Case Studies in Constructive Mathematics

Erik Parmann

Dissertation for the degree of philosophiae doctor (PhD)
at the University of Bergen

2016

Dissertation date: 22.01.2016

Scientific Environment

The work of this thesis was done at the department of informatics, University of Bergen in the ICT Research School. I was under the supervision of Professor Marc Bezem.

Acknowledgements

I am deeply indebted to my supervisor, Marc Bezem. He has patiently listened to all my ideas, thoroughly examined and questioned my every thought, and meticulously read everything I have written. I deeply enjoyed working together with Marc and consider myself very fortunate to have had him as my supervisor.

I also wish to thank the members of my evaluation committee in advance—Prof. Peter Dybjer, Dr. Martín Escardó, and Dr. Uwe Egbert Wolter—for reading and evaluating my thesis.

I would like to thank my coauthors, Marc Bezem and Thierry Coquand.

I also want to thank Michał Walicki, for whom I have had the fortune of being a teaching assistant for most of my PhD. He teaches the introductory course in logic which made me first love logic 8 years ago, and he sent the crucial email recruiting me to study logic at the Master's level. My life would have been very different without these interventions.

There is a long list of friends and coworkers who deserve thanks. First, I want to thank Dr. Pål Grønås Drange for the previous 9 years. Both life inside and outside of academia would have been less fun without you. The acknowledgement I wrote you in my Master's thesis still stands.

I also want to thank Eivind Jahren, Markus S. Dregi, Dr. Pim van 't Hof, and Hannah A. Hansen for the many lunches and conversations we have had. I also want to thank Ida Rosenlund for chatting with me when I needed a break, smørbukk, and some tea.

A big thanks to the modal logic gang (the majority of which no longer lives in Bergen, unfortunately): Dr. Truls A. Pedersen, Dr. Piotr Kaźmierczak, and Dr. Sjur Dyrkolbotn, for both work and pleasure.

Outside of the computer science faculty, Øyvind Døskeland, Øystein Rolland and Eilin Erevik have all worked hard to keep me sane these years. Thank you. And a big thanks to Martin, Morten and Adam for the all good times.

A big thanks to my whole family, and a special thanks to my parents, Øystein and Nina Parmann. They have managed to strike a balance between encouragement and generosity; encouraging me to work hard while at the same time leaving me safe in the knowledge that they would be proud of me independently of where I

ended up.

Thanks to ma belle-mère Michèle Jaakson, who took time off during her vacation to proofread my introduction.

Finally, I would like to thank my girlfriend Maja Maria Dawn Jaakson for not only being an amazing girlfriend, but also for proofreading my whole thesis. She changes between commas, semicolons and "—", between "is" and "are", between "which" and "who", and she insists on putting the period *inside* the parenthesizes. She is—at the time of submission—the only person in addition to Marc and me who have read the whole thesis. All except this paragraph that is; all the mistakes will surely drive her mad (and if not—the following period will).

Thank you all.

Abstract

The common theme in this thesis is the study of constructive provability: in particular we investigate aspects of *finite sets* and *Kan simplicial sets* from a constructive perspective.

There are numerous definitions of finiteness which are classically equivalent but not constructively so. In other words, constructive mathematics is able to distinguish between more notions of finiteness. We start by investigating some relationships between several ways of defining finiteness for sets of natural numbers. As a new result, we give *strictly bounded* a precise placement in a hierarchy of definitions of finiteness.

We also investigate *streamless sets*, which constitutes another notion of finiteness. Streamless sets require neither decidable equality nor that the set is a subset of an enumerable set, and they are as such more general than strictly bounded sets. It is an open problem whether the Cartesian product of two streamless sets is itself streamless. We show that this holds if at least one of the sets has decidable equality or is of bounded size. The problem remains open for the case where both streamless sets have undecidable equality and fail to be of bounded size. We also show that—in certain constructive systems—the addition of function extensionality makes equality within streamless sets decidable.

Another notion of finiteness is *Noetherian*. Both streamless and Noetherian can be generalized to properties of binary relations, whereby such sets are those where equality is respectively streamless or Noetherian. We provide a proof that all Noetherian relations are streamless—notably, in a type system without inductively defined equality. This result immediately entails that all Noetherian sets are streamless within that type system.

We proceed to investigate aspects of Kan simplicial sets, a notion coming from topology. Kan simplicial sets have recently caught the eye of the type theory community since they can be used to build models of Martin-Löf type theory that validate the Univalence axiom. All known proofs of the following well-known theorem use classical logic: if simplicial sets X and Y are Kan simplicial sets then Y^X is also a Kan simplicial set. This theorem plays an important role in the Kan simplicial set model of type theory. We investigate whether this theorem

also holds constructively. The classical definition of the Kan property has at least two non-equivalent constructive interpretations, and we provide countermodels showing the constructive non-provability of the classical theorem above for both of these definitions of Kan simplicial sets.

List of Papers

Erik Parmann. *Strictly Bounded Sets.* Manuscript.

Erik Parmann. *Investigating Streamless Sets* [Par15]. In *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, volume 39 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 187–201. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

Erik Parmann. *A proof that Noetherian relations are streamless in a type theory without identity.* Manuscript.

Marc Bezem, Thierry Coquand, and Erik Parmann. *Non-Constructivity in Kan Simplicial Sets* [BCP15]. In *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 92–106. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

Erik Parmann. *Functional Kan Simplicial Sets: Non-Constructivity of Exponentiation.* Submitted to *21th International Conference on Types for Proofs and Programs (TYPES 2015).*

Contents

List of Figures

Part I

Introduction and preliminaries

Introduction

This chapter provides an introduction to logic and constructive mathematics. We begin by explaining the absolute fundamentals, hopefully making these understandable for even the most novice reader. Readers with some experience in logic might want to skip to Section 1.2 for an introduction to intuitionistic logic, or all the way to Section 1.3, where we discuss constructive mathematics. In Section 1.4, we give a brief overview of the results covered in this thesis.

1.1 The fundamentals of logic

It is no easy feat to define what logic is. Although the following quote from [BdRV01] concerns modal logic, it certainly rings true of logic in general:

> Ask three modal logicians what modal logic is, and you are likely to get at least three different answers. The authors of this book are no exception, so we won't try to start off with a neat definition.

Given this, we will give readers a sense of what logic is by taking a "hands-on" approach, introducing logic through actual examples.

There are a multitude of logics, but these (typically) have a few things in common. Firstly, they use a well-defined *language*. This means that it is properly specified what one can, and what one cannot write in the logic. As a simple example, "9+4" is a meaningful expression of arithmetic, while "4 6++" is not. Most of us have learned through experience what counts as meaningful expressions of arithmetic, while with logics we usually define—in a formal and exact way—the language of the logic. For example, we can formally define a very simple language Play containing *terms* and *formulae*. The terms are given by the following grammar:

$$t := 0 \mid s(t)$$

The grammar says that 0 is a term, and that if you have a term you obtain a new term by putting parentheses around it and an "s" in front of it. So $s(0)$ is a

term, as is $s(s(s(0)))$ and so on. The formulae of Play are all strings of the form $a \circ b = c$ where a, b, c are terms.

Both "5+2=7" and "5+2=9" are sentences of arithmetic, but the first is special. It is not only a sentence of arithmetic, but a *true* sentence of arithmetic. In the same way, many (but not all) logics have a notion of truth. It is also quite common to have several notions of truth for the same language; then we say that the different notions of truth correspond to different logics.

To give a logic a *semantics* is to give it a notion of meaning: and we often build a notion of *truth* on top of this. The language Play is made with a particular semantic in mind, where the term 0 is interpreted as the number 0, and $s(a)$ is interpreted as 1 plus the interpretation of a. So $s(0)$ is interpreted as 1 and $s(s(s(0)))$ as 3. A formula $a \circ b = c$ is *true* if the interpretation of a *plus* the interpretation of b equals the interpretation of c. So both $s(0) \circ 0 = 0$ and $0 \circ s(s(0)) = s(s(0))$ are formulae of Play, but only the latter is a true formula.

There is another subset of the formulae which often interests us, and that is the set of *deducible* formulae. These are formulae which we are able to create using a specified set of *rules* and *axioms*. We can, for instance, define a very simple deductive system for the language Play containing one axiom and two rules. An axiom is a sentence which is deducible by assumption—axioms form the starting point of any deduction—while rules allow the deduction of new sentences from already deduced sentences. Knowing this, we can start by introducing our only axiom:

$$0 \circ 0 = 0 \tag{1.1}$$

In addition to the axiom, we provide two rules:

$$\frac{a \circ b = c}{s(a) \circ b = s(c)}\ (\mathrm{L}) \qquad\qquad \frac{a \circ b = c}{a \circ s(b) = s(c)}\ (\mathrm{R}) \tag{1.2}$$

The symbols "a", "b" and "c" are meta-variables ranging over the terms of Play. The rules are to be read as follows: "if we have deduced the expression above the line we can deduce the expression below the line". The rules are named "L" and "R" respectively. Below, we show an example of how one can deduce $1 \circ 2 = 3$ by using a line of deductions. We start our deductions with the axiom at the top and write which rule was applied on the left.

$$\begin{array}{ll} & 0 \circ 0 = 0 \\ \mathrm{L} & \hline 1 \circ 0 = 1 \\ \mathrm{R} & \hline 1 \circ 1 = 2 \\ \mathrm{L} & \hline 1 \circ 2 = 3 \end{array}$$

Our simple deduction system is such that all of the sentences it can deduce are also true sentences with regard to the semantics given above. Deduction systems

with this property related to a semantics are said to be *sound* with relation to that semantics. Play also has the beautiful property that *all* of its true formulae are deducible; the system is thus *complete* with respect to the semantics. So the logics defined by the true and deducible formulae of Play turn out to be the same! This is no coincidence; most deduction systems are made with a particular semantics in mind, and Play is no different in this regard. As we will see later, having a corresponding semantics and deduction system for a logic can be quite useful, as it is sometimes significantly easier working with one rather than the other.

1.2 Intuitionistic logic

We are now ready to move on from the simple, "baby" logic presented in the previous section to real logic. In this section we will present *intuitionistic logic*, in both its propositional and first-order form.

1.2.1 Propositional logic

Starting with the *propositional language*, we assume that we have some set Σ of propositional variables, usually denoted as $p, q, p_1, \ldots$. The language of propositional logic is given by the following grammar, where $p \in \Sigma$:

$$\phi, \psi := p \mid \phi \vee \psi \mid \phi \wedge \psi \mid \phi \to \psi \mid \bot$$

In addition we use $\neg\phi$ as an abbreviation for $\phi \to \bot$.

Natural deduction system for intuitionistic propositional logic

We can construct many different deduction systems for the language of propositional logic, yielding (possibly) different logics. We will now define the deduction system of *intuitionistic propositional logic (IPC)*. There exist several equivalent definitions; the one we provide is found in [SU06]. The rules are given in Figure 1.1, and they are applicable for every propositional formula ϕ, ψ, χ. A *judgment* consists of a pair, one part being a finite set of formulae Γ, the other being a single formula ϕ. We write the judgment as $\Gamma \vdash \phi$, to be read as "Γ proves ϕ". We will use certain conventions when writing the set on the left; instead of writing $\Gamma \cup \{\phi\} \vdash \psi$ we just write $\Gamma, \phi \vdash \psi$, and instead of $\emptyset \vdash \phi$ we write $\vdash \phi$. A *proof* of $\Gamma \vdash \phi$ consists of a finite tree of judgments satisfying certain requirements; the root of the tree must be $\Gamma \vdash \phi$, all the leaves must be empty, and each step must be the application of one of the rules in Figure 1.1. We write these proof-trees with the root at the bottom.

$$\frac{}{\Gamma, \phi \vdash \phi}\ (\mathrm{AX})$$

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}\ (\to\mathrm{I}) \qquad \frac{\Gamma \vdash \phi \qquad \Gamma \vdash \phi \to \psi}{\Gamma \vdash \psi}\ (\to\mathrm{E})$$

$$\frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi}\ (\wedge\mathrm{E1}) \qquad \frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}\ (\wedge\mathrm{I})$$

$$\frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}\ (\wedge\mathrm{E2}) \qquad \frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi}\ (\vee\mathrm{I1})$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \phi}\ (\bot\mathrm{E}) \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}\ (\vee\mathrm{I2})$$

$$\frac{\Gamma, \phi \vdash \chi \qquad \Gamma, \psi \vdash \chi \qquad \Gamma \vdash \phi \vee \psi}{\Gamma \vdash \chi}\ (\vee\mathrm{E})$$

Figure 1.1: Natural deduction system NJ for intuitionistic propositonal logic.

Definition 1.1 (IPC)**.** The logic IPC consists of all propositional formulae ϕ such that $\vdash \phi$, the so-called *validities* of IPC. We sometimes write $\vdash_{IPC} \phi$ instead of $\vdash \phi$.

As an example of a proof, we show $\vdash_{IPC} (p \to r) \to (p \to (q \to r))$:

$$\cfrac{\cfrac{\cfrac{\cfrac{(\mathrm{AX})\ \overline{} \qquad (\mathrm{AX})\ \overline{}}{(\to\mathrm{E})\ p \to r, p, q \vdash r}}{(\to\mathrm{I})\ p \to r, p \vdash q \to r}}{(\to\mathrm{I})\ p \to r \vdash p \to (q \to r)}}{\vdash (p \to r) \to (p \to (q \to r))}$$

Notice how each of three branches ends in an application of the AX-rule, this is the only rule giving an empty leaf.

Kripke semantics for intuitionistic propositional logic

We now present a semantic for intuitionistic propositional logic. This gives a completely different way of defining a set of propositional formulae, but we will see that it actually corresponds with the set of deducible formulae defined above.

A Kripke model for the propositional language over the alphabet Σ consists of a triple $\langle W, \leq, V \rangle$ where W is a set of states (sometimes called "days"), $\leq$ is a

preorder (i.e., a reflexive and transitive binary relation) over W, and $V : W \to \mathcal{P}(\Sigma)$ is an assignment from states to the set of propositional letters holding in that state. We demand monotonicity of V; for every $w, w' \in W$ and $p \in \Sigma$, if $p \in V(w)$ and $w \leq w'$ then $p \in V(w')$.

We then define when a propositional formula is *true* at a point in a Kripke model $M = \langle W, \leq, V \rangle$ by induction on the shape of the formula:

Definition 1.2 (Truth of propositional formulae in a pointed Kripke model)**.**

$$
\begin{aligned}
&M, w \models p \text{ iff } p \in V(w)\\
&M, w \models \bot \text{ never holds}\\
&M, w \models \phi \vee \psi \text{ iff } M, w \models \phi \text{ or } M, w \models \psi\\
&M, w \models \phi \wedge \psi \text{ iff } M, w \models \phi \text{ and } M, w \models \psi\\
&M, w \models \phi \to \psi \text{ iff } \forall u \geq w, \text{ if } M, u \models \phi \text{ then } M, u \models \psi
\end{aligned}
$$

A propositional formula ϕ is said to hold in a model $M = \langle W, \leq, V \rangle$, written $M \models \phi$ when $M, w \models \phi$ for all $w \in W$, and we write $\models \phi$ if $M \models \phi$ for all Kripke models. The set of all propositional formulae ϕ such that $\models \phi$ forms a logic, and—amazingly—it is exactly the logic IPC.

Notice how the truth-condition for $\phi \to \psi$ is different from the classical condition; the implication must not only hold in the current state, but in all following states as well.

Theorem 1.1 (Soundness and completeness of propositional Kripke Semantics)**.** *For all propositional formulae ϕ we have $\models \phi$ if and only if $\vdash_{IPC} \phi$.*

This way of seeing the same logic from two different viewpoints can be quite useful, and in this thesis we use a similar correspondence to show unprovability; if we can provide a Kripke model where the formula ψ is not true, then the above theorem says that ψ cannot be provable in IPC.

As an example, we provide a very simple model $M = \langle W, \leq, V \rangle$ demonstrating the non-provability of $p \vee \neg p$, shown in Figure 1.2. The nodes in the picture are the states in W, with their names written inside the node, and the arrow from w to u means that we have $w \leq u$. Next to the state we have the set of propositional letters true in that state (the output of V on the state).

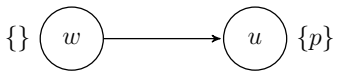


Figure 1.2: Kripke countermodel falsifying the law of excluded middle.

Evaluating $M, w \models p \vee \neg p$ we see see that it holds when $M, w \models p$ or $M, w \models \neg p$. $M, w \models p$ certainly does not hold, since $p \notin V(w)$. But we also don't have

$M, w \models \neg p$ since this is the same as $M, w \models p \to \bot$, which would require $\forall u \geq w$, if $M, u \models p$ then $M, u \models \bot$. But we have $u \geq w$ with $M, u \models p$ but not $M, u \models \bot$ (since no state satisfies $\bot$).

One way of thinking about Kripke models is as representing possible states, where there is an arrow from one state to another, if the latter state is seen as possible from the first. For something to be provable it needs to hold in *all* possible states. The monotonicity of V can be understood as capturing the intuition that we can only regard a state as possible if it satisfies the basic things which we know are true in the current state.

Classical propositional logic

As seen above, $p \vee \neg p$ is not deducible in IPC. In light of the correspondence between provability in IPC and the Kripke semantics for IPC, this can be viewed in two different ways. One way is that the rules given in Definition 1.1 are too weak to deduce $p \vee \neg p$ without further assumptions, and the other way is that we are so liberal with what we count as a model that we also have models in which $p \vee \neg p$ is false. If we want, we can add the axiom scheme $\phi \vee \neg\phi$ to IPC, giving classical propositional logic (CPC).

Definition 1.3 (CPC)**.** The logic CPC consists of the propositional formulae which are provable using the rules in Figure 1.1 with the additional rule:

$$\frac{}{\Gamma \vdash \phi \vee \neg\phi}\,(\text{LEM})$$

The law of excluded middle provides us with a new way to construct empty leaves to "close off" our proof trees. Adding the law of excluded middle has two effects; more formulae are provable, and the logic has fewer models.

