized CwFs, neutral-polar CwFs, directed CwFs, $(1,1)$-truncated directed CwFs, and directed CwFs with features like polarized dependent types and a directed univalent set universe. Working within the directed type theory of these models, we found that it was possible to work informally with the powerful directed path induction principle to make basic constructions in category theory, with our careful discipline about variable negation preventing the directed type theory from collapsing into undirected type theory.

The reviewers have claimed that our system isn't closed under substitution: the negation operation is only applicable in a neutral context, but there may be a substitution from a context that is not neutral and then reducing the substitution would result in a non-welltyped term. However, since we formulated our theory as a GAT, substitution is an explicit operation and hence this isn't an issue for our presentation. Indeed the naturality equation $(-t)[\sigma] = -(t[\sigma])$ only applies if σ is a substitution between neutral contexts. This does entail for an implementation that applying substitutions to negated terms has to be delayed, i.e. we need to deal with explicit closures of the form $(-t)[\sigma]$, if the domain of σ is not a neutral context.

Much remains to be done.[12] The category theory of Section 4 serves as a proof-of-concept, but needs to be fleshed out into a full theory. As mentioned, work is needed to articulate natural transformations in the theory; our current investigations concern possible generalization of Π-types to di-variant *end types* which address some of the above-mentioned variance issues with natural transformations. For reasons of space, we omitted dependent sum types from the theory. But with them added, much of basic category theory should be expressible in this language, such as isomorphisms, (co)slice categories, (co)limits, exponentials, and adjunctions.[13] Better development of the category of sets should put representability and some properties of presheaf categories into reach, though internal statement and proof of the Yoneda Lemma will likely rely on the resolution of the above-mentioned dilemma regarding natural transformations. We also leave it to future work to study whether this theory can capture higher category theory by weakening or dropping the assumption of (1,1)-truncation, and, if so, how it compares to existing synthetic higher category frameworks, such as [30].

[12] Some details which are omitted here are included in the extended arxiv version of this paper, [26]. A broader and more thorough treatment of this material is given in the first author's forthcoming PhD thesis [25]. There, some of the issues listed here are addressed.

[13] See [25, Chap. 4].

There are further avenues for developing the type theory of DCwFs. Two important metatheoretic results about the syntax of DCwFs currently being pursued are *canonicity* and *normalization* (modulo the concern above about closure under substitution). Moreover, we would like to verify the correctness of these results by formalizing them in a computer proof assistant. A further goal would be to implement the syntax of DCwFs as a computer proof language itself, hopefully with syntax nearly as convenient as the constructions of Section 4, and formalize larger swaths of category theory in it.

A further motivation for the present work's focus on generalized algebraic theories is the possibility of expressing this style of directed types theory in a *second-order* generalized algebraic theory, following [8, 33, 32, 4, 19]. Since our notions of PCwFs and NPCwFs include explicit operations on contexts (as seen in the substructural character of the (Var Neg) rule), it's clear that either an extension to the SOGAT signature language of [19], and/or a partial internalization of the first-order theory into the second-order theory – à la [4] – will be necessary. Another task is to compare our $(\_)^{-}$ modality (and the $(\_)^{0}$ modality definable in the category model using core groupoids) to the modalities studied in [13], particularly the kinds employed in directed type theory [37]. We indicated above that the core modality ought to be a dependent right adjoint in the appropriate sense, but work remains to spell out this explicitly. Another possible avenue: this work is conducted with a possible future connection to *higher observational type theory* [31, 5, 4] in mind; to begin to prepare the way for such a project, at the very least further study of the observational equivalences of this theory (e.g. a characterization of the Hom-types of Π-types) is needed.

Finally, the present framework provides a setting for studying *directed higher-inductive types* – inductively-defined types with both term constructors and *hom constructors*. Some simple examples would be the directed interval (as is studied in [36, 30]) and a directed analogue of the *circle* type [34, Section 6.2]. These examples are modelled by the category model, and can therefore be soundly added to the present theory. Higher examples (such as directed versions of higher tori and spheres) would require more careful metatheoretic work to justify, but could perhaps lead to a number of interesting considerations.

References

1 Benedikt Ahrens, Paige Randall North, and Niels van der Weide. Bicategorical type theory: semantics and syntax. *Mathematical Structures in Computer Science*, 33(10), 2023. doi:10.1017/s0960129523000312.

2 Thorsten Altenkirch. Extensional equality in intensional type theory. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 412–420. IEEE, 1999. doi:10.1109/LICS.1999.782636.

3 Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, Christian Sattler, and Filippo Sestini. Constructing a universe for the setoid model. In *FoSSaCS*, pages 1–21, 2021. doi:10.1007/978-3-030-71995-1_1.

4 Thorsten Altenkirch, Yorgo Chamoun, Ambrus Kaposi, and Michael Shulman. Internal parametricity, without an interval. *Proceedings of the ACM on Programming Languages*, 8(POPL):2340–2369, 2024. doi:10.1145/3632920.

5 Thorsten Altenkirch, Ambrus Kaposi, and Michael Shulman. Towards higher observational type theory. In *28th International Conference on Types for Proofs and Programs (TYPES 2022). University of Nantes*, 2022.

6 John C. Baez and Michael Shulman. Lectures on n-categories and cohomology, 2007. arXiv:math/0608420.

7 Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science*, 30(2):118–138, 2020. doi:10.1017/S0960129519000197.

8 Rafaël Bocquet. External univalence for second-order generalized algebraic theories. *CoRR*, abs/2211.07487, 2022. doi:10.48550/arXiv.2211.07487.

9 John Cartmell. Generalised algebraic theories and contextual categories. *Annals of pure and applied logic*, 32:209–243, 1986. doi:10.1016/0168-0072(86)90053-9.

10 Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827:135–180, 2019. doi:10.48550/arXiv.1904.00827.

11 Fernando Chu, Éléonore Mangel, and Paige Randall North. A directed homotopy type theory for 1-categories. In *30th International Conference on Types for Proofs and Programs (TYPES 2024). IT University of Copenhagen*, 2024.

12 Peter Dybjer. Internal type theory. In *International Workshop on Types for Proofs and Programs*, pages 120–134. Springer, 1995. doi:10.1007/3-540-61780-9_66.

13 Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 492–506. ACM, 2020. doi:10.1145/3373718.3394736.

14 Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. The yoneda embedding in simplicial type theory, 2025. doi:10.48550/arXiv.2501.13229.

15 Martin Hofmann. A simple model for quotient types. In *International Conference on Typed Lambda Calculi and Applications*, pages 216–234. Springer, 1995. doi:10.1007/BFb0014055.

16 Martin Hofmann. Syntax and semantics of dependent types. In *Extensional Constructs in Intensional Type Theory*, pages 13–54. Springer, 1997.

17 Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. *Twenty-five years of constructive type theory (Venice, 1995)*, 36:83–111, 1995.

18 Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL), January 2019. doi:10.1145/3290315.

19 Ambrus Kaposi and Szumi Xie. Second-order generalised algebraic theories: Signatures and first-order semantics. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction, FSCD 2024, July 10-13, 2024, Tallinn, Estonia*, volume 299 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.FSCD.2024.10.

20 András Kovács. *Type-Theoretic Signatures for Algebraic Theories and Inductive Types.* PhD thesis, Eötvös Loránd University, 2022.

21 Nikolai Kudasov, Emily Riehl, and Jonathan Weinberger. Formalizing the ∞-categorical yoneda lemma, 2023. doi:10.48550/arXiv.2309.08340.

22 Daniel R. Licata and Robert Harper. 2-dimensional directed type theory. In Michael W. Mislove and Joël Ouaknine, editors, *Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011*, volume 276 of *Electronic Notes in Theoretical Computer Science*, pages 263–289. Elsevier, 2011. doi:10.1016/j.entcs.2011.09.026.

23 Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. Elsevier, 1975.

24 Per Martin-Löf. Constructive mathematics and computer programming. In L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski, editors, *Logic, Methodology and Philosophy of Science VI*, volume 104 of *Studies in Logic and the Foundations of Mathematics*, pages 153–175. Elsevier, 1982.

25 Jacob Neumann. *A Generalized Algebraic Theory of Directed Equality.* PhD thesis, University of Nottingham, 2025.

26 Jacob Neumann and Thorsten Altenkirch. Synthetic 1-categories in directed type theory. *CoRR*, abs/2410.19520, 2024. doi:10.48550/arXiv.2410.19520.

27 Max S. New and Daniel R. Licata. A formal logic for formal category theory. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 113–134. Springer, 2023. doi:10.1007/978-3-031-30829-1_6.

28 Paige Randall North. Towards a directed homotopy type theory. *Electronic Notes in Theoretical Computer Science*, 347:223–239, 2019. doi:10.1016/j.entcs.2019.09.012.

29 Andreas Nuyts. Towards a directed homotopy type theory based on 4 kinds of variance. *Mém. de mast. Katholieke Universiteit Leuven*, 2015.

30 E. Riehl and M. Shulman. A type theory for synthetic ∞-categories. *Higher Structures*, 1(1):116–193, 2017. doi:10.2140/hs.2017.1.116.

31 Michael Shulman. Towards a third-generation HOTT, 2022. Carnegie Mellon University HoTT Seminar. URL: https://www.cmu.edu/dietrich/philosophy/hott/seminars/index.html.

32 Taichi Uemura. *Abstract and concrete type theories*. PhD thesis, University of Amsterdam, 2021.

33 Taichi Uemura. A general framework for the semantics of type theory. *Mathematical Structures in Computer Science*, 33(3), March 2023. doi:10.1017/s0960129523000208.

34 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, Institute for Advanced Study, 2013. URL: https://homotopytypetheory.org/book.

35 Benno Van Den Berg and Richard Garner. Types are weak ω-groupoids. *Proceedings of the london mathematical society*, 102(2):370–394, 2011.

36 Matthew Z. Weaver and Daniel R. Licata. A constructive model of directed univalence in bicubical sets. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, LICS '20, pages 915–928, New York, NY, USA, 2020. ACM. doi:10.1145/3373718.3394794.

37 Jonathan Weinberger and Ulrik Buchholtz. Type-theoretic modalities for synthetic $(\infty, 1)$-categories, 2019. International Conference on Homotopy Type Theory (HoTT 2019).

Effective Kan Fibrations for W-Types in Homotopy Type Theory

Shinichiro Tanaka
Institute for Logic, Language and Computation, University of Amsterdam, The Netherlands

Abstract

We investigate effective Kan fibrations in the context of the semantics of Homotopy Type Theory (HoTT). Effective Kan fibrations were proposed by Benno van den Berg and Eric Faber as constructive alternatives to classical Kan fibrations for modeling HoTT. Our work specifically explores their interaction with W-types in HoTT, which are inductive types representing well-founded trees, and extends this exploration to variants such as M-types. By using the categorical properties of W-types, we show that effective Kan fibrations model them. Additionally, we examine the behavior of quotient maps and discuss that certain cases can also be classified as effective Kan fibrations.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Homotopy Type Theory, Effective Kan Fibrations, W-types

Digital Object Identifier 10.4230/LIPIcs.TYPES.2024.8

Acknowledgements I would like to thank Dr. Benno van den Berg for his insightful advice and support.

1 Introduction

1.1 Background and motivation

Homotopy Type Theory (HoTT) started with the homotopy-theoretic interpretation of Martin-Löf's dependent type theory and the discovery of the univalence axiom by Voevodsky in the 2000s [11] [12]. Awodey and Warren, and independently Voevodsky, provided this homotopy interpretation. A certain form of Martin-Löf type theory can be modeled in any Quillen model category [1], giving HoTT a semantics with topological intuition. The category of simplicial sets, one of the most significant model categories, models types as Kan complexes and dependent types as Kan fibrations, as proposed by Voevodsky.

Kan fibrations, a key construct in this model, possess a lifting property analogous to the homotopy lifting property in classical homotopy theory. However, the initial Kan fibration model is non-constructive, as closure under pushforward of Kan fibrations is unprovable [2].

To develop constructive models, cubical sets were proposed as an alternative to simplicial sets. Nevertheless, the original Kan fibration concept is still appealing, and there have been endeavors to create a constructive version that maintains Voevodsky's ideas. Gambino and Sattler proposed a model treating Kan fibrations as structure rather than property [6]. Then, to overcome difficulties in this approach, van den Berg and Faber introduced effective Kan fibrations [2], which is the primary focus of our paper.

Verification of effective Kan fibrations modeling HoTT includes modeling all type formers, such as Π-types, and a primary focus of this thesis is W-types, which represent well-founded trees (e.g., natural numbers) introduced by Martin-Löf in [10]. W-types, already modeled by classical HoTT semantics using Kan fibrations [3], are examined here using effective Kan fibrations.

Prior research regarding this concept includes [5] and [7]. The latter defined a special case of effective Kan fibrations, called symmetric effective Kan fibrations. The primary difference between effective Kan fibrations and symmetric effective Kan fibrations lies in the fact that,

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg; Article No. 8; pp. 8:1–8:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

except for certain exceptions, each horn problem in effective Kan fibrations has two solutions – positive and negative – whereas this distinction does not exist in symmetric effective Kan fibrations. This paper focuses on effective Kan fibrations and employs a slightly modified version of the framework introduced in [7] to incorporate the distinction between positive and negative solutions.

As will be mentioned, constructing W-types categorically involves transfinite induction. To ensure the entire argument is fully constructive, we must deal with this constructively, which is currently an open problem.

1.2 Outline

The first section introduces preliminary information about simplicial sets, providing essential foundational knowledge for the subsequent sections.

Then, based on the paper [3] on W-types in HoTT in the classical setting, we will extend this study using a constructive approach. We define the categories of effective Kan fibrations and their variants. This categorical framework is crucial for describing W-types and M-types as certain kinds of colimits and limits, respectively.

Next, we explore whether effective Kan fibrations can model W-types and M-types by examining the existence and behavior of filtered colimits and limits in these categories. We then explain how W-types are generated by polynomial functors, and investigate the role of these functors as maps in the categorical structures defined in previous sections.

Finally, we proceed to the main theorem: Effective Kan fibrations can indeed model W-types and their variants, providing a comprehensive conclusion to the theoretical framework developed.

Additionally, for future work, to explore potential applications of W-types in constructive mathematics, we will examine their role in defining quotients and discuss related challenges. Specifically, we address issues involving epimorphisms by introducing additional structures on effective Kan fibrations. A potential goal of this approach is to show that such structures could enable effective Kan complexes and fibrations to model quotient types and quotient maps, respectively, in a constructive framework.

2 Effective Kan fibrations/complexes

2.1 Simplicial sets

▶ **Definition 1.** *The simplex category Δ is a category whose objects are the finite, non-empty, linearly ordered sets $[n] = \{0, \dots, n\}$ for each $n \geq 0$, and whose morphisms $[n] \to [m]$ are nondecreasing maps.*

For each $n \geq 0$, there are two important types of morphisms in Δ. An injection $d_i \colon [n-1] \to [n]$ that omits the i-th element is called the i-th *face map*. A surjection $s_i \colon [n+1] \to [n]$ that maps two distinct elements to i is called the i-th *degeneracy map*. Specifically,

$$d_i(j) = \begin{cases} j & \text{if } j < i, \\ j+1 & \text{if } j \geq i, \end{cases} \qquad s_i(j) = \begin{cases} j & \text{if } j \leq i, \\ j-1 & \text{if } j > i. \end{cases}$$

These maps satisfy the *simplicial identities*:

$$s_j \circ d_k = \begin{cases} d_{k-1} \circ s_j & \text{if } k > j+1, \\ \text{id} & \text{if } k \in \{j, j+1\}, \\ d_k \circ s_{j-1} & \text{if } k < j, \end{cases}$$

$d_j \circ d_k = d_{k+1} \circ d_j$ (if $k \geq j$), and $s_j \circ s_k = s_k \circ s_{j+1}$ (if $j \geq k$).

▶ **Definition 2.** *A contravariant functor $\Delta \to$ **Set**, namely a presheaf on Δ, is called a simplicial set. The category of simplicial sets is denoted by* **sSet**.

Given a simplicial set X, we write $X_n := X([n])$, making X a graded set $\{X_n\}_{n\in\mathbb{N}}$. Since X is contravariant, the face maps and degeneracy maps induce actions $X(d_i)\colon X_n \to X_{n-1}$ and $X(s_i)\colon X_n \to X_{n+1}$, respectively. For any locally small category $\mathcal{C}$ and $C \in \mathrm{ob}(\mathcal{C})$, the Yoneda lemma identifies the natural bijection $\mathrm{Hom}_{\mathbf{Set}^{\mathcal{C}^{\mathrm{op}}}}(y_C, F) \cong FC$ for every presheaf F with the representable presheaf $y_C = \mathrm{Hom}_{\mathcal{C}}(-, C)$. Setting $\mathcal{C} = \Delta$, we have:

▶ **Definition 3.** *In* **sSet**, *for each $[n]$, the representable functor is called the n-standard simplex and denoted by Δ^n instead of $y_{[n]}$, i.e., $\Delta^n([m]) = \mathrm{Hom}_\Delta([m],[n])$.*

A face map $d_k \in \mathrm{Hom}_\Delta([n-1],[n])$. This corresponds to a face map $d_k\colon \Delta^{n-1} \to \Delta^n$ in **sSet**. Similarly, we have degeneracy maps in **sSet**.

▶ **Definition 4.** *The boundary $\partial\Delta^n$ of Δ^n is the union of all its $(n-1)$-dimensional faces:*

$$\partial\Delta^n = \bigcup_{i=0}^{n} d_i(\Delta^{n-1}).$$

▶ **Definition 5.** *Let $n \geq 1$ and $0 \leq m \leq n$. The m-th horn of Δ^n is the simplicial subset:*

$$\Lambda^n_m = \bigcup_{\substack{0\leq i\leq n\\ i\neq m}} d_i(\Delta^{n-1})$$

If $0 < m < n$, Λ^n_m is called an inner horn, and if $m \in \{0, n\}$, it is called an outer horn.

By the density theorem, every presheaf is a colimit of representable presheaves. That is, a horn Λ^n_m is a colimit of standard simplices. This is expressed as follows [8]:

▶ **Lemma 6.** *A horn Λ^n_m can be expressed by the following coequalizer in the following commutative "fork".*

$$\begin{tikzcd}
\Delta^{n-2} \arrow[r, "d_{j-1}"] \arrow[d, hook, "\mathsf{in}_{i<j}"] & \Delta^{n-1} \arrow[d, hook, "\mathsf{in}_i"'] \arrow[dr, "d_i"] & \\
\coprod_{0\leq i<j\leq n} \Delta^{n-2} \arrow[r, hook, shift left] \arrow[r, hook, shift right] & \coprod_{i\neq m} \Delta^{n-1} \arrow[r, dashed, "{[d_k]_{k\neq m}}"] & \Lambda^n_m \\
\Delta^{n-2} \arrow[r, "d_i"'] \arrow[u, hook', "\mathsf{in}_{i<j}"] & \Delta^{n-1} \arrow[u, hook', "\mathsf{in}_j"'] \arrow[ur, "d_j"] &
\end{tikzcd}$$

Besides this lemma, we will also see another perspective of constructing horns in the next section. The following lemma will be used frequently, so we state it here for reference. The proof follows from the universal property of the coequalizer and coproduct in **Lemma 6**.

▶ **Lemma 7.** *In* **sSet**, *given $f, g\colon \Lambda^n_m \to S$, if $f \circ d_k = g \circ d_k$ for all k with $0 \leq k \leq n$ and $k \neq m$, then $f = g$, i.e., $(d_k)_{k\neq m}$ are jointly epic.*

2.2 Preliminaries on effective Kan fibrations/complexes

Now, let us see an alternative construction of a horn described in [7].

▶ **Construction 8.** *Let $\epsilon \in \{0,1\}$. In the commutative squares (1) each of which is a pullback square, a horn $\Lambda^{n+1}_{i+1-\epsilon}$ is alternatively constructed as the pushout of the left square.*

$$\begin{tikzcd}
\partial\Delta^n \arrow[r] \arrow[d, hook] \arrow[dr, phantom, "\lrcorner", very near start] & s_i^*(\partial\Delta^n) \arrow[r] \arrow[d, hook] \arrow[dr, phantom, "\lrcorner", very near start] & \partial\Delta^n \arrow[d, hook] \\
\Delta^n \arrow[r, "d_{i+\epsilon}"'] & \Delta^{n+1} \arrow[r, "s_i"'] & \Delta^n
\end{tikzcd} \tag{1}$$

Notice that except for the edge cases ($m = 0$ or $m = n+1$), there are two ways to obtain Λ^{n+1}_m, since two pairs $(i,\epsilon) = (m-1, 0)$ and $(m,1)$ satisfy $i+1-\epsilon = m$ for $0 < m < n+1$. At the edge cases, however, only one pair satisfies the condition: $(i,\epsilon) = (0,1)$ when $m = 0$ and $(i,\epsilon) = (n,0)$ when $m = n+1$. Thus Λ^{n+1}_m for $0 < m < n+1$ is created by d_{m-1} (when $\epsilon = 0$) or d_{m+1} (when $\epsilon = 1$) in (1). We call a horn *negative* if $\epsilon = 0$ and *positive* if $\epsilon = 1$, and denote it by $(\Lambda^{n+1}_m, \pm)$. Note that at the edges, $(\Lambda^{n+1}_0, +)$ and $(\Lambda^{n+1}_{n+1}, -)$ do not exist.