An example of a formula which is provable with the law of excluded middle is $\neg\neg\phi \to \phi$. Remember that $\neg\phi$ is shorthand for $\phi \to \bot$, and for sake of brevity we have left out the label AX on the three leaves where it is used.

$$\dfrac{\dfrac{}{\neg\neg\phi, \phi \vdash \phi}\quad \dfrac{\dfrac{\dfrac{}{\neg\neg\phi, \neg\phi \vdash \neg\phi}\quad \dfrac{}{\neg\neg\phi, \neg\phi \vdash \neg\neg\phi}}{\neg\neg\phi, \phi \to \bot \vdash \bot}(\to\text{E})}{\neg\neg\phi, \phi \to \bot \vdash \phi}(\bot\text{E})\quad \dfrac{}{\neg\neg\phi \vdash \phi \vee \neg\phi}(\text{LEM})}{\neg\neg\phi \vdash \phi}(\vee\text{E})}{\vdash \neg\neg\phi \to \phi}(\to\text{I})$$

CPC is not sound and complete with respect to Kripke models, but it *is* sound and complete with respect to 1-point Kripke models. When thinking about Kripke models as representing possible states, this is not surprising; the law of excluded middle lets us remove any ambiguity regarding the state of the world, so there are no other possible states.

1.2.2 First-order logic

Propositional logic is interesting, but for certain use its expressive power is too limited. For example, when attempting to capture the reasoning of Aristotle's syllogism, the lack of quantification ("All" and "Exists") becomes a problem:

All men are mortal.
Socrates is a man.
Therefore, Socrates is mortal.

Intuitionistic first-order logic is an extension of the propositional version: it has a significantly more expressive language which can quantify over variables. This enables us to express more complicated notions, but it does so at the expense of making reasoning more complicated.

The first-order language

First-order languages are defined with respect to a signature—a family of function and relation symbols with a designated arity—and set of variables. All the variables are *terms*, as are expressions of the form $f(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol.

An *atomic formula* is either the logical symbol $\bot$ or an expression of the form $P(t_1, \ldots, t_n)$, where P is an n-ary relation symbol and $t_1, \ldots, t_n$ are terms.

The first-order formulae are given by the following grammar, where Q is an atomic formula and a is a variable:

$$\phi, \psi := Q \mid \phi \vee \psi \mid \phi \wedge \psi \mid \phi \to \psi \mid \forall a\phi \mid \exists a\phi \mid \bot$$

Natural deduction system for intuitionistic first-order logic

To extend the natural deduction system for intuitionistic propositional logic such that it also handles the first-order formulae, one simply needs to add four new rules, given in Figure 1.3. To understand the rules, we need to introduce a bit of notation. A variable x is *free* in a formula if there is no occurrence of $\exists x$ or $\forall x$ which ranges over it, and $FV(\Gamma)$ collects all variables occurring freely in at least one of the formulae in Γ. $\phi[a := t]$ means ϕ with all occurrences of a replaced with t.

We define a logic of the deducible formulae.

Definition 1.4 (IQC)**.** If we can prove a first-order formula $\vdash \phi$ using the rules in Figure 1.1 and Figure 1.3 we write $\vdash_{IQC} \phi$. The logic IQC consists of all first-order formulae ϕ such that $\vdash_{IQC} \phi$, and we then say that ϕ is valid in the logic IQC.

$$\frac{\Gamma \vdash \phi[a := t]}{\Gamma \vdash \exists a\phi}\,(\exists\mathrm{I}) \qquad\qquad \frac{\Gamma \vdash \forall a\phi}{\Gamma \vdash \phi[a := t]}\,(\forall\mathrm{E})$$

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \forall a\phi}\,(\forall\mathrm{I}),\ \ (\mathrm{a} \notin FV(\Gamma))$$

$$\frac{\Gamma \vdash \exists a\phi \qquad \Gamma, \phi \vdash \psi}{\Gamma \vdash \psi}\,(\exists\mathrm{E}),\ \ (\mathrm{a} \notin FV(\Gamma, \psi))$$

Figure 1.3: Natural deduction system NJ for intuitionistic first-order logic.

Kripke semantics for intuitionistic first-order logic

Kripke semantics for intuitionistic first-order logic is based on Kripke semantics for intuitionistic propositional logic, but it is somewhat more complex.[1] A Kripke model for a first-order language Σ is a triple $M = \langle W, \leq, \{N_w\}_{w \in W} \rangle$ where W is a set of states/days, $\leq$ is a preorder over W and $\{N_w\}_{w \in W}$ is a family of classical Σ-structures satisfying a number of requirements. Being a classical Σ-structure means that each N_w must contain a domain and give an interpretation to each of the relations symbols and functions in Σ. The additional requirements are that these interpretations are monotone in the following sense: For all $u, v \in W$, if $v \leq u$ then

- the domain of N_v must be a subset of the domain of N_u,
- the interpretation of function symbols in N_v and N_u must agree on the domain of N_v, and
- the interpretation of relations symbols in N_v and N_u must agree on the domain of N_v.

We can now evaluate a first-order formula in a point w in a Kripke model $M = \langle W, \leq, \{N_w\}_{w \in W} \}$ under a evaluation e, mapping variables to elements the domain of w.

[1] For those familiar with classical semantics of first-order logic, Kripke semantics is a pre-order of classical models with a similar monotonicity requirement as the one for the propositional case.

Definition 1.5 (Truth of first-order formulae in a pointed Kripke model)**.**

$M, w \models P(t_1, \ldots, t_n)[e]$ iff $P(t_1, \ldots, t_n)[e]$ holds in N_w
$M, w \models \bot$ never holds
$M, w \models (\phi \vee \psi)[e]$ iff $M, w \models \phi[e]$ or $M, w \models \psi[e]$
$M, w \models (\phi \wedge \psi)[e]$ iff $M, w \models \phi[e]$ and $M, w \models \psi[e]$
$M, w \models (\phi \to \psi)[e]$ iff $\forall u \geq w$, if $M, u \models \phi[e]$ then $M, u \models \psi[e]$
$M, w \models (\exists x \phi)[e]$ iff there exists an $a \in N_w$ such that: $M, w \models \phi[e(x \mapsto a)]$
$M, w \models (\forall x \phi)[e]$ iff for every $u \geq w$ and every $a \in N_u : M, u \models \phi[e(x \mapsto a)]$

Again we write $M \models \phi$ for a Kripke model $M = \langle W, \leq, \{N_w\}_{w \in W} \rangle$ if $M, w \models \phi$ for all $w \in W$, and $\models \phi$ if $M \models \phi$ for all Kripke models M.

In addition to implication, it is particularly the truth-conditions for $\forall x \phi$ which differs in an interesting way from the classical semantics: for $\forall x \phi$ to hold, it is not enough that every element in the present state satisfies ϕ—it must hold in all future states as well, even for elements not yet introduced. Thinking about Kripke semantics in terms of possible worlds, this means that $\forall x \phi$ must hold not only for elements we know about, but for all elements we see as possible.

As with the propositional case, we have a beautiful correspondence between deduction system and semantics, shown in [Kri65], which we can use to show the unprovability of first-order formulae in quantified intuitionistic logic.

Theorem 1.2 (Soundness and completeness of first-order Kripke Semantics)**.** *For all first-order formulae ϕ, we have $\models \phi$ if and only if $\vdash_{IQC} \phi$.*

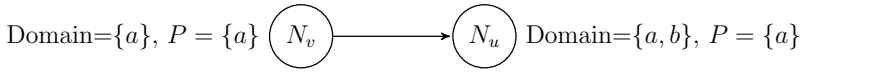


Figure 1.4: Kripke countermodel falsifying $\neg \forall x P(x) \to \exists x \neg P(x)$.

We will use this correspondence to show the unprovability of the classic tautology $\neg \forall x P(x) \to \exists x \neg P(x)$ in IQC by means of a Kripke countermodel. The model, shown in Figure 1.4, goes over two days, u and v, with $v \leq u$. The domain of N_v is $\{a\}$, and the interpretation of P in N_v is $\{a\}$. The domain of N_u is $\{a, b\}$, and the interpretation of P in N_u is also $\{a\}$, crucially *not* containing b. Expanding the definition of truth, we see that $M, v \models \neg \forall x P(x)$ is the same as $M, v \models \forall x P(x) \to \bot$; and this holds, since we have both $M, v \not\models \forall x P(x)$ and $M, u \not\models \forall x P(x)$. The latter follows directly from the fact that we have an element b in the domain of N_u such that we don't have it in the interpretation of P in N_u. We also have $M, v \not\models \forall x P(x)$, since $M, v \models \forall x P(x)$ would require $\forall x P(x)$ to hold

in all following states—including u—but as we have shown, this is not the case. But we don't have $M, v \models \exists x \neg P(x)$, since all elements of N_v are in the interpretation of P. Thus, we have a model M such that $M \not\models \neg\forall x P(x) \to \exists x \neg P(x)$, giving $\not\models \neg\forall x P(x) \to \exists x \neg P(x)$, which again gives $\not\vdash_{IQC} \neg\forall x P(x) \to \exists x \neg P(x)$ by soundness.

Classical first-order logic

Adding the law of excluded middle to IQC, in the same way as we did to IPC, we get classical first-order logic.

Definition 1.6 (FOL)**.** The logic FOL consists of all first-order formulae which are provable using the rules in Figure 1.3, with the additional rule:

$$\frac{}{\Gamma \vdash \phi \vee \neg\phi}\ (\text{LEM})$$

Adding the law of excluded middle restricts us in certain ways. With IPC, we can include assumptions which are incompatible with the law of excluded middle. We could, for example, add that all functions are computable and see what would follow from this, essentially investigating the models where this is satisfied. If we try adding the same assumption to classical logic however we introduce a contradiction.

We also have—as in the propositional case—that the addition of the law of excluded middle means that IPC is no longer sound with respect to Kripke models. But—as earlier—IPC is sound and complete with respect to 1-point Kripke models, also called Tarski semantics.

1.2.3 The BHK interpretation

Intuitionistic logic can be understood through the *Brouwer–Heyting–Kolmogorov* interpretation of what a proof consists of, explaining proofs as algorithms. This gives a third way to read the formulae, in addition to the formal "explanation" given by semantic and deduction systems.

- A proof of $\phi \vee \psi$ consists of a pair $\langle k, p \rangle$ where either k is 0 and p is a proof of ϕ, or k is 1 and p is a proof of ψ.
- A proof of $\phi \wedge \psi$ consists of a pair $\langle a, b \rangle$ such that a is a proof of ϕ and b is a proof of ψ.
- A proof of $\phi \to \psi$ consists of an algorithm transforming any proof of ϕ to a proof of ψ.

- $\bot$, has no proof.
- A proof of $\neg\phi$ is a proof of $\phi \to \bot$, i.e., a transformation from a hypothetical proof of ϕ into a proof of $\bot$.
- A proof of $\forall x, \phi(x)$ consists of an algorithm transforming any $a \in S$ into a proof of $\phi(a)$.
- A proof of $\exists x, \phi(x)$ consists of a pair $\langle a, b\rangle$ such that $a \in S$ and b is a proof of $\phi(a)$.

Note that the BHK interpretation does not give a precise notion of what constructive logic is. It does not, for example, define what it means to be an "algorithm", and it does not account for what constitutes a proof of a propositional letter or a predicate applied to terms. Several different constructive systems adhere to the BKH interpretation to different degrees and in different ways: we will talk about a couple of these in the next section.

The BHK interpretation provides us with a new, computational way to read formulae. Let us look at, for instance, the following intuitionistic provable formula

$$(p \wedge z) \to (q \to p)$$

The BHK reading of this is that, given a proof of $p \wedge z$—let's call such a proof "proof 1"—we can transform that proof into one of $q \to p$. A proof of $q \to p$ is just a transformation which, given a proof of q, returns a proof of p. A proof of p can be found by turning our attention to proof 1, which, according to the BHK interpretation, consists of a pair $\langle a, b\rangle$ where a is a proof of p and b is a proof of z. The "transformation" which ignores q but returns a is a proof of $q \to p$; so we have shown $(p \wedge z) \to (q \to p)$.

Remember that $\neg\forall x P(x) \to \exists x \neg P(x)$ is not provable in intuitionistic logic. When given a BHK reading, this seems reasonable; the antecedent gives a procedure which transforms $\forall x P(x)$ into $\bot$, but you will be hard-pressed to construct—from this procedure alone—a concrete element a such that $\neg P(a)$.

Similarly for $\neg\neg p \to p$, which is the same as $((p \to \bot) \to \bot) \to p$; a way of transforming $(p \to \bot)$ into $\bot$, does not provide a way to construct a proof of p.

1.3 Constructive mathematics

This thesis concerns itself with *constructive proofs*, contrary to non-constructive proofs. The major difference lies in the interpretation of disjunction and existence: constructive proofs of an existential contain a way of constructing or computing the witness, and constructive proofs of a disjunction tell us which of the disjuncts

hold. As classical mathematics can be built on top of classical logic, constructive mathematics can be built on a constructive logic. We have seen one example of a constructive logic already, the logic of intuitionistic first-order logic IQC. Although the two words "intuitionistic" and "constructive" have had different meanings historically, in this thesis, we will use them synonymously.

There is not only *one* formal system in which all of constructive mathematics is done, there are several different systems with different properties. Two important ones are Intuitionistic Zermelo-Fraenkel set theory (IZF) and Martin-Löf Type Theory (MLTT) [ML75]. IZF is set theory built on top of IQC—as defined above—and it is one of the stronger constructive systems. MLTT is a relatively modern system which has gained much attention in the last decades, and has a semantics quite close to the BHK interpretation of formulae. A proper introduction to MLTT is outside of the scope of this section (see [Cro15], [Coq15] and [BP13, Section 3.4] for good introductions), but some points are worth mentioning. MLTT takes the computational reading of constructive mathematics to its heart, and proofs are *actual* programs. These programs/proofs are first class citizens of the system, and proofs of implications are actual programs which take a proof (program) of the assumption as an argument and transform it into a proof of the antecedent. The prominent proof assistant Coq [CDT12] is based on MLTT and we make extensive use of Coq in this thesis. See [TVD88] for an overview of constructive mathematics in general.

To avoid getting too bogged down in the details of a particular constructive framework, it is not uncommon to present constructive proofs in a rigorous but informal matter consistent with the BHK interpretation, in such a way that it can readily be expressed in several of the constructive frameworks. We follow this style in this thesis, but most of the results are either additionally formalized in Coq, or accompanied by a sketch of how to formalize them in a formal system.

To illustrate the differences between constructive and non-constructive proofs we will provide two examples. The first example is used so extensively that it can almost be regarded as compulsory in any introduction text on constructive mathematics. It probably appeared in print for the first time in [Jar53].

Lemma 1.7. *There exist irrational numbers a and b such that a^b is rational.*

Non-constructive proof: $\sqrt{2}^{\sqrt{2}}$ is either rational or irrational. If it is rational, our statement is proved. If it is irrational, $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}*\sqrt{2}} = \sqrt{2}^2 = 2$ proves our statement. □

Constructive proof. Let $a = \sqrt{2}$, $b = \log_2 9$. Using well-known logarithmic identities we can conduct the following line of reasoning:

$$\sqrt{2}^{\log_2 9} = 2^{\log_2(\sqrt{2})\times\log_2 9} = 2^{\log_2(\log_2 9\times\sqrt{2})} = 9^{\log_2\sqrt{2}} = 9^{\log_2(2^{0.5})} = 9^{0.5} = \sqrt{9} = 3$$

□

Notice the difference between these two proofs. The first shows the truth of the statement, but does not provide us with actual witnesses for a and b. It could be that $a = b = \sqrt{2}$ or it could be that $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$; the proof gives us no information in this regard. It is the use of the law of excluded middle which lets us divide the proof into two cases, depending on whether $\sqrt{2}^{\sqrt{2}}$ is rational or irrational.

The constructive proof consists of actual witnesses, and then simple high-school arithmetic shows that they satisfy the desired properties. To complete the proof we need to show constructively that $\sqrt{2}$ and $\log_2 9$ are irrational numbers. But the standard proof that $\sqrt{2}$ is irrational is constructive, and the irrationality of $\log_2 9$ is also constructively deducible.[2]

The second example is a bit more involved, but highlights the very computational nature of constructive proofs. This example uses notions which we will discuss in greater detail in Chapter 2. A *stream* over Boolean is a function from natural numbers into the values $\{\text{True}, \text{False}\}$, and we define two properties of streams: *eventually always false* and *bounded*. A stream f is eventually always false when we have an index i such that we can guarantee that every index above i has the value False:

$$\exists i : \mathbb{N}, \forall m : \mathbb{N}, ((m \geq i) \to (f(m) = \text{False})).$$

A stream f is bounded by n when we know that there are no more than n true positions in the stream.

$$\exists n : \mathbb{N}, \forall k : \mathbb{N}, \#\text{True}_f(k) \leq n.$$

$\#\text{True}_f(k)$ is a function which counts the number of true positions in f from position 0 up to k.

Both constructively and classically, we have that any stream that is eventually always false is bounded. Constructively, we can see this by constructing a terminating algorithm, a procedure, which converts a proof of a stream being eventually always false into a proof that the same stream is bounded. We are given an eventually always false stream f, and the constructive reading of this is that we are then given a stream f and an $i : \mathbb{N}$ with the guarantee that $\forall m : \mathbb{N}, (m \geq i) \to (f(m) = \text{False})$. Our goal is to provide an $n : \mathbb{N}$, together with a guarantee that $\forall k : \mathbb{N}, \#\text{True}_f(k) \leq n$. We simply let n be the i we received

[2] If $\log_2 9$ were representable as a rational number $\frac{p}{q}$ then $2^{\log_2 9} = 9$ iff $2^{\frac{p}{q}} = 9$, iff $(2^{\frac{p}{q}})^q = 2^p = 9^q$, but $9^q = 2^p$ is impossible since $9 = 3^2$, and 3^n is always odd while 2^m is always even.

as an argument. There cannot be any more true positions than i, since we are guaranteed that all values after i are false.

The interesting question is whether we can also go the other way. Can we show that all bounded streams are eventually always false? Classically, we can: assuming a bounded stream f, we can show that it is impossible that f is not eventually always false. Since $\neg\neg\phi \to \phi$ holds classically (but not constructively), this shows that f is eventually always false.

But can we provide a *constructive proof*, a terminating algorithm? This seems difficult; we are given a bound k on the number of true positions, and we can start counting true positions from 0, but when do we stop? Since the bound is not strict, there is no way of knowing—when we have found some number of true positions—whether there will be another position with True. We cannot guarantee that all following positions will be false until we have found the k^{th} true position, but we are not guaranteed that there even is such a position.

More specifically, if we had an algorithm implementing this implication we could solve the halting problem. The halting problem asks whether one can decide, for *every* possible program, whether it will halt, and this was shown to be impossible by Alan Turing [Tur36]. Given a program we make a stream which is True on position n if the program halts after being run for *exactly* n steps, and False otherwise. This stream clearly has a bounded number of true positions: either 0 or 1. But by getting a position after which every position after is false we can check in finite time whether the whole stream is false or not, determining whether the program halts or not.

To show that the implication is not provable in a specific constructive system, one can provide a falsifying Kripke model as in Section 1.2.2—which would show that the implication is not provable in IQC—or show that if we had the implication, then we would get some consequence which we already know are not constructively provable, as done in [BNU12].

These two notions are the same classically, but algorithmically they are not. Constructive mathematics allows one to work in a framework where these algorithmically nonequivalent notions remain nonequivalent. One can of course investigate these notions algorithmically using classical logic by carefully refraining from doing anything non-computational, but constructive mathematics provides a framework where this comes naturally. As an added bonus, most constructive frameworks provide a way to extract algorithms directly from proofs.

In [Ric90], Richman defends the view that constructive mathematics should be seen as a generalization of classical mathematics which "accommodates both classical and computational models", and not as something inferior. Richman separates between what he calls "Type 1" and "Type 2" theories. The first—exemplified by arithmetic—has *one* intended model; the axiomatization is an

attempt at characterizing this one model and the inability[3] to do so perfectly is seen as a defect of the formalization. In the second type of axiomatic theory—group theory being a prominent example—the power lies *exactly* in the number of different models. Constructive set theory has more models than classical set theory; all that is provable constructively (without further assumptions) hold classically as well. When viewing set theory as a Type 2 theory, this is a virtue.

Constructive mathematics not only allows for more models than classical mathematics; it also allows one to pick axioms which directly contradict classical mathematics, but which hold in other models. In this way, one can "zoom" into these models—e.g., models where all functions are computable—and investigate them. But it is a mistake to think that constructive mathematics is *about* these models—it merely allows them to exist and for us to investigate them. This axiomatic freedom makes constructive mathematics go hand-in-hand with *reverse mathematics*, the program of exactly establishing which axioms are needed to prove theorems of mathematics.

I want to mention a property of constructive mathematics which is neglected far too often: constructive mathematics is both beautiful and fun. Restraining from using the law of excluded middle forces one to work in a radically different way, but in a way which feels quite natural to a computer scientist. The proofs can end up taking unexpected turns—when seen from a classical perspective—but they are often beautiful and hold in more models.

1.4 Overview of the thesis

In this thesis, we investigate aspects of *finite sets* and *simplicial sets* from a constructive perspective. In this chapter we have already looked at two notions which can be used to define finiteness: *bounded* gives us one way to characterize a set of true positions as finite, while *eventually always false* gives us another. Chapter 2 investigates a third notion, *strictly bounded sets*, and how this notion relates to the two previous ones. We find that it falls nicely between them; it is strictly stronger than *bounded* but strictly weaker than *eventually always false*, and we discover exactly which assumptions are needed to make them equivalent. All of these results are formalized in Coq.

In Chapter 3, we continue our exploration of constructive finiteness and examine *streamless* sets. Contrary to the previous notions of finiteness, streamlessness does not presume the set to have (decidable) equality, nor to be a subset of an enumerable set. One possible use case for this is finite sets of real numbers; the equality of real numbers is undecidable. We investigate whether streamless sets

[3]Gödel showed in 1931 that any consistent axiomatizations of arithmetic must be incomplete.

are closed under Cartesian products: if A and B are streamless sets, is $A \times B$ necessarily streamless? We also see how the assumption of function extensionality gives streamless sets decidable equality in certain constructive frameworks. These results have appeared in [Par15].

Another notion of finiteness is *Noetherian*. In Chapter 4, we generalize the notion of streamless and Noetherian to binary relations; streamless and Noetherian sets are then sets where equality is a streamless or Noetherian relation, respectively. We provide two proofs that Noetherian relations are streamless, where one of these proofs can be expressed in a type system without inductively defined equality.

In Chapters 5 and 6, we move on to *simplicial sets*; in particular, those satisfying the *Kan condition*. This notion, coming from topology, has piqued the interest of the type theory community, as Kan simplicial sets can be used to build models of Martin-Löf type theory that validate the Univalence Axiom.

The following theorem holds classically: if Y and X are Kan simplicial sets, then Y^X is also a Kan simplicial set. This theorem is important for the Kan simplicial set model of type theory. We investigate the non-constructivity of this theorem for several interpretations of what it means to be Kan in a constructive setting, and provide Kripke counter models showing that the result is not constructively provable for any of the interpretations. This has consequences for homotopy type theory, had it been constructively provable, it could have led the way to a computational interpretation of the Univalence Axiom. Some of these results have appeared as [BCP15], and others are under review.

Part II

Scientific results

Strictly Bounded Sets

Erik Parmann

In this report we investigate the relationship between several kinds of finiteness. In particular, we are interested in the finiteness of decidable subsets of natural numbers. We start by recalling two definitions given by Bezem et al. [BNU12], and we formulate a natural version of finiteness lying strictly between the two. All results are formulated and verified in the Coq proof assistant.

2.1 Introduction

Constructive logic can prove fewer logical equivalences than classical logic; or, alternatively, it allows for more distinctions. This also holds with respect to finiteness; there are several notions of finiteness which are classically equivalent, but which can be separated constructively. This report looks at a few such notions.

We look at some ways to constructively define the set of true positions in a Boolean stream as finite. Since Boolean streams can be seen as representing decidable sets of natural numbers, we can also be seen as defining decidable sets of natural numbers as finite. In particular, the notions of finiteness we investigate here are applicable to sets which:

- have an enumerable superset with decidable equality; and
- have decidable membership.

It should be noted that the present report has significance for any such set, not only for decidable subsets of the natural numbers.

We extend work done by Bezem, Nakata and Uustalu [BNU12] by identifying a new formalization which lies between two known ones, and we show reductions "downwards" and "upwards" in the hierarchy. We show that the two upward reductions are equivalent to *Markov's principle* and the *weak limited principle of omniscience*, respectively. Both of these principles are constructively consistent weak forms of the law of excluded middle, but not constructively provable, showing that the different notions of finiteness form a proper, non-collapsing hierarchy constructively. All new results in this report have been formalized and verified using the Coq [CDT12] proof assistant.

We start in Section 2.2 by introducing the needed machinery and the two preexisting notions of finiteness. In Section 2.3, we introduce the new notion of finiteness and place it in the hierarchy, before we sum up in Section 2.4.

2.2 Preliminaries

We start by providing the basic definitions. A stream over A is a function of type $\mathbb{N} \to A$. For a stream $f : \mathbb{N} \to \text{bool}$, we define the *complement stream* $f^- : \mathbb{N} \to \text{bool}$ as $f^-(x) = \neg f(x)$. For a stream $f : \mathbb{N} \to \text{bool}$, $\#\text{True}_f : \mathbb{N} \to \mathbb{N}$ is the function which on input k returns the number of natural numbers i such that $i \leq k$ and $f(i) = \text{True}$; that is,

$$\#\text{True}_f(k) = |\{i \mid i \leq k \wedge f(i) = \text{True}\}|.$$

Given a stream $g : \mathbb{N} \to \text{bool}$ we construct the stream TrueOnFirst_g, which is true on at most the first index where g is true and false everywhere else:

$$\text{TrueOnFirst}_g(n) = \begin{cases} g(n) & \forall i < n, g(i) = \text{False} \\ \text{False} & \text{otherwise} \end{cases}$$

The condition $\forall i < n, g(i) = \text{False}$ is constructively checkable, since it is a bounded search down from n. It is easy to both see and show that:

$$\forall n : \mathbb{N}, \#\text{True}_{\text{TrueOnFirst}_g}(n) \leq 1$$

The following three principles are all constructively consistent, but not provable. *Markov's principle (MP)* states that, for any decidable predicate P over natural numbers, if it is impossible that there is no such number satisfying P, then there exists a natural number satisfying P.

Definition 2.1 (Markov's principle, MP)**.**

$$(\forall n : \mathbb{N}, P(n) \vee \neg P(n)) \to (\neg\neg(\exists n : \mathbb{N}, P(n)) \to \exists n : \mathbb{N}, P(n))$$

Intuitively, Markov's principle is realized by an unbounded search, which we can convince ourself (from outside the system) will terminate since it is impossible that it will have to run forever.

The weak limited principle of omniscience (WLPO) states that any decidable predicate P is everywhere true or not everywhere true.

Definition 2.2 (Weak limited principle of omniscience, WLPO)**.**

$$(\forall n : \mathbb{N}, P(n) \vee \neg P(n)) \to ((\forall n : \mathbb{N}, P(n)) \vee \neg\forall n : \mathbb{N}, P(n))$$

The even stronger limited principle of omniscience (LPO) states that every decidable predicate is either everywhere true, or we have an $n : \mathbb{N}$ falsifying P.

Definition 2.3 (Limited principle of omniscience, LPO)**.**

$$(\forall n : \mathbb{N}, P(n) \vee \neg P(n)) \to ((\forall n : \mathbb{N}, P(n)) \vee \exists n : \mathbb{N}, \neg P(n))$$

It is rather easy to see that

$$(\text{MP} \wedge \text{WLPO}) \Rightarrow \text{LPO},$$

and in fact we even have the stronger equivalence

$$(\text{MP} \wedge \text{WLPO}) \iff \text{LPO}.$$

Since the predicate P in WLPO, MP, and LPO is a decidable predicate over $\mathbb{N}$, the above formulations have equivalent formulations for Boolean streams. For example, Markov's principle states that, for any Boolean stream f, we have

$$\neg\neg(\exists n : \mathbb{N}, f(n) = \text{True}) \to \exists n : \mathbb{N}, f(n) = \text{True}\,.$$

We will freely switch between the two formulations, depending on which is easier to work with.

We now provide two properties that can hold of Boolean streams, both given in [BNU12]. A function $f : \mathbb{N} \to \text{bool}$ is *eventually always false*, written $\text{eaf}(f)$, when there is some index such that every index above it is false:

Definition 2.4 (Eventually always false ($\text{eaf}(f)$))**.**

$$\exists n : \mathbb{N}, \forall m : \mathbb{N}, m \geq n \to f(m) = \text{False}\,.$$

Next, we say that a stream $f : \mathbb{N} \to \text{bool}$ is *bounded*, written $\text{bounded}(f)$, when there is a bound to the number of true positions:

Definition 2.5 (Bounded ($\text{bounded}(f)$))**.**

$$\exists n : \mathbb{N}, \forall k : \mathbb{N}, \#\text{True}_f(k) \leq n.$$

Bezem et al. refer to $\text{bounded}(f)$ as both "Equation (2)" and "$\exists n.\, \text{len}_n\, s$", and to $\text{eaf}(f)$ as both "Equation (1)" and "$\mathcal{F}(\mathcal{G}\ \text{blue})s$".

2.3 Strictly bounded

In this section we introduce a natural strengthening of bounded, expressing that we not only have a *bound* on the number of true positions, but a *strict* bound. Formally we say that a stream $f : \mathbb{N} \to \text{bool}$ is *strictly bounded*, written $\text{sb}(f)$, when:

Definition 2.6 (Strictly bounded ($\text{sb}(f)$))**.**

$$\exists n : \mathbb{N}, (\forall k : \mathbb{N}, \#\text{True}_f(k) \leq n \wedge \neg(\forall k : \mathbb{N}, \#\text{True}_f(k) < n)).$$

An equivalent formulation to $\text{sb}(f)$, easily provable equivalent by induction, is the following.

$$\exists n{:}\mathbb{N}, (\forall k{:}\mathbb{N}, \#\text{True}_f(k) \leq n \wedge (\forall m : \mathbb{N}, m < n \to (\neg\forall k : \mathbb{N}, \#\text{True}_f(k) \leq m))).$$

We will switch freely between the two formulations. In the following, we will investigate how strictly bounded relates to both bounded and eventually always false.

It is clear that $\mathrm{sb}(f) \Rightarrow \mathrm{bounded}(f)$, and $\mathrm{eaf}(f) \Rightarrow \mathrm{sb}(f)$ is also easily proven, as the bound on the index in $\mathrm{eaf}(f)$ gives us a bound where all the true values must reside, letting us find the exact number of them in finite time. Furthermore, Bezem et al. show that $(\forall f : \mathbb{N} \to \mathrm{bool}, \mathrm{bounded}(f) \Rightarrow \mathrm{eaf}(f)) \iff \mathrm{LPO}$. LPO is consistent but not provable constructively, so the same holds for $(\forall f : \mathbb{N} \to \mathrm{bool}, \mathrm{bounded}(f) \Rightarrow \mathrm{eaf}(f))$.

The main result in this report is that the new notion of strictly bounded falls neatly between the two previous notions, completing the picture. The relationships are summed up in Figure 2.1, where the solid lines are the new results in this report and the dashed lines were shown in [BNU12]. As neither Markov's principle, WLPO nor LPO hold constructively, we get a strict hierarchy where eaf $\Rightarrow$ sb and sb $\Rightarrow$ bounded holds constructively, but none of the other directions hold without further assumptions.

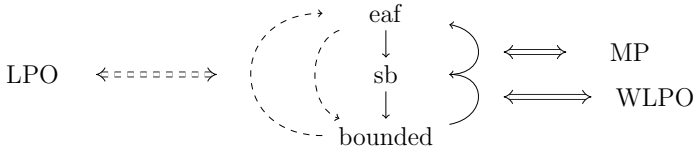


Figure 2.1: The relationship between different properties of Boolean streams. Dashed lines were known; solid lines are new results.

We start by showing the equivalence

$$(\forall f : \mathbb{N} \to \mathrm{bool}, \mathrm{sb}(f) \Rightarrow \mathrm{eaf}(f)) \iff \mathrm{MP}.$$

Before we give the formal proof, however, we provide some intuition. For the direction right to left we must be able to get—from a *strict bound* on the number of true elements—the concrete *index* of the last true element. With a strict bound, we are able to show that it is impossible that there is no index which is the index of the last true position, which combined with Markov's principle gives us the concrete index of the last true position.

For the left to right direction, we get the antecedent of Markov's principle—a proof that $\neg\neg(\exists n : \mathbb{N}, g(n) = \mathrm{True})$ for a stream $g : \mathbb{N} \to \mathrm{bool}$—and we must provide an $n : \mathbb{N}$ such that $g(n) = \mathrm{True}$. We use the stream $\mathrm{TrueOnFirst}_g$, which is true on the first true position of g (if this exists), and is false everywhere else. From the assumption $\neg\neg(\exists n : \mathbb{N}, g(n) = \mathrm{True})$, we are able to show that $\mathrm{TrueOnFirst}_g$ has more than 0, so *exactly* 1, true position, giving a strict bound on the number of true positions. Since this, by assumption, implies that $\mathrm{TrueOnFirst}_g$ is eventually always false, we are able to find the concrete index of the first true position, which is an $n : \mathbb{N}$ such that $g(n) = \mathrm{True}$.

Theorem 2.1. $(\forall f : \mathbb{N} \to \text{bool}, \text{sb}(f) \Rightarrow \text{eaf}(f)) \iff MP$.

Proof. *(⇐):* We assume Markov's principle, and an arbitrary stream $f : \mathbb{N} \to \text{bool}$ such that we have $\text{sb}(f)$:

$$\exists n : \mathbb{N}, (\forall k : \mathbb{N}, \#\text{True}_f(k) \leq n \wedge (\forall m : \mathbb{N}, m < n \to (\neg\forall k : \mathbb{N}, \#\text{True}_f(k) \leq m))),$$

and we proceed to find the n witnessing the statement

$$\exists n : \mathbb{N}, \forall m : \mathbb{N}, m \geq n \to f(m) = \text{False}\,. \tag{2.1}$$

From the assumption $\text{sb}(f)$ we get a $nr : \mathbb{N}$ such that

$$\forall k : \mathbb{N}, \#\text{True}_f(k) \leq nr \wedge \neg(\forall k : \mathbb{N}, \#\text{True}_f(k) < nr), \tag{2.2}$$

and we proceed to show that there is an index $i : \mathbb{N}$ such that $\#\text{True}_f(i) = nr$. For this we apply Markov's principle, so we need to deduce a contradiction from the assumption

$$\neg(\exists i : \mathbb{N}, \#\text{True}_f(i) = nr).$$

From this assumption, we immediately get $\forall n : \mathbb{N}, \#\text{True}_f(n) \neq nr$. Then, using the left conjunct of assumption 2.2, we get $\forall n : \mathbb{N}, \#\text{True}_f(n) < nr$. But this contradicts the right conjunct of assumption 2.2, giving us

$$\neg\neg(\exists i : \mathbb{N}, \#\text{True}_f(i) = nr),$$

which by Markov's principle gives us

$$\exists i : \mathbb{N}, \#\text{True}_f(i) = nr.$$

We set $i + 1$ as the witness needed for $\text{eaf}(f)$, and we are left showing

$$\forall m : \mathbb{N}, m \geq i + 1 \to f(m) = \text{False}\,. \tag{2.3}$$

This is an easy consequence from the monotonicity of $\#\text{True}_f$ combined with the following property:

$$\forall n : \mathbb{N}, (f(n + 1) = \text{True} \to (\#\text{True}_f(n + 1) = (\#\text{True}_f(n)) + 1))$$

□

(⇒): We assume that we have

$$\forall f : \mathbb{N} \to \text{bool}, \text{sb}(f) \Rightarrow \text{eaf}(f),$$

and that we have

$$\neg\neg(\exists n : \mathbb{N}, g(n) = \text{True})$$

for a Boolean stream g, which is equivalent to $\neg\forall n : \mathbb{N}, g(n) = \text{False}$.