▶ **Definition 9.** *In* **sSet***, let p be a Kan fibration, i.e., for every horn inclusion $i\colon (\Lambda^n_m, \pm) \to \Delta^n$, every pair of maps x and y with $px = yi$ as in the right square in (4), there exists a lift. For each such p, we can define a function* lift_p *that assigns a lift* $\mathsf{lift}_p(x,y)$ *for each lifting problem which is represented as a pair (x,y).*

As mentioned in the introduction, this paper uses a framework from [7], which is described below, with a modification to distinguish between positive and negative solutions discussed above.

▶ **Definition 10.** *Let p be a Kan fibration equipped with* lift_p*. For every lifting problem (x,y) and $s_j\colon \Delta^{n+1} \to \Delta^n$ with $0 \le j \le n$ as in (4), a horn map $s_j^*(x)\colon (\Lambda^{n+1}_{m^*}, \pm) \to X$, where m^* is as given in (2), is defined by (3).*

$$m^* \in \begin{cases} \{m\} & \text{if } m < j \\ \{m, m+1\} & \text{if } m = j \\ \{m+1\} & \text{if } m > j, \end{cases} \tag{2}$$

$$s_j^*(x) \circ d_k = \begin{cases} x \circ s_j \circ d_k & \text{if } k \neq j, j+1, m^* \\ \mathsf{lift}_p(x,y) & \text{if } \begin{array}{l} k \in \{j, j+1\} \\ \text{and } k \neq m^*. \end{array} \end{cases} \tag{3}$$

▶ **Definition 11.** *A map $p\colon X \to Y$ in* **sSet** *is an effective Kan fibration if it comes equipped with* lift_p*, and satisfies the compatibility (or uniformity) condition:* $\mathsf{lift}_p(s_j^*(x), ys_j) = \mathsf{lift}_p(x,y) \circ s_j$ *for every $0 \le j \le n$, m^*, and $s_j^*(x)$. In (4), two horns have the same ± signs.*

$$\begin{tikzcd}
\Lambda^{n+1}_{m^*} \arrow[d, hook, "i'"'] \arrow[rr, bend left, "s_j^*(x)"] & \Lambda^n_m \arrow[r, "x"] \arrow[d, hook, "i"] & X \arrow[d, squiggly, "p"] \\
\Delta^{n+1} \arrow[r, "s_j"'] \arrow[urr, dashed, "\mathsf{lift}_p(s_j^*(x), ys_j)" description] & \Delta^n \arrow[r, "y"'] \arrow[ur, dashed, "\mathsf{lift}_p(x,y)" description] & Y
\end{tikzcd} \tag{4}$$

When $Y = \{*\}$, the one-point simplicial set, the lower triangle is trivial and X is called an effective Kan complex.

The ± distinction is important for bookkeeping since mere existence of a lift is not sufficient anymore in the effective setting. However, for simplicity, we will mostly drop + and − signs from horns throughout the rest of this paper.

▶ Remark 12. Formally, an effective Kan fibration is a pair (p, lift_p). Also, in the case of an effective Kan complex, the fibration is the unique map $!_X \colon X \to \{*\}$ but we write it (X, lift_X) instead of $(!_X, \mathsf{lift}_{!_X})$. Also, we just denote $\mathsf{lift}_X(x)$ instead of $\mathsf{lift}_{!_X}(x, !_{\Delta^n})$.

By a slight abuse of notation, we sometimes refer to X in the pair (X, lift_X) simply as an effective Kan complex, and we say that X is in **EffKanCplx**. Similarly, the same convention applies in the case of fibrations.

▶ **Definition 13.** **EffKanFib** *is the category of effective Kan fibrations whose objects and morphisms are as follows.*

Objects: *Effective Kan fibrations* (p, lift_p)*.*
Morphisms: *A morphism* $(\varphi, \psi) \colon p \to q$ *is a pair of maps* $\varphi \colon X \to Y$ *and* $\psi \colon A \to B$ *such that* $q \circ \varphi = \psi \circ p$*, and for an arbitrary horn* $(\Lambda^n_m, \pm)$*, a map* $x \colon (\Lambda^n_m, \pm) \to X$ *and a map* $a \colon \Delta^n \to A$ *such that* $px = a\iota$*, each triangle of the following diagram commutes. In particular,* $\varphi \circ \mathsf{lift}_p(x, a) = \mathsf{lift}_q(\varphi x, \psi a)$*.*

$$\begin{tikzcd}
\Lambda^n_m \arrow[r, "x"] \arrow[d, hook, "\iota"'] & X \arrow[r, "\varphi"] \arrow[d, "p"] & Y \arrow[d, "q"] \\
\Delta^n \arrow[r, "a"'] \arrow[ur, dashed, "\mathsf{lift}_p(x,a)"] \arrow[urr, dashed, "\mathsf{lift}_q(\varphi x, \psi a)"'] & A \arrow[r, "\psi"'] & B
\end{tikzcd}$$

Note that the classical Kan fibrations are maps in **sSet**. This **EffKanFib** is further endowed with additional structures.

▶ **Definition 14.** *The category* **EffKan**$/A$ *of effective Kan fibrations whose codomain is a simplicial set* A *is defined in a similar way to* ***Definition 13****. In that definition, let* $B := A$ *and* $\psi := 1_A$*.*

▶ **Definition 15.** *The category* **EffKanCplx** *of effective Kan complexes is defined as* **EffKan**$/1$*.*

As mentioned in **Remark12**, we just regard the objects in **EffKanCplx** as (X, lift_X) instead of $(!_X, \mathsf{lift}_{!_X})$.

2.3 Properties of effective Kan fibrations/complexes

In this section, we will examine filtered colimits and limits in the categories in the previous section, as they play important roles in the construction of W-types and the variants later. Recall that a *filtered diagram* $F \colon \mathcal{I} \to \mathcal{C}$ satisfies that (1) $\mathcal{I}$ has at least one object, (2) for each pair $i, j \in \mathcal{I}$, there is $k \in \mathcal{I}$ with a pair of morphisms $i \to k$ and $j \to k$ of $\mathcal{I}$, (3) for each pair $i, j \in \mathcal{I}$ and each pair $a, b \colon i \to j$, there is a morphism $c \colon j \to k$ of $\mathcal{I}$ such that $F(c \circ a) = F(c \circ b)$ in $\mathcal{C}$. And a *filtered colimit* is the colimit of a filtered diagram. Also, in **Set**, filtered colimits can be explicitly calculated: For a filtered diagram $F \colon \mathcal{I} \to \mathbf{Set}$, we have $\mathrm{colim}_{i \in \mathcal{I}} F_i = \left(\coprod_{i \in \mathcal{I}} F_i\right)/\sim$, where $x_i \sim x_j$ if and only if there exist $k \in \mathcal{I}, \varphi \colon i \to k$ and $\psi \colon j \to k$ such that $F(\varphi)(x_i) = F(\psi)(x_j)$, for $i, j \in I$, $x_i \in F_i$ and $x_j \in F_j$.

Applying the above calculation to hom-sets, we can obtain useful facts for **sSet**. First, more generally, recall that we have $\mathrm{Hom}_{\mathbf{sSet}}(\Delta^n, X) \cong X_n$ by the Yoneda lemma, and colimits in **sSet** are computed degree-wise. Thus, every evaluation functor $\mathrm{Hom}_{\mathbf{sSet}}(\Delta^n, -) \colon \mathbf{sSet} \to \mathbf{Set}$ preserves all small colimits, i.e., for any small diagram $F \colon \mathcal{I} \to \mathbf{sSet}$, we have

$\mathrm{Hom}_{\mathbf{sSet}}(\Delta^n, \mathrm{colim}_{i\in\mathcal{I}} F_i) \cong \mathrm{colim}_{i\in\mathcal{I}} \mathrm{Hom}_{\mathbf{sSet}}(\Delta^n, F_i)$. Moreover, suppose F is also filtered. A horn Λ^n_m is a finite colimit of standard n-simplices, namely finitely presentable, and thus the functor $\mathrm{Hom}_{\mathbf{sSet}}(\Lambda^n_m, -)\colon \mathbf{sSet} \to \mathbf{Set}$ preserves all filtered colimits. In particular, we have the following.

▶ **Lemma 16.** *Let $F\colon \mathcal{I} \to$ **sSet** be a filtered diagram with filtered colimit X. Denote $F(i) = X_i$ for $i \in \mathcal{I}$ and write $\beta_i\colon X_i \to X$ for the coprojections. For each $n \geq 1$ and each $0 \leq m \leq n$, every morphism $x\colon \Lambda^n_m \longrightarrow X$ into the colimit factors through a coprojection: there exists $i \in \mathcal{I}$ and a map $x_i\colon \Lambda^n_m \to X_i$ such that $x = \beta_i \circ x_i$.*

▶ **Lemma 17.** *With the same notation, suppose a map $x\colon \Lambda^n_m \to X$ admits two factorizations $x = \beta_i \circ x_i$ and $x = \beta_j \circ x_j$ through X_i and X_j. Then there exists $k \in \mathcal{I}$ and morphisms $f\colon i \to k$ and $g\colon j \to k$ in $\mathcal{I}$ such that $x_k = F(f) \circ x_i = F(g) \circ x_j$ and $x = \beta_k \circ x_k$.*

Let us rewrite the maps of the form $F(f)$ or $F(g)$ in the **Lemma 17** as φ_{ik} or φ_{jk}, respectively. Also, let us call them *mediating maps.*

The next proposition is an important tool for our main theorem. Similar results hold for **EffKan**/A and for **EffKanFib** in place of **EffKanCplx**. While the result for **EffKanFib** would immediately imply the results for **EffKanCplx** and **EffKan**/A, since the core idea behind the proofs of all three versions is essentially the same, we choose to focus on the simplest case for **EffKanCplx**.

▶ **Proposition 18.** *The forgetful functor* **EffKanCplx** $\to$ **sSet** *creates small filtered colimits.*

Proof. Let $F\colon \mathcal{I} \to$ **EffKanCplx** be a small filtered diagram. Let $F(i) := (X_i, \mathsf{lift}_{X_i})$. Let $U\colon$ **EffKanCplx** $\to$ **sSet** denote the forgetful functor that sends $(X_i, \mathsf{lift}_{X_i})$ to X_i. Suppose X is the filtered colimit of $(X_i\colon i \in \mathcal{I})$ in **sSet** (under $U \circ F$). Our goal is to equip X with a horn-filler operation lift_X, thereby obtaining an effective Kan complex (X, lift_X) and showing that it is a filtered colimit in **EffKanCplx**.

Fix an arbitrary horn Λ^n_m and a map $x\colon \Lambda^n_m \to X$. First, let us define $\mathsf{lift}_X(x)$ for x. By **Lemma 16**, we observed that x factors through some X_i with the coprojection β_i. We define

$$\mathsf{lift}_X(x) := \beta_i \circ \mathsf{lift}_{X_i}(x_i). \tag{5}$$

As discussed earlier, the factorization is not unique. However, this is well-defined, since we can show that it does not depend on the particular choice. For, suppose there are two different factorizations via X_i and X_j. By **Lemma 17**, there is X_k with $x_k = x_i \circ \varphi_{ik} = x_j \circ \varphi_{jk}$ as follows:

$$\begin{tikzcd}
\Lambda^n_m \arrow[rrrr, "x"] \arrow[rrrd, "x_k"] \arrow[rdd, "x_i"'] \arrow[rrdd, "x_j"] \arrow[ddd, hook] & & & & X \\
& & & X_k \arrow[ru, "\beta_k"] & \\
& X_i \arrow[rru, bend left, "\varphi_{ik}"] \arrow[rrruu, "\beta_i"] & X_j \arrow[ru, "\varphi_{jk}"] \arrow[rruu, "\beta_j"'] & & \\
\Delta^n \arrow[ru, dashed, "\mathsf{lift}_{X_i}(x_i)"] \arrow[rrruu, dashed, "\mathsf{lift}_{X_k}(x_k)"'] \arrow[rru, dashed, bend right, "\mathsf{lift}_{X_j}(x_j)"'] & & & &
\end{tikzcd}$$

Then, since φ_{ik} and φ_{jk} are morphisms in **EffKanCplx**, we have $\varphi_{ik} \circ \mathsf{lift}_{X_i}(x_i) = \mathsf{lift}_{X_k}(x_k) = \varphi_{jk} \circ \mathsf{lift}_{X_j}(x_j)$. From this, we can observe that $\beta_i \circ \mathsf{lift}_{X_i}(x_i) = \beta_k \circ \varphi_{ik} \circ \mathsf{lift}_{X_i}(x_i) = \beta_k \circ \varphi_{jk} \circ \mathsf{lift}_{X_j}(x_j) = \beta_j \circ \mathsf{lift}_{X_j}(x_j)$. That is, the definition of $\mathsf{lift}_X(x)$ is not dependent on the particular factorization.

To verify the compatibility condition, we pull back the horn along s_p for any $0 \leq p \leq n$, and we also consider $s_p^*(x) \colon \Lambda^{n+1}_{m^*} \to X$ defined in **Definition 10**, and we will show $\mathsf{lift}_X(s_p^*(x)) = \mathsf{lift}_X(x) \circ s_p$ as in **Definition 11**. For any $i \in \mathcal{I}$, consider the map $s_p^*(x_i)$ of x_i along s_p as shown in the diagram below. We claim that $s_p^*(x)$ factors through X_i with $s_p^*(x) = \beta_i \circ s_p^*(x_i)$. To see this, it suffices to show that $s_p^*(x) \circ d_q = \beta_i \circ s_p^*(x_i) \circ d_q$ for all $q \in [n] \setminus \{m^*\}$ by **Lemma 7**. By **Definition 10**,

$$s_p^*(x) \circ d_q = \begin{cases} x \circ s_p \circ d_q & \text{if } q \neq p, p+1, m^* \\ \mathsf{lift}_X(x) & \text{if } q \in \{p, p+1\} \setminus \{m^*\}, \end{cases}$$

and also

$$\beta_i \circ s_p^*(x_i) \circ d_q = \begin{cases} \beta_i \circ x_i \circ s_p \circ d_q & \text{if } q \neq p, p+1, m^* \\ \beta_i \circ \mathsf{lift}_{X_i}(x_i) & \text{if } q \in \{p, p+1\} \setminus \{m^*\}. \end{cases}$$

Now, comparing both cases for q, we see that $x \circ s_p \circ d_q = \beta_i \circ x_i \circ s_p \circ d_q$ for the first case, since $x = \beta_i \circ x_i$. Also, $\mathsf{lift}_X(x) = \beta_i \circ \mathsf{lift}_{X_i}(x_i)$ for the second case by the definition of $\mathsf{lift}_X(x)$ provided earlier in (5). Hence, this proves the claim about the factorization.

$$\begin{tikzcd}
\Lambda^{n+1}_{m^*} \arrow[rrr, bend left, "s_p^*(x)"] \arrow[d, hook] \arrow[rrd, bend right, "s_p^*(x_i)"'] & \Lambda^n_m \arrow[rr, "x"'] \arrow[rd, "x_i"] \arrow[d, hook] & & X \\
& & X_i \arrow[ru, "\beta_i"'] & \\
\Delta^{n+1} \arrow[r, "s_p"'] \arrow[rru, dashed, "\mathsf{lift}_{X_i}(s_p^*(x_i))"] & \Delta^n \arrow[ru, dashed, "\mathsf{lift}_{X_i}(x_i)"'] & &
\end{tikzcd}$$

Due to this factorization, we can use (5) to define $\mathsf{lift}_X(s_p^*(x)) = \beta_i \circ \mathsf{lift}_{X_i}(s_p^*(x_i))$ independently of i. Then, using the compatibility for X_i, we have $\mathsf{lift}_X(s_p^*(x)) = \beta_i \circ \mathsf{lift}_{X_i}(s_p^*(x_i)) = \beta_i \circ \mathsf{lift}_{X_i}((x_i) \circ s_p) = \mathsf{lift}_X(x) \circ s_p$, which is the compatibility for X, and thus (X, lift_X) is an effective Kan complex.

Next, we check the coprojections are maps in **EffKanCplx**. For each β_i and each $\overline{x_i} \colon \Lambda^n_m \to X_i$, we have $\mathsf{lift}_X(\beta_i \circ \overline{x_i}) = \beta_i \circ \mathsf{lift}_{X_i}\bar{x_i}$ by definition, meaning exactly that β_i is in **EffKanCplx**.

Finally, we show that the cocone $\big((X, \mathsf{lift}_X), \{\beta_i \colon X_i \to X\}_{i \in I}\big)$ is colimiting. Let $\big((Y, \mathsf{lift}_Y), \{\gamma_i \colon X_i \to Y\}_{i \in I}\big)$ be any cocone of the morphisms in **EffKanCplx** over F. Forgetting the data regarding the lifting structure, the maps γ_i form a cocone of simplicial-set maps over $U \circ F$. Since $X = \mathrm{colim}_{i \in I} X_i$ in **sSet**, the universal property of the colimit gives a unique map $\theta \colon X \to Y$ such that $\theta \circ \beta_i = \gamma_i$ for all $i \in \mathcal{I}$. Then θ preserves lifts as follows: $\theta \circ \mathsf{lift}_X(x) = \theta \circ \beta_i \circ \mathsf{lift}_{X_i}(x_i) = \gamma_i \circ \mathsf{lift}_{X_i}(x_i) = \mathsf{lift}_Y(\gamma_i \circ x_i) = \mathsf{lift}_Y(\theta \circ \beta_i \circ x_i) = \mathsf{lift}_Y(\theta \circ x)$. Hence θ is a morphism in **EffKanCplx**. For the uniqueness, suppose $\psi \colon X \to Y$ is another morphism with $\psi \circ \beta_i = \gamma_i$ for all $i \in \mathcal{I}$. But then $U(\psi) = U(\theta)$ implies $\psi = \theta$ by the faithfulness of U.

Consequently the cocone $\big((X, \mathsf{lift}_X), \{\beta_i\}_{i\in I}\big)$ is colimiting in **EffKanCplx**, and thus (X, lift_X) is a filtered colimit in **EffKanCplx**. ◀

▶ Remark 19. As mentioned before, using analogous reasoning, this proposition can be extended to the category **EffKan**/A easily, and further generalized to **EffKanFib**, provided that similar assumptions hold for the mediating maps in both cases, although we only require the cases for **EffKanCplx** and **EffKan**/A in this paper.

Let us also look at limits.

▶ **Proposition 20.** *The forgetful functor* **EffKanCplx** → **sSet** *creates small limits.*

Proof. Let $F\colon \mathcal{I} \to \textbf{EffKanCplx}$ be a diagram for a small category $\mathcal{I}$. Denote $F(i) = (X_i, \mathsf{lift}_{X_i})$. Using the forgetful functor $U\colon \textbf{EffKanCplx} \to \textbf{sSet}$ that sends $(X_i, \mathsf{lift}_{X_i})$ to X_i, we want to show that if X is the limit of $(X_i\colon i \in \mathcal{I})$ in **sSet** (under $U \circ F$), and each X_i has the structure of an effective Kan complex $(X_i, \mathsf{lift}_{X_i})$, then so does X.

For each $i \in \mathcal{I}$, let $\pi_i\colon X \to X_i$ denote the projection. Fix an arbitrary horn problem $x\colon \Lambda^n_m \to X$. For each X_i, the composition $\pi_i \circ x\colon \Lambda^n_m \to X_i$ admits a lift $\mathsf{lift}_{X_i}(\pi_i \circ x)\colon \Delta^n \to X_i$. Now, since each $\varphi_{ii'}\colon X_i \to X'_i$ is a morphism in **EffKanCplx**, we have $\varphi_{ii'} \circ \mathsf{lift}_{X_i}(\pi_i \circ x) = \mathsf{lift}_{X'_i}(\varphi_{ii'} \circ \pi_i \circ x) = \mathsf{lift}_{X'_i}(\pi_{i'} \circ x)$ for all $i, i' \in \mathcal{I}$. Hence, the collection of maps $\mathsf{lift}_{X_k}(\pi_k \circ x)$ over $k \in \mathcal{I}$ forms a cone over the diagram F. By the universal property of X, there exists the unique map $\langle \mathsf{lift}_{X_k}(\pi_k \circ x)\rangle_{k\in\mathcal{I}}\colon \Delta^n \to X$ such that $\pi_j \circ \langle \mathsf{lift}_{X_k}(\pi_k \circ x)\rangle_{k\in\mathcal{I}} = \mathsf{lift}_{X_j}(\pi_j \circ x)$ for all $j \in \mathcal{I}$.

$$\begin{tikzcd}
\Lambda^n_m \arrow[r, "x"'] \arrow[dd, hook] & X \arrow[d, "\pi_i"] \arrow[dr, "\pi_{i'}"] & \\
& X_i \arrow[r, "\varphi_{ii'}"] & X'_i \\
\Delta^n \arrow[uur, dotted, "\langle \mathsf{lift}_{X_k}(\pi_k \circ x)\rangle_{k\in\mathcal{I}}"] \arrow[ur, dashed, "\mathsf{lift}_{X_i}(\pi_i \circ x)"'] \arrow[uurr, phantom] \arrow[urr, dashed, "\mathsf{lift}_{X'_i}(\pi_{i'} \circ x)"'] & &
\end{tikzcd}$$

We claim that $\mathsf{lift}_X(x) := \langle \mathsf{lift}_{X_k}(\pi_k \circ x)\rangle_{k\in\mathcal{I}}$ gives the desired structure. We need to show its compatibility. Consider $\Lambda^{n+1}_{m^*}$ and an arbitrary s_p. We need to show $\mathsf{lift}_X(s_p^*(x)) = \mathsf{lift}_X(x) \circ s_p$, i.e., using the definition of lift_X that we have just given, we need to show $\langle \mathsf{lift}_{X_k}(\pi_k \circ s_p^*(x))\rangle_{k\in\mathcal{I}} = \langle \mathsf{lift}_{X_k}(\pi_k \circ x)\rangle_{k\in\mathcal{I}} \circ s_p$.

$$\begin{tikzcd}
\Lambda^{n+1}_{m^*} \arrow[rr, bend left, "s_p^*(x)"] \arrow[dd, hook] & \Lambda^n_m \arrow[r, "x"'] \arrow[dd, hook] & X \arrow[d, "\pi_i"] \arrow[dr, "\pi_j"] & \\
& & X_i \arrow[r, "\varphi_{ij}"] & X_j \\
\Delta^{n+1} \arrow[r, "s_p"'] \arrow[uurr, dotted, "\langle \mathsf{lift}_{X_k}(\pi_k \circ s_p^*(x))\rangle_{k\in\mathcal{I}}"] & \Delta^n \arrow[uur, dotted, "\langle \mathsf{lift}_{X_k}(\pi_k \circ x)\rangle_{k\in\mathcal{I}}"] \arrow[ur, dashed, "\mathsf{lift}_{X_i}(\pi_i \circ x)"'] \arrow[urr, dashed, "\mathsf{lift}_{X_j}(\pi_j \circ x)"'] & &
\end{tikzcd}$$

First, we claim $s_p^*(\pi_j \circ x) = \pi_j \circ s_p^*(x)$ for each $j \in \mathcal{I}$. As before, we examine all possible faces d_q to see $s_p^*(\pi_j \circ x) \circ d_q = \pi_j \circ s_p^*(x) \circ d_q$.

$$s_p^*(\pi_j \circ x) \circ d_q = \begin{cases} \pi_j \circ x \circ s_p \circ d_q & \text{if } q \neq p, p+1, m^* \\ \mathsf{lift}_X(\pi_j \circ x) & \text{if } q \in \{p, p+1\} \setminus \{m^*\}. \end{cases}$$

$$\pi_j \circ s_p^*(x) \circ d_q = \begin{cases} \pi_j \circ x \circ s_p \circ d_q & \text{if } q \neq p, p+1, m^* \\ \pi_j \circ \mathsf{lift}_{X_j}(x) & \text{if } q \in \{p, p+1\} \setminus \{m^*\}. \end{cases}$$

In the first case they coincide. In the second case, using lift_X defined earlier, we have $\pi_j \circ \mathsf{lift}_X(x) = \pi_j \circ \langle \mathsf{lift}_{X_k}(\pi_k \circ x)\rangle_{k \in I} = \mathsf{lift}_X(\pi_j \circ x)$. Thus, both sides are equal for all possible faces and the claim follows.