Recall that we have $\forall n : \mathbb{N}, \#\text{True}_{\text{TrueOnFirst}_g}(n) \leq 1$. It is fairly easy to see that from $\neg\forall n : \mathbb{N}, g(n) = \text{False}$, we get

$$\neg\forall n : \mathbb{N}, \#\text{True}_{\text{TrueOnFirst}_g}(n) = 0.$$

This means that we have

$$(\forall k : \mathbb{N}, \#\text{True}_{\text{TrueOnFirst}_g}(k) \leq 1) \wedge (\neg\forall k : \mathbb{N}, \#\text{True}_{\text{TrueOnFirst}_g}(k) < 1),$$

giving $\text{sb}(\text{TrueOnFirst}_g)$ with the witness 1. From the main assumption, we get

$$\exists l : \mathbb{N}, \forall m : \mathbb{N}, m \geq l \to \text{TrueOnFirst}_g\ \ m = \text{False},$$

giving, by a bounded search downwards from l:

$$(\forall m : \mathbb{N}, m \leq l \to \text{TrueOnFirst}_g\, m = \text{False}) \vee (\exists m : \mathbb{N}, \text{TrueOnFirst}_g\, m = \text{True}).$$

Since the left implies $\forall n : \mathbb{N}, \#\text{True}_{\text{TrueOnFirst}_g}(n) = 0$, which is a contradiction, we can conclude $\exists m : \mathbb{N}, \text{TrueOnFirst}_g\ \ m = \text{True}$, giving the index $m : \mathbb{N}$ such that $g(m) = \text{True}$. □

We proceed to show informally

$$(\forall f : \mathbb{N} \to \text{bool}, \text{bounded}(f) \Rightarrow \text{sb}(f)) \iff \text{WLPO}.$$

For the right to left direction, we need to find a strict bound for f from a (weak) bound, with the help of WLPO. We define the predicate $B(n)$ which holds for an n if n is a bound on the number of true elements in f; that is,

$$B(N) := \forall m : \mathbb{N}, \#\text{True}_f(m) \leq n.$$

Importantly, we have $B(n) \vee \neg B(n)$ with the help of WLPO. We then simply perform a search downwards from the bound we got from the hypothesis that f is bounded, checking each smaller number until we find a *strict* bound.

For the direction left to right, we have to decide $(\forall n : \mathbb{N}, g(n) = \text{True}) \vee \neg(\forall n : \mathbb{N}, g(n) = \text{True})$ for a $g : \mathbb{N} \to \text{bool}$. We solve the following equivalent formulation:

$$(\forall n : \mathbb{N}, g^-(n) = \text{False}) \vee \neg(\forall n : \mathbb{N}, g^-(n) = \text{False}).$$

We use the TrueOnFirst construction, and observe that TrueOnFirst_{g^-} has at most 1 true position, so TrueOnFirst_{g^-} is bounded. By the assumption we get a *strict* bound. If the strict bound of TrueOnFirst_{g^-} is 1 then $\neg(\forall n : \mathbb{N}, g^-(n) = \text{False})$; if it is 0, then $(\forall n : \mathbb{N}, g^-(n) = \text{False})$.

Theorem 2.2. $(\forall f : \mathbb{N} \to \text{bool}, \text{bounded}(f) \Rightarrow \text{sb}(f)) \iff$ *WLPO.*

Proof. (⇐): We assume WLPO

$$(\forall n : \mathbb{N}, P(n)) \vee (\neg\forall n : \mathbb{N}, P(n)),$$

and an arbitrary bounded Boolean stream f:

$$\exists n : \mathbb{N}, \forall k : \mathbb{N}, \#\text{True}_f(k) \leq n.$$

We first note that we can prove the following theorem for decidable predicates P over the natural numbers by a simple induction over the witness in the antecedent.

$$(\exists n : \mathbb{N}, P(n)) \to (\neg P(0) \to \exists n : \mathbb{N}, P(n) \wedge \neg P(n-1)) \tag{2.4}$$

By applying the functional version of WLPO on the complement stream of f, we can decide whether all indexes of f are False (i.e., whether $\forall k : \mathbb{N}, \#\text{True}_f(k) = 0$.) If yes, then we have the witness for $\text{sb}(f)$, and we are done. So for the rest of the proof, assume otherwise:

$$\neg\forall k : \mathbb{N}, \#\text{True}_f(k) = 0. \tag{2.5}$$

We define the predicate B over natural numbers, holding when n is a bound on the number of true positions:

$$B(n) := \forall m : \mathbb{N}, \#\text{True}_f(m) \leq n,$$

and we note that B is decidable with WLPO: given an n we can apply WLPO to decide whether $\forall m : \mathbb{N}, \#\text{True}_f(m) \leq n$ or $\neg\forall m : \mathbb{N}, \#\text{True}_f(m) \leq n$. From the fact that f is bounded we know that $\exists n : \mathbb{N}, B(n)$, and from 2.5 above we know that $\neg B(0)$, giving us from 2.4

$$\exists n : \mathbb{N}, (\forall m : \mathbb{N}, \#\text{True}_f(m) \leq n) \wedge \neg(\forall m : \mathbb{N}, \#\text{True}_f(m) \leq n-1),$$

which is equivalent to

$$\exists n : \mathbb{N}, (\forall m : \mathbb{N}, \#\text{True}_f(m) \leq n) \wedge \neg(\forall m : \mathbb{N}, \#\text{True}_f(m) < n),$$

giving us the witness for $\text{sb}(f)$. □

(⇒): We will show the functional version of WLPO, so we assume a Boolean stream g, looking to decide whether $(\forall n : \mathbb{N}, g(n) = \text{True}) \vee \neg(\forall n : \mathbb{N}, g(n) = \text{True})$. This is equivalent with deciding

$$\forall n : \mathbb{N}, g^-(n) = \text{False} \vee \neg\forall n : \mathbb{N}, g^-(n) = \text{False}\,.$$

Observe that we have $\forall k{:}\mathbb{N}, \#\mathrm{True}_{\mathrm{TrueOnFirst}_{g^-}}(k) \leq 1$, so $\mathrm{TrueOnFirst}_{g^-}$ is bounded, and by the main assumption we get a witness for $\mathrm{sb}(\mathrm{TrueOnFirst}_{g^-})$:

$$\exists n{:}\mathbb{N}, ((\forall k{:}\mathbb{N}, \#\mathrm{True}_{\mathrm{TrueOnFirst}_{g^-}}(k) \leq n) \wedge \neg(\forall k{:}\mathbb{N}, \#\mathrm{True}_{\mathrm{TrueOnFirst}_{g^-}}(k) < n)).$$

Now observe that we have $\forall k : \mathbb{N}, \#\mathrm{True}_{\mathrm{TrueOnFirst}_{g^-}}(k) = 0$ if and only if $\forall n : \mathbb{N}, g^-(n) = \mathrm{False}$, and by inspecting whether the $n : \mathbb{N}$ witnessing $\mathrm{sb}(\mathrm{TrueOnFirst}_{g^-})$ is 0 or 1 we are done. □

Notice how the final proof can be modified easily to show

$$(\forall f : \mathbb{N} \to \mathrm{bool}, \mathrm{bounded}(f) \Rightarrow \mathrm{eaf}(f)) \Rightarrow \mathrm{LPO}.$$

Instead of $\mathrm{sb}(\mathrm{TrueOnFirst}_{g^-})$ we would get $\mathrm{eaf}(\mathrm{TrueOnFirst}_{g^-})$, which would tell us not only whether there were true elements in g^-, but provide us with the actual last index. Since $\forall i : \mathbb{N}, g^-(i) = \mathrm{True} \Leftrightarrow g(i) = \mathrm{False}$, this allows us to decide $(\forall n : \mathbb{N}, g(n) = \mathrm{True}) \vee (\exists n : \mathbb{N}, g(n) = \mathrm{False})$.

All of the above results are verified in Coq.[1] The formalization is a straightforward transcript of the proofs given here.

2.4 Conclusion

We have provided a formalization of finiteness which fits robustly between the two existing notions of eventually always false and bounded. We placed it firmly in the hierarchy, and showed that the reductions one way are provable without assumptions, while reductions upwards are equivalent with well-known constructive principles.

[1] See `https://github.com/epa095/strictly-bounded-streams` for the Coq script.

Investigating Streamless Sets

Erik Parmann

In this paper we look at *streamless sets*, recently investigated by Coquand and Spiwack [CS10]. A set is *streamless* if every stream over that set contain a duplicate. It is an open question in constructive mathematics whether the Cartesian product of two streamless sets is streamless.

We look at some settings in which the Cartesian product of two streamless sets is indeed streamless; in particular, we show that this holds in Martin-Löf intentional type theory when at least one of the sets have decidable equality. We go on to show that the addition of functional extensionality give streamless sets decidable equality, and then investigate these results in a few other constructive systems.

3.1 Introduction

One of the interesting aspects of working in constructive mathematics is that notions often become more nuanced than they do in classical mathematics. This holds for the notion of finiteness, for instance; there are a multitude of possible definitions of a set being finite which would be equivalent classically, but are different constructively.

In this paper, we will look at a particular definition of finite sets in a constructive context, given in terms of streamless sets. This is essentially a constructive version of the classical statement that a set is finite if there are no injections from $\mathbb{N}$ into it. It is formulated positively: a set A is *streamless* when

$$\forall f : \mathbb{N} \to A, \exists i, j : \mathbb{N}, i < j \wedge f(i) = f(j).$$

It is not known who first looked at finiteness in a constructive setting, but it was recently investigated by Coquand and Spiwack [CS10], who look at four different definitions of a set being finite. These four are, in decreasing order of strength:

- Enumerated: there is a list containing all the elements in the set;
- Bounded: there is an $n : \mathbb{N}$ such that every list with more than n elements has duplicates;
- Noetherian: no matter how one adds elements from the set to a list, one eventually gets duplication in the list; and
- Streamless: every stream over the set contains duplicates.

They show that these notions form a strict hierarchy, except in the streamless and noetherian cases (where strictness is left open): they show that any noetherian set is streamless, and conjecture that the converse does not hold.

It is relatively easy to see that not all bounded sets are enumerated: with enumerated sets we actually have all of its elements, while with bounded sets we only know a bound on the size of the set. In fact, emptiness is in general undecidable for bounded sets, but decidable for enumerated sets. Coquand and Spiwack[CS10] cite an example offered by F. Richman of a way to generate subsets of natural numbers with the property that one cannot *a priori* know the size of the subset, but if one gets any element in the set then one knows the size of the set. These sets are noetherian but not bounded.

Marc Bezem and other authors have a model of Martin-Löf Type Theory [MLS84] in which there is a streamless set which is not provably noetherian, thus showing that the noetherian property is strictly stronger than streamlessness (personal

communication, September 2014). This model is rather complicated and has yet to be published. The authors construct a set parameterized by a undecidable predicate on $\mathbb{N}$. Equality on this set is decidable, which is important for the proof that it is streamless. They assume Markov's principle, and use that as the "engine" which finds duplicates in the streams. They are also able to show that this set cannot be noetherian. In this way they show—since Markov's principle is consistent with Type Theory—that it is not possible to prove that streamlessness implies noetheriannes.

In addition to giving the hierarchy, Coquand and Spiwack [CS10] also prove several closure properties of the different notions of finiteness. They show that all four are closed under sum; that is, for any of the notions of finiteness, the sum of two finite sets is itself finite by the same notion. The situation is more complicated for Cartesian products. The two strongest notions, enumerated and bounded, are shown to be closed under products, and noetherian sets are closed under products, as long as one of the sets has decidable equality. The use of decidable equality in one of the sets in the proof in [CS10] was first pointed out in [BNU12]. Whether streamless sets are closed under Cartesian products was left as an open problem.

Our main result will be the following: in Martin-Löf intentional type theory (ITT)[ML75] streamlessness is closed under Cartesian products, granted that one of the sets has decidable equality or is bounded.

An important feature of ITT is strong Σ-elimination. Consequently, from a proof of $\forall x \exists y.\phi(x, y)$ we are able to get, for any x, an actual y which can be used in the construction of new functions/streams. This plays an important role in the proof of our main result. In other systems, like HA which we will look at in Section 3.6, we need to assert a axiom of choice to get the same.

In Coq we have the choice of formalizing statements either in Set, which enjoy strong Σ-elimination, or Prop which does not. The proof we provide here will, on the face of it, only hold when streamlessness is formalized in Set; but we will see that, as long as *both* sets have decidable equality, the two formalizations actually correspond.

Decidable equality plays an important role in our proof, and we conjecture that streamlessness is not closed under products when both sets have undecidable equality. We show that, in ITT with functional extensionality, streamless sets have decidable equality, meaning that a potential counter-model must reject functional extensionality.

The main motivation behind this work is curiosity as to the strength of streamless sets, but there is also potential for practical applications. One such example is outlined in Coquand and Spiwack [CS10], namely automaton reachability testing. They give the regular depth-first graph algorithm for finding reachable states, and then proceed to show that if one assumes that the set of states in the au-

tomaton is finite in the sense of streamless, then this algorithm terminates. It is not uncommon to take the Cartesian product of two automata to create one which has as its language the intersection of the two original languages. Given that streamlessness is closed under product, one can show that the reachability algorithm also terminates on this new automaton.

In Section 3.2, we introduce streamless sets and some machinery which lets us find any number of duplicate elements. In Section 3.3, we prove the main theorem: that streamless sets with decidable equality are closed under Cartesian products in ITT. In Section 3.4, we see that adding functional extensionality gives streamless sets decidable equality. Section 3.5 relates our findings to Coq and its Set vs Prop distinction; it also briefly touches upon Homotopy Type Theory with Univalence. In Section 3.6, we relate our finding to Heyting arithmetic in the systems (E-) HA^ω. Section 3.7 provides a brief overview of related works; Section 3.8 highlights some remaining questions; and we conclude in Section 3.9.

3.1.1 Notation

We work in Martin-Löf intensional type theory (ITT)[ML75], where both propositions and sets are modeled as types.

We assume a inductive type $\mathbb{N}$ for the natural numbers, and we have the usual type constructors: If A is a type and B is a type family over A then both $\Pi_{x:A}B(x)$ and $\Sigma_{x:A}B(x)$ are types, the dependent function type and the dependent pair type with the usual computation rules. We use $\pi_1 : (\Sigma_{x:A}B(x)) \to A$ and $\pi_2 : \Pi_{p:\Sigma_{x:A}B(x)}B(\pi_1(p))$ as the two projections of dependent pairs. In the special cases where $B(x)$ does not depend on x, we abbreviate $\Pi_{x:A}B(x)$ as $A \to B$ and $\Sigma_{x:A}B(x)$ as $A \times B$, the latter being the Cartesian product of A and B. If A and B are types, then $A + B$ is their disjoint union with the constructors $\mathrm{inl} : A \to A + B$ and $\mathrm{inr} : B \to A + B$.

We will use the notation $\mathrm{Dec}\ =_A$ to stand for the type $\Pi_{x:A}\Pi_{y:A}(I_A(x,y) + \neg I_A(x,y))$, where $I_A(x,y)$ is the inductive identity type. We will use $=_A$ as an infix version of I_A, or just $=$ if the type A is clear from context. With A having decidable equality we mean that we have an inhabitant of $\mathrm{Dec}\ =_A$.

A stream over a set A is any function of type $\mathbb{N} \to A$. Given a stream $g : \mathbb{N} \to A$ we also have "cut" streams $g|_n : \mathbb{N} \to A$ for every $n : \mathbb{N}$ defined by

$$g|_n(x) := g(x + n).$$

When we say that we have duplicates in a stream $g : \mathbb{N} \to A$, we mean that we have two indices $i < j$ such that $g(i) =_A g(j)$.

Given a stream g over $A \times B$, we can project out two streams $g_1 : \mathbb{N} \to A$ and $g_2 : \mathbb{N} \to B$ being $g_i = \pi_i \circ g$. As usual, two elements in $A \times B$ are equal if

both their first and second projection are equal. We also say that two elements in $A \times B$ are A-equal (resp. B-equal) if their first (resp. second) projections are equal.

3.2 Introduction to streamless sets

A set A is *streamless* if all streams over it contains duplicates; that is, for all streams $g : \mathbb{N} \to A$, we have indices $i < j$ with $g(i) =_A g(j)$. Formally, it means that we have a inhabitant of the type

$$\text{Streamless}(A) := \Pi_{f:\mathbb{N}\to A}\Sigma_{p:\mathbb{N}\times\mathbb{N}}(\pi_1(p) < \pi_2(p) \times f(\pi_1(p)) =_A f(\pi_2(p))).$$

In what follows we will mostly be interested in the pair $p : \mathbb{N} \times \mathbb{N}$, and not the proof that it has the desired features. To avoid having to project out the number and clutter up the construction more than needed, we will assume that if we have a streamless set A, we have a witness

$$M_A : (\mathbb{N} \to A) \to \mathbb{N} \times \mathbb{N},$$

which, given a stream g over A, gives out two indices $i < j$ such that $g(i) = g(j)$.

First, we show that if we have a stream over a streamless set B, we can find not only duplicates, but for any n we can find elements occurring at least n times. This is clear classically; we just have to look at the first $|B| \times n$ elements in the stream. Constructively it is less clear, as we do not know the actual size of the set— only that it is streamless. As seen in the introduction, one cannot, in general, deduce the size of a set from the fact that it is streamless. The first part of this construction, for $n = 2$, is also used to prove that streamless is closed under sum in [CS10].

Given a stream g over streamless B, we make a new stream g^2 over $B \times \mathbb{N} \times \mathbb{N}$, such that for every $\langle b, i, j\rangle$ we have $i < j$ and $g(i) = g(j) = b$, and for all $g^2(n) = \langle b, i_1, j_1\rangle$ and $g^2(n+1) = \langle c, i_2, j_2\rangle$ we have $j_1 < i_2$. We get this by letting g^2 begin with $\langle g(j), i, j\rangle$ where $\langle i, j\rangle = M_B(g)$, and then continue likewise on the stream $g|_{j+1}$.

Formally, g^2 is defined as follows, where _ indicates a value which we do not use (and thus prefer not to name).

Definition 3.1 ($g^2 : \mathbb{N} \to B \times \mathbb{N} \times \mathbb{N}$)**.**

$$\begin{aligned} g^2(0) &= \langle g(j), i, j\rangle && \text{where } \langle i, j\rangle = M_B(g), \\ g^2(n+1) &= \langle g(j+p), i+p, j+p\rangle && \text{where } \langle \_, \_, p\rangle = g^2(n) \\ & && \text{and } \langle i, j\rangle = M_B(g|_{p+1}) \end{aligned}$$

Figure 3.1 contains a visual representation of g^2, the top being g and the bottom g^2. The two blue boxes make up the first duplicate pair found by $M_B(g)$. The vertical red line indicates that this is where we "cut" the stream, and by using M_B again on this new stream we get a new duplicate pair, the purple diamonds. This process continues, defining a new stream of representatives of duplicates in g.