Now, because of this claim, our goal is to show $\langle \mathsf{lift}_{X_k}(s_p^*(\pi_j \circ x))\rangle_{k \in \mathcal{I}} = \langle \mathsf{lift}_{X_k}(\pi_k \circ x)\rangle_{k \in \mathcal{I}} \circ s_p$. By the compatibility condition for X_i, we have $\mathsf{lift}_{X_i}(s_p^*(\pi_i \circ x)) = \mathsf{lift}_{X_i}(\pi_i \circ x) \circ s_p$. Then by the universal property of X, we see that $\langle \mathsf{lift}_{X_k}(s_p^*(\pi_j \circ x))\rangle_{k \in \mathcal{I}} = \langle \mathsf{lift}_{X_k}(\pi_k \circ x)\rangle_{k \in \mathcal{I}} \circ s_p$, as desired. Thus, (X, lift_X) is an effective Kan complex.

It is immediate that the projections are morphisms in **EffKanCplx**, as $\pi_i \circ \mathsf{lift}_X(x) = \pi_i \circ \left\langle \mathsf{lift}_{X_k}(\pi_k \circ x)\right\rangle_{k \in I} = \mathsf{lift}_{X_i}(\pi_i \circ x)$ by the definition of our lift lift_X.

Let $\big((Y, \mathsf{lift}_Y), \{\tau_i : Y \to X_i\}_{i \in I}\big)$ be any other cone in **EffKanCplx** over F. Forgetting fillers, the maps $\{\tau_i\}$ form a cone in **sSet** over $U \circ F$, so the universal property of the limit gives a unique map $\theta \colon Y \to X$ such that $\pi_i \circ \theta = \tau_i$ for all $i \in \mathcal{I}$. To show that θ preserves lifts, we have $\pi_i \circ \theta \circ \mathsf{lift}_Y(x) = \tau_i \circ \mathsf{lift}_Y(x) = \mathsf{lift}_{X_i}(\tau_i \circ x) = \mathsf{lift}_{X_i}(\pi_i \circ \theta \circ x) = \pi_i \circ \mathsf{lift}_X(\theta \circ x)$. Since the projections are jointly monic in **sSet**, we have $\theta \circ \mathsf{lift}_Y(x) = \mathsf{lift}_X(\theta \circ x)$. Thus, θ is a morphism in **EffKanCplx**. For the uniqueness, if $\psi : Y \to X$ is another morphism in **EffKanCplx** with $\pi_i \circ \psi = \tau_i$, then $U(\psi) = U(\theta)$ by the uniqueness of the limit in **sSet**, and by the faithfulness of U, we have $\psi = \theta$. Therefore, the cone $\big((X, \mathsf{lift}_X), \{\pi_i\}_{i \in I}\big)$ is limiting in **EffKanCplx**, and thus (X, lift_X) is a limit in **EffKanCplx**. ◀

3 W-types and variations

We introduce the categorical definition of W-types to show that effective Kan complexes model them. While our primary goal is this categorical perspective, a brief note about the type-theoretic development is as follows: W-types were introduced by Martin-Löf in the late 1970s in [10], and most recently their type-theoretic aspects are summarized in "the HoTT book" [12]. These types generalize structures such as natural numbers, lists, and binary trees, encapsulating the recursive properties of inductive types.

3.1 W-types

We summarize the categorical formulation of W-types as described in [3]. For an endofunctor $F \colon \mathcal{E} \to \mathcal{E}$, an *algebra* is a pair (X, α) with $\alpha \colon FX \to X$, and morphisms $\varphi \colon (X, \alpha) \to (Y, \beta)$ satisfy $\varphi \circ \alpha = \beta \circ F\varphi$. The *initial algebra* is the initial object in this category of F-algebras, if it exists.

In a locally cartesian closed category $\mathcal{E}$, for every $f \colon A \to B$, the pullback functor $f^* \colon \mathcal{E}/B \to \mathcal{E}/A$ has left and right adjoints, Σ_f and Π_f, forming $\Sigma_f \dashv f^* \dashv \Pi_f$. In particular, for the unique morphism $! \colon A \to 1$, the adjunction $\Sigma_! \dashv !^* \dashv \Pi_!$ is written as $\Sigma_A \dashv A^* \dashv \Pi_A$. Note that A^* is also written as $- \times A$.

▶ **Definition 21.** *In a locally cartesian closed category $\mathcal{E}$, given $f\colon A \to B$, the polynomial functor P_f is the composite $\Sigma_A \circ \Pi_f \circ B^*$.*

$$P_f\colon \mathcal{E} \xrightarrow{B^*} \mathcal{E}/B \xrightarrow{\Pi_f} \mathcal{E}/A \xrightarrow{\Sigma_A} \mathcal{E}, \tag{6}$$

The W-type $W(f)$ is the initial algebra of P_f, if it exists.

To describe W-types in **sSet**, it suffices to describe them in presheaves.

3.1.1 W-types in Set

Given a polynomial functor $P_f(X) = \sum_{a\in A} X^{B_a}$, where $B_a = f^{-1}(a)$, a W-type consists of labeled, well-founded trees with edges directed towards the root. Each $a \in A$ is a node, and $b \in B_a$ is an edge leading to the node a. Each $\ell \in A$ with $B_\ell = \varnothing$ is the end of a branch (namely, a leaf node) in the tree. Thus, for example, a branch would look like $\ell \xrightarrow{b} a \xrightarrow{b'} a' \xrightarrow{b} a \cdots \to r$ with the root r being some non-leaf node.

For a P_f-algebra (X, μ) with structure map $\mu\colon P_f(X) \to X$, restricting μ to the a-th summand induces the component map $\mu_a\colon X^{B_a} \to X$. With this in mind, we see how an important structure map called sup is obtained. The collection $W(f)$ of well-founded trees is inductively constructed as follows: Each tree is of the form $\mathrm{sup}_a(t)$, where $a \in A$ and $t\colon B_a \to W(f)$ assigns subtrees to branches labeled by B_a. The base case includes trees with no branches ($B_a = \emptyset$). Larger trees are constructed by attaching $t(b) \in W(f)$ for each $b \in B_a$. The map $\mathrm{sup}\colon P_f(W(f)) \to W(f)$ given by $\mathrm{sup}_a\colon W(f)^{B_a} \to W(f)$ equips $W(f)$ with a P_f-algebra structure. Then, inductively we can see that for any other P_f-algebra (X, μ), there is a unique map $\varphi\colon W(f) \to X$ with $\varphi(\mathrm{sup}_a(t)) = \mu_a(\varphi \circ t)$ for any $a \in A$ and $t \in W(f)^{B_a}$, which shows that $W(f)$ is the W-type associated to f. Concretely, the rank function $\mathrm{rk}\colon W(f) \to \mathit{Ord}$ assigns ordinals to trees based on their well-foundedness: $\mathrm{rk}(\mathrm{sup}_a(t)) = \sup\{\mathrm{rk}(t(b)) + 1 \mid b \in B_a\}$. Furthermore, we define $W(f)_{<\alpha} := \{w \in W(f) : \mathrm{rk}(w) < \alpha\}$. Then, $W(f)_{<0} = \emptyset$, $W(f)_{<\alpha+1} \cong P_f(W(f)_{<\alpha})$, and $W(f)_{<\lambda} = \mathrm{colim}_{\alpha<\lambda} W(f)_{<\alpha}$, for limit ordinals λ. By transfinite induction on $w \in W(f)$, we can show that $\mathrm{rk}(w) < \kappa$ for some sufficiently large regular cardinal κ, implying $W(f) = W(f)_{<\kappa}$. This transfinite chain of sets:

$$0 \xrightarrow{!} P_f(0) \xrightarrow{P_f(!)} P_f(P_f(0)) \xrightarrow{P_f(P_f(!))} P_f(P_f(P_f(0))) \xrightarrow{P_f(P_f(P_f(!)))} \cdots \tag{7}$$

converges to $W(f)$. Here, 0 denotes the initial object, which is $\varnothing$ in **Set**. Note that $P_f^\alpha(0) = W(f)_{<\alpha}$.

3.1.2 W-types in $\mathbf{Set}^{\mathcal{C}^{\mathrm{op}}}$

We show that the category of presheaves $\mathbf{Set}^{\mathcal{C}^{\mathrm{op}}}$, where $\mathcal{C}$ is a small category, has all W-types. Given a map $f\colon B \to A$ in $\mathbf{Set}^{\mathcal{C}^{\mathrm{op}}}$, define $\hat{A} = \{(C, a) \mid C \in \mathcal{C},\ a \in A(C)\}$ and $\hat{B}_{(C,a)} = \{(\alpha\colon D \to C,\ b \in B(D)) \mid f_D(b) = \alpha^*(a)\}$, where $\alpha^*(a) := A\alpha(a)$. Let $\hat{f}\colon \sum_{(C,a)\in\hat{A}} \hat{B}_{(C,a)} \to \hat{A}$ be the projection. Since $\hat{A}$ and $\hat{B}_{(C,a)}$ are sets, we consider $W(\hat{f})$. Since $(W(\hat{f}), \mathrm{sup})$ is the initial $P_{\hat{f}}$-algebra, by Lambek's lemma, the structure map $\mathrm{sup}\colon P_{\hat{f}}W(\hat{f}) \xrightarrow{\cong} W(\hat{f})$ is an isomorphism, so every $w \in W(\hat{f})$ is uniquely of the form $\mathrm{sup}_{(C,a)}(t)$ with $t \in W(\hat{f})^{\hat{B}_{(C,a)}}$. We give $W(\hat{f})$ a presheaf structure over $\mathcal{C}$. On objects, $W(\hat{f})(C)$ consists of trees rooted at C. On morphisms, for $\alpha\colon D \to C$, define $\alpha^*\big(\mathrm{sup}_{(C,a)}(t)\big) = \mathrm{sup}_{(D,\alpha^*(a))}(\alpha^* t)$, where $\alpha^* t$ is defined by $(\alpha^* t)(\beta, b) = t(\alpha\beta, b)$. Then, we can see that this is functorial, so $W(\hat{f})$ is a presheaf. Assign ranks to elements by $\mathrm{rk}(\mathrm{sup}_{(C,a)}(t)) = \sup\{\mathrm{rk}(t(\beta, b)) + 1 \mid (\beta, b) \in \hat{B}_{(C,a)}\}$. Define subpresheaves as $W(\hat{f})_{<\alpha} = \{w \in W(\hat{f}) \mid \mathrm{rk}(w) < \alpha\}$. To obtain $W(f)$, select hereditarily natural trees: A tree

$\sup_{(C,a)}(t)$ is *natural* if for all $(\alpha\colon D \to C,\ b) \in \hat{B}_{(C,a)}$ and $\beta\colon E \to D$, it lives in the fiber over $\mathrm{dom}(\alpha)$, and $t\big(\alpha\beta,\ \beta^*(b)\big) = \beta^*\big(t(\alpha, b)\big)$. A tree is *hereditarily natural* if all its subtrees are natural. These hereditarily natural trees form a subpresheaf $W(f) \subseteq W(\hat{f})$, and $W(f)_{<\alpha} := W(\hat{f})_{<\alpha} \cap W(f)$ is also a presheaf. Then, again, $W(f)_{<0} = 0$, $W(f)_{<\alpha+1} = P_f(W(f)_{<\alpha})$ and $W(f)_{<\lambda} = \mathrm{colim}_{\alpha<\lambda} W(f)_{<\alpha}$(for limit λ). This chain, the same as (7), converges to $W(f)$, since $W(f) = W(f)_{<\kappa}$ for large enough κ.

Setting $\mathcal{C} = \Delta$, we view W-types $W(f)$ in **sSet** as filtered colimits, which motivated our prior investigation.

▶ Remark 22. The categorical construction of W-types matches their definition in HoTT [12]. Given a type A and a type family $B\colon A \to \mathcal{U}$, the inductive type W-type $\mathsf{W}_{(a:A)}B(a)$ is defined by the constructor sup and induction principle. The constructor is specified as follows: For each $a : A$ and function $f\colon B(a) \to \mathsf{W}_{(a:A)}B(a)$, there exists a term $\mathsf{sup}(a, f) : \mathsf{W}_{(a:A)}B(a)$.

3.2 Initial algebras for dependent polynomial functors

Now let us introduce the first variation. As seen above, a W-type is the initial algebra for a polynomial functor. We can generalize the notion of polynomial functors as follows:

▶ **Definition 23.** *Suppose we are given a diagram of the form*

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

in a locally cartesian closed category $\mathcal{E}$. Then this diagram determines an endofunctor on $\mathcal{E}/C$, the dependent polynomial functor

$$D_f\colon \mathcal{E}/C \xrightarrow{h^*} \mathcal{E}/B \xrightarrow{\Pi_f} \mathcal{E}/A \xrightarrow{\Sigma_g} \mathcal{E}/C.$$

In **Set**, it behaves as $D_f(X)_c = \sum_{a\in A_c} \prod_{b\in B_a} X_{h(b)}$. The initial algebra for D_f is a subset of $W(f)$, defined by the condition that an edge labeled by $b \in B$ with source node labeled by $a \in A$ satisfies $g(a) = h(b)$. The concept of rank extends naturally from W-types, and this initial algebra generalizes from **Set** to presheaves, in particular **sSet**.

3.3 M-types

As a dual to a W-type, an *M-type* is defined as the final coalgebra of a polynomial functor associated with a morphism $f : B \to A$ in a locally cartesian closed category $\mathcal{E}$.

In both **Set** and **sSet**, this final coalgebra, denoted $M(f)$, captures both well-founded and non-well-founded trees labeled according to f. The construction of $M(f)$ can be understood as the limit of the following chain:

$$\cdots \longrightarrow P_f(P_f(P_f(1))) \longrightarrow P_f(P_f(1)) \longrightarrow P_f(1) \longrightarrow 1,$$

where P_f is the polynomial functor associated with f, and 1 denotes the terminal object. Notably, this sequence stabilizes at the ordinal ω, leading to the isomorphism $M(f) \cong \lim_{n\in\mathbb{N}} P_f^n(1)$. Thus, unlike certain W-types which require transfinite recursion to generate new well-founded trees, M-types do not require it.

4 P_f on effective Kan fibrations

Before stating the main theorem, we lift each functor in the composite D_f from a functor on the slice categories of **sSet** to a functor on the categories of **EffKan**$/(-)$. In particular, we obtain the result for P_f. Recall that D_f is of the form $\Sigma_g \Pi_f h^*$. First let us look at the second component, Π_f. The result was established by van den Berg and Faber [2]:

▶ **Lemma 24.** *The functor* $\Pi_f \colon \mathbf{sSet}/B \to \mathbf{sSet}/A$ *lifts to* $\Pi_f \colon \mathbf{EffKan}/B \to \mathbf{EffKan}/A$ *with an effective Kan fibration* $f \colon B \to A$*.*

For the first component (the pullback functor) of D_f, we first

▶ **Lemma 25.** *Effective Kan fibrations are stable under pullbacks along any map.*

Proof. Let (p, lift_p) with $p \colon X \to A$ be an effective Kan fibration, and $a \colon B \to A$ be any simplicial map. Consider an arbitrary horn problem (x, b) against the map a^*p. Then, we have the lift $\mathsf{lift}_p(\varphi \circ x, a \circ b)$, which we denote by γ.

In the inner left commutative square below, by the universal property of $B \times_A X$, there exists a unique morphism $\langle \gamma, b \rangle$, as shown in the diagram. We claim $\mathsf{lift}_{a^*p}(x, b) := \langle \gamma, b \rangle$ is well-defined.

$$\begin{tikzcd}
\Lambda^n_m \arrow[r, "x"] \arrow[d, "\iota"'] & B \times_A X \arrow[r, "\varphi"] \arrow[d, "a^*p"'] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[d, "p"] \\
\Delta^n \arrow[r, "b"'] \arrow[ur, dashed, "\langle \gamma, b \rangle"] \arrow[urr, dashed, "\gamma"'] & B \arrow[r, "a"'] & A
\end{tikzcd}$$

First, to see it is a lift, clearly the lower triangle commutes ($a^*p \circ \langle \gamma, b \rangle = b$). For the upper triangle, we have $\varphi \circ \langle \gamma, b \rangle \circ \iota = \gamma \circ \iota = \varphi \circ x$, and $a^*p \circ \langle \gamma, b \rangle \circ \iota = b \circ \iota = a^*p \circ x$. Then, by universal property of $B \times_A X$, we obtain $\langle \gamma, b \rangle \circ \iota = x$.

Next, we need to show this lift satisfies the compatibility condition. For every possible j and m^*, we have the following diagram.

$$\begin{tikzcd}
\Lambda^{n+1}_{m^*} \arrow[rr, bend left, "s_j^*(x)"] \arrow[d] & \Lambda^n_m \arrow[r, "x"] \arrow[d, "\iota"] & B \times_A X \arrow[r, "\varphi"] \arrow[d, "a^*p"] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[d, "p"] \\
\Delta^{n+1} \arrow[r, "s_j"'] \arrow[urr, dashed, "\langle \gamma \circ s_j, b \circ s_j \rangle"] \arrow[urrr, dashed, "\gamma \circ s_j"] & \Delta^n \arrow[r, "b"'] \arrow[urr, dashed, "\gamma"] & B \arrow[r, "a"'] & A
\end{tikzcd}$$

By the compatibility of p, $\mathsf{lift}_p(\varphi \circ s_j^*(x), a \circ b \circ s_j) = \gamma \circ s_j$. Then, by the universal property of $B \times_A X$, we obtain $\langle \gamma \circ s_j, b \circ s_j \rangle$, and $\mathsf{lift}_{a^*p}(s_j^*(x), b \circ s_j) = \langle \gamma \circ s_j, b \circ s_j \rangle$ for the same reason as above. Now, since $\varphi \circ \langle \gamma \circ s_j, b \circ s_j \rangle = \gamma \circ s_j = \varphi \circ \langle \gamma, b \rangle \circ s_j$ and $a^*p \circ \langle \gamma \circ s_j, b \circ s_j \rangle = b \circ s_j = a^*p \circ \langle \gamma, b \rangle \circ s_j$, by the universal property, we obtain $\langle \gamma \circ s_j, b \circ s_j \rangle = \langle \gamma, b \rangle \circ s_j$, which shows the compatibility of a^*p. ◀

▶ **Lemma 26.** *For a simplicial-set map* $h \colon B \to C$*, the functor* $h^* \colon \mathbf{sSet}/C \to \mathbf{sSet}/B$ *lifts to* $h^* \colon \mathbf{EffKan}/C \to \mathbf{EffKan}/B$*.*

Proof. First, let us see how it is defined on objects. Given an effective Kan fibration (k, lift_k) with $k\colon X \to C$, let us define $h^*\big((k, \mathsf{lift}_k)\big) := (h^*k, \mathsf{lift}_{h^*k})$, where lift_{h^*k} is described in **Lemma 25**, by extending $h^*\colon \mathbf{sSet}/C \to \mathbf{sSet}/B$. This is well-defined: The first component is inherited from the underlying simplicial sets, and the second component is uniquely determined by the universal property of the pullback.

Next, we see that it is well-defined on morphisms. Take $(k, \mathsf{lift}_k), (l, \mathsf{lift}_l) \in \mathbf{EffKan}/C$. Suppose we are given a pair of maps $\varphi\colon X \to Y$ and $1_C\colon C \to C$ which together form a morphism $(k, \mathsf{lift}_k) \to (l, \mathsf{lift}_l)$ in $\mathbf{EffKan}/C$. We need to show that a pair of maps $h^*\varphi$ and 1_B is a morphism in $\mathbf{EffKan}/B$. That is, for an arbitrary horn problem as in the following, we need to show $h^*\varphi \circ L_1 = L_2$, where $L_1 = \mathsf{lift}_{h^*k}(x, b)$ and $L_2 = \mathsf{lift}_{h^*l}(h^*\varphi \circ x, b)$.

$$\begin{tikzcd}
 & & X \arrow[rr, "\varphi"] \arrow[dd, "k"] & & Y \arrow[dd, "l"] \\
\Lambda^n_m \arrow[r, "x"] \arrow[dd, hook, "\iota"'] & B \times_C X \arrow[ur, "k^*h"] \arrow[rr, "h^*\varphi"] \arrow[dd, "h^*k"] & & B \times_C Y \arrow[ur, "l^*h"] \arrow[dd, "h^*l"] & \\
 & & C \arrow[rr, "1_C"] & & C \\
\Delta^n \arrow[r, "b"] \arrow[uur, dashed, "L_1"] \arrow[uurrr, dashed, "L_2"] & B \arrow[ur, "h"] \arrow[rr, "1_B"] & & B \arrow[ur, "h"] &
\end{tikzcd}$$

As a prism diagram, each face of this cube is commutative. We see that

$$\begin{aligned}
l^*h \circ h^*\varphi \circ \mathsf{lift}_{h^*k}(x, b) &= \varphi \circ k^*h \circ \mathsf{lift}_{h^*k}(x, b) && \text{(top face of the cube)}\\
&= \varphi \circ \mathsf{lift}_k(k^*h \circ x, h \circ b) && \text{(by } \textbf{Lemma 25}\text{)}\\
&= \mathsf{lift}_l(\varphi \circ k^*h \circ x, h \circ b) && (\varphi \text{ is a map in } \mathbf{EffKan}/C)\\
&= \mathsf{lift}_l(l^*h \circ h^*\varphi \circ x, h \circ b) && \text{(top face of the cube)}\\
&= l^*h \circ \mathsf{lift}_{h^*l}(h^*\varphi \circ x, b) && \text{(by } \textbf{Lemma 25}\text{).}
\end{aligned}$$

Also,

$$\begin{aligned}
h^*l \circ h^*\varphi \circ \mathsf{lift}_{h^*k}(x, b) &= h^*k \circ \mathsf{lift}_{h^*k}(x, b) && \text{(front face of the cube)}\\
&= b && \text{(by } \textbf{Lemma 25}\text{)}\\
&= h^*l \circ \mathsf{lift}_{h^*l}(h^*\varphi \circ x, b). && \text{(by } \textbf{Lemma 25}\text{)}
\end{aligned}$$

Thus, by the universal property of $B \times_C Y$, we have $h^*\varphi \circ \mathsf{lift}_{h^*k}(x, b) = \mathsf{lift}_{h^*l}(h^*\varphi \circ x, b)$ as desired, and h^* preserves a morphism in $\mathbf{EffKan}/B$. It obviously preserves the identity. Also, for $\psi\colon Y \to Z$, we have $h^*\big((\varphi, 1_C) \circ (\psi, 1_C)\big) = h^*(\varphi, 1_C) \circ h^*(\psi, 1_C)$, especially because φ and ψ are morphisms in $\mathbf{EffKan}/C$, and they respect lifts. ◀

Setting $C = 1$, we have the following.

▶ **Corollary 27.** *We obtain the functor* $B^*\colon \mathbf{EffKanCplx} \to \mathbf{EffKan}/B$*.*

For the last component $\Sigma_{(-)}$ of D_f, we first show this lemma.

▶ **Lemma 28.** *Effective Kan fibrations are stable under composition.*

Proof. Let (p, lift_p) with $p\colon X \to A$ be an effective Kan fibration, and $g\colon A \to C$ be an effective Kan fibration. Consider an arbitrary horn problem (x, c) against the map $g \circ p$. Using lift_g we have $\mathsf{lift}_g(p \circ x, c)$. Then we obtain the lift $\mathsf{lift}_p(x, \mathsf{lift}_g(p \circ x, c))$, which we call α in the diagram.

$$\begin{tikzcd}
\Lambda^n_m \arrow[r, "x"] \arrow[dd, "\iota"'] & X \arrow[d, "p"] \\
& A \arrow[d, "g"] \\
\Delta^n \arrow[r, "c"'] \arrow[uur, dashed, "\alpha"] \arrow[ur, "\mathsf{lift}_g(p\circ x, c)"'] & C
\end{tikzcd}$$

Then, this α is also a lift for $g \circ p$, since the lower triangle commutes: $g \circ p \circ \alpha = g \circ \mathsf{lift}_g(p \circ x, c) = x$. Thus, we can define $\mathsf{lift}_{g\circ p}(x, c) := \alpha$. The compatibility follows from the compatibility of lift_p and lift_q: for every possible j and m^*, we have $\mathsf{lift}_{g\circ p}(x, c) \circ s_j = \mathsf{lift}_p(x, \mathsf{lift}_g(p \circ x, c)) \circ s_j = \mathsf{lift}_p(s_j^*(x), \mathsf{lift}_g(p \circ x, c) \circ s_j) = \mathsf{lift}_p(s_j^*(x), \mathsf{lift}_g(p \circ s_j^*(x), c \circ s_j)) = \mathsf{lift}_{g\circ p}(s_j^*(x), c \circ s_j)$. The last equation is by our definition of $\mathsf{lift}_{g\circ p}$. ◀

▶ **Corollary 29.** *Given an effective Kan fibration $p\colon X \to A$, if A is an effective Kan complex, then so is X.*

Proof. By **Lemma 28**, the unique map $!_A \circ p = !_X$ is effective Kan, i.e., X is effective Kan. ◀

▶ **Lemma 30.** *$\Sigma_g\colon$ **EffKan**$/A \to$ **EffKan**$/C$ for an effective Kan fibration $g\colon A \to C$ is a functor.*

Proof. For objects in **EffKan**$/A$, Σ_g is defined by postcomposition with g. Thus, Σ_g is well-defined on objects as seen in **Lemma 28**.

For morphisms, let $(p, \mathsf{lift}_p), (p', \mathsf{lift}_{p'}) \in$ **EffKan**$/A$. Suppose we have maps $\varphi\colon X \to Y$ and $1_A\colon A \to A$ such that $(\varphi, 1_A)$ is a morphism in **EffKan**$/A$ from (p, lift_p) to $(p', \mathsf{lift}_{p'})$. We need to show that $\Sigma_g(\varphi, 1_A)$ is a morphism in **EffKan**$/C$. This follows immediately. By construction, as established in **Lemma 28**, the lifts associated with $g \circ p$ and $g \circ p'$ are created by p and p', respectively. Also it is immediate that Σ_g preserves the identities and respects composition. ◀

Setting $C = 1$, we have the following.

▶ **Corollary 31.** *We obtain the functor $\Sigma_A\colon$ **EffKan**$/A \to$ **EffKanCplx** for an effective Kan complex A.*

Due to the functoriality of each component of P_f in **Lemma 24**, **Corollary 27**, and **Corollary 31**, a polynomial functor P_f on **sSet** is extended to a functor on **EffKanCplx**, preserving morphisms in **EffKanCplx** in particular.

▶ **Corollary 32.** *We have a functor $P_f\colon$ **EffKanCplx** $\to$ **EffKanCplx** for an effective Kan fibration $f\colon B \to A$.*

Let us observe the following as well.

▶ **Lemma 33.** *The map $P_f(X) \to A$, where $f\colon B \to A$ is an effective Kan fibration and X and A are effective Kan complexes, is an effective Kan fibration.*

Proof. The composite $\Pi_f !_B^*$ preserves effective Kan fibrations, and $\Pi_f !_B^*(X \to 1) = P_f(X) \to A$. ◀

Then, by **Corollary 29**, the following holds.

▶ **Corollary 34.** *For an effective Kan fibration $f\colon B \to A$ between effective Kan complexes, if X is an effective Kan complex, so is $P_f(X)$.*

5 Main results

5.1 W-types

Now, we present the main theorem in this paper. However, readers are advised to refer to **Remark 36** following the proof, which offers additional considerations on the limitations of the argument presented.

▶ **Theorem 35.** *If $f\colon B \to A$ is an effective Kan fibration between effective Kan complexes, then, for any ordinal α, the map $W(f)_{<\alpha} \to A$ is also an effective Kan fibration with $W(f)_{<\alpha}$ being an effective Kan complex; in particular, $W(f) \to A$ is an effective Kan fibration with $W(f)$ being an effective Kan complex.*

Proof. We argue by transfinite induction. For the base case $\alpha = 0$, the claim vacuously holds. Also, the unique mediating map $!\colon 0 \to P_f(0)$, which is trivially a morphism in **EffKanCplx**, is carried to the morphism $\Pi_f B^*(!)$ in **EffKan**$/A$.

For a successor ordinal $\alpha + 1$, suppose $P_f^\alpha(0) \to A$ is effective Kan, and $P_f^\alpha(!)\colon P_f^\alpha(0) \to P_f^{\alpha+1}(0)$ is a morphism in **EffKanCplx**. Then, by **Lemma 33** and **Corollary 34**, $P_f(P_f^\alpha(0)) \to A$, namely $W(f)_{<\alpha+1} \to A$, is an effective Kan fibration with $W(f)_{<\alpha+1}$ being an effective Kan complex. Also $P_f^{\alpha+1}(!)$ is a morphism in **EffKanCplx** by **Corollary 32**, so the mediating map $\Pi_f B^*\big(P_f^{\alpha+1}(!)\big)$ is in **EffKan**$/A$.

For a limit ordinal α, suppose $P_f^\lambda(0) \to A$ is effective Kan for all $\lambda < \alpha$, and all the relevant mediating maps are in **EffKan**$/A$. Then, by **Remark 19**, the filtered colimit $W(f)_{<\alpha} \to A$ in **EffKan**$/A$ is also an effective Kan fibration, with $W(f)_{<\alpha}$ being an effective Kan complex.

As a result of this transfinite induction, since $W(f) = W(f)_{<\kappa}$ for a large enough κ as seen in the construction of W-types, $W(f) \to A$ is an effective Kan fibration and $W(f)$ is an effective Kan complex, as desired. ◀

▶ Remark 36. As will be discussed in the conclusion, the transfinite induction employed here relies on the classical concept of ordinals. Therefore, a future challenge is to develop a constructive approach in this context.

5.2 Initial algebras for dependent polynomial functors

▶ **Proposition 37.** *If we have a diagram*

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

of Kan fibrations in **EffKan**$/C$*, then the initial D_f-algebra is in* **EffKan**$/C$.

Proof. We saw that this endofunctor D_f is well-defined on $\mathbf{EffKan}/C$ in the previous section (each component of the composite D_f preserves morphisms in its category, which is by **Lemma 24**, **26** and **30**). $\mathbf{EffKan}/C$ has a filtered colimit by **Remark 19**, and it has an initial algebra which can be built as the filtered colimit of a sufficiently long chain of $D_f(0)$. Thus, this initial algebra is in $\mathbf{EffKan}/C$. ◀

5.3 M-types

▶ **Theorem 38.** *If $f\colon B \to A$ is an effective Kan fibration between effective Kan complexes, then $M(f)$ is an effective Kan complex as well.*

Proof. We have $P_f(1) \cong A$, which is in $\mathbf{EffKanCplx}$ by assumption. We know $M(f) \cong \lim_{n\in\mathbb{N}} P_f^n(1)$ as mentioned in section 3.3, and P_f is an endofunctor on $\mathbf{EffKanCplx}$. Thus, by **Proposition 20** this $M(f)$ is in $\mathbf{EffKanCplx}$, i.e., it is an effective Kan complex. ◀

6 Future work

Constructively extending the scope of the original research on W-types [3], we focused on the modeling of W-types using Kan fibrations. As an application of W-types, the original research discussed quotients. Specifically, it addressed the universal Kan fibration $\pi\colon E \to U$, characterized by the property that any other Kan fibration can be obtained as a pullback of π. Then, it explored the application of W-types associated with π, namely $W(\pi)$. As discussed in [3], quotients are required in this application (under certain conditions, quotient types and maps are modeled by fibrant objects and fibrations, respectively). Inspired by this, our potential application would be to develop similar arguments within a constructive framework. Also, regardless of the context, considering quotient types still remain significant. However, this poses challenges, because dealing with epimorphisms in $\mathbf{sSet}$ requires selecting individual elements from their fibers, a process that depends on the axiom of choice. To avoid this, one must establish constructive rules for such selections. These rules, however, need to be carefully designed to maintain consistency across all dimensions while remaining compatible with the constructive model we adopt.

The most challenging part is to show the constructive version of the following proposition.

▶ **Proposition 39.** *In* $\mathbf{sSet}$*, suppose in a commutative triangle*

$$\begin{tikzcd}
X \arrow[rr, "p"] \arrow[dr, "g"'] & & Y \arrow[dl, "f"] \\
& A &
\end{tikzcd}$$

where p is an epi, and both p and g are Kan fibrations. Then f is also a Kan fibration.

The proof for this can be found in [3]. However, in a constructive setting, unless we impose additional structures on the fibrations, it is likely that this cannot even be proven for effective Kan fibrations. As mentioned above, this p needs to be equipped with some structured way to have a section. For this, [5] and [7] formulated the following version of effective Kan fibrations with additional structures:

▶ **Definition 40.** *Let (p, lift_p) be an effective Kan fibration. We say that lift_p gives the structure of a degenerate-preferring Kan fibration, if for all $n \in \mathbb{N}$, all $g \in X_n$, all possible j in s_j and m in Λ^{n+1}_m that makes the following diagram commute, the lift satisfies $\mathsf{lift}_p(gs_j\iota, pgs_j) = gs_j$.*

*We say a fibration with such lifting structure is in **D-pkf** (for the case of fibrant objects X we say X is in **D-pkf**).*

$$\begin{tikzcd}
\Lambda^{n+1}_m \arrow[rr, "g s_j \iota"] \arrow[d, hook, "\iota"'] & & X \arrow[d, "p"] \\
\Delta^{n+1} \arrow[r, "s_j"'] & \Delta^n \arrow[ur, "g"] \arrow[r, "pg"'] & Y
\end{tikzcd}$$

If we give the maps in **Proposition 39** **D-pkf** structure, we can prove the constructive version of this proposition [5].

▶ **Proposition 41.** *Suppose in a commutative triangle*

$$\begin{tikzcd}
X \arrow[rr, "p"] \arrow[dr, "g"'] & & Y \arrow[dl, "f"] \\
& A &
\end{tikzcd}$$

*where p is an epi, and both p and g are **D-pkf**. Also, suppose p has a section $\tilde{p}_0 \colon Y_0 \to X_0$ at the vertices level. Then f is also **D-pkf**.*

Since we need more results besides **Proposition 39** for quotient types, first we need to define the following (by [7]).

▶ **Definition 42.** *Let $q \colon X \to Y$ be a map of simplicial sets. Let $\tilde{q} = (\tilde{q}_n \colon Y_n \to X_n)_{n \in \mathbb{N}}$ be a collection of functions. We call $\tilde{q}$ a degeneracy-section of q if and only if*

- *q has a right inverse $\tilde{q}$ such that $q \circ \tilde{q} = 1_Y$.*
- *for all $n \in \mathbb{N}$ and all $0 \leq j \leq n$, we have $\tilde{q}_{n+1}(s_j^*(y)) = s_j^*\big(\tilde{q}_n(y)\big)$.*

Here, y is a map $y \colon \Delta^n \to Y$ identified as $y \in Y_n$ by Yoneda lemma (and by abuse of notation).

Note that the second condition can be also viewed just as $\tilde{q}_{n+1}(y \circ s_j) = \tilde{q}_n(y) \circ s_j$.

▶ **Definition 43.** *If an effective Kan fibration p has this degeneraracy-section, we say that p is in **DS**.*

First of all, even with the assumption of degeneracy-sections alone, we can at least prove that quotient maps are modeled by effective Kan fibrations.

▶ **Proposition 44.** *If R is an equivalence relation on X, $q \colon X \to X/R$ is in **DS** with a degeneracy-section $\tilde{q}$ and both projections $\pi_i \colon R \to X$ $(i = 1, 2)$ are effective Kan fibrations (with structures lift_{π_1} and lift_{π_2}, respectively), then so is $X \to X/R$.*

For the proof, we can modify the proof of the classical version in [3]. This extra assumption of degeneracy-sections allows us to check the compatibility condition of effective Kan fibrations.

However, to see that the quotient type X/R is modeled by a fibrant object, we need the constructive version of **Proposition 39**, and as mentioned earlier, at least **D-pkf** proves it. Thus, let us first consider the following.

▶ **Definition 45.** *In **Definition 40**, if the **D-pkf** map p has a degeneracy-section $\tilde{p}$, we call it **D-pkf** $\cap$ **DS** (also we use this $\cap$ notation for other instances).*

Using this, we could attempt to prove the constructive version of the corollary for X/R (originally it is "Corollary 4.4." in [3]).

First recall that a pseudo-equivalence relation $(s, t) : R \to X \times X$ satisfies:

- Reflexivity: There exists $\rho : X \to R$ such that $(s, t)\rho$ is the diagonal map Δ_X.
- Symmetry: There exists $\sigma : R \to R$ with $s\sigma = t$ and $t\sigma = s$.
- Transitivity: In the pullback diagram below, there exists $\tau : P \to R$ such that $s\tau = sp_{12}$ and $t\tau = tp_{23}$.

$$\begin{tikzcd}
P \arrow[r, "p_{12}"] \arrow[d, "p_{23}"'] \arrow[dr, phantom, "\lrcorner", very near start] & R \arrow[d, "t"] \\
R \arrow[r, "s"'] & X
\end{tikzcd}$$

The constructive version of "Corollary 4.4." in [3] would then be formulated as follows:

[Quotient : D-pkf ∩ DS] *"Suppose R is a pseudo-equivalence relation on X, the quotient map $q \colon X \to X/R$ has a degeneracy-section $\tilde{q}$, and $R \hookrightarrow X \times X$ is an effective Kan fibration. Moreover, suppose that X is in **D-pkf**. Then the quotient map $X \xrightarrow{q} X/R$ is in **D-pkf**∩ **DS**, and X/R is in **D-pkf**."*

Once we can show that q is in **D-pkf** ∩ **DS** in this statement, the last part about X/R being in **D-pkf** is immediate by setting $A = 1$ in **Proposition 41**. However, it may not be possible to show q is in **D-pkf** (while showing that it is effective Kan is immediate by **Proposition 44**, showing that it is degenerate-preferring is not). Thus, to achieve more ideal conditions, we try to weaken the assumptions of **Proposition 41**, thereby establishing a more comprehensive result.

Then, we propose yet another version of effective Kan fibrations with additional structure as follows:

▶ **Definition 46.** *Assume (p, lift_p) is an effective Kan fibration with a degeneracy-section $\tilde{p}$. (p, lift_p) is said to be degeneracy-section-preferring, if $\mathsf{lift}_p(\tilde{p}_{n+1}(ys_j)\iota, ys_j) = \tilde{p}_{n+1}(ys_j)$ for a horn problem as in the following commutative diagram for all $n \in \mathbb{N}$, all possible j in s_j and m in Λ^{n+1}_m. We call this **DS-pkf**.*

$$\begin{tikzcd}
\Lambda^{n+1}_m \arrow[rr, "\tilde{p}_{n+1}(ys_j)\iota"] \arrow[d, hook, "\iota"'] & & X \arrow[d, "p"] \\
\Delta^{n+1} \arrow[r, "s_j"'] & \Delta^n \arrow[ur, "\tilde{p}_n(y)"] \arrow[r, "y"'] & Y \arrow[u, bend right, dotted, "\tilde{p}"']
\end{tikzcd}$$

Note that it is straightforward to check that this lift_p indeed satisfies the compatibility condition of effective Kan fibrations. It is also straightforward to see that any **D-pkf** ∩ **DS** is **DS-pkf**, although **D-pkf** $\not\subset$ **DS-pkf** and **DS-pkf** $\not\subset$ **D-pkf**.

We have checked that if we assume a similar statement to **Proposition 41** where we replace **D-pkf** with **DS-pkf** instead of holds, then we are able to show the constructive version of "Corollary 4.4." in [3], which would be formulated as follows:

[Quotient : DS-pkf] *"Suppose R is a pseudo-equivalence relation on X, the quotient map $q \colon X \to X/R$ has a degeneracy-section $\tilde{q}$, and $R \overset{\pi}{\hookrightarrow} X \times X$ is an effective Kan fibration. Moreover, suppose that X is in **DS-pkf**. Then the quotient map $X \xrightarrow{q} X/R$ is in **DS-pkf**, and X/R is in **DS-pkf**."*

We have also defined even more general structures which do not depend on degeneracy-sections as follows:

▶ **Definition 47.** *Assume (p, lift_p) is an effective Kan fibration with a section $\tilde{p_0}\colon Y_0 \to X_0$ at vertices. (p, lift_p) is called vertex-section-preferring, if $\mathsf{lift}_p(\tilde{p_0}(y), ys_0) = \tilde{p_0}(y)s_0$ for a horn problem of the following commutative diagram with $m = 0, 1$. We call this **VS-pkf** (and **VS-pkc** for the case of fibrant objects).*

$$\begin{tikzcd}
\Lambda^1_m \arrow[rr, "\tilde{p_0}(y)"] \arrow[d, hook, "\iota"'] & & X \arrow[d, "p"] \\
\Delta^1 \arrow[r, "s_0"'] & \Delta^0 \arrow[ur, "\tilde{p_0}(y)"] \arrow[r, "y"'] & Y
\end{tikzcd}$$

Note that although $\Lambda^1_m = \Delta^0$ for both $m = 0, 1$, the inclusion ι depends on m. Also, it is immediate that any **DS-pkf** is **VS-pkf**.