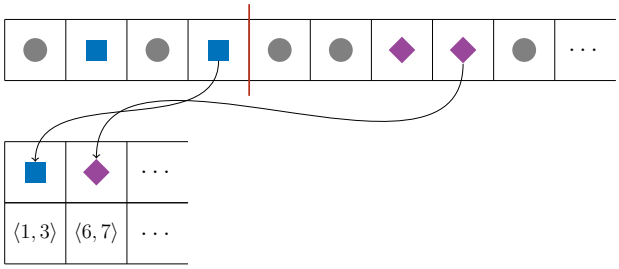


Figure 3.1: g^2, the stream of duplicates in g.

The first projection of g^2 is itself a B-stream, and we can then use the same process on this stream. This provides duplicate duplicates, giving us elements which occur four times in g.

We can iterate this process and, for every $n : \mathbb{N}$ and stream $g : \mathbb{N} \to B$, we get a stream $g^n : \mathbb{N} \to B \times (List\ \mathbb{N})$ such that every element in the new stream gives a $\langle b, l\rangle$ such that b occurs at least n times in g, at the n different indices given in l.

To formally define g^n it is easiest to first define a slightly stronger function $f^n : \mathbb{N} \to B \times (List\ \mathbb{N}) \times \mathbb{N}$. The last natural number is used when defining $f^{m+1}(n+1)$, it tells it where in $(f^m)_1$ the n^{th} duplicate was found, enabling us to cut $(f^m)_1$ at the right place.

$$\begin{aligned}
f^2(n) =& \langle b, [i_1, i_2], i_2\rangle && \text{where } \langle b, i_1, i_2\rangle = g^2(n)\\
f^{m+1}(0) =& \langle (f^m)_1(i), (f^m)_2(i) + +(f^m)_2(j), j\rangle && \text{where } \langle i, j\rangle = M_B((f^m)_1),\\
f^{m+1}(n+1) =& \langle (f^m)_1(i), (f^m)_2(i) + +(f^m)_2(j), j\rangle && \text{where } \langle _, _, p\rangle = f^{m+1}(n)\\
& && \text{and } \langle i, j\rangle = M_B((f^m)_1|_p)
\end{aligned}$$

This process is illustrated in Figure 3.2, where we show how to calculate f^3 from f^2.

Having f^n we define g^n by simply dropping the third number:

Definition 3.2 ($g^n : \mathbb{N} \to B \times List\ \mathbb{N}$)**.**

$$g^n(x) = \langle e, l\rangle \qquad \text{where } \langle e, l, \_\rangle = f^n(x)$$

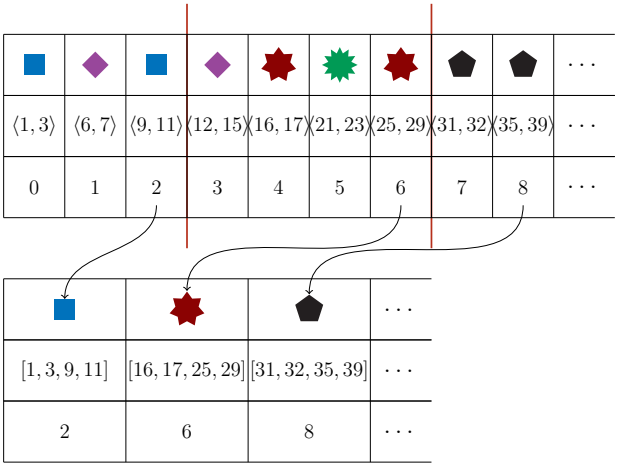


Figure 3.2: Calculating f^3 from f^2.

The attentive reader notices that this does not actually produce linearly many indices, but exponentially many. g^3 is actually a stream of items occurring 4 times, and g^4 is a stream of objects occurring 8 times. We will not make use of this property, and we will, for the sake of simplicity, assume that g^n contains elements that occur n times.

Observe that we use strong $\exists$ elimination for this construction. Not only do we know that there are indices, but we know what they are; we are also free to use them in the construction of a new stream, to which we can apply M_B once more. As mentioned above, [CS10] uses a stream which is the first projection of g^2 in the proof that streamless is closed under sum. We do not know of a proof that streamlessness is closed under sum which does not assume strong $\exists$ elimination.

3.3 Products of streamless sets

This section applies the machinery developed in the previous section to the product of streamless sets.

We will first see that the Cartesian product of a bounded set with a streamless set is streamless. It is worth noting that this is independent of whether any of the sets has decidable equality or not.

Lemma 3.3. *In ITT we have: If at least one of A and B is bounded and the other is streamless then $A \times B$ is streamless.*

Proof. We assume that A is bounded by n. (If it were B then the construction below would be "mirrored".) Given the stream $g : \mathbb{N} \to A \times B$ we look at $g_2 : \mathbb{N} \to B$, its second projection. By looking at $(g_2)^{n+1}(0)$ we get a pair $\langle b, [i_0, \ldots, i_n]\rangle$ such that b occurs at all the indices $i_0, \ldots, i_n$ in g_2. Note that $g_1(i_0), \ldots, g_1(i_n)$ are $n+1$ elements of A, so since A is bounded by n, there must be at least two indices $i_k < i_l$ such that $g_1(i_k) = g_1(i_l)$. As $g_2(i_k) = b = g_2(i_l)$, we get $g(i_k) = g(i_l)$. □

We now show that Markov's principle and decidable equality of one of the sets imply that streamlessness is closed under product. This result is a warm up for the later, more general result shown in Theorem 3.1. The proofs have interesting similarities, especially in how we can use the streamlessness of a set to "emulate" Markov's principle.

First a reminder of Markov's principle.

Definition 3.4 (Markov's principle)**.** For decidable predicates P on $\mathbb{N}$ we have $\neg\neg\Sigma_{x:\mathbb{N}}P(x) \to \Sigma_{x:\mathbb{N}}P(x)$.

Markov's principle has a quite computational flavour, which unsurprisingly makes it easier to prove a set streamless. All we need to do to find the duplicate indices is to show that it cannot be the case that they do not exist.

Lemma 3.5. *In ITT with MP we have: If at least one of A and B has decidable equality and A and B are both streamless then $A \times B$ is streamless.*

Proof. We assume a stream $g : \mathbb{N} \to A \times B$. We also assume, without loss of generality, that A is the set with decidable equality. (If it were B then the construction below would be "mirrored".)

We define the following predicate on $\mathbb{N}$:

$$P(n) := \text{For } \langle \_, [i_0, \ldots, i_{n-1}]\rangle = (g_2)^n(0) \text{ we have duplicates in } [g_1(i_0), \ldots, g_1(i_{n-1})].$$

Remember that g_2 gets the B-stream, and $(g_2)^n$ finds n indices with equal elements. Note that if A has decidable equality, $P(n)$ is decidable.

We now proceed to show that (1) $\neg\neg\exists n P(n)$ and that (2) from $\exists n P(n)$ we can get $\langle i, j\rangle$, with $i < j$ and $g(i) = g(j)$.

Proof of (1). We assume $\neg\exists n P(n)$ and proceed to produce a contradiction. $\neg\exists n P(n)$ implies $\forall n \neg P(n)$, which says that for any n we have that for $\langle b, [i_0, \ldots, i_{n-1}]\rangle = (g_2)^n(0)$ the list $[g_1(i_0), \ldots, g_1(i_{n-1})]$ has no duplicates. Notice that the list $[g_1(i_0), \ldots, g_1(i_{n-1})]$ has n elements, all from A.

We now make a duplicate-free stream $f : \mathbb{N} \to A$; that is, for every $n : \mathbb{N}$, we have for all $j < n$ that $f(n) \neq f(j)$, contradicting that A is streamless. Defining

$f(n)$ we first find $n+1$ indices with the same b element, $\langle \_, [i_0, \ldots, i_n] \rangle = (g_2)^{n+1}(0)$. We now let

$$f(n) = ([g_1(i_0), \ldots, g_1(i_n)] \setminus [f(0), \ldots f(n-1)])(0).$$

That is, $f(n)$ is the first element in the list resulting from removing any element from $[g_1(i_0), \ldots, g_1(i_n)]$ that the stream already contains. As $[g_1(i_0), \ldots, g_1(i_n)])$ contains $n+1$ *different* elements from A, we know that the resulting list is non-empty. Since this stream only outputs elements which have not been output up until that point, it will never introduce a duplicate pair. Thus we have contradicted that A is streamless, enabling us to conclude $\neg\neg\exists n P(n)$. □

Proof of (2). From $\exists n P(n)$ we have that there is an n such that for $\langle b, [i_0, \ldots, i_{n-1}] \rangle = (g_2)^n(0)$ the list $[g_1(i_0), \ldots, g_1(i_{n-1})]$ has duplicates. Let those indices be $i_k < i_l$. Since every element in $g(i_0), \ldots, g(i_{n-1})$ has b as its second coordinate, we get that $g(i_k) = g(i_l)$. □

Having proved $\neg\neg\exists n P(n)$, we apply Markov's principle and get $\exists n P(n)$. By (2) above, this gives us the indices $\langle i, j \rangle$ with $i < j$ and $g(i) = g(j)$. □

We will now proceed to get rid of Markov's principle. Several parts of the previous proof will be recognisable, but we use the streamlessness of the two underlying sets to do the work that Markov's principle did in the previous proof.

Theorem 3.1. *In ITT we have: If at least one of A and B has decidable equality and A and B are both streamless, then $A \times B$ is streamless.*

Proof. We assume that A is the set with decidable equality, and we want to construct

$$M_{A \times B} : (\mathbb{N} \to A \times B) \to \mathbb{N} \times \mathbb{N},$$

which, given any $A \times B$-stream g, finds a pair of indices $i < j$ such that $g(i) = g(j)$.

Given an $A \times B$-stream g, we inductively define an A-stream f by letting $f(n)$ first look at $f(m)$ for all $m < n$, and see if two equal elements are outputted. This can be done since A has decidable equality. If there is a duplicate element, $f(n)$ outputs it. If there are no duplicates outputted so far, we let $f(n)$ look at all the A-elements corresponding to the n B-elements given by $(g_2)^n(0)$. Remember, $(g_2)^n(0) = \langle b, l \rangle$ where $l = [i_0, \ldots, i_{n-1}]$ is a list of n indices. Looking up these indices in g_1 gives us a list $la : List\ A$ of n A-elements.

By using the decidability of A, we can check whether there are duplicate elements in la. In the case of no duplicates, we know that there must be at least one of the n elements which does not already occur in f so far (as we have only produced $n-1$ elements so far). We can check which one this is, as we have

already defined f up to $n-1$. We then let $f(n)$ be one of those elements which has not occurred in f so far. More precisely,

$$f(n) = ([g_1(i_0), \ldots, g_1(i_{n-1})] \setminus [f(0), \ldots f(n-1)])(0).$$

If, on the other hand, there is some duplicate element in the list, we let $f(m)$ be that element for all $m \geq n$. Notice that if this is the case, this is the first time a duplicate is introduced in f. This completes the construction of $f : \mathbb{N} \to A$, and we will now use f to find duplicates in $g : \mathbb{N} \to A \times B$.

From the construction of f, we have the following property:

Lemma 3.6. *For the smallest i such that $f(i) = f(i+1)$ we have duplicates in the list $[g(l_0) \ldots, g(l_{i-1})]$, where $\langle b, [l_0 \ldots, l_{i-1}]\rangle = (g_2)^i(0)$.*

As A is streamless and f is a A-stream, we can use M_A to find indices $k < d$ of duplicates in f. Since A had decidable equality, we can do a bounded search downward from k to find the first index i such that $f(i) = f(i+1)$. By Lemma 3.6, we have duplicates in $[g(l_0) \ldots, g(l_{i-1})]$ where $\langle b, [l_0 \ldots, l_{i-1}]\rangle = (g_2)^i(0)$. Thus, we have two indices $l_k < l_m$ in l such that $g_1(l_k) = g_1(l_m)$. By construction all the indices in $[l_0 \ldots, l_{i-1}]$ are B-equal, so $g_2(l_k) = g_2(l_m)$, giving $g(l_k) = g(l_m)$.

□

Finally observe that if it were B and not A that had decidable equality, the construction above would be "mirrored"; f would have to be a B-stream, and we would use $(g_1)^n(0)$ instead of $(g_2)^n(0)$.

As an example of the construction, let us look at a particular calculation of $f(4)$, where no duplicates have been found so far. That means that so far f looks like

$$f = a_0, a_1, a_2$$

with none of them being equal any other.

$(g_2)^4(0)$ is $\langle b, [n_0, n_1, n_2, n_3\rangle]$, giving four indices in g with the same b-element. This means that g looks somewhat like

$$g = \ldots, \langle b, a'_0\rangle \ldots, \langle b, a'_1\rangle, \ldots, \langle b, a'_2\rangle, \ldots, \langle b, a'_3\rangle, \ldots$$

By the decidability of A, we can check whether there is a duplicate among $[a'_0, a'_1, a'_2, a'_3]$. If not, then we know that there is some element in $[a'_0, a'_1, a'_2, a'_3] \setminus [a_0, a_1, a_2]$, and we let $f(4)$ be the first such element. If there are duplicates, e.g $a'_1 = a'_3$, we let $f(n) = a'_1$ for all $n \geq 4$.

Comparing this proof with the proof using Markov's principle we see that we can use the streamlessness of one of the underlying sets to search for the n which gives us A-duplicates. The trick is to control exactly when duplicates are

introduced in the f-stream, and then use the streamlessness of A to recover this point.

We combine Lemma 3.3 and Theorem 3.1 to get the following corollary.

Corollary 3.7. *In ITT we have: If at least one of A and B has decidable equality or is bounded, and A and B are both streamless, then $A \times B$ is streamless.*

3.4 Streamlessness and decidable equality

It should be clear by now that decidable equality of the underlying set is quite important for the ability to produce streamless sets; we will see another indication of this in this section. We will show that in ITT, functional extensionality give streamless sets decidable equality. In addition to showing the close relation between finiteness and decidable equality, it is relevant to the search for a potential counter-model to the claim that streamlessness is closed under Cartesian products even without decidable equality.

As a warm-up, we look at the situation where the set is not only streamless, but bounded. Remember that this means that we have an $n : \mathbb{N}$ such that, for every A-list of more than n elements, we can find a duplicate pair. Formally, this means that we have an inhabitant of the type

$$\text{Bounded}(A) := \Sigma_{n:\mathbb{N}}(\Pi_{l:listA}(\text{len}(l) > n \to \Sigma_{i,j:\mathbb{N}}(i < j \times l[i] = l[j]))).$$

If we want to determine whether a_1 is equal to a_2 we make a list l of $n+1$ instances of a_1, and get a pair of indices $i_1 < j_1$ with duplicates in this list. We then proceed to swap the element at $l[i_1]$ with a_2, giving a new list. The original list is equal the new list if and only if $a_1 = a_2$.

We then proceed to get two indices $i_2 < j_2$ of duplicate elements in this new list. If this process is assumed to be a function, and thus provide equal outputs for equal inputs, we get $\langle i_1, j_1\rangle = \langle i_2, j_2\rangle$ if and only if $a_1 = a_2$; and since equality on $\mathbb{N}$ is decidable, we are done.

Our proof turned on the facts that (1) the second projection of a witness of Bounded(A) is a function, (2) this function can be assumed to respect equality on its input, and (3) two lists are equal if and only if they are pointwise equal.

We will now mirror this with streamless sets. One major difference between lists and streams is the following: while lists are equal whenever their elements are equal, this only holds for streams if we assume so. It is consistent to assume an inhabitant of the following type in ITT, and if we do so for all types, we say that we have *functional extensionality*.

Definition 3.8 (FunExt(A))**.**

$$\text{FunExt}(A) := \Pi_{f,g:\mathbb{N}\to A}(\Pi_{n:\mathbb{N}}(f(n) =_A g(n)) \to f =_{\mathbb{N}\to A} g)$$

Lemma 3.9. *In ITT with functional extensionality we have: If A is streamless then it has decidable equality.*

Proof. We assume an inhabitant of $\mathrm{FunExt}(A)$ and two elements $a, b : A$, and we proceed to determine their equality. Let the stream f_a be the constant A-stream consisting of only a, and let $\langle i, j\rangle$ be the indices returned by $M_A(f_a)$. We now make the stream f'_a which is constantly a, except at index i, where it is b:

$$f'_a(n) = \begin{cases} b & \text{if } n =_{\mathbb{N}} i \\ a & \text{otherwise} \end{cases}$$

Notice that if $a =_A b$ we have $\Pi_{n:\mathbb{N}} f_a(n) =_A f'_a(n)$, so from functional extensionality we then have $f_a = f'_a$. So, by functionality of M_A, we get $a = b \to M_A(f_a) = M_A(f'_a)$, and thus

$$M_A(f_a) \neq M_A(f'_a) \to a \neq b.$$

Concluding, if $M_A(f'_a) \neq \langle i, j\rangle$ then $a \neq b$, and if $M_A(f'_a) = \langle i, j\rangle$ then $a = b$ (as $f'_a(i) = b$ and $f'_a(j) = a$), and since equality on $\mathbb{N}$ is decidable we are done. □

Lemma 3.9 is relevant for the search of a counter-model to the general claim that streamlessness is closed under product. From section 3.3, we know that such a counter-model must have two streamless sets with undecidable equality. This section shows that the model must also reject functionality extensionality for us to have a streamless set with undecidable equality.

It also highlights some of the difficulty of defining finiteness for sets with undecidable equality in a computational setting, and since the other notions of finiteness given in [CS10] imply streamlessness, this result also covers them. All the definitions of finiteness have some sort of equality/duplication check at their core. Given this it seems plausible that a proof of finiteness can, in certain situations, lead to decidability. On the other hand, it is quite unsatisfactory that, in certain settings, we are unable to define finite sets of elements with undecidable equality.

In the next section we look at how to formalize both this and the previous results in Coq.

3.5 Formalization in Coq and HoTT

3.5.1 Coq: Prop and Set

In this section we will relate the above results to the proof assistant Coq [CDT12], where we have to deal with the distinction between Prop and Set. Functions, which is how we defined streams, live in the universe Set, while there is a separate universe Prop for propositions. The intention is, roughly, to separate between

types where we care about the internal structure of the inhabitants (Set) and where we care only about the existence of the inhabitant (Prop).