In this way, we can broaden our search for the candidate in the constructive version of "Corollary 4.4." in [3]. However, it is also required that those maps have to satisfy **Proposition 41** too.

To sum up, we have

$$\begin{array}{ccc}
\textbf{VS-pkf} & \supset & \textbf{VS-pkf} \cap \textbf{DS} \\
 & & \cup \\
 & & \textbf{DS-pkf} \\
 & & \cup \\
\textbf{D-pkf} & \supset & \textbf{D-pkf} \cap \textbf{DS}
\end{array}$$

Under the condition of **D-pkf**, we can prove **Proposition 41**. However, this degenerate-preferring structure is quite strong, making it hard to solve [Quotient : D-pkf ∩ DS]. On the other hand, **VS-pkf** might be too general for the condition in **Proposition 41**. The condition of **D-pkf** with a section on vertices in **Proposition 41** is vastly generalized, and that would make the proof more combinatorially complicated. However, it is still expected to be achievable: it has a section only at the vertices level, but a lift construction algorithm is expected to work from the vertices level to higher levels using the compatibility condition of the fibrations and the universal properties of horn construction. It would then improve [Quotient : DS-pkf] by replacing **DS-pkf** by **VS-pkf**.

7 Conclusion

In this paper, we have demonstrated that effective Kan fibrations indeed model W-types as expected. The transition from classical to constructive proofs was generally smooth, though certain challenges arose. In particular, constructively, choices can be subtle. In **Proposition 18**, we addressed the issue of arbitrariness in the choice of factorizations by imposing conditions such that the mediating maps are in **EffKanCplx** or **EffKan**$/A$. Fortunately, these conditions posed no issues for W-types, as their categorical interpretation ensures that all the relevant mediating maps in W-types are such maps.

One significant challenge is that as mentioned in **Remark 36**, the W-types employed here are non-constructive due to their reliance on linearly ordered transfinite ordinals, invoking the Law of Excluded Middle (LEM). Addressing this issue was beyond the scope of this paper, leading us to adopt a classical transfinite argument. Constructive alternatives to ordinals remain a promising avenue for future work. Various attempts, such as those in [9] and more recently in [4], focus on countable ordinals. However, there are difficulties representing other limit ordinals. Nevertheless, our strategy of interpreting W-types as certain filtered colimits suggests that it may apply to constructive ordinals, which, though not linearly ordered, are anticipated to form a filtered poset. In this context, **Proposition 18** and the result that polynomial functors can be set on the category of effective Kan complexes are expected to remain relevant.

Also, for future work, we discussed various additional structures on effective Kan fibrations, such as **D-pkf**, **DS-pkf**, and **VS-pkf**, to constructively model quotient types and establish key propositions in constructive settings. This direction is beneficial not only for applications of W-types but also for broadening the applicability of these methods within the related fields of constructive semantics for HoTT.

References

1 Steve Awodey and Michael A. Warren. *Homotopy theoretic models of identity types.* Math. Proc. Cambridge Philos. Soc., 146(1): pp. 45–55, 2009. doi:10.1017/S0305004108001783.

2 Benno van den Berg and Eric Faber. *Effective Kan Fibrations in Simplicial Sets.* Springer, 2022. doi:10.1007/978-3-031-18900-5.

3 Benno van den Berg and Ieke Moerdijk. *W-types in homotopy type theory.* Mathematical Structures in Computer Science, 25(5): pp. 1100–1115, 2015. doi:10.1017/S0960129514000516.

4 Thierry Coquand, Henri Lombardi, and Stefan Neuwirth. *Constructive theory of ordinals.* arXiv, 2022. arXiv preprint arXiv:2201.04352. doi:10.48550/arXiv.2201.04352.

5 Storm Diephuis. Effective kan fibrations for simplicial groupoids, semisimplicial sets and ex$^\infty$, 2023. https://eprints.illc.uva.nl/id/eprint/2247/1/MoL-2023-06.text.pdf.

6 Nicola Gambino and Christian Sattler. *The Frobenius Condition, Right Properness, and Uniform Fibrations.* Journal of Pure and Applied Algebra, 221(12): pp. 3027–3068, 2017. doi:10.1016/j.jpaa.2017.02.013.

7 Freek Geerligs. Symmetric effective and degenerate-preferring kan complexes, 2023. https://studenttheses.uu.nl/handle/20.500.12932/43910.

8 Paul Goerss and John F. Jardine. *Simplicial homotopy theory.* Birkhäuser, 1999. doi:10.1007/978-3-0348-8707-6.

9 Per Martin-Löf. *Notes on constructive mathematics.* Almqvist & Wiksell, Stockholm, 1970. URL: https://worldcat.org/oclc/472257944.

10 Per Martin-Löf. *Constructive mathematics and computer programming.* In *Logic, Methodology and Philosophy of Science VI*, volume VI, pages pp. 153–175. Elsevier, 1982. doi:10.1016/S0049-237X(09)70189-2.

11 Egbert Rijke. *Introduction to homotopy type theory.* Cambridge University Press, 2022. doi:10.48550/arXiv.2212.11082.

12 Univalent Foundations Project. Homotopy type theory -– univalent foundations of mathematics, 2013. Accessed: 2025-05-02. URL: https://homotopytypetheory.org/book/.

Complexity of Cubical Cofibration Logics I: coNP-Complete Examples

Robert Rose

Dept. of Mathematics & Computer Science, Wesleyan University, Middletown, CT, USA

Daniel R. Licata

Dept. of Mathematics & Computer Science, Wesleyan University, Middletown, CT, USA

Abstract

We provide a comprehensive classification of the cofibration entailment problem, COFENT, for the cofibration logics of various cubical type theories in use today. The problem COFENT arose from the need of cubical proof assistants to automate reasoning about cubical complexes included in an n-dimensional hypercube. Intuitively, it asks: given logical descriptions of two such complexes, is one a subcomplex of the other? We show that the common variants of COFENT are coNP-complete.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases cubical sets, internal language, intuitionistic logic, dependent type theory, homotopy type theory, decision procedures

Digital Object Identifier 10.4230/LIPIcs.TYPES.2024.9

Funding This material is based upon work supported by the Air Force Office of Scientific Research under Grant No.FA9550-21-0009 (Tristan Nguyen, program manager). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the Air Force Office of Scientific Research.

Robert Rose: US Air Force Research Lab, FA-95502110009 and FA-95501510053

Daniel R. Licata: US Air Force Research Lab, FA-95502110009 and FA-95501510053

1 Introduction

A well-known tool in topos theory for reasoning about a given topos is its Mitchell-Bénabou language [5] (or simply, its "language"), a formal abstraction of the topos as a typed (i.e., multi-sorted) first-order intuitionistic language with a fixed interpretation in that topos. Because the syntax of the Mitchell-Bénabou language is defined directly from the topos (e.g., the types of the logic are just the objects of the topos), this interpretation is nearly trivial: terms denote morphisms and are represented by syntax for certain categorical operations which produce morphisms, such as pairing or post-composition with a given morphism; formulas are, idiosyncratically, terms whose type is the subobject classifier of the topos (or a subobject thereof).

Proving or disproving that a topos has a given property expressed in the Mitchell-Bénabou language is greatly facilitated by the Kripke-Joyal semantics (or "sheaf semantics", for a topos of sheaves), which provides a rigorous translation of arguments in this language to ordinary classical mathematics. The semantics has the familiar form of a Tarskian truth definition. (However, because the interpretation is fixed, the forcing relation is defined a priori in terms of this definition, and the semantics is presented as a theorem.)

This semantics affords a convenient point of departure for investigating the computability and complexity theoretic properties of formal first-order reasoning in a topos. Specifically, this area gives rise to generalizations of the *model-checking problem.* It is a rich area which has received relatively little attention.

30th International Conference on Types for Proofs and Programs (TYPES 2024).
Editors: Rasmus Ejlers Møgelberg and Benno van den Berg; Article No. 9; pp. 9:1–9:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Language fragments for cubical homotopy theory

One application of Mitchell-Bénabou language is to the homotopy theory of cubical sets, where by cubical set we mean a contravariant functor on a particular indexing category $\mathcal{C}$ (a "cube category"). The topos is the category of cubical sets which are sheaves for a given Grothendieck topology on $\mathcal{C}$. In this paper, this topology is trivial: the toposes are presheaf toposes.

The language of a topos of cubical sets is expressive enough to define important homotopical structure: notably, to specify which morphisms of cubical sets qualify as maps of spaces (i.e., "fibrations") and which maps of spaces further qualify as equivalences (i.e., "trivial fibrations"). This is usually achieved first by selecting a collection of subfunctors of representable functors (e.g., in the case of simplicial homotopy theory, a standard choice is the "horn inclusions") and hence by taking a fibration to be a morphism f for which any map from a select subfunctor inclusion to f affords an extension to the base representable. Trivial fibrations have this extension property with respect to a larger selection of such subfunctors. These collections of subfunctors form the bases for generating classes of morphisms called, respectively, "trivial cofibrations" and "cofibrations". Extension of maps along trivial cofibrations is the fundamental operation provided by homotopical structure.

While these extension properties are readily expressible in the language of the topos, a more restricted usage of the language may suffice in certain frameworks (e.g., homotopy type theory). For example, the language of the topos may be used just for the purposes of specifying cofibrations into representables and expressing their inclusion relation. Since cofibrations are classified by Ω, the inclusion relation is naturally expressed by entailment in the language. The sacrifice in expressivity brings significant computational benefits. In Section 3, we will define a number of such language fragments.

1.2 Applications for cubical type theory

Our interest in decision problems arising from the language of a topos indeed began with a practical need. A key component of a proof assistant we were designing would automatically decide formula entailment in a fragment of the language of a topos: namely, presheaves on the finite-product theory of distributive lattices, an important definitional variant of the category of cubical sets. Analogous decision procedures are core components in existing implementations of cubical type theories. For example, entailment in a fragment of the language of the topos of presheaves on the finite-product theory of De Morgan algebras is decided in the proof assistant Agda's cubical mode, which is an implementation of the cubical type theory presented in [3]. Entailment in a fragment of the language of the topos of presheaves on the finite-product theory of bipointed sets is decided in the proof assistant `cooltt`, which is an implementation of the cubical type theory presented in [1]. Our original motivation began with investigating the feasibility of implementing the directed cubical type theory presented in [7], which is an adaptation of the directed simplicial type theory presented in [6].

2 Motivating examples

For the sake of intuition, we present some instances and non-instances of the decision problems of interest before launching into the technical development.

2.1 Cube boundary inclusion

Consider a 3-dimensional cube, with 8 corner points $(0,0,0),\ldots,(1,1,1)$. As a cubical set, it is modeled by a representable functor $\mathrm{Hom}(-,\mathrm{X}^3)$. The object X is a formal element which corresponds to a single dimension. The identity morphism on X corresponds to a variable x_1 along that dimension. Hence, the object X^3 corresponds to three dimensions; and x_1, x_2 and x_3 each correspond to a dimension in the given cube. Let us picture x_1 as varying from left to right; x_2 as varying bottom to top; and x_3 as varying from front to back.

By means of equations involving these variables and end points 0 and 1, we can describe certain parts of this cube. For example, the equation $(x_1 = 0)$ describes just the left face, and $(x_1 = 1)$ describes just the right face. We can combine these using $\vee$ to get exactly the left and the right faces: $(x_1 = 0) \vee (x_1 = 1)$. The entire 2-dimensional boundary of the 3-cube is described by the formula

$$(x_1 = 0) \vee (x_1 = 1) \vee (x_2 = 0) \vee (x_2 = 1) \vee (x_3 = 0) \vee (x_3 = 1) \tag{1}$$

We can also describe the 1-dimensional edges of the 3-cube by combining formulas describing faces using $\wedge$. For example, the top-front edge is described by the formula $(x_2 = 1) \wedge (x_3 = 0)$. The 1-dimensional boundary of the 3-cube is described by the formula

$$\begin{array}{lllll} & (x_1 = 0) \wedge (x_2 = 0) & \vee & (x_1 = 0) \wedge (x_2 = 1) & \vee & (x_1 = 0) \wedge (x_3 = 0) \\ \vee & (x_1 = 0) \wedge (x_3 = 1) & \vee & (x_1 = 1) \wedge (x_2 = 0) & \vee & (x_1 = 1) \wedge (x_2 = 1) \\ \vee & (x_1 = 1) \wedge (x_3 = 0) & \vee & (x_1 = 1) \wedge (x_3 = 1) & \vee & (x_2 = 0) \wedge (x_3 = 0) \\ \vee & (x_2 = 0) \wedge (x_3 = 1) & \vee & (x_2 = 1) \wedge (x_3 = 0) & \vee & (x_2 = 1) \wedge (x_3 = 1) \end{array} \tag{2}$$

(Recall that $\wedge$ has higher precedence than $\vee$.)

Let ϕ_1 be the formula (1), and let ϕ_2 be the formula (2). Here are two queries for a decision problem: does ϕ_1 entail ϕ_2? or does ϕ_2 entail ϕ_1? Geometric intuition tells us no and yes. The internal language of the topos of cubical sets will also tell us this.

2.2 Simplex and pyramid

The last pair of problems are common to all of the cubical type theories mentioned in Section 1.2. We now present an example which, among the three, can be expressed only in the language fragment used in [7].

We just saw how to combine equations using $\wedge$ and $\vee$ to describe parts of a cube. It is also possible to combine variables directly to describe the *minimum* of two variables or the *maximum* of two variables. The notation is the same: $x_1 \wedge x_2$ denotes the min of x_1 and x_2, while $x_1 \vee x_2$ denotes their max.

The 2-simplex below the diagonal from $(0,0)$ to $(1,1)$ of the solid square (2-cube) is described by $(x_1 = x_1 \vee x_2)$ (or alternatively by $(x_2 = x_1 \wedge x_2)$). The same formula can be used to describe part of the 3-cube: it is the wedge containing the right face and the bottom face. Similarly, $(x_2 = x_2 \vee x_3)$ describes the wedge formed between the 2-simplices above the diagonal on the left and right faces. Combining the two wedges with $\wedge$ yields a 3-simplex:

$$(x_1 = x_1 \vee x_2) \wedge (x_2 = x_2 \vee x_3)$$

The pyramid whose apex is the corner point $(0,0,0)$ and whose base is the right face of the 3-cube is described by $(x_1 = x_1 \vee x_2 \vee x_3)$. Thus, we have two more queries for a decision problem. Is this formula valid?

$$(x_1 = x_1 \vee x_2) \wedge (x_2 = x_2 \vee x_3) \;\Rightarrow\; (x_1 = x_1 \vee x_2 \vee x_3) \tag{3}$$

Or this one?

$$(x_1 = x_1 \vee x_2 \vee x_3) \;\Rightarrow\; (x_1 = x_1 \vee x_2) \wedge (x_2 = x_2 \vee x_3) \tag{4}$$

2.3 Example reduction

In what follows, we will show that the instances (2) ⇒ (1), (1) ⇒ (2), (3), and (4) may be decided by an efficient translation into second-order logic, where the problem of checking the truth of the corresponding formula in a suitable model is in the complexity class coNP. For example, the translation of (3) to second-order logic is

$$\begin{aligned}\forall R \big(&\chi_3 \to \forall x\,(R(x) \to x_1 = x_1 \vee x_2) \;\wedge\; \forall x\,(R(x) \to x_2 = x_2 \vee x_3) \\ &\quad \to \forall x\,(R(x) \to x_1 = x_1 \vee x_2 \vee x_3)\big)\end{aligned}$$

where χ_3 is a formula encoding the constraint that ternary relations assigned to R have minimal and maximal elements. The entailment (3) holds if and only if the formula above is true in the model $\mathcal{A} = (\{0,1\}, 0^{\mathcal{A}} = 0, 1^{\mathcal{A}} = 1, \wedge^{\mathcal{A}} = \min, \vee^{\mathcal{A}} = \max)$.

The translation alone provides the core of a practical decision procedure. What remains is boilerplate and optimizations. The model-checking instance above may in turn be efficiently encoded in propositional logic and checked with a state-of-the-art SAT-solver. Alternatively, the model-checking instance may be handed almost as-is (after some lightweight wrangling) to an SMT-solver like `z3` or to an answer-set programming system like `clingo`.

3 Decision problems

Because terms in the language of a topos are built directly from its morphisms, this language is not in general recursively enumerable, and is often a proper class. Thus, it may not be that the entire language of a topos can be encoded as a countable set of strings over a finite alphabet. In computability and complexity theory, however, decision problems are ordinarily framed as such. For example, in a sheaf topos, any set at all induces a sheaf, and hence a type in the language. But for the purposes of defining a conventional decision problem, we can reasonably expect to encode no more than a countable number of types.

Furthermore, because terms in the language of a topos are formal combinations of morphisms, morphism equality is reducible to term equality: in particular, if morphism equality is undecidable, so is term equality. In contrast, equality of instances of a conventional decision problem is efficiently reducible to equality of strings over a finite alphabet.

Hence, one straightforward strategy is simply to limit one's focus to a suitable fragment of the language of the topos, and to define our decision problems relative to this fragment. To this end, we will next consider fragments which are "finitely presented" by formula languages.

3.1 Formula languages

Given a set A, we will denote a generic element of A^m by $a \in A^m$ where $a = (a_1, \ldots, a_m)$. We will extend this notational convention to variables, parameters, and terms: x will denote an m-tuple of variables $x = (x_1, \ldots, x_m)$; and likewise for $\alpha = (\alpha_1, \ldots, \alpha_m)$ and for $s = (s_1, \ldots, s_m)$. Annotations or constructors may be "broadcast" over an m-tuple: for example, $\forall x\,\psi$ abbreviates $\forall x_1 \cdots \forall x_m\,\psi$. The concatenation of a k-tuple a and an m-tuple b will be denoted by $a\,;b$. We will freely omit parentheses when they may be inferred.

3.1.1 Types and terms

We will define a term language T prior to the formula language in the conventional way. Fix a finite set F of morphisms in $\mathcal{E}$. The term types of the fragment are generated by the domains and codomains appearing in F. The terms of the formula language will be the fragment of the language of the topos generated by the identity morphisms on term types under the term-forming operations corresponding to post-composition by morphisms $f \in F$, post-composition by projections π_i, and pairing [5, Section VI.5]. Thus, terms can be viewed as composable chains of tuples of morphisms, generated from a finite set of morphisms and projections. For any given domain type $S_1 \times \cdots \times S_m$, we fix a variable set $\{x_1, \ldots, x_m\}$ where each x_i is interpreted by the corresponding projection. We will refer to such a term language as *finitely presented*. Equality of terms ("syntactic equality") is clearly efficient to decide.

3.1.2 Parameterized first-order formulas

We consider formulas with parameters, or metavariables. From the point of view of the language of the topos, parameters are just variables defined on a subobject of Ω. In addition to term types, the definition of the fragment includes a finite collection of subobjects $\Phi_i \subseteq \Omega$. For convenience, we assume that formulas are given with domain in the form $S_1 \times \cdots \times S_m \times \Phi_1 \times \cdots \times \Phi_k$. We fix a variable set $\{\alpha_1, \alpha_2, \ldots\}$ where each α_i is interpreted by the corresponding projection from Φ_i.

▶ **Definition 1.** *Let T be a finitely presented term language. The corresponding language of formulas $L_{T,S\times\Phi}$ is inductively defined by:*

$$\begin{array}{rcll}
\bot & \in & L_{T,S\times\Phi} & \\
\top & \in & L_{T,S\times\Phi} & \\
\alpha_i & \in & L_{T,S\times\Phi} & \textit{where } 1 \leq i \leq k \\
(f = g) & \in & L_{T,S\times\Phi} & \textit{where } f, g : S \to R \in T \textit{ for some } R \in \mathcal{E} \\
\phi_1 \wedge \phi_2 & \in & L_{T,S\times\Phi} & \textit{for } \phi_1, \phi_2 \in L_{T,S\times\Phi} \\
\phi_2 \vee \phi_2 & \in & L_{T,S\times\Phi} & \textit{for } \phi_1, \phi_2 \in L_{T,S\times\Phi} \\
\phi_2 \Rightarrow \phi_2 & \in & L_{T,S\times\Phi} & \textit{for } \phi_1, \phi_2 \in L_{T,S\times\Phi} \\
\exists x_{m+1}\,\psi & \in & L_{T,S\times\Phi} & \textit{for } \psi \in L_{T,S\times S_{m+1}\times\Phi} \\
\forall x_{m+1}\,\psi & \in & L_{T,S\times\Phi} & \textit{for } \psi \in L_{T,S\times S_{m+1}\times\Phi}
\end{array}$$

We define the language $L_T = \bigcup_{S\times\Phi} L_{T,S\times\Phi}$. For a given $\phi \in L_T$ denotes the set of subformulas of ϕ.

We will usually abbreviate $L_{T,S\times\Phi}$ to L_S when T and Φ are inferable from context or are arbitrary. When the term type language is generated by a single type, we will index L by the length of S instead.

3.2 Semantics

Our focus in this paper is on the problem of deciding when a formula in the language of the topos holds when interpreted "locally". One way to make this concept precise is in terms of a forcing relation, which is the basis the sheaf semantics.