Given a inhabitant of a type in Prop one is generally not allowed to eliminate on it to construct elements in Set; thus we can not build the new stream g_2 of duplicates using indexes found from a witness of a type in Prop. This means that the constructions given in this paper can not be implemented in Coq *as they stand* if streamless is written as follows:

```
Definition StreamlessEx(A:Set):= forall g:nat → A,
          exists i j, i<j ∧ g(i) = g(j).
```

One way to remedy the situation is to define the notion of a set being streamless in the following way, closer to the way it was encoded in ITT. The notation "$\{x : nat \mid P(x)\}$" is Coq's notation for $\Sigma_{x:\mathbb{N}} P(x)$.

```
Definition StreamlessSig (A:Set):= forall g:nat→ A,
          {ij : nat*nat | fst ij < snd ij ∧ g(fst ij)=g(snd ij)}.
```

StreamlessSig enables us to use the proof of a set being streamless in a computation; in particular it enables us to construct the stream g_2 needed to prove Corollary 3.7 in Coq. The disadvantage is that it can make it harder to prove sets to be streamless in the first place. There is reason to believe that there are fewer sets satisfying StreamlessSig than StreamlessEx.

In general, whether one wants the statement in Prop or in Set reflects whether one wants to work proof relevant or not; formalizing it as StreamlessSig enables us to use the proof (of a set being streamless) in a computation.

StreamlessSig A implies StreamlessEx A, while the provability of the converse implication is unknown. Interestingly, it *is* know for sets with decidable equality, since we are able to prove the following lemma in Coq for A with decidable equiality, making the two notions of streamless coincide in those cases.

```
Lemma streamlessExToStrSig(A:Set)(A_dec: DecidableEq A) :
                        StreamlessEx A → StreamlessSig A
```

Essential for the proof is the following lemma, holding for decidable predicates P on $\mathbb{N}$, and shown in the Coq library *Coq.Logic.ConstructiveEpsilon*[1].

```
Lemma constructive_indefinite_ground_description_nat :
      (exists x : nat, P x) → {x : nat | P x}.
```

With the indefinite ground description the proof is straightforward. We assume that we have some pairing/decoding functions enabling us to encode pairs of natural numbers as single natural numbers. We then define versions of both StreamlessEx and StreamlessSig using single numbers, prove that the single and

[1] http://coq.inria.fr/library/Coq.Logic.ConstructiveEpsilon.html

paired versions are equivalent, and then it is a simple application of the indefinite ground description given above.

The conclusion is the following corollary:

Corollary 3.10. *In Coq we can prove that StreamlessEx (and StreamlessSig) of sets with decidable equality is closed under Cartesian products.*

A natural question is whether we can strengthen this to say that StreamlessEx is closed under Cartesian products as long as *at least one* of the sets have decidable equality. Unfortunately, this does not follow from the current construction. To see this, assume an $A \times B$-stream g. The construction in Proof 3.3 uses $(g_2)^n$ to find n-indices with B-equal elements. But for this to be definable in Coq using the technique above, B needs decidable equality. The proof then uses the decidability of A to eliminate on whether there are duplicates among the resulting A-elements or not. It does not seem possible to manipulate the construction such that it is enough for only one of the sets to have decidable equality.

We are also able to reproduce Lemma 3.9 in Coq for StreamlessSig, the proof is simply a direct Coq formalization of the proof given in Section 3.4.

```
Lemma strSigAndFuncExtImpliesDecA (A:Set) (Ma:StreamlessSig A)
    (fext: functional_extensionality nat A): forall a b :A, {a=b}+{not(a=b)}.
```

Again, we are not able to simply adapt the proof to StreamlessEx, since the proof crucially uses the indexes returned from M_A in the construction of new functions.

All the Lemmas in this section have been formalized and proved in Coq[2].

3.5.2 HoTT

Closely related to the Prop/Set distinction is the truncated and non-truncated statements one encounters in Homotopy Type Theory (HoTT). Truncation is a type former which "truncates" a type — removing all information contained in the inhabitants of that type except their existence — and it is written as $||A||$ for a type A. (For more information we refer the reader to the freely available book [Uni13].) We will not go further into HoTT here; but what is relevant for us is that we have a HoTT version of the indefinite ground description above. For decidable predicates P we have

$$||\Sigma_{n:\mathbb{N}} P(n)|| \to \Sigma_{n:\mathbb{N}} P(n)$$

as stated by exercise 3.19 in [Uni13]. One should be able to reproduce a version of Corollary 3.10 in this setting, getting that for the non-truncated version of streamless it is enough for one of the sets to have decidable equality for streamlessness to be closed under Cartesian products.

[2]The Coq script can be found at `https://github.com/epa095/streamless-in-coq`.

With our current knowledge we need both sets to have decidable equality for the truncated version to be closed under Cartesian products without further assumptions, and we conjecture that this is in fact a strict requirement. If we choose to assume the HoTT-version of the axiom of choice,

$$(\Pi_{x:X}||\Sigma_{a:A(x)}P(x,a)||) \to ||\Sigma_{g:\Pi_{(x:X)}A(x)}\Pi_{x:X}P(x,g(x))||,$$

we can show that truncated-streamless sets are closed under products as long as one of the sets has decidable equality.

In HoTT we can also assume the Univalence axiom, giving that isomorphic structures can be identified. Importantly, the univalence axiom implies functional extensionality. Lemma 3.9 makes it clear that — unless we want every streamless set to have decidable equality—we must use the truncated version of streamlessness in this setting.

3.6 HA$^\omega$

It is natural to ask how closely coupled the above results are to the particular constructive setting we are working in, and whether we can reproduce them in a different setting. We will now look at how the results fit in the system HA$^\omega$, an extension of Heyting Arithmetic to the language of finite types, see [TVD88] for more information on HA$^\omega$.

HA$^\omega$ is proof-irrelevant and does not have strong Σ elimination; instead, we have to use the axiom of choice to extract a function, giving the witnesses which we can then use as terms in the logic.

The set of finite types $\mathcal{T}$ is built from the basic type 0 ($\mathbb{N}$) and is closed under $\times$ and $\to$. HA$^\omega$ is "neutral" in the terminology of [TVD88]; we do not assume decidability of $=_\tau$ for any other types than 0, nor do we assume that equality between functions is extensional.

Sets are not a primitive notion in HA$^\omega$, so when talking about sets we mean functions of the type $A : \tau \to 0$; such functions represent the set of elements on which it returns 1. This means that all sets will have decidable membership and sets can only contain elements of one and the same type. For a set $A : \tau \to 0$, we call τ the enclosing type of A. Following [TVD88] we will write $a < b$ in place of $< (a, b) = 1$, where the latter is the characteristic function of the less-than relation. With "a stream over A" we mean a function $f^{0\to\tau}$ where τ is the enclosing type of A such that $\forall n^0(A(f(n)) = 1)$.

Streamlessness of A_τ in this setting is expressed as

$$\text{Streamless}(A_\tau) := \forall g^{0\to\tau}((\forall n^0 A(g(n)) = 1) \to \exists i^0 j^0(i < j \wedge g(i) = g(j))).$$

In order to formalize our results in HA^ω, we first need to define some axioms. $\mathrm{AC}_{\sigma,\tau}$ is the following axiom schema,

$$\mathrm{AC}_{\sigma,\tau} := \forall x^\sigma \exists y^\tau \phi(x,y) \to \exists z^{\sigma\to\tau} \forall x^\sigma \phi(x, zx),$$

and AC is the axiom schema consisting of $\mathrm{AC}_{\sigma,\tau}$ for all types $\sigma, \tau \in \mathcal{T}$. $\mathrm{EXT}_{\sigma,\tau}$ is the following axiom schema,

$$\mathrm{EXT}_{\sigma,\tau} := \forall y^{\sigma\to\tau} z^{\sigma\to\tau}((\forall x^\sigma, yx = zx) \to y = z),$$

and if we add $\mathrm{EXT}_{\sigma,\tau}$ for all types $\sigma, \tau \in \mathcal{T}$ we get the system E-HA^ω.

To reproduce the proof of Lemma 3.5 in HA^ω, we need to construct the function g^2 of duplicates, and for this we need access to, for every stream, a pair of indexes with duplicate elements in that stream. The following instance of AC for every type τ enclosing a streamless set is enough to mirror Lemma 3.5 in HA^ω.

$$\mathrm{AC}_{0\to\tau,0} := \forall x^{0\to\tau} \exists y^0 \phi(x,y) \to \exists z^{(0\to\tau)\to 0)} \forall x^\sigma \phi(x, zx)$$

Let the $\phi(x,y)$ stand for the predicate "$(\forall i^0 A(x(i)) = 1) \to y$ encodes a pair of indexes $i < j$ such that $x(i) = x(j)$". Then the antecedent of $\mathrm{AC}_{0\to\tau,0}$ follows immediately from A being streamless, and the result is the function M_A, needed to reconstruct the machinery in the proof of Lemma 3.5.

Corollary 3.11. *In* $\mathrm{HA}^\omega + \mathrm{AC}$ *we have that streamless sets are closed under products.*

Encoding sets by their characteristic functions yields decidable membership, but in general not decidable equality. The extensionality of E-HA^ω, giving that streams are equal when they are pointwise equal, enables us to mirror Lemma 3.9:

Corollary 3.12. *In* E-$\mathrm{HA}^\omega + \mathrm{AC}$ *we have that streamless sets have decidable equality.*

Note that E-$\mathrm{HA}^\omega + \mathrm{AC}$ does not prove the law of excluded middle, as it is conservative over HA. For further details, see [Bee79].

3.7 Related work

One of the first investigations of streamlessness known to the author is by Richman and Stolzenberg [RS93]. In their terms, a streamless set is called **2**-good, where **2** is the set of two-element subsets of the natural numbers. They show that the sum of two **B**-good sets, of which **2**-good is an instance, is **B**-good, but leave it open for products. This paper does not resolve any of their open questions, as they work in

a more general setting than equality. They also give another notion, that of a set being *bar-good*, and they show that the Cartesian product of a bar-good set with a **B**-good set is **B**-good. It is not clear what the relation between streamlessness and bar-good is, and whether there are natural axioms one can assume to make a streamless set bar-good.

Veldman and Bezem [VB93] investigate the constructive content of the Ramsey theorem [Ram30], giving a constructive proof of a reformulation of it. For this, they use what they call *almost-full* binary relations; relations R on $\mathbb{N}$ where, for every *increasing* function $f : \mathbb{N} \to \mathbb{N}$,

$$\exists m, n : \mathbb{N}, m < n \wedge R(f(m), f(n)).$$

They postulate the axiom of bar-induction, and with it they prove that almost-full relations are closed under intersection. They name this the Intuitionistic Ramsey Theorem, and show that it is classically equivalent to Ramsey's Theorem.

Using equality as the relation R, one gets a notion which comes quite close to streamlessness, apart from Veldman and Bezem's requirement that the functions are increasing, and the fact that streamlessness is a concept applicable for any type (not only $\mathbb{N}$), possibly with undecidable equality.

In light of this, it is natural to ask whether the proofs in this paper can be generalized to relations other than equality. We define what it means for a reflexive and transitive relation R on A to be a well-quasi-ordering:

$$\mathrm{Wqo}_A(R) := \forall g : \mathbb{N} \to A, \exists i, j : \mathbb{N}, i < j \wedge R(g(i), g(j)).$$

Note that a set A is streamless exactly when we have $\mathrm{Wqo}_A(=_A)$. We can ask if the intersection of two such relations is itself a Wqo and whether the proof of Lemma 3.1 suggest how this could be shown. Unfortunately, we do not see how. We are still able to use the construction g^n to find n elements $a_1, \ldots a_n$ such that $a_1 R a_2 \ldots R a_n$, but we do not have the property that with n elements $b_1, \ldots b_n$ such that *none* of them are R-related, and $n-1$ elements $b'_1, \ldots b'_{n-1}$ such that none of them are R-related, there must be one of the $b_1, \ldots b_n$ which is not R related to any of the $b'_1, \ldots b'_{n-1}$. We have this property when the relation R is equality, and this is used in the proof of Lemma 3.1.

If we *did* have that Wqo relations were closed under intersection we would immediately get that streamless sets are closed under products: define the relation R_1 on $A \times B$ as $(a_1, b_1) R_1 (a_2, b_2)$ if and only if $a_1 =_A a_2$, and likewise for R_2, looking at the second projections. If A and B are streamless sets, then R_1 and R_2 are Wqo relations and their intersection is equality on $A \times B$.

Vytiniotis, Coquand and Wahlstedt [VCW12] provide an inductive formulation of almost full relations on arbitrary types. They show—if we instantiate their

proofs with the relation being equality—that it implies streamlessness, and show that almost-full relations are closed under intersection.

Streamlessness works in a quite general setting, with few assumptions on the underlying set. Bezem et al. [BNU12] impose further restrictions, and the result is a interesting hierarchy of finiteness notions. The restrictions imposed are that equality is decidable; that the subset is defined by some decidable predicate; and that the set is a subset of some set that can be enumerated. This holds for decidable subsets of natural numbers in particular. The authors find six different formalizations and put them into a hierarchy.

3.8 Remaining questions

There are several questions remaining. The main one is whether one can show that streamlessness is closed under Cartesian products in ITT without assuming decidable equality. Secondly, to what degree can one show similar results in systems without strong Σ elimination— for example, for StreamlessEx in Coq or the truncated statement in HoTT? And what is the relationship between StreamlessEx and StreamlessSig for sets with undecidable equality?

We conjecture that there exists a model showing that, in ITT, the product of two streamless sets with undecidable equality is not necessarily streamless. From Lemma 3.9 we know that such a model must reject functional extensionality, and from Lemma 3.3, we know that neither of the sets can be bounded.

At this point there are, to this author's knowledge, only two sets which are known to be streamless but not bounded. One is the set presented in [CS10], originally suggested by F. Richman, showing that not all noetherian sets are bounded. As noetherian sets are streamless, this is also a streamless set. But this set has the interesting property that, once one looks at any of the elements in the set, one knows the size of the set! So it is not bounded *a priori*, but if one is given a stream of elements from the set, one can deduce its size and then continue as in the proof of Lemma 3.3.

The second set, presented in the still unpublished article by Bezem et al. showing that not all streamless sets are noetherian, does not have this property. On the other hand, it has decidable equality, rendering it useless as a counter-model. There does not seem to be an easy way to tweak the model to get rid of this decidable equality; it is essential for the proof that the set is streamless as the authors use Markov's Principle to find the duplicate pair, and Markov's Principle is only applicable for decidable predicates.

To conclude, we currently have no good candidate for a streamless set with a non-streamless Cartesian product. Constructing a suitable streamless set, non-bounded and with undecidable equality, appears to be quite complicated. Neither

of the ways used to prove a set streamless—that is, by gathering information about the size of the set encoded in the elements themselves, or using Markov's principle—is likely to work. It seems the most promising route to a counter-model involves finding novel ways to construct streamless sets.

Lastly, we would like to encourage other to look for new notions of finiteness, especially trying to find notions that works nicely and robustly for sets with undecidable equality.

3.9 Conclusion

We showed that, in Martin-Löf intensional type theory, if at least one of the streamless sets A and B has decidable equality or is bounded, then the Cartesian product $A \times B$ is streamless. We also saw that adding functional extensionality to ITT gives streamless sets decidable equality; and we mirrored these results in both (E-) $\mathrm{HA}^{\omega} + \mathrm{AC}$ and in Coq.

Acknowledgements I want to thank Arnaud Spiwack and Thierry Coquand for their valuable feedback and questions after my presentation on this topic at TYPES 2014, and Marc Bezem for both introducing this problem to me and discussing it with me. I also wish to thank the anonymous reviewers for their challenging feedback which led to significant changes to this paper, and Maja Jaakson which kindly proofread it.

Noetherian Relations Are Streamless Even Without Identity Types

Erik Parmann

We give two proofs showing that all Noetherian relations are streamless. The first proof is simple, but uses a notion of equality on lists and substitution of equal elements in types. The second proof avoids these requirements and is expressible in type systems without equality.

4.1 Introduction

In a yet unpublished paper by Bezem, Coquand, Nakata, and Parmann [BCNP], we present a realizability model for Martin-Löf dependent type theory and use this model to show that not all streamless relations are Noetherian, even though the converse holds. The type theory does not have the usual, inductively defined identity types, which simplifies the soundness proof. But as a result, no known proof that Noetherian relations are streamless was expressible in the type theory since all known proofs use equality. In that paper, streamless and Noetherian are properties applicable to arbitrary binary relations, not only equality; so this state of affairs was rather unsatisfactory, in that we seemingly needed to introduce equality to show that all Noetherian relations are streamless.

The following note constitutes my main contribution to the aforementioned paper; it shows that all Noetherian relations are streamless even in a type theory *without* identity types, giving a proof that is expressible in our weak type theory.

This note begins with a presentation of the type theory in question in Section 4.2, followed by a presentation of Noetherian and streamless relations in Section 4.3. In Section 4.4 we give two proofs, one using equality and one expressible without equality, before we conclude in Section 4.5.

4.2 Preliminaries

We work in Martin-Löf dependent type theory without inductively defined identity, and with one universe U. In addition, we will use a number of types needed to express Bar induction and the two notions streamless and Noetherian. Notably, the type theory does *not* have an inductively defined equality.

We use the following indutive types, which are standard (A and B are arbitrary types).

- N_0 is the empty type with no constructor and elimination rule $\mathsf{ExF} : \mathsf{N}_0 \to A$;
- N is the type of natural numbers with constructors $\mathsf{0} : \mathsf{N}$ and $\mathsf{S} : \mathsf{N} \to \mathsf{N}$, and elimination rule $\mathsf{ind} : C\,\mathsf{0} \to (\Pi n{:}\mathsf{N}.\, C\,n \to C(\mathsf{S}n)) \to \Pi n{:}\mathsf{N}.\, C\,n$ for $C : \mathsf{N} \to \mathsf{U}$;
- $[A]$ is the type of lists over A, with constructors $\mathsf{nil} : [A]$ and $\mathsf{cons} : A \to [A] \to [A]$. We usually write $a :: l$ for $\mathsf{cons}\,a\,l$;
- $A + B$ is the sum type over A and B, with constructors $\mathsf{Inl} : A \to A + B$ and $\mathsf{Inr} : B \to A + B$, and elimination rule $\mathsf{Case} : (A \to C) \to (B \to C) \to A + B \to C$ for $C : \mathsf{U}$;

- $\Sigma x{:}A.\, D$ is the dependent-pair type over type A and a type-family D over A, and with the constructor "$(\_,\_)$" given by $(W, P) : \Sigma x{:}A.\, D$ when $\Sigma x{:}A.\, D$ is a type, $W : A$ and $P : D(W)$, and elimination rule $\mathsf{sInd} : (\Pi x{:}A.\, \Pi p{:}D.\, C(x,p)) \to \Pi y{:}(\Sigma x{:}A.\, D).\, C\, y$ for $C : (\Sigma x{:}A.\, D) \to \mathsf{U}$.