3.2.1 Forcing relation

For an object C in a category $\mathcal{C}$, $\mathrm{t}(C)$ will denote the sieve $\bigcup_{D \in \mathcal{C}} \mathrm{Hom}(D, C)$. We first recall the definition of forcing in toposes of sheaves from [5, Section VI.7]:

▶ **Definition 2.** *Let $\mathcal{E}$ be a topos of sheaves with indexing category $\mathcal{C}$, and let $S \in \mathcal{E}$. Let ϕ be a formula defined on S in the language of $\mathcal{E}$. Let $X \in \mathcal{C}$, and let $s \in SX$. The* forcing relation *is defined by*

$$X \Vdash \phi\,(s) \quad \textit{iff} \quad s \in \{x \mid \phi(x)\}X$$

where

$$\begin{tikzcd}
\{x \mid \phi(x)\} \arrow[r] \arrow[d, phantom, "\subseteq" rotate=90] & 1 \arrow[d, "\mathrm{t}"] \\
S \arrow[r, "\phi"] & \Omega
\end{tikzcd}$$

is a pullback square.

We will refer to the element $s \in SX$ in Definition 2 as a *local assignment.* When we say "X forces ϕ with local assignment s" we mean that $X \Vdash \phi\,(s)$.

The forcing relation models a form of "truth definition", which is attributed to Kripke and Joyal, and which we also call the sheaf semantics. Furthermore, the truth definition may be usefully refined in light of specific features of the topos of sheaves in question.

3.2.2 Presheaf semantics

Next we present the *presheaf semantics* for a language fragment L (Definition 1). The following theorem is adapted from [5, Theorem VI.7.1]. It will be our main tool for proving the correctness of mapping reductions. We recall that for an element $s \in SX$ and a morphism $f : Y \to X$, the *restriction of s along f*, denoted by $s \cdot f$, is defined as $S(f)(s)$.

▶ **Theorem 3.** *Let $\mathcal{C}$ be a category, and let $\mathcal{E}$ be the topos of presheaves on $\mathcal{C}$. Let $S = S_1 \times \cdots \times S_m, S_{m+1} \in \mathcal{E}$. Let $\Phi = \Phi_1 \times \cdots \times \Phi_k \in \mathcal{E}$ with each $\Phi_i \subseteq \Omega$. Let $\phi_1, \phi_2 : S \times \Phi \to \Omega$ and $\psi : S \times S_{m+1} \times \Phi \to \Omega$ be formulas. Let $X \in \mathcal{C}$, let $s \in SX$, and let $h \in \Phi X$.*

$$\begin{array}{ll}
X \Vdash \top\,(s\,;h) & \\
X \nVdash \bot\,(s\,;h) & \\
X \Vdash (f = g)\,(s\,;h) & \textit{iff } f_X(s) = g_X(s) \\
X \Vdash \alpha_i\,(s\,;h) & \textit{iff } h_i = \mathrm{t}(X) \\
X \Vdash \phi_1 \wedge \phi_2\,(s\,;h) & \textit{iff } X \Vdash \phi_1\,(s\,;h) \textit{ and } X \Vdash \phi_2\,(s\,;h) \\
X \Vdash \phi_1 \vee \phi_2\,(s\,;h) & \textit{iff } X \Vdash \phi_1\,(s\,;h) \textit{ or } X \Vdash \phi_2\,(s\,;h) \\
X \Vdash \phi_1 \Rightarrow \phi_2\,(s\,;h) & \textit{iff for all } Y \in \mathcal{C} \textit{ and } t \in \mathrm{Hom}(Y, X), \\
& \quad Y \Vdash \phi_1\,(s \cdot t\,;h \cdot t) \textit{ implies } Y \Vdash \phi_2\,(s \cdot t\,;h \cdot t) \\
X \Vdash \exists x_{m+1}\,\psi\,(s\,;h) & \textit{iff there exists } s_{m+1} \in S_{m+1}X \textit{ such that} \\
& \quad X \Vdash \psi\,(s\,;s_{m+1}\,;h) \\
X \Vdash \forall x_{m+1}\,\psi\,(s\,;h) & \textit{iff for all } Y \in \mathcal{C}, t \in \mathrm{Hom}(Y, X), \textit{and } s_{m+1} \in S_{m+1}Y, \\
& \quad Y \Vdash \psi\,(s \cdot t\,;s_{m+1}\,;h \cdot t)
\end{array}$$

When $S_{m+1} = \mathrm{Hom}_{\mathcal{C}}(-, U)$ for some $U \in \mathcal{C}$ and $\mathcal{C}$ is Cartesian, the semantics of the universal quantifier simplifies to

$$X \Vdash \forall x_{m+1}\, \psi\, (s\,;h) \qquad \textit{iff } X \times U \Vdash \psi\, (s \cdot \pi_1\,; \pi_2\,; h \cdot \pi_1)$$

▶ Remark 4. We may extend Theorem 3 to formulas in the language of the topos of the form $\forall \alpha_1 \cdots \forall \alpha_k\, \phi$ where $\phi \in L$ and k is the maximal parameter index in ϕ:

$$\begin{aligned} X \Vdash \forall \alpha_1 \cdots \forall \alpha_k\, \phi\, (s) \quad & \text{iff for all } Y \in \mathcal{C}, t \in \mathrm{Hom}(Y, X), \text{ and } h \in \Phi_1 \times \cdots \times \Phi_k Y, \\ & \quad Y \Vdash \phi\, (s \cdot t\,; h) \end{aligned}$$

▶ **Corollary 5.** *In the context of Theorem 3, the semantics of the universal quantifier when $S_{m+1} = \mathrm{Hom}_{\mathcal{C}}(-, U)$ for some $U \in \mathcal{C}$ and $\mathcal{C}$ is Cartesian validates the following equivalence:*

$$X \Vdash \forall x_{m+1}\, (\psi_1 \vee \psi_2) \Leftrightarrow \forall x_{m+1}\, \psi_1 \vee \forall x_{m+1}\, \psi_2\, (x)$$

3.3 Languages of representable types

One natural way to obtain a finitely presented formula language is to restrict attention to sheaves of the form $S = \mathrm{Hom}(-, U)$ for a selection of $U \in \mathcal{C}$. Elements of S would just be morphisms in $\mathcal{C}$ from X to U. Thus, we may induce a formula language from a collection of morphisms from $\mathcal{C}$.

Coincidentally, elements of $SX = \mathrm{Hom}(X, U)$ then have a compositional structure analogous to that induced by a formal term system. Below, this coincidence will be taken further: the indexing category $\mathcal{C}$ will be presented as a quotiented term language, and the term language defining $\mathcal{C}$ will take the place of the term language used to define the formula language. Conveniently, restrictions will then be computed by term substitution.

3.4 Indexing category from a finite algebra

The indexing categories for cubical sets – "cube categories" – are readily and usefully definable as quotients of formal term languages.

Beyond encodability, another requirement for bringing a decision problem under the purview of complexity theory is that it be decidable. But an indexing category $\mathcal{C}$ may fail to have decidable equality of morphisms; and so even validity of the atomic formulas of the language may fail to be decidable. The following method of construction however ensures that the morphisms of $\mathcal{C}$ at least have decidable equality.[1]

3.4.1 Generation by a finite algebra

Fix a finite algebraic signature σ. Let $\mathcal{T}_{\sigma,n}$ be the term algebra of type σ over the variable set $\{x_1, \ldots, x_n\}$. We will abbreviate $\mathcal{T}_{\sigma,n}$ to $\mathcal{T}_n$. For $m \geq 0$, the m-fold (left-associated) product algebra $\mathcal{T}_n \times \cdots \times \mathcal{T}_n$ will be denoted by $\mathcal{T}_n^m$. The term algebra $\mathcal{T}$ is defined by $\mathcal{T} = \bigcup_n \mathcal{T}_n$.

For $s \in \mathcal{T}_m$ and $t \in \mathcal{T}_n^m$, the simultaneous substitution of the terms t_i for variables x_i in s will be denoted by $s[t_1/x_1, \ldots, t_m/x_m]$, by $s[t/x]$, or simply by st. Let $\mathcal{A}$ be a finite σ-algebra. The equational theory of $\mathcal{A}$ contains the universally quantified equations of terms satisfied by $\mathcal{A}$. The variety of this theory, denoted by $\mathrm{V}(\mathcal{A})$, contains the σ-algebras which satisfy this theory (i.e., its models). Examples of varieties generated by a finite algebra $\mathcal{A}$

[1] We are grateful to Alex Kruckman for directing our attention to the central role played by finite algebras in this area. See also [2].

include Boolean algebras, De Morgan algebras, (bounded) distributive lattices, (bounded) meet (join) semi-lattices, bi-pointed sets, pointed sets, and sets. All save Boolean algebras and sets have been used in the construction of indexing categories of cubical sets. Bounded distributive lattices have also been used to construct the indexing category in a variant definition of simplicial sets.

We may use the term algebra $\mathcal{T}_n$ to give a formal definition of a free algebra in $\mathrm{V}(\mathcal{A})$ generated by variables $x_1, \ldots, x_n$. We will denote these algebras by $\mathcal{F}_{\mathcal{A},n}$, or more briefly by $\mathcal{F}_n$. Given terms $r, s \in \mathcal{T}_n$, we say that r and s denote the same elements in $\mathcal{F}_n$ if and only if $r^{\mathcal{A}} = s^{\mathcal{A}}$ (i.e., their interpretation as term functions on $\mathcal{A}^n$ are equal). Recall that this relation respects term substitution: for $s \in \mathcal{T}_m$ and $t \in \mathcal{T}_n^m$, $s[t/x]^{\mathcal{A}} = s^{\mathcal{A}} \circ t^{\mathcal{A}}$.

3.4.2 Finite product theories

From here, we may use the free algebras $\mathcal{F}_n$ to give a formal definition of an indexing category $\mathcal{C}$. We will denote these categories by $\mathcal{C}_{\mathcal{A}}$, or more briefly by $\mathcal{C}$. The object set of $\mathcal{C}_{\mathcal{A}}$ contains a formal object X, and its remaining objects are formal n-fold (left-associated) products of X for $n \geq 0$. The set of morphisms from X^n to X^m is simply the (left-associated) product algebra $\mathcal{F}_n^m$. Thus, morphisms are denoted by m-tuples of terms in n-variables, and two morphisms denoted by tuples of terms r and s are equal if and only if their interpretations as term functions on $\mathcal{A}^n$ are equal. For $n \geq 0$, the identity morphism on X^n is $(x_1, \ldots, x_n)$. Composition of morphisms is given by composition of denoted term functions; which factors through the substitution of the representing tuples of terms.

The category $\mathcal{C}_{\mathcal{A}}$ is an example of a finite-product theory (or Lawvere theory). A well-known alternative view of this construction is as the category opposite to a chosen skeleton of the category of free algebras on a finite number of generators and homomorphisms. Note however that the language of a topos is interpreted by contravariant functors on $\mathcal{C}_{\mathcal{A}}$. This usage is different than the usage for which the concept of a finite-product theory was developed by Lawvere: the categorification of universal algebra. Models of this finite-product theory would be product-preserving functors on $\mathcal{C}_{\mathcal{A}}$, not arbitrary functors on $\mathcal{C}_{\mathcal{A}}^{op}$ satisfying sheaf conditions.

3.5 Cofibration formulas

From now on, we will assume that the formula language L is defined over a term algebra $\mathcal{T}$ corresponding to a finite algebra $\mathcal{A}$ which induces the indexing category $\mathcal{C}$, and that the term translation is the natural one. The letter c will denote a generic constant term in the signature of $\mathcal{A}$.

Standard cofibration formulas (i.e., those used in cubical type theories) constitute only a small fragment of L: specifically, the entailment connective $\Rightarrow$ and existential quantifier $\exists$ do not appear in cofibration formulas. Another restriction which appears concerns allowable atomic equations: in some cubical type theories, an equation may compare a given non-constant term f only to a constant.

▶ **Definition 6.** *We define several fragments of L by inductive generation:*

- L_{cof} *is generated by* $\bot$, $\top$, $(f = g)$, $\wedge$, $\vee$, *and* $\forall$
- L_{coftc} *is generated by* $\bot$, $\top$, $(f = c)$, $\wedge$, $\vee$, *and* $\forall$
- L_{cofvc} *is generated by* $\bot$, $\top$, $(x_i = c)$, $\wedge$, $\vee$, *and* $\forall$

Each fragment above also has a variant – L_{pcof}, L_{pcoftc}, L_{pcofvc} – whose generators also contain α_i.

A formula in L_{cof} is called a *cofibration formula.* The variants L_{coftc} and L_{cofvc} are called *term-constant* and *variable-constant.* The variants containing α_i are called *parameterized.* Although the focus of our study is on these fragments, it is natural to investigate related decision problems for the full language. In several cases, the methods we use below do apply more generally. In the future, the provisional definition above of what is a cofibration formula may merit revisiting.

3.6 Entailment problems

We are at last in position to define the decision problems we aim to classify in this paper.

▶ **Definition 7.** *Let $\mathcal{A}$ be a finite algebra with signature σ. Let $\mathcal{T}$ be the term algebra of type σ over variable set $\{x_1, x_2, \dots\}$, let $\mathcal{C}$ be the finite-product theory induced by $\mathcal{A}$, and let* X *be the generating object of $\mathcal{C}$. The* cofibration entailment problem for $\mathcal{A}$ *is the subset of $L_{cof} \times L_{cof}$ defined by*

$$\text{COFENT} = \big\{(\phi_1, \phi_2) \mid \exists m \ \mathrm{X}^m \Vdash \phi_1 \Rightarrow \phi_2\,(x)\big\}$$

The parameterized cofibration entailment problem for $\mathcal{A}$ and Φ *is the subset of $L_{pcof} \times L_{pcof}$ defined by*

$$\text{PCOFENT} = \big\{(\phi_1, \phi_2) \mid \exists m \ \mathrm{X}^m \Vdash \forall\alpha\,\big(\phi_1 \Rightarrow \phi_2\big)\,(x)\big\}$$

Lastly, for algebras $\mathcal{A}$ with at least one constant, we denote the term-constant *and* variable-constant *variants of each problem with the suffixes TC and VC.*

3.7 Problem characterization in terms of derivability

Given the proof-theoretic character of many presentations of cubical type theories, it is reasonable to wonder if there is an alternative, logical characterization of COFENT. Is there a suitable notion of derivability such that COFENT $= \{(\phi_1, \phi_2) \mid \phi_1 \vdash \phi_2\}$?

For this discussion, let us consider the cube category corresponding to the theory of bounded distributive lattices. We know that the language of a topos can be viewed as an intuitionistic first-order logic. A natural first pass is intuitionistic derivability from this theory. This notion of derivability is sound with respect to the Kripke-Joyal semantics. Unsurprisingly, it is not complete. For example,

$$(x_1 \wedge x_2 = 0) \Rightarrow (x_1 = 0) \vee (x_2 = 0)$$

is valid in the language of the topos; but a corresponding derivation does not exist (not even classically: consider the smallest congruence on the free distributive lattice on two generators which identifies the meet of the generators and 0). Another example is Corollary 5, which is certainly not derivable. The notion of derivability above, it seems, would need to be significantly augmented with axioms to accommodate the structural peculiarities of categories of cubical sets as well as semantic properties specific to the choice of generating algebra and of language fragment.

However, even given a complete axiomatization, we do not expect a notion of derivability to yield an efficient-enough decision procedure to prove our main results. In practice, where this approach has been made to work, it remains the case that the smallest-known derivability proofs may be exponential in the length of the given instance. So instead, we ask, what form might a “short witness” of $\phi_1 \nvdash \phi_2$ take? The answer in logic traditionally has been a countermodel.

4 Results

To classify COFENT and related problems for various cube categories, we will first show uniformly that they are all coNP-hard by a reduction from the canonical coNP-hard problem UNSAT. Next we will show more-or-less on a case-by-case basis that they belong to coNP via efficient translations to the fragment of second-order logic ∀SO. Here we rely on Fagin's theorem, the foundational result in descriptive complexity theory (cf. [4, Section 9.2]). Note that the Cook-Levin theorem is an easy corollary of Fagin's theorem, and its typical proof contains an efficient translation to UNSAT (ibid.); so these translations may also be viewed as indirect reductions to UNSAT.

4.1 Reduction from UNSAT

▶ **Proposition 8.** *If σ_0 has constants 0 and 1 denoting distinct elements of $\mathcal{A}$, then the decision problem* COFENTVC *is* coNP*-hard.*

Proof. Let F be a formula of propositional logic in disjunctive normal form. Assume wlog that $\mathrm{Var}(F) \subseteq \{x_1, \ldots, x_m\}$. Let $F' \in L_{cof,m}$ be the formula induced by mapping positive literals x_i to equations $(x_i = 1)$ and negative literals to equations $(x_i = 0)$. The edge cases: empty clauses are mapped to $\top$ while empty formulas are mapped to $\bot$. We claim that

$$F \in \mathrm{UNSAT} \quad \text{iff} \quad \mathrm{X}^m \Vdash F' \Rightarrow \bot\,(x)$$

First, we prove the case where F is a single clause C. Suppose that C is unsatisfiable. Then C contains literals x_i and $\neg x_i$ for some i. So $(x_i = 1), (x_i = 0) \in \mathrm{SF}(F')$. Let $s \in \mathcal{T}_n^m$. Note that $\mathrm{X}^n \Vdash C\,(s)$ implies both $s_i = 0$ and $s_i = 1$, which is a contradiction. So $\mathrm{X}^n \nVdash C\,(s)$, as desired. Conversely, suppose that C is satisfiable. Let $b \in \{0,1\}^m$ be a satisfying assignment (whereby $x_i \mapsto b_i$). Note that for $x_i \in \mathrm{Var}(C)$, $b_i = 0$ if and only if C contains $\neg x_i$. Hence, for $x_i \in \mathrm{Var}(C)$, $b_i = 0$ if and only if $(x_i = 0) \in \mathrm{SF}(C')$ and $b_i = 1$ if and only if $(x_i = 1) \in \mathrm{SF}(C')$. Therefore, $\mathrm{X}^0 \Vdash C'\,(b)$.

We finish with an induction on F. If F is the empty formula, then F is unsatisfiable while $\mathrm{X}^m \Vdash \bot \Rightarrow \bot\,(x)$. Now suppose $F = F_1 \vee F_2$. On the one hand, observe that $F_1 \vee F_2$ is unsatisfiable if and only if F_1 and F_2 are unsatisfiable. On the other hand, $\mathrm{X}^m \Vdash F_1' \vee F_2' \Rightarrow \bot\,(x)$ if and only if $\mathrm{X}^m \Vdash F_1' \Rightarrow \bot\,(x)$ and $\mathrm{X}^m \Vdash F_2' \Rightarrow \bot\,(x)$. The conclusion follows by induction. ◀

▶ **Corollary 9.** *If σ_0 has constants 0 and 1 denoting distinct elements of $\mathcal{A}$, then any of the decision problems defined in Definition 7 is* coNP*-hard.*

4.2 Reduction to ∀SO

4.2.1 Translation of L_{cof} to second-order logic

Consider the equation $(f = g) \in L$. We would like to transform this equation into the sentence $\forall x \left(R(x) \to f(x) = g(x)\right)$. Informally, this sentence holds if the equation holds for all m-tuples satisfying R. Given $\phi \in L_{cof}$, the purpose of the following transformation is to replace equations $(f = g) \in \mathrm{SF}(\phi)$ with such formulas.

Below, $\vDash$ denotes the conventional formula satisfaction relation from mathematical logic for second-order logic ([4, Section 7.1]). The term function interpreting a tuple of terms $s \in \mathcal{T}_n^m$ will be denoted by $s^{\mathcal{A}} : \mathcal{A}^n \to \mathcal{A}^m$.

▶ **Definition 10.** *Let $m, n \geq 0$, and let $\phi \in L_{cof,m}$. We define the second-order formula ϕ'_n with a free second-order variable R of arity n as follows:*

$$\phi'_n = \begin{cases} \bot & \phi = \bot \\ \top & \phi = \top \\ \forall x \left(R(x) \to f(x) = g(x)\right) & \phi = (f = g) \\ \phi'_{1,n} \wedge \phi'_{2,n} & \phi = \phi_1 \wedge \phi_2 \\ \phi'_{1,n} \vee \phi'_{2,n} & \phi = \phi_1 \vee \phi_2 \\ \forall x_{m+1}\, \psi'_n & \phi = \forall x_{m+1}\, \psi \end{cases}$$

When $n = m$, we will abbreviate ϕ'_n to ϕ'. Note that ϕ' has no free first-order variables.

▶ Remark 11. The transformation in Definition 10 admits a reduction of the universal quantifier:

$$\mathcal{A} \vDash \forall x_{m+1}\, \psi'_m\, (M) \quad \text{iff} \quad \mathcal{A} \vDash \psi'_{m+1}\, (M \times \mathcal{A})$$

4.2.2 First schema

As it turns out, if a formula in the fragment L_{cof} is forced by a given index with local assignment s, it is forced by any other index and local assignment provided it has the same term-functional image as s. We now present a reduction schema which takes advantage of this fact to obtain a bound on the number of tuples of terms which must be considered to force an entailment.

▶ **Lemma 12** (Term image invariance). *Let $\phi \in L_{cof,m}$. Let $s \in \mathcal{T}^m_n$ and let $t \in \mathcal{T}^m_p$. If $\operatorname{img}(s^{\mathcal{A}}) = \operatorname{img}(t^{\mathcal{A}})$ and $\mathrm{X}^n \Vdash \phi\,(s)$, then $\mathrm{X}^p \Vdash \phi\,(t)$.*

Proof. Induction on ϕ. The only interesting case is when $\phi = (f = g)$. Then it suffices to prove that if $fs^{\mathcal{A}} = gs^{\mathcal{A}}$, then $ft^{\mathcal{A}} = gt^{\mathcal{A}}$. Let $a \in \mathcal{A}^p$. By assumption, there exists $b \in \mathcal{A}^n$ such that $s^{\mathcal{A}}(b) = t^{\mathcal{A}}(a)$. But then

$$ft^{\mathcal{A}}(a) \quad = \quad fs^{\mathcal{A}}(b) \quad = \quad gs^{\mathcal{A}}(b) \quad = \quad gt^{\mathcal{A}}(a)$$ ◀

As a consequence, forcing of a cofibration formula ϕ with a local assignment s is reducible to checking that ϕ' holds with respect to the term-functional image of s given an assignment to the second-order variable R.

▶ **Lemma 13.** *Let $m, n \geq 0$, let $\phi \in L_{cof,m}$, and let $s \in \mathcal{T}^m_n$. Let $M = \operatorname{img}(s^{\mathcal{A}})$. Then*

$$X^n \Vdash \phi\,(s) \quad \textit{iff} \quad \mathcal{A} \vDash \phi'(M)$$

Proof. Induction on ϕ. We omit the cases which follow directly by induction.

- Suppose $\phi = \bot$ or $\phi = \top$. Immediate.
- Suppose $\phi = (f = g)$. Fix $a \in \mathcal{A}^m$, and suppose that $a \in M$. Let $b \in \mathcal{A}^n$ satisfy $s^{\mathcal{A}}(b) = a$. By assumption, $X^n \Vdash (f = g)\,(s)$, and so $fs^{\mathcal{A}} = gs^{\mathcal{A}}$. Hence $f^{\mathcal{A}}(a) = fs^{\mathcal{A}}(b) = gs^{\mathcal{A}}(b) = g^{\mathcal{A}}(a)$.

  For the converse, we need to show that $fs^{\mathcal{A}} = gs^{\mathcal{A}}$. Fix $b \in \mathcal{A}^n$. Because $s^{\mathcal{A}}(b) \in M$, $fs^{\mathcal{A}}(b) = gs^{\mathcal{A}}(b)$ by assumption.
- Suppose $\phi = \forall x_{m+1}\, \psi$. Recall that $\mathrm{X}^n \Vdash \forall x_{m+1}\, \psi\,(s)$ if and only if $\mathrm{X}^{n+1} \Vdash \psi\,(s, x_{n+1})$; while the induction hypothesis is that

  $$X^{n+1} \Vdash \psi\,(s, x_{n+1}) \quad \text{iff} \quad \mathcal{A} \vDash \psi'_{m+1}(M \times \mathcal{A})$$

  The conclusion follows by Remark 11. ◀

In the event that the cofibration ϕ is a conjunction of atomic formulas, we can dispense with the translation (Definition 10) altogether:

▶ **Corollary 14.** *Let $m, n \geq 0$, and let $\phi \in L_{cof,m}$ be a conjunction of atomic formulas. Let $s \in \mathcal{T}_n^m$, and let $M = \mathrm{img}(s^{\mathcal{A}})$. Then*

$$X^n \Vdash \phi\,(s) \quad \textit{iff} \quad \textit{for all } a \in M, \mathcal{A} \vDash \phi(a)$$

To extend the translation (Definition 10) of formulas to entailment instances, we must reduce the quantification over all m-tuples of terms. Term image invariance allows us to reduce this to quantification over all term-functional images included by $\mathcal{A}^m$. Whence stems the decidability of the problem as well as its inefficiency. Notice that while belonging to a term-functional image of a given $s \in \mathcal{T}^m$ defines an m-ary relation over $\mathcal{A}$, being a term-functional image defines a property of m-ary relations over $\mathcal{A}$, one which we may try to capture by a second-order formula:

▶ **Definition 15.** *Let $m \geq 0$. Let χ_m be a formula in the second-order language over the signature of $\mathcal{A}$ whose only free variable is the second-order variable R with arity m. We say that χ_m* characterizes images for $\mathcal{A}$-term functions *whenever the following conditions are equivalent for all $M \subseteq \mathcal{A}^m$:*

1. $\mathcal{A} \vDash \chi_m\,(M)$
2. *there exist $n \geq 0$ and $s \in \mathcal{T}_n^m$ such that $M = \mathrm{img}(s^{\mathcal{A}})$*

Indeed, a second-order characterization of term-functional images is all that is needed for this reduction schema.

▶ **Theorem 16.** *Let $m \geq 0$, let $\phi_1, \phi_2 \in L_{cof,m}$. Let χ_m be a formula which characterizes images for $\mathcal{A}$-term functions. Then*

$$\mathrm{X}^m \Vdash \phi_1 \Rightarrow \phi_2\,(x) \quad \textit{iff} \quad \mathcal{A} \vDash \forall R\left(\chi_m \to \phi_1' \to \phi_2'\right)$$

Proof. Fix $M \subseteq \mathcal{A}^m$ and $a \in \mathcal{A}^m$. Suppose that $\mathcal{A} \vDash \chi_m\,(M)$. Fix $n \geq 0$ and $s \in \mathcal{T}_n^m$ according to Definition 15. Specializing the assumption, we have that $\mathrm{X}^n \Vdash \phi_1\,(s)$ implies $\mathrm{X}^n \Vdash \phi_2\,(s)$. Hence, by Lemma 13, we have that $\mathcal{A} \vDash \phi_1'\,(M)$ implies $\mathcal{A} \vDash \phi_2'\,(M)$.

For the converse, fix $n \geq 0$ and $s \in \mathcal{T}_n^m$. Let $M = \mathrm{img}(s^{\mathcal{A}})$. According to Definition 15, we have that $\mathcal{A} \vDash \chi_m\,(M)$. And by assumption, $\mathcal{A} \vDash \phi_1'\,(M)$ implies $\mathcal{A} \vDash \phi_2'\,(M)$. So, by Lemma 13, $\mathrm{X}^n \Vdash \phi_1'\,(s)$ implies $\mathrm{X}^n \Vdash \phi_2'\,(s)$. ◀

4.2.3 Second schema

We will now present a reduction schema which applies to generating algebras with language fragments which have the property of *local factorization* (Definition 17). To abstract over the choice of language fragment, we let the variable γ denote a generic element of $\{\mathit{cof}, \mathit{coftc}, \mathit{cofvc}\}$.

▶ **Definition 17.** *Let $m \geq 0$. A formula $\rho \in L_{\gamma,m}$ is* locally prime *if ρ does not entail $\bot$, and for all $\phi_1, \phi_2 \in L_{\gamma,m}$, if ρ entails $\phi_1 \vee \phi_2$, then either ρ entails ϕ_1 or ρ entails ϕ_2. An algebra $\mathcal{A}$ paired with a language fragment L_γ has* local factorization *if for all m and formulas $\phi \in L_{\gamma,m}$, ϕ is equivalent to the disjunction (in $L_{\gamma,m}$) of locally prime formulas which entail ϕ.*

We assume for the remainder of this section that $(\mathcal{A}, L_\gamma)$ has local factorization.

▶ **Lemma 18.** *Let $m \geq 0$, and let $\phi, \rho \in L_{\gamma,m}$. Let $M = \{a \in \mathcal{A}^m \mid \mathcal{A} \vDash \rho\,(a)\}$. Assume that for all terms $f, g \in \mathcal{T}_m$,*

$$\mathrm{X}^m \Vdash \rho \Rightarrow (f = g)\,(x) \quad \textit{iff} \quad \mathcal{A} \vDash (f = g)'\,(M)$$

If ρ is locally prime, then

$$\mathrm{X}^m \Vdash \rho \Rightarrow \phi\,(x) \quad \textit{iff} \quad \mathcal{A} \vDash \phi'\,(M)$$

Proof. Induction on ϕ.

- Suppose $\phi = \bot$ or $\phi = \top$. Trivial.
- Suppose $\phi = (f = g)$. By assumption.
- Suppose $\phi = \phi_1 \wedge \phi_2$. Directly by induction.
- Suppose $\phi = \phi_1 \vee \phi_2$. Suppose that $\mathrm{X}^m \Vdash \rho \Rightarrow \phi_1 \vee \phi_2\,(x)$. Because ρ is $\vee$-prime, either $\mathrm{X}^m \Vdash \rho \Rightarrow \phi_1\,(x)$ or $\mathrm{X}^m \Vdash \rho \Rightarrow \phi_2\,(x)$. The conclusion follows by induction. The converse also follows directly by induction.
- Suppose $\phi = \forall x_{m+1}\,\psi$. Recall that $\mathrm{X}^m \Vdash \rho \Rightarrow \forall x_{m+1}\,\psi\,(x)$ if and only if $\mathrm{X}^{m+1} \Vdash \rho \Rightarrow \psi\,(x)$; while the induction hypothesis is that

  $$X^{n+1} \Vdash \rho \Rightarrow \psi\,(x) \quad \text{iff} \quad \mathcal{A} \vDash \psi'_{m+1}(M \times \mathcal{A})$$

  The conclusion follows by Remark 11. ◀

▶ **Lemma 19.** *Let $m \geq 0$, and let $\phi_1, \phi_2 \in L_{\gamma,m}$. Let P be the set of locally prime formulas in $L_{\gamma,m}$. Then*

$$\mathrm{X}^m \Vdash \phi_1 \Rightarrow \phi_2\,(x) \quad \textit{iff}$$
$$\textit{for all } \rho \in P,\ \mathrm{X}^m \Vdash \rho \Rightarrow \phi_1\,(x) \textit{ implies } \mathrm{X}^m \Vdash \rho \Rightarrow \phi_2\,(x)$$

Proof. The forward direction follows directly by the semantics. For the converse, let $s \in \mathcal{T}_n^m$ and suppose that $\mathrm{X}^n \Vdash \phi_1\,(s)$. Let $P' = \{\rho \in P \mid \mathrm{X}^m \Vdash \rho \Rightarrow \phi_1\,(x)\}$. From local factorization, we have

$$\mathrm{X}^m \Vdash \phi_1 \Leftrightarrow \bigvee_{\rho \in P'} \rho\,(x)$$

Fix $\rho \in P'$ such that $\mathrm{X}^n \Vdash \rho\,(s)$. By definition of P' and hypothesis, $\mathrm{X}^m \Vdash \rho \Rightarrow \phi_2\,(x)$. In particular, $\mathrm{X}^m \Vdash \phi_2\,(s)$. ◀

▶ **Definition 20.** *Let $m \geq 0$. Let P be the set of locally prime formulas in $L_{\gamma,m}$. Let χ_m be a formula in the second-order language over the signature of $\mathcal{A}$ whose only free variable is the second-order variable R with arity m. We say that χ_m* defines P (the locally prime formulas in $L_{\gamma,m}$) *when the following are equivalent for all $M \subseteq \mathcal{A}^m$:*

- $\mathcal{A} \vDash \chi_m\,(M)$
- *there exists $\rho \in P$ such that $M = \{a \in \mathcal{A}^m \mid \mathcal{A} \vDash \rho\,(a)\}$*

▶ **Theorem 21.** *Let $m \geq 0$, and let $\phi_1, \phi_2 \in L_{\gamma,m}$. Let P be the set of locally prime formulas in $L_{\gamma,m}$. Let the formula χ_m define P (Definition 20). If for all $\rho \in P$, and for all terms $f, g \in \mathcal{T}_m$,*

$$\mathrm{X}^m \Vdash \rho \Rightarrow (f = g)\,(x) \quad \textit{iff} \quad \mathcal{A} \vDash (f = g)'\,(M)$$

then

$$\mathrm{X}^m \Vdash \phi_1 \Rightarrow \phi_2\,(x) \quad \textit{iff} \quad \mathcal{A} \vDash \forall R\,(\chi_m \to \phi'_1 \to \phi'_2)$$

Proof. By Lemma 19, it suffices to show that

$$\begin{aligned}&\text{for all } \rho \in P,\ \mathrm{X}^m \Vdash \rho \Rightarrow \phi_1\,(x) \text{ implies } \mathrm{X}^m \Vdash \rho \Rightarrow \phi_2\,(x)\\ &\text{iff} \quad \mathcal{A} \vDash \forall R\,(\chi_m \rightarrow \phi_1' \rightarrow \phi_2')\end{aligned}$$

Let $M \subseteq \mathcal{A}^m$, and suppose that $\mathcal{A} \vDash \chi_m\,(M)$ and $\mathcal{A} \vDash \phi_1'\,(M)$. Then, by Lemma 18, there exists $\rho \in P$ such that $\mathrm{X}^m \Vdash \rho \Rightarrow \phi_1\,(x)$; and by assumption, $\mathcal{A} \vDash \phi_2'\,(M)$.

For the converse, suppose $\rho \in P$, and that $\mathrm{X}^m \Vdash \rho \Rightarrow \phi_1\,(x)$. Let $M = \{a \in \mathcal{A}^m \mid \mathcal{A} \vDash \rho\,(a)\}$. Then $\mathcal{A} \vDash \chi_m\,(M)$, and by Lemma 18, $\mathcal{A} \vDash \phi_1'\,(M)$; and by assumption, $\mathrm{X}^m \Vdash \rho \Rightarrow \phi_2\,(x)$. ◀

4.3 Parameter elimination

In this section we will prove the following proposition, which will allow us to transport a classification of COFENT to PCOFENT:

▶ **Proposition 22.** *There exist efficient mapping reductions from* PCOFENT *to* COFENT*, from* PCOFENTTC *to* COFENTTC*, and from* PCOFENTVC *to* COFENTVC.

Proof. Corollaries 24 and 29 ◀

For the remainder of this subsection, let $\mathrm{B} \subseteq \Omega$ be defined by $\mathrm{B}\,C = \{\emptyset, \mathrm{t}(C)\}$ for all $C \in \mathcal{C}$. Let Φ be an arbitrary subobject satisfying the constraint $\mathrm{B} \subseteq \Phi \subseteq \Omega$.

4.3.1 Bivalent parameters

▶ **Lemma 23.** *Let $\phi \in L_{pcof,m}$. Let k be the maximal parameter index of ϕ. Let $n \geq 0$, let $s \in \mathrm{Hom}_{\mathcal{C}}(\mathrm{X}^n, \mathrm{X}^m)$, and let $h \in \Phi\,\mathrm{X}^n$. Define $b_h \in \mathrm{B}^k\,\mathrm{X}^n$ by*

$$b_{h,i} = \begin{cases} h_i & h_i = \mathrm{t}(\mathrm{X}^n) \\ \emptyset & \textit{otherwise} \end{cases}$$

Then $\mathrm{X}^n \Vdash \phi\,(s\,;b_h)$ if and only if $\mathrm{X}^n \Vdash \phi\,(s\,;h)$.

Proof. Induction on ϕ. When $\phi = \alpha_i$, we have

$$\begin{aligned}\mathrm{X}^n \Vdash \alpha_i\,(s\,;b_h) \quad &\text{iff} \quad b_{h,i} = \mathrm{t}(\mathrm{X}^n)\\ &\text{iff} \quad h_i = \mathrm{t}(\mathrm{X}^n)\\ &\text{iff} \quad \mathrm{X}^n \Vdash \alpha_i\,(s\,;h)\end{aligned}$$

When ϕ is otherwise atomic, then $\mathrm{X}^n \Vdash \phi\,(s\,;b_h)$ if and only if $\mathrm{X}^n \Vdash \phi\,(s)$; and likewise for $\mathrm{X}^n \Vdash \phi\,(s\,;h)$. The induction cases are straightforward. ◀

▶ **Corollary 24.** *Let $\phi_1, \phi_2 \in L_{pcof,m}$. Let k be the maximal parameter index of $\phi_1 \Rightarrow \phi_2$. Then*

$$\mathrm{X}^m \Vdash \forall\alpha^{\Phi}\,\big(\phi_1 \Rightarrow \phi_2\big)\,(x) \quad \textit{iff} \quad \mathrm{X}^m \Vdash \forall\alpha^{B}\,\big(\phi_1 \Rightarrow \phi_2\big)\,(x)$$

Proof. The forward direction is trivial. For the converse, fix $n \geq 0$, $s \in \mathrm{Hom}_{\mathcal{C}}(\mathrm{X}^n, \mathrm{X}^m)$, and $h \in \Phi^k\,\mathrm{X}^n$. Our goal is to show that $\mathrm{X}^n \Vdash \phi_1 \Rightarrow \phi_2\,(s\,;h)$. Hence, fix $p \geq 0$ and $t \in \mathrm{Hom}_{\mathcal{C}}(\mathrm{X}^p, \mathrm{X}^n)$, and suppose that $\mathrm{X}^p \Vdash \phi_1\,(st\,;f\cdot t)$. We must show that $\mathrm{X}^p \Vdash \phi_2\,(st\,;f\cdot t)$. By Lemma 23, it suffices to check that $\mathrm{X}^p \Vdash \phi_2\,(st\,;b_{f\cdot t})$; and by assumption, that $\mathrm{X}^p \Vdash \phi_1\,(st\,;b_{f\cdot t})$; which follows again from Lemma 23. ◀

4.3.2 Formula lifting

Next we introduce a piece of technical machinery to weaken a formula in L_m: that is, add to the list of available free variables, irrespective of their non-occurrence. Because variables are bound in order of decreasing index, dummy free variables should be inserted before existing variables.

▶ **Definition 25.** *Let $l, m \geq 0$, and let $\phi \in L_m$. We define $\mathrm{lift}_{l,m} : L_m \to L_{l+m}$ by*

$$\mathrm{lift}_{l,m}(\phi) = \begin{cases} \bot & \phi = \bot \\ \top & \phi = \top \\ (f = g)[x_{l+1}/x_1, \ldots, x_{l+m}/x_m] & \phi = (f = g) \\ \alpha_i & \phi = \alpha_i \\ \mathrm{lift}_{l,m}(\phi_1) \wedge \mathrm{lift}_{l,m}(\phi_2) & \phi = \phi_1 \wedge \phi_2 \\ \mathrm{lift}_{l,m}(\phi_1) \vee \mathrm{lift}_{l,m}(\phi_2) & \phi = \phi_1 \vee \phi_2 \\ \mathrm{lift}_{l,m}(\phi_1) \Rightarrow \mathrm{lift}_{l,m}(\phi_2) & \phi = \phi_1 \Rightarrow \phi_2 \\ \exists x_{l+m+1}\ \mathrm{lift}_{l,m+1}(\psi) & \phi = \exists x_{m+1}\, \psi \\ \forall x_{l+m+1}\ \mathrm{lift}_{l,m+1}(\psi) & \phi = \forall x_{m+1}\, \psi \end{cases}$$

▶ **Proposition 26.** *Let $k, l, m \geq 0$. Let $\phi \in L_m$ with maximal index k. Let $\Phi = \Phi_1 \times \cdots \times \Phi_k$, where $\Phi_i \subseteq \Omega$ for all i. Let $R = R_1 \times \cdots \times R_l$ and let $S = S_1 \times \cdots \times S_m$. Let $X \in \mathcal{C}$, let $h \in \Phi X$, let $r \in RX$, and let $s \in SX$. Then $X \Vdash \phi\,(s\,;h)$ if and only if $X \Vdash \mathrm{lift}(\phi)\,(r\,;s\,;h)$.*

4.3.3 Grounding

In Lemmas 27 and 28 and Corollary 29 below, given $k \geq 0$, let $\epsilon \in L_{2k}^k$ be the k-tuple of equations defined so that $\epsilon_i = (x_i = x_{k+i})$.