We also have a few types specifically needed to express the notion of streamless and Noetherian relations. We fully explain these, since they are less known than the standard types given above.

Given a predicate P on A and a list $l : [A]$, we define $\mathsf{exists}\, P\, l$ to be true when there is an element in l satisfying P:

$$\frac{\Gamma \vdash A : \mathsf{U}}{\Gamma \vdash \mathsf{exists} : (A \to \mathsf{U}) \to [A] \to \mathsf{U}}$$

with ι-reduction given by:

$$\begin{aligned} \mathsf{exists}\, P\, \mathsf{nil} &= \mathsf{N}_0 \\ \mathsf{exists}\, P\, (a :: l) &= P\, a + \mathsf{exists}\, P\, l \end{aligned}$$

We also define the predicate $\mathsf{good}\, R\, l$ for a relation $R : A \to A \to \mathsf{U}$ and a list $l : [A]$ to be true when l contains elements that are related by R in the order "head" to "tail":

$$\frac{\Gamma \vdash A : \mathsf{U}}{\Gamma \vdash \mathsf{good} : (A \to A \to \mathsf{U}) \to [A] \to \mathsf{U}}$$

with ι-reduction given by:

$$\begin{aligned} \mathsf{good}\, R\, \mathsf{nil} &= \mathsf{N}_0 \\ \mathsf{good}\, R\, (a :: l) &= \mathsf{exists}\, (R\, a)\, l + \mathsf{good}\, R\, l \end{aligned}$$

We define $\mathsf{f2l}\, f\, n$, which, given a function $f : \mathsf{N} \to A$, gives a list of the first n elements in f:

$$\frac{\Gamma \vdash A : \mathsf{U}}{\Gamma \vdash \mathsf{f2l} : (\mathsf{N} \to A) \to \mathsf{N} \to [A]}$$

with ι-reduction given by:

$$\begin{aligned} \mathsf{f2l}\, f\, 0 &= \mathsf{nil} \\ \mathsf{f2l}\, f\, (\mathsf{S}n) &= fn :: (\mathsf{f2l}\, f\, n) \end{aligned}$$

We will use $\overline{f}$ as a notation for $\mathsf{f2l}\, f$. Note that $\overline{f}$ "reverses" the order of the elements; earlier elements of the stream (indexed by a lower number) occur later in the resulting list. We also have the length function on lists:

$$\frac{\Gamma \vdash A : \mathsf{U}}{\Gamma \vdash \mathsf{length} : [A] \to \mathsf{N}}$$

with ι-reduction given by:

$$\mathsf{length}\ \mathsf{nil} = \mathsf{0}$$
$$\mathsf{length}\ (h :: t) = \mathsf{S}(\mathsf{length}\ t)$$

Finally, we have the type for Bar induction:

$$\frac{\Gamma \vdash A : \mathsf{U}}{\Gamma \vdash \mathsf{bar} : ([A] \to \mathsf{U}) \to [A] \to \mathsf{U}}$$

$$\frac{\Gamma \vdash A : \mathsf{U} \qquad \Gamma \vdash l : [A] \qquad \Gamma \vdash P : [A] \to \mathsf{U} \qquad \Gamma \vdash X : Pl}{\Gamma \vdash \mathsf{base}\, X : \mathsf{bar}\, P\, l}$$

$$\frac{\Gamma \vdash A : \mathsf{U} \qquad \Gamma \vdash l : [A] \qquad \Gamma \vdash P : [A] \to \mathsf{U} \qquad \Gamma \vdash Y : \Pi a{:}A.\, \mathsf{bar}\, P\, (a :: l)}{\Gamma \vdash \mathsf{step}\, Y : \mathsf{bar}\, P\, l}$$

The elimination principle for Bar induction is given by:

$$\frac{\Gamma \vdash A : \mathsf{U} \qquad \Gamma \vdash P : [A] \to \mathsf{U} \qquad \Gamma \vdash C : [A] \to \mathsf{U}}{\Gamma \vdash \mathsf{barInd}{:}(\Pi l{:}[A].\, P\, l \to C\, l) \to (\Pi l{:}[A].\, (\Pi a{:}A.\, C(a{::}l)) \to C\, l) \to \Pi l{:}[A].\, \mathsf{bar}\, P\, l \to C\, l}$$

4.3 Streamless and Noetherian

Both streamless and Noetherian have the type $((A \to A \to \mathsf{U}) \to \mathsf{U})$, so they are properties of binary relations. But they can also be used to define finiteness of sets, see e.g. [CS10], where a *set* is Noetherian (or streamless) when *equality* on the set is Noetherian (or streamless). In this note, we will work in the more general setting of arbitrary binary relations.

A relation is Noetherian when the empty list is a bar for $\mathsf{good}\, R$. Note that, Bar induction gives us an induction principle for Noetherian relations.

Definition 4.1 (Noetherian relation)**.** A relation $R : A \to A \to \mathsf{U}$ on a type $A : \mathsf{U}$ is *Noetherian* when $\mathsf{bar}\, (\mathsf{good}\, R)\, \mathsf{nil}$ holds.

There are several definitions of streamless relations in the literature. Some of these are given below, where R^{-1} is the inverse relation of R.

$$\Pi f{:}\mathsf{N} \to A.\, \Sigma i\, j{:}\mathsf{N}.\, i < j \wedge R(f(i), f(j)) \tag{4.1}$$
$$\Pi f{:}\mathsf{N} \to A.\, \Sigma i\, j{:}\mathsf{N}.\, i < j \wedge R^{-1}(f(i), f(j)) \tag{4.2}$$
$$\Pi f{:}\mathsf{N} \to A.\, \Sigma n{:}\mathsf{N}.\, \mathsf{good}\, R\, (\overline{f}n) \tag{4.3}$$
$$\Pi f{:}\mathsf{N} \to A.\, \Sigma n{:}\mathsf{N}.\, \mathsf{good}\, R^{-1}\, (\overline{f}n) \tag{4.4}$$

For symmetric relations, all of the above formalizations are equivalent; but for non-symmetric relations, formalisation 4.1 and formalisation 4.4 are equivalent, as are 4.2 and 4.3.

With the definition of Noetherian relations used in this note, we cannot expect Noetherian relations to be streamless in the form of formalisation 4.1. The relation $R : \mathsf{N} \to \mathsf{N} \to \mathsf{U}$ given by

$$R(a, b) := a \geq b$$

is Noetherian; lists grow to the left and there are no infinite descending chains of natural numbers. But R is certainly not streamless in terms of formalisation 4.1 above, as a constantly increasing stream falsifies it.

This note makes use of formalisation 4.3:

Definition 4.2 (Streamless relation)**.** A relation $R : A \to A \to \mathsf{U}$ on a type $A : \mathsf{U}$ is *streamless* if every function $f : \mathsf{N} \to A$ from natural numbers to A has a initial segment that is R-good, i.e., $\Pi f{:}\mathsf{N} \to A.\, \Sigma n{:}\mathsf{N}.\, \mathsf{good}\, R\, (\overline{f}n)$.

Coquand and Spiwack [CS10] sketch a proof that Noetherian *sets* are streamless, and show this for formalization 4.1. Since equality is symmetric, all the different formalizations become equivalent in the context of streamless sets. Even if we change the notion of streamless to one suiting a non-symmetric relation, it is not clear how to express their proof without using equality and substitution.

4.4 Noetherian implies Streamless

We now present two proofs that Noetherian relations are streamless. The first proof requires a theory with equality on lists and substitution. We cannot express this proof in our weak type theory, since we don't have these notions; but the proof is still interesting as an intuitive and straightforward proof. Our final proof—which *is* expressible in our type theory—is based on the former proof, but replaces equality with weaker notions.

In addition to the presentation here, both proofs are formalized in Coq.[1] Coq has a significantly stronger theory than the type theory we work in here, so the formalization only shows that the final proof is correct, not that it is expressible in our type theory. By manually examining the final proof however, we can see that it only uses features of our type theory.

4.4.1 Proof using equality

For this proof, we assume a bit of extra machinery—in particular, inductively defined equality. We write $\vdash_e$ in place of $\vdash$ to indicate that we are working in the

[1] See `https://github.com/epa095/noetherian-implies-streamless` for the Coq script.

extended type theory. We start with identity types:

$$\frac{\Gamma \vdash_e A : \mathsf{U}}{\Gamma, a : A, a' : A \vdash_e \mathsf{eq}_A\, a\, a' : \mathsf{U}} \qquad \frac{\Gamma \vdash_e A : \mathsf{U} \quad \Gamma \vdash_e a : A}{\Gamma \vdash_e \mathsf{refl}\, a : \mathsf{eq}_A\, a\, a}$$

with the eliminator allowing us to substitute equal terms in types:

$$\frac{\Gamma \vdash_e A : \mathsf{U} \quad \Gamma, a : A, a' : A, \alpha : \mathsf{eq}_A\, a\, a' \vdash_e C(a, a', \alpha) : \mathsf{U}}{\Gamma \vdash_e \mathsf{eqInd} : (\Pi a{:}A.\, C\, a\, a\, (\mathsf{refl}\, a)) \to \Pi a\, a'{:}A.\, \Pi \alpha{:}\mathsf{eq}_A\, a\, a'.\, C\, a\, a'\, \alpha}$$

Note that from the recursor above, known as the J-combinator, we can get the Leibniz property of equality which we will use in the proof.

$$\frac{\Gamma \vdash_e A : \mathsf{U} \quad \Gamma \vdash_e a : A \quad \Gamma \vdash_e P : A \to \mathsf{U}}{\Gamma \vdash_e \mathsf{eqInd}' : Pa \to (\Pi a'{:}A.\, \mathsf{eq}_A\, a\, a' \to Pa')}$$

We define the abbreviation $\mathsf{initialP}$ to be true of a stream $f : \mathsf{N} \to A$ and a list $l : [A]$ when l is equal to the initial segment of length $\mathsf{length}\, l$ of f :

$$\mathsf{initialP}\, l\, f := \mathsf{eq}_{[A]}\, (\overline{f}\, (\mathsf{length}\, l))\, l.$$

Lemma 4.3. *There is a proof M such that*

$$A : \mathsf{U}, R : A \to A \to \mathsf{U}, f : \mathsf{N} \to A \vdash_e$$
$$M : \left[\Pi l{:}[A].\, ((\mathsf{bar}\, (\mathsf{good}\, R)\, l) \to \mathsf{initialP}\, l\, f \to \Sigma m{:}\mathsf{N}.\, \mathsf{good}\, R\, (\overline{f}\, m))\right]$$

Proof. We perform induction on $\mathsf{bar}\, (\mathsf{good}\, R)\, l$ using the elimination principle barInd, with P being $\lambda l{:}[A].\mathsf{good}\, R\, l$ and C being

$$\lambda l{:}[A].\mathsf{initialP}\, l\, f \to \Sigma m{:}\mathsf{N}.\, \mathsf{good}\, R\, (\overline{f}\, m).$$

We show both the base case $H_b : \Pi l{:}[A].\, P\, l \to C\, l$ and the induction step $H_i : \Pi l{:}[A].\, (\Pi a{:}A.\, C(a :: l)) \to C\, l$, with the consequence that $\mathsf{barInd}\, H_b\, H_i$ is the desired M and hence prove the lemma.

To construct H_b assume $l : [A]$ such that $\mathsf{good}\, R\, l$ and $\mathsf{initialP}\, l\, f$.[2] Since the latter is $\mathsf{eq}_{[A]}\, (\overline{f}\, (\mathsf{length}\, l))\, l$ we get $\mathsf{good}\, R\, (\overline{f}\, \mathsf{length}\, l)$ immediately by eqInd'.

For H_i we assume $l : [A]$, and $\Pi a{:}A.\, C(a :: l)$, and proceed to show $C(l)$. The latter expands to $\mathsf{initialP}\, l\, f \to \Sigma m{:}\mathsf{N}.\, \mathsf{good}\, R\, (\overline{f}\, m)$, so we assume $\mathsf{initialP}\, l\, f$.

Expanding $\Pi a{:}A.\, C(a :: l)$ gives

$$\Pi x{:}A.\, (\mathsf{initialP}\, (x :: l)\, f \to \Sigma m{:}\mathsf{N}.\, \mathsf{good}\, R\, (\overline{f}\, m)).$$

[2] With this we mean that we have inhabitants of the types $\mathsf{good}\, R\, l$ and $\mathsf{initialP}\, l\, f$. We will mostly avoid the phrase "an inhabitant of", except in situations where giving the inhabitant a name eases the reading of the proof. The formalization in Coq contains all inhabitants.

We apply this to $f(\mathsf{length}\, l)$, which after expanding $\mathsf{initialP}$ leaves us with showing

$$\mathsf{eq}_{[A]}\,(\overline{f}\,(\mathsf{length}\,(f(\mathsf{length}\, l) :: l)))\,(f(\mathsf{length}\, l) :: l),$$

which ι-reduces to

$$\mathsf{eq}_{[A]}\,(f\,(\mathsf{length}\, l) :: \overline{f}\,(\mathsf{length}\, l))\,(f(\mathsf{length}\, l) :: l),$$

which we get by eqInd' and the assumption $\mathsf{initialP}\, l\, f$, completing the proof. □

Since $\mathsf{initialP}\,\mathsf{nil}\, f$ holds for all $f : \mathsf{N} \to A$, we immediately get that all Noetherian relations are streamless.

Corollary 4.4. *There is a proof M such that*

$$A : \mathsf{U}, R : A \to A \to \mathsf{U} \vdash_e M : \mathit{Noetherian}\, R \to \mathit{streamless}\, R$$

4.4.2 Proof avoiding equality

In this section, we present a proof which is expressible even without a notion of equality on lists and substitution of equal terms in types, making this proof expressible in our weak type theory.

We first define two abbreviations, both applicable to $R : A \to A \to \mathsf{U}$, $f : \mathsf{N} \to A$ and $l : [A]$:

$$\begin{aligned}\mathsf{subG}\, R\, f\, l &:= \mathsf{good}\, R\, l \to \mathsf{good}\, R\, (\overline{f}\,(\mathsf{length}\, l))\\ \mathsf{subEx}\, R\, f\, l &:= \Pi a{:}A.\,(\mathsf{exists}\,(R\, a)\, l \to \mathsf{exists}\,(R\, a)\,(\overline{f}\,(\mathsf{length}\, l)))\end{aligned}$$

Intuitively, these capture what is needed to show the desired implication, and nothing more. $\mathsf{subG}\, R\, f\, l$ is used to show the base case of the induction, while $\mathsf{subEx}\, R\, f\, l$ is used to show the induction step. Also note that they both hold trivially for nil.

Lemma 4.5. *There is a proof M such that*

$$A : \mathsf{U}, R : A \to A \to \mathsf{U}, f : \mathsf{N} \to A \vdash$$
$$M{:}\Pi l{:}[A].\mathsf{bar}\,(\mathsf{good}\, R)\, l \to (\mathsf{subG}\, R\, f\, l \to (\mathsf{subEx}\, R\, f\, l \to \Sigma m{:}\mathsf{N}.\,\mathsf{good}\, R\,(\overline{f}\, m))).$$

Proof. We perform induction on $\mathsf{bar}\,(\mathsf{good}\, R)\, l$ using the elimination principle barInd, with the predicate P being $\lambda l : [A].\mathsf{good}\, R\, l$, and the predicate C being $\lambda l : [A].\mathsf{subG}\, R\, f\, l \to \mathsf{subEx}\, R\, f\, l \to \Sigma m{:}\mathsf{N}.\,\mathsf{good}\, R\,(\overline{f}\, m)$. We show $H_b : \Pi l{:}[A].\, P\, l \to C\, l$ and $H_i : \Pi l{:}[A].\,((\Pi a{:}A.\, C(a :: l)) \to C l)$, with the consequence that $\mathsf{barInd}\, H_b\, H_i$ is the desired M, and hence prove the lemma.

To construct H_b assume $l : [A]$ such that $\mathsf{good}\, R\, l$, $\mathsf{subG}\, R\, f\, l$, and $\mathsf{subEx}\, R\, f\, l$. From $\mathsf{subG}\, R\, f\, l$ and $\mathsf{good}\, R\, l$ we immediately get the goal $\mathsf{good}\, R\, (\overline{f}\, (\mathsf{length}\, l))$.

For H_i we assume $l : [A]$ with $(\Pi a{:}A.\, C(a :: l))$, and proceed to show $C\, l$. The latter expands to $\mathsf{subG}\, R\, f\, l \to \mathsf{subEx}\, R\, f\, l \to \Sigma m{:}\mathsf{N}.\, \mathsf{good}\, R\, (\overline{f}\, m)$, so we assume $\mathsf{subG}\, R\, f\, l$, and $\mathsf{subEx}\, R\, f\, l$, and proceed to show $\Sigma m{:}\mathsf{N}.\, \mathsf{good}\, R\, (\overline{f}\, m)$. Expanding the induction hypotheses $(\Pi a{:}A.\, C(a :: l))$ gives

$$\Pi x{:}A.\, (\mathsf{subG}\, R\, f\, (x :: l) \to (\mathsf{subEx}\, R\, f\, (x :: l) \to \Sigma m{:}\mathsf{N}.\, \mathsf{good}\, R\, (\overline{f}\, m)))$$

We apply this to $f(\mathsf{length}\, l)$, which yields the goal if we can populate the two assumptions. Starting with the first, we need to show (where we have expanded the abbreviation subG):

$$\mathsf{good}\, R\, (f(\mathsf{length}\, l) :: l) \to \mathsf{good}\, R\, (\overline{f}\, (\mathsf{length}\, (f(\mathsf{length}\, l) :: l)))$$

Since $(\mathsf{length}\, (f(\mathsf{length}\, l) :: l))$ reduces to $S(\mathsf{length}\, l)$, the conclusion of the above formula reduces to $\mathsf{good}\, R\, (f(\mathsf{length}\, l) :: \overline{f}\, (\mathsf{length}\, l))$, which again reduces to the sum-type

$$\mathsf{exists}\, (R\, (f(\mathsf{length}\, l)))\, (\overline{f}\, (\mathsf{length}\, l)) + \mathsf{good}\, R\, (\overline{f}\, (\mathsf{length}\, l)),$$

which we proceed to show.