▶ **Lemma 27.** *Let $\phi \in L_{pcof,m}$. Let k be the maximal parameter index of ϕ. Let $n, p \geq 0$, let $s \in \mathrm{Hom}_{\mathcal{C}}(\mathrm{X}^n, \mathrm{X}^m)$, and let $t \in \mathrm{Hom}_{\mathcal{C}}(\mathrm{X}^p, \mathrm{X}^{k+n})$. Define $b_t \in \mathrm{B}^k\,\mathrm{X}^p$ by*

$$b_{t,i} = \begin{cases} \mathrm{t}(\mathrm{X}^p) & t_i = 1 \\ \emptyset & \textit{otherwise} \end{cases}$$

The following are equivalent:

- $\mathrm{X}^p \Vdash \phi\,(s(t_{k+1}, \ldots, t_{k+n})\,; b_t)$
- $\mathrm{X}^p \Vdash \mathrm{lift}(\phi)\,\big[\epsilon/\alpha\big]\,(t_1, \ldots, t_k\,; 1, \ldots, 1\,; s(t_{k+1}, \ldots, t_{k+n}))$

Proof. Induction on ϕ. When $\phi = \alpha_i$, we have

$$\begin{aligned} &\mathrm{X}^p \Vdash \alpha_i\,(s(t_{k+1}, \ldots, t_{k+n})\,; b_t) \\ &\text{iff } b_{t,i} = \mathrm{t}(\mathrm{X}^n) \\ &\text{iff } t_i = 1 \\ &\text{iff } \ \mathrm{X}^p \Vdash (x_i = x_{k+i})\,(t_1, \ldots, t_k\,; 1, \ldots, 1\,; s(t_{k+1}, \ldots, t_{k+n})) \end{aligned}$$

When ϕ is otherwise atomic, then

$$\mathrm{X}^p \Vdash \phi\,(s(t_{k+1}, \ldots, t_{k+n})\,; b_t) \quad \text{iff} \quad \mathrm{X}^p \Vdash \phi\,(s(t_{k+1}, \ldots, t_{k+n}))$$

This holds, by Proposition 26, if and only if

$$\mathrm{X}^p \Vdash \mathrm{lift}(\phi)\,\big[\epsilon/\alpha\big]\,(t_1, \ldots, t_k\,; 1, \ldots, 1\,; s(t_{k+1}, \ldots, t_{k+n}))$$

The induction cases are straightforward. ◀

▶ **Lemma 28.** *Let $\phi \in L_{pcof,m}$. Let k be the maximal parameter index of ϕ. Let $n \geq 0$, let $s \in \mathrm{Hom}_{\mathcal{C}}(\mathrm{X}^n, \mathrm{X}^m)$, and let $b \in \mathrm{B}^k\,\mathrm{X}^n$. Define $a_b \in \mathrm{B}^k\,\mathrm{X}^n$ by*

$$a_{b,i} = \begin{cases} 1 & b_i = \mathrm{t}(\mathrm{X}^n) \\ 0 & \textit{otherwise} \end{cases}$$

Then

$$\mathrm{X}^n \Vdash \phi\,(s\,;b) \quad \textit{iff} \quad \mathrm{X}^n \Vdash \mathrm{lift}(\phi)\big[\epsilon/\alpha\big]\,(a_b\,;1,\ldots,1\,;s)$$

Proof. Induction on ϕ. When $\phi = \alpha_i$, we have

$$\begin{aligned} \mathrm{X}^n \Vdash \alpha_i\,(s\,;b) \quad & \text{iff} \quad b_i = \mathrm{t}(\mathrm{X}^n) \\ & \text{iff} \quad a_{b,i} = 1 \\ & \text{iff} \quad \mathrm{X}^n \Vdash (x_i = x_{k+i})\,(a_b\,;1,\ldots,1\,;s) \end{aligned}$$

When ϕ is otherwise atomic, then $\mathrm{X}^n \Vdash \phi\,(s\,;b)$ if and only if $\mathrm{X}^n \Vdash \phi\,(s)$; which holds, by Proposition 26, if and only if $\mathrm{X}^n \Vdash \mathrm{lift}(\phi)[\epsilon/\alpha]\,(a_b\,;1,\ldots,1\,;s)$. The induction cases are straightforward. ◀

▶ **Corollary 29.** *Let $\phi_1, \phi_2 \in L_{pcof,m}$. Let k be the maximal parameter index of $\phi_1 \Rightarrow \phi_2$. The following are equivalent:*

- $\mathrm{X}^m \Vdash \forall\alpha^B\,\big(\phi_1 \Rightarrow \phi_2\big)\,(x)$
- $\mathrm{X}^{k+m} \Vdash \mathrm{lift}(\phi_1 \Rightarrow \phi_2)\big[\epsilon/\alpha\big]\,(x_1,\ldots,x_k\,;1,\ldots,1\,;x_{k+1},\ldots,x_{k+m})$

Proof. Fix $n \geq 0$ and $s \in \mathrm{Hom}_{\mathcal{C}}(\mathrm{X}^n, \mathrm{X}^{k+m})$, and suppose that

$$\mathrm{X}^n \Vdash \mathrm{lift}(\phi_1)\big[\epsilon/\alpha\big]\,(s_1,\ldots,s_k\,;1,\ldots,1\,;s_{k+1},\ldots,s_{k+m})$$

By Lemma 27 and by assumption, $\mathrm{X}^n \Vdash \phi_2\,(s_{k+1},\ldots,s_{k+m}\,;b_t)$. And again by Lemma 27,

$$\mathrm{X}^n \Vdash \mathrm{lift}(\phi_2)\big[\epsilon/\alpha\big]\,(s_1,\ldots,s_k\,;1,\ldots,1\,;s_{k+1},\ldots,s_{k+m})$$

For the converse, fix $n \geq 0$, $s \in \mathrm{Hom}_{\mathcal{C}}(\mathrm{X}^n, \mathrm{X}^m)$, and $b \in \mathrm{B}^k\,\mathrm{X}^n$. Our goal is to check that $\mathrm{X}^n \Vdash \phi_1 \Rightarrow \phi_2\,(s\,;b)$. So fix $p \geq 0$ and $t \in \mathrm{Hom}_{\mathcal{C}}(\mathrm{X}^p, \mathrm{X}^n)$, and suppose that $\mathrm{X}^p \Vdash \phi_1\,(st\,;b\cdot t)$. By Lemma 28 and by assumption,

$$\mathrm{X}^p \Vdash \mathrm{lift}(\phi_2)\big[\epsilon/\alpha\big]\,(a_{b\cdot t}\,;1,\ldots,1\,;st)$$

So by Lemma 28, $\mathrm{X}^p \Vdash \phi_2\,(st\,;b\cdot t)$. ◀

4.4 Applications

To apply the first reduction schema to a concrete algebra $\mathcal{A}$, it suffices to supply a formula χ (Definition 15) characterizing local images for $\mathcal{A}$. To apply the second reduction schema to a concrete algebra $\mathcal{A}$ and language L_γ, it suffices to supply a formula χ (Definition 20) which defines the locally prime formulas of L_γ.

For either schema, if the prenex normal form of χ contains no universal second-order quantifiers, and it is efficiently computable from m, then the sentence checked in Theorem 16 is in ∀SO; and by Fagin's theorem, the corresponding model-checking problem is in coNP (cf. [4, Section 9.2]). This is the case for all of the following applications.

Below, $\leq$ will denote the standard ordering on the set $\{0,1\}$, as well as this ordering extended component-wise to $\{0,1\}^n$. We will denote the minimal and maximal elements of the poset $\{0,1\}^n$ also by 0 and 1.

4.4.1 Bipointed sets

For a theory of bipointed sets, the signature will be $\sigma_0 = \{0, 1\}$. We take the generating algebra $\mathcal{A}$ to be $\{0, 1\}$ with the constants interpreted as $0^{\mathcal{A}} = 0$ and $1^{\mathcal{A}} = 1$. When needed, we will annotate constructions associated to this algebra with *bp*.

We apply the first schema. The following second order formula encodes each term function $s_i^{\mathcal{A}} : \mathcal{A}^m \to \mathcal{A}$ as a second-order variable T^{m+1}: the first conjunct ensures that T encodes either a projection or a constant, and the second conjunct ensures that R is indeed the image of functions encoded by the T_i.

▶ **Definition 30.** *Let $m \geq 0$. Define χ_m (for bp terms) to be the formula*

$$\begin{aligned}\exists T_1^{m+1} \ldots \exists T_m^{m+1} \Big(\bigwedge_i \big(\bigvee_j & \forall y \forall x \left(T_i(y\,;x) \;\leftrightarrow\; x = y_j\right) \\ & \vee\; \forall y \forall x \left(T_i(y\,;x) \;\leftrightarrow\; x = 0\right) \\ & \vee\; \forall y \forall x \left(T_i(y\,;x) \;\leftrightarrow\; x = 1\right)\big) \\ \wedge\; \forall x \big(R(x) \;\leftrightarrow\; & \exists y \big(\bigwedge_i T_i(y\,;x_i)\big)\big)\Big)\end{aligned}$$

▶ Remark 31. Let $s \in \mathcal{T}_n^m$ be an m-tuple of bipointed-set terms. There exist indices $i_1 < i_2 < \cdots < i_m$ in $\{1, \ldots, n\}$ and $s' \in \mathcal{T}_m^m$ such that $s = s'[x_{i_1}/x_1, \ldots, x_{i_m}/x_m]$. In particular, for all $n \geq 0$ and $s \in \mathcal{T}_n^m$, there exists $s' \in \mathcal{T}_m^m$ such that $\mathrm{img}(s'^{\mathcal{A}}) = \mathrm{img}(s^{\mathcal{A}})$.

▶ **Proposition 32.** *Let $m \geq 0$. The formula χ_m in Definition 30 characterizes images of bp term functions.*

▶ **Corollary 33.** *The decision problem* $\mathrm{PCOFENT}_{bp}$ *is* coNP*-complete.*

4.4.2 Distributive lattices

For the theory of bounded distributive lattices, the signature will be $\sigma_0 = \{0, 1\}$ and $\sigma_2 = \{\wedge, \vee\}$. We take the generating algebra $\mathcal{A}$ to be $\{0, 1\}$ with the constants interpreted as $0^{\mathcal{A}} = 0$ and $1^{\mathcal{A}} = 1$, and with the basic operations defined as $\wedge^{\mathcal{A}} = \min$ and $\vee^{\mathcal{A}} = \max$. When needed, we will annotate constructions associated to this algebra with *dl*.

▶ **Lemma 34.**

$$\{s^{\mathcal{A}} : \mathcal{A}^n \to \mathcal{A} \mid s \in \mathcal{T}_n\} \quad\cong\quad \{f : \mathcal{A}^n \to \mathcal{A} \mid f \text{ is monotone }\}$$

Proof. Let $s \in \mathcal{T}_n$. That $s^{\mathcal{A}}$ is monotone follows from the fact that the constant functions, the projections, min and max are all monotone.

Conversely, let f be a monotone function. There are many ways to define $s \in \mathcal{T}_n$ so that $s^{\mathcal{A}} = f$, and here is one way. Define s by

$$s = \bigvee_{f(a)=1} \bigwedge_{a_k=1} x_k$$

Suppose $f(a) = 1$. Since $\left(\bigwedge_{a_k=1} x_k\right)^{\mathcal{A}}(a) = 1$, $s^{\mathcal{A}}(a) = 1$. Conversely, suppose that $s^{\mathcal{A}}(a) = 1$. Then there exists a' such that $f(a') = 1$ and $(\bigwedge_{a'_k=1} x_k)^{\mathcal{A}}(a) = 1$. Hence, $a' \leq a$. But f is monotone, so $f(a) = 1$. ◀

▶ **Lemma 35.** *Let $B \subseteq \mathcal{A}^m$. Then B contains* $\min(B)$ *and* $\max(B)$ *if and only if there exist $n \geq 0$ and $s \in \mathcal{T}_n^m$ such that* $\mathrm{img}(s^{\mathcal{A}}) = B$.

Proof. Let $\{b_1, \dots, b_n\}$ be an enumeration of B. Define the monotone function $f : \mathcal{A}^n \to \mathcal{A}^m$ by

$$f(a) = \begin{cases} \min(B) & a = 0 \\ b_j & a_k = 1 \text{ iff } j = k \\ \max(B) & \text{otherwise} \end{cases}$$

By Lemma 34, there exists an m-tuple of terms $s \in \mathcal{T}_n^m$ such that $s^{\mathcal{A}} = f$.

For the converse, observe that $\min(\mathrm{img}(s^{\mathcal{A}})) = s^{\mathcal{A}}(0)$ and $\max(\mathrm{img}(s^{\mathcal{A}})) = s^{\mathcal{A}}(1)$. ◀

▶ **Definition 36.** *Let $m \geq 0$. Define χ_m (for dl terms) to be the formula*

$$\exists x \left(R(x) \wedge \forall y \left(R(y) \to \textstyle\bigwedge_i (x_i = x_i \wedge y_i)\right)\right)$$
$$\wedge\ \exists x \left(R(x) \wedge \forall y \left(R(y) \to \textstyle\bigwedge_i (y_i = x_i \wedge y_i)\right)\right)$$

▶ **Proposition 37.** *Let $m \geq 0$. The formula χ_m in Definition 36 characterizes images for dl term functions.*

▶ **Corollary 38.** *The decision problem* $\mathrm{PCOFENT}_{dl}$ *is* coNP*-complete.*

4.4.3 Variable-constant fragments

We will now apply the second schema to show that irrespective of the choice of algebra $\mathcal{A}$, PCOFENTVC belongs to coNP.

▶ **Lemma 39.** *Let $m \geq 0$. On the one hand, if $\mathcal{A} \subseteq \{c^{\mathcal{A}}\}$, then $\forall x_{m+1} (x_{m+1} = c)$ is valid; and if not, $\forall x_{m+1} (x_{m+1} = c)$ is equivalent to $\bot$. On the other hand, for $(x_i = c) \in L_{cof,m}$, $\forall x_{m+1} (x_i = c)$ is equivalent to $(x_i = c)$.*

▶ Remark 40. Let $m \geq 0$, and let $\phi \in L_{cofvc,m}$. By Corollary 5 and Lemma 39, we may assume wlog that ϕ is, up to equivalence, either $\bot$, $\top$, or a disjunction of conjunctions of equations.

▶ **Proposition 41.** *Let $\phi \in L_{cofvc,m}$. Then*

$$\mathrm{X}^m \Vdash \phi \Rightarrow (x_j = c)\,(x) \quad \textit{iff} \quad \mathcal{A} \vDash \forall x \left(\phi \to x_j = c\right)$$

Proof. Induction on ϕ (in disjunctive form).

- Suppose $\phi = \bot$ or $\phi = \top$. Immediate.
- Suppose that $\phi = \bigwedge_{i \in I} (x_i = c_i)$ where $I \subseteq \{1, \dots, m\}$.
  - Suppose that $j \in I$ and $c_j = c$. Immediate.
  - Suppose that $j \notin I$ and $c_j = c$. Suppose that $\mathrm{X}^m \Vdash \bigwedge_{i \in I} (x_i = c_i) \Rightarrow (x_j = c)\,(x)$. In particular, $\mathrm{X}^m \Vdash (x_j = c)\,(s)$ when

    $$s_k = \begin{cases} c & k \in I \\ x_k & \text{otherwise} \end{cases}$$

    Therefore, $\mathcal{A} = \{c^{\mathcal{A}}\}$, and the conclusion follows trivially. The converse is analogous.
  - Suppose that $j \in I$ and $c_j \neq c$. But $\mathrm{X}^m \nVdash \bigwedge_{i \in I} (x_i = c_i) \Rightarrow (x_j = c)\,(x)$ because $\mathrm{X}^0 \Vdash (x_j = c_j)\,(c_j)$ while $\mathrm{X}^0 \nVdash (x_j = c)\,(c_j)$. And likewise, $\mathcal{A} \nvDash \bigwedge_{i \in I} (x_i = c_i) \to (x_j = c)\,(c_j)$.
- Suppose $\phi = \phi_1 \vee \phi_2$. Directly by induction. ◀

▶ **Proposition 42.** *Let $m \geq 0$, and let $S_1, S_2 \in \Omega\,\mathrm{X}^m$. Let E be the sieve denoted by the equation $(x_i = c) \in L_{cofvc,m}$: that is,*

$$E = \{s : \mathrm{X}^n \to \mathrm{X}^m \mid n \geq 0,\ s_i = c\} \in \Omega\,\mathrm{X}^m$$

If $E \subseteq S_1 \cup S_2$, then either $E \subseteq S_1$ or $E \subseteq S_2$.

Proof. Observe that E is principal: i.e., every morphism in E factors through

$$s = (x_1, \ldots, x_{i-1}, c, x_{i+1}, \ldots, x_m)$$

So if $s \in S_1$ ($s \in S_2$), then $E \subseteq S_1$ ($E \subseteq S_2$). ◀

▶ **Corollary 43.** *Equations in L_{cofvc} are locally prime.*

▶ **Lemma 44.** *Let $m \geq 0$. A formula $\rho \in L_{cofvc,m}$ is locally prime if and only it is equivalent to a formula of the form*

$$\rho \;=\; \bigwedge_{i \in I} (x_i = c_i) \quad \textit{where } c_1, \ldots, c_m \in \sigma_0$$

for some non-empty $I \subseteq \{1, \ldots, m\}$.

▶ **Definition 45.** *Let $m \geq 0$. Define χ_m (for variable-constant fragments) to be the formula*

$$\begin{aligned}\exists C_1 \ldots \exists C_m \Big(&\bigwedge_i \bigvee_j (C_i(c_j) \vee (x_i = c_j)) \\ &\wedge \bigwedge_i \bigwedge_j \bigwedge_{j<k} (C_i(c_j) \vee C_i(c_k)) \\ &\wedge \big(\bigvee_{i,j} \neg C_i(c_j)\big)\Big)\end{aligned}$$

where $c_1, \ldots, c_{|\sigma_0|}$ is an enumeration of σ_0.

▶ **Proposition 46.** *Let $m \geq 0$. The formula χ_m in Definition 45 defines the locally prime formulas in L_{cofvc}.*

▶ **Corollary 47.** *If σ_0 has constants 0 and 1 denoting distinct elements of $\mathcal{A}$, the decision problem* PCOFENTVC *is* coNP*-complete.*

Proof. By Propositions 8, 41, and 46 and Theorem 21. ◀

4.4.4 De Morgan algebras

Lastly, we turn to the theory of De Morgan algebras and the term-constant fragment L_{coftc}, which provide the cofibration layer of the cubical type theory in [3]. The strategy below is adapted from [3]: it amounts to reducing the term-constant variant of COFENT to the variable-constant variant.

The signature of De Morgan algebras will be $\sigma_0 = \{0, 1\}$, $\sigma_1 = \{\neg\}$, and $\sigma_2 = \{\wedge, \vee\}$. We take the generating algebra $\mathcal{A}$ to be $\{0,1\}^2$ with the constants interpreted as $0^{\mathcal{A}} = (0,0)$ and $1^{\mathcal{A}} = (1,1)$, and with the basic binary operations defined as $\wedge^{\mathcal{A}} = \min$ and $\vee^{\mathcal{A}} = \max$. The operation of De Morgan negation on $\{0,1\}^2$ is

x	$\neg^{\mathcal{A}}(x)$
$(0,0)$	$(1,1)$
$(1,0)$	$(1,0)$
$(0,1)$	$(0,1)$
$(1,1)$	$(0,0)$

Note that while Boolean negation swaps $(0,1)$ and $(1,0)$ as well as $(0,0)$ and $(1,1)$, De Morgan negation fixes $(0,1)$ and $(1,0)$, and swaps just $(0,0)$ and $(1,1)$. When needed, we will annotate constructions associated to this algebra with *dm*.

The following definition is adapted from [3]:

▶ **Definition 48.** *Let $m \geq 0$, and let $\phi \in L_{coftc,m}$.*

$$\phi^+ = \begin{cases} \bot & \phi = \bot \text{ or } (c' = c) \text{ with } c' \neq c \\ \top & \phi = \top \text{ or } (c = c) \\ (f = 1)^- & \phi = (f = 0) \\ (x_i = 1) & \phi = (x_i = 1) \\ (f = 1)^- & \phi = (\neg f = 1) \\ (f_1 = 1)^+ \wedge (f_2 = 1)^+ & \phi = (f_1 \wedge f_2 = 1) \\ (f_1 = 1)^+ \vee (f_2 = 1)^+ & \phi = (f_1 \vee f_2 = 1) \\ \phi_1^+ \wedge \phi_2^+ & \phi = \phi_1 \wedge \phi_2 \\ \phi_1^+ \vee \phi_2^+ & \phi = \phi_1 \vee \phi_2 \\ \forall x_{m+1}\, \psi^+ & \phi = \forall x_{m+1}\, \psi \end{cases}$$

$$\phi^- = \begin{cases} (f = 1)^+ & \phi = (f = 0) \\ (x_i = 0) & \phi = (x_i = 1) \\ (f = 1)^+ & \phi = (\neg f = 1) \\ (f_1 = 1)^- \vee (f_2 = 1)^- & \phi = (f_1 \wedge f_2 = 1) \\ (f_1 = 1)^- \wedge (f_2 = 1)^- & \phi = (f_1 \vee f_2 = 1) \end{cases}$$

▶ **Proposition 49.** *Let $m \geq 0$, and let $\phi \in L_{coftc,m}$. The formula $\phi^+ \in L_{cofvc,m}$ is equivalent to ϕ and is proportional in length.*

▶ **Corollary 50.** *The decision problem* PCOFENTTC$_{dm}$ *is* coNP*-complete.*

5 Conclusion

In this paper, we have assembled a capable framework for studying the computational character of key fragments of the internal languages of cubical sets. We have also put it to use and obtained theoretical results with practical importance: not only have we determined the complexity of variants of COFENT, we have reasonable decision procedures. That said, this framework is capable of more:

- We plan to extend our results to simplicial sets. The cofibration language in [6] corresponds not to all presheaves on the cube category, but to sheaves for a specific Grothendieck topology which defines a topos equivalent to simplicial sets. Preliminary work indicates that our results, summarized above, on presheaves on cubical categories will extend to simplicial sets via this equivalence. This would provide the means to automate the cofibration language layer of a directed type theory based on simplicial sets. It would also provide the means to automate the dependently-typed internal language of simplicial sets.
- We plan to extend our results to full first-order cofibration languages. For certain finite algebras $\mathcal{A}$, either or both schemas will apply: for the first schema, it needs to be shown that term-invariance holds for the more expressive fragment; and for the second schema, that the more expressive fragment has local factorization. Deciding validity in this context will give rise to fresh examples of PSPACE-complete problems.

- Finally, we conjecture that there are finite algebras for which the corresponding forcing relation (in presheaves) is undecidable. Our expectation is that counter-examples will help to explain the efficacy of the key properties already discovered: image invariance and local factorization.

References

1 Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Robert Harper, Kuen-Bang Hou (Favonia), and Daniel R. Licata. Syntax and models of cartesian cubical type theory. *Mathematical Structures in Computer Science*, 31(4):424–468, 2021. doi:10.1017/S0960129521000347.

2 Ulrik Buchholtz and Edward Morehouse. Varieties of cubical sets. *Lecture Notes in Computer Science*, pages 77–92, 2017. doi:10.1007/978-3-319-57418-9_5.

3 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom, 2016. arXiv:1611.02108.

4 Leonid Libkin. *Elements of Finite Model Theory*. Springer Publishing Company, Incorporated, 1st edition, 2010.

5 Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer New York, New York, NY, 1994.

6 Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories, 2017. arXiv:1705.07442.

7 Matthew Z. Weaver and Daniel R. Licata. A constructive model of directed univalence in bicubical sets. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 915–928, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394794.