The antecedent $\mathsf{good}\, R\, (f(\mathsf{length}\, l) :: l)$ reduces to $\mathsf{exists}\, (R\, (f\, (\mathsf{length}\, l)))\, l + \mathsf{good}\, R\, l$, and we apply Case, the eliminator for sum. If $\mathsf{exists}\, (R\, (f\, (\mathsf{length}\, l)))\, l$ then we apply the proof of $\mathsf{subEx}\, R\, f\, l$ to $f(\mathsf{length}\, l)$, and get the left part of the goal directly. In the case of $\mathsf{good}\, R\, (f(\mathsf{length}\, l) :: l)$ from $\mathsf{good}\, R\, l$, we get $\mathsf{good}\, R\, (\overline{f}\, (\mathsf{length}\, l))$ from $\mathsf{subG}\, R\, f\, l$, giving the right part of the goal and finishing the proof of $\mathsf{subG}\, R\, f\, (x :: l)$.

The second assumption, $\mathsf{subEx}\, R\, f\, (x :: l)$, expands to:

$$\Pi a{:}A.\mathsf{exists}\, (R\, a)\, (f(\mathsf{length}\, l) :: l) \to \mathsf{exists}\, (R\, a)\, (\overline{f}\, (\mathsf{length}\, (f(\mathsf{length}\, l) :: l)))$$

We assume an $a : A$ with $\mathsf{exists}\, (R\, a)\, (f(\mathsf{length}\, l) :: l)$. Since $(\overline{f}\, (\mathsf{length}\, (f(\mathsf{length}\, l) :: l)))$ reduces to $f(\mathsf{length}\, l) :: \overline{f}\, (\mathsf{length}\, l)$, the goal reduces to the sum-type

$$(R\, a\, (f(\mathsf{length}\, l))) + \mathsf{exists}\, (R\, a)\, (\overline{f}\, (\mathsf{length}\, l)).$$

If $\mathsf{exists}\, (R\, a)\, (f(\mathsf{length}\, l) :: l)$ stems from $R\, a\, (f\, (\mathsf{length}\, l))$ then we have left part of the goal directly. If we have $\mathsf{exists}\, (R\, a)\, (f(\mathsf{length}\, l) :: l)$ from $\mathsf{exists}\, (R\, a)\, l$, then applying the proof of $\mathsf{subEx}\, R\, f\, l$ to a gives $\mathsf{exists}\, (R\, a)\, l \to \mathsf{exists}\, (R\, a)\, \overline{f}\, (\mathsf{length}\, l)$, giving the right part of the goal. This finishes the proof of $\mathsf{subEx}\, R\, f\, (x :: l)$, finishing the construction of H_i, and we are done. □

Since $\mathsf{subG}\, R\, f\, \mathsf{nil}$ and $\mathsf{subEx}\, R\, f\, \mathsf{nil}$ holds for all $f : \mathsf{N} \to A$, we get that all Noetherian relations are streamless.

Corollary 4.6. *There is a proof M such that*

$$A : \mathsf{U}, R : A \to A \to \mathsf{U} \vdash M : \mathit{Noetherian}\, R \to \mathit{streamless}\, R$$

4.5 Conclusion

In this note, we have presented a proof that all Noetherian relations are streamless, even without a notion of identity. Avoiding equality makes it easier to construct a model of the type theory in question, which again can be used to show that the converse implication is not provable in type theory.

Non-Constructivity in Kan Simplicial Sets

Marc Bezem, Thierry Coquand, and Erik Parmann

We give an analysis of the non-constructivity of the following basic result: if X and Y are simplicial sets and Y has the Kan extension property, then Y^X also has the Kan extension property. By means of Kripke countermodels we show that even simple consequences of this basic result, such as edge reversal and edge composition, are not constructively provable. We also show that our unprovability argument will have to be refined if one strengthens the usual formulation of the Kan extension property to one with explicit horn-filler operations.

5.1 Introduction

Brouwer's Programme is the constructive reformulation of (as much as possible of) classical mathematics. In [BC15] it has been shown that the following theorem, though classically true (cf. [May93, Corollary 7.11]), cannot be proved constructively.

Theorem 5.1 (classical)**.** *The fibers of* 0 *and* 1 *of a Kan fibration* $p : E \to \Delta^1$ *are homotopy equivalent.*

In this paper we show that the following basic theorems cannot be proved constructively.

Theorem 5.2 (classical)**.** *If* X *and* Y *are Kan simplicial sets, then any edge in* Y^X *can be reversed.*

Theorem 5.3 (classical)**.** *If* X *and* Y *are Kan simplicial sets, then compatible edges in* Y^X *can be composed.*

The above two theorems follow immediately and constructively from the following.

Theorem 5.4 (classical)**.** *If* X *and* Y *are Kan simplicial sets, then also* Y^X *is so.*

Hence we obtain that also Theorem 5.4, though classically true even without requiring that X is Kan (cf. [May93, Theorem 6.9]), cannot be proved constructively.

The importance of these results is twofold. First, it is of evident importance for Brouwer's Programme to understand which results of classical mathematics already are constructive and which results are not. Second, Theorem 5.4 plays a crucial role in the construction of models of type theory with the Univalence Axiom, see [KLV12]. The use of classical logic in proving this crucial property implies in particular that the model construction cannot be used to give a computational interpretation of univalence. Actually, Theorem 5.4 is a necessary step in the semantics of the simply typed λ-calculus based on Kan simplicial sets. In what follows we expand on these points; for more motivation we refer to [BC15].

We would like to use the occasion to say a few not-too-technical words on the role of the Kan extension property of simplicial sets in relation to univalence. Let MLTT be Martin-Löf type theory with universe U and inductive equality $=_U$ on U. Assume we have two distinct copies of the natural numbers, inductively defined by constructors $0 : N$ $(0' : N')$ and $S : N \to N$ $(S' : N' \to N')$. MLTT proves $N =_U N$ and $N' =_U N'$, but not $N =_U N'$. The Univalence Axiom (UA) implies that (homotopy) equivalent types are equal, and in particular $N =_U N'$. By the

Leibniz property of inductive equality, this implies that N and N' have the same properties and that all structure on N can be transported to N' and vice versa. This holds even uniformly, for example, $\Pi P : U \to U.\ (PN \to PN')$ is inhabited under UA. On the other hand, without UA, $\Pi P : U \to U.\ (PN \to PN')$ is not inhabited in MLTT. (One reason is that MLTT has models in which $N \neq N'$, so one can take $P \equiv \lambda X : U.\ (N =_U X)$ and get PN but not PN').

The above observation concerns not only the rather artificial type N' but also any other type that is equivalent to N, such as the type of lists over a unit type with one object. In fact the observation concerns all equivalent types. A less artificial example is perhaps the equivalence of the unit type to $\Sigma x : A.\ (a =_A x)$ for given $a : A : U$. The upshot is that validating UA requires an interpretation of $=_U$ that carries much more information than in MLTT without UA, since the elimination rule for $=_U$ (roughly, the Leibniz property, or substitutivity of equals for equals) has to be much stronger. In our simple example, the interpretation of $=_U$ must be leveraged to give an inhabitant of $\Pi P : U \to U.\ (PN \to PN')$.

Simplicial sets can be used to build a presheaf-style [Hof97] model of MLTT. In this model the interpretation of $=_U$ does not validate UA. It turns out that if one builds a model of MLTT based on Kan simplicial sets, then it is possible to validate UA. The crucial notion here is that of a Kan fibration. A Kan *fibration* $p : E \to B$ is a map of simplicial sets with a specific lifting property. This lifting property *lifts* a path from b_0 to b_1 in B to a transport function from the fiber $p^{-1}(b_0)$ to the fiber $p^{-1}(b_1)$. In the model based on Kan simplicial sets, an inhabitant of $N =_U N'$ is interpreted as a path from N to N' in U. (Here and below we omit the correct but tedious phrase *the interpretation of* $N, N', U, \ldots$). Any $P : U \to U$ is interpreted as a Kan fibration with fibers PT for any $T : U$. (NB the fibration, being a projection on the base type, has a direction opposite to the arrow in $P : U \to U$). Then the transport function obtained from the lifting property is the desired function $PN \to PN'$. In short, one can say that the transport functions interpret substitutivity of equals by equals.

Finally, to come back to the topic of this paper: if all types are to have Kan structure, one has to prove this inductively following the rules of type formation. One of the induction steps is Theorem 5.4. The unprovablility of Theorem 5.4 shows that, from the constructive point of view, there is a problem with using the exponent Y^X in the category of Kan simplicial sets to interpret function types $X \to Y$.

The type theoretic (synthetic) formulation of homotopy equivalence and the Univalence Axiom, as well as the model of MLTT plus UA using Kan simplicial sets are all due to Voevodsky [Voe09, KLV12]. This model confirms the homotopical interpretation proposed by Awodey and Warren [AW09].

Theorem 5.4 (without requiring that X is Kan) has an interesting history. The

first appearance seems to be [Moo56, Appendix A, p. 1A-8, Theorem 3]. Moore credits A. Heller for the definition of the function space Y^X on page 1A-4. Moore's proof is combinatorial, using the excluded middle in distinguishing the cases a non/degenerate on page 1A-9, l. 17ff. (Typo: on page 1A-7, l. 12 and 15, the map F is missing on the rhs; evidently $F_{(\mu,\nu)}$ was intended to depend on F.) The proof in [May93, Theorem 6.9] is much the same as the one by Moore (with the F's in place). Several variations of this argument can be found in the literature.

An essentially more abstract proof using anodyne extensions is given by Gabriel and Zisman in [GZ67, Chapter Four, 3.1.2] (take $B = \Delta^0$). Here the classical reasoning shows up when in 2.1.2 amalgamated sums over sets of non-degenerate simplices are taken.

The results of Moore and Heller imply that Kan simplicial sets form a cartesian closed category, which can be seen as a germ of the fact that they model dependent type theory.

The rest of the paper is structured as follows. In Section 5.2 we give an introduction to simplicial sets, and in Section 5.3 we provide several examples of simplicial sets which will be in use in the rest of the article. In Section 5.4 we take a closer look at Theorem 5.2, and provide a Kripke model showing that a constructive consequence, Lemma 5.10 cannot be proven constructively. Section 5.5 deals with edge compostion, much in the same way as Section 5.4 deals with edge reversal. A summary and evaluation of the results obtained so far is given in Section 5.6. In Section 5.7 we strengthen the Kan condition and prove constructively a weak version of Lemma 5.10. This shows that our unprovability argument will have to be refined for the stronger Kan condition. We sum up our findings and discuss further research in Section 5.8.

5.2 Preliminaries

Definition 5.1 (Simplicial set)**.** A *simplicial set* A is a collection of sets $A[i]$ for $i \in \mathbb{N}$ such that for every $0 < n$ and $j \leq n$ we have a function (*face map*) $d^n_j : A[n] \to A[n-1]$, and for every $0 \leq n$ and $j \leq n$ we have a function (*degeneracy map*) $s^n_j : A[n] \to A[n+1]$, satisfying the following *simplicial identities* for all suitable superscripts, which we happily omit:

$$d_i d_j = d_{j-1} d_i \qquad \text{if } i < j \tag{5.1}$$

$$d_i s_j = s_{j-1} d_i \qquad \text{if } i < j \tag{5.2}$$

$$d_i s_j = id \qquad \text{for } i = j, j+1 \tag{5.3}$$

$$d_i s_j = s_j d_{i-1} \qquad \text{if } i > j+1 \tag{5.4}$$

$$s_i s_j = s_j s_{i-1} \qquad \text{if } i > j \tag{5.5}$$

An element of $A[i]$ is called an *i-simplex*, or just simplex when we don't wish to stipulate the dimension. A *degenerate* element is any element $a \in A[i+1]$ in the image of a degeneracy map.

Note that a simplicial identity like, e.g., $d_i^n d_j^{n+1} = d_{j-1}^n d_i^{n+1}$ actually means

$$\forall x \in A[n+1].\ d_i^n(d_j^{n+1}(x)) = d_{j-1}^n(d_i^{n+1}(x)).$$

With a countably infinite signature, the above definition can be expressed completely in many-sorted first-order logic. That means that we can see first-order models which satisfy the above requirement as simplicial sets, and instead of simplicial sets we could talk about first-order models satisfying the above requirements.

Simplicial sets form a category. For two simplicial sets A and B, $Hom_S(A, B)$ is the set of all natural transformations from A to B. A natural transformation is a collection of maps $g[n] : A[n] \to B[n]$ commuting with the face and degeneracy maps of A and B: $g[n]s_i = s_i g[n-1]$ for all $0 \leq i < n$ and $g[n+1]d_i = d_i g[n]$ for all $0 \leq i \leq n+1$. We freely omit the dimension $[n]$ when it can be inferred from the other arguments. For more information on simplicial sets we refer to, for example, [May93, GJ09, Fri08].

Definition 5.2 (Kan simplicial set)**.** A simplicial set Y satisfies the Kan condition if for any collection of simplices $y_0, \ldots, y_{k-1}, y_{k+1}, \ldots, y_n$ in $Y[n-1]$ such that $d_i y_j = d_{j-1} y_i$ for any $i < j$ with $i \neq k$ and $j \neq k$, there is an n-simplex y in Y such that $d_i y = y_i$ for all $i \neq k$. The Kan condition is also called the Kan extension property, and a simplicial set is called a *Kan simplicial set* if it satisfies the Kan condition.

Definition 5.3 (Kan graph)**.** A *reflexive multigraph* consists of C_1, C_0, d_0, d_1, s where C_0 is a set of points, C_1 a set of edges, $d_i : C_1 \to C_0$, d_1 the *source* and d_0 the *target* function, and $s : C_0 \to C_1$ the function mapping each $c \in C_0$ to a selfloop of c. We write $e : a \to b$ if e is in C_1 such that $d_1(e) = a$ and $d_0(e) = b$ (note the direction!). In particular we have $d_i(s(c)) = c$ for all $c \in C_0$. A *Kan graph* is a reflexive multigraph having the property that for all a, b, c in C_0, if $e : a \to b$ and $f : a \to c$, then there exists an edge $g : b \to c$ in C_1.

Kan graphs can be viewed as truncated Kan simplicial sets, modelling a truncated proof-relevant equality relation. Note that we don't require the Kan graph to have explicit functions giving the required edges like in [BC15], we merely require that the edges exists. We discuss this distinction further in Section 5.7. The special requirement of the edges for the Kan graph is in the literature often called *Euclidean*. Euclidean combined with reflexivity gives both transitivity and symmetry.

5.3 Examples of simplicial sets

We give some examples of simplicial sets that are used in the sequel.

5.3.1 Standard simplicial k-simplex Δ^k

Δ^k is the simplicial set with $\Delta^k[j]$ consisting of all non-decreasing sequences of numbers $0, \ldots, k$ of length $j+1$. Equivalently, $\Delta^k[j]$ is the set of order-preserving functions $[j] \to [k]$, where $[i]$ denotes $0, \ldots, i$ with the natural ordering. Examples are $\Delta^1[0] = \{0, 1\}$, $\Delta^1[1] = \{00, 01, 11\}$, $\Delta^2[1] = \{00, 01, 02, 11, 12, 22\}$ and

$$\Delta^2[2] = \{000, 001, 002, 011, 012, 022, 111, 112, 122, 222\}.$$

The degeneracy map $s^j_k : \Delta^i[j] \to \Delta^i[j+1]$ duplicates the k-th element in its input. So, $s^j_k(x_0 \ldots x_k \ldots x_{j+1}) = x_0 \ldots x_k x_k \ldots x_{j+1}$. The face map $d^j_k : \Delta^i[j] \to \Delta^i[j-1]$ deletes the k-th element. So, $d^j_k(x_0 \ldots x_j) = x_0 \ldots x_{k-1} x_{k+1} \ldots x_j$.

5.3.2 The k-horns Λ^k_j

Λ^k_j is the j'th horn of the standard k-simplex Δ^k, and defined by $\Lambda^k_j[n] = \{f \in \Delta^k[n] \mid [k] - \{j\} \not\subseteq Im(f)\}$. Alternatively, it is $\Delta^k[n]$ except every element must avoid some element not equal to j. For example, $\Lambda^2_0[1] = \{00, 01, 02, 11, 12, 22\} = \Delta^2[1] - \{12\}$ (excluding 12, since 12 does not avoid any element not equal to 0). We also have:

$$\Lambda^2_0[2] = \{000, 001, 002, 011, 012, 022, 111, 112, 122, 222\}.$$

The Kan extension condition for a simplicial set Y can also be formulated as: every map $F : \Lambda^k_j \to Y$ can be extended to a map $F' : \Delta^k \to Y$. This is equivalent to Definition 5.2.

5.3.3 Cartesian products

For two simplicial sets A and B, $A \times B$ is the simplicial set given by $(A \times B)[i] = A[i] \times B[i]$, and the structural maps d and s use d^A and d^B component-wise (and likewise for s^A and s^B). So if $a \in A[i]$ and $b \in B[i]$ then $(a, b) \in (A \times B)[i]$, and $d_i((a, b)) = (d^A_i(a), d^B_i(b))$. In particular, the degenerate simplices of $A \times B$ are pairs $(s^A_j(a), s^B_j(b)) \in (A \times B)[i+1]$. (Caveat: this is stronger than both components being degenerate.)

5.3.4 Function spaces

We give the standard definition [Moo56, p. 1A-4]: Y^X is the simplicial set given by $Y^X[i] = Hom_S(\Delta^i \times X, Y)$, where Hom_S denotes morphisms (natural transformations) of simplicial sets, and structural maps as follows. The face maps $d_k[i] : Y^X[i] \to Y^X[i-1]$ need to map elements of $Hom_S(\Delta^i \times X, Y)$ to $Hom_S(\Delta^{i-1} \times X, Y)$ and the degeneracy maps vice versa. For their definition it is convenient to view a k-simplex in Δ^i as an order-preserving function $a : [k] \to [i]$. Let d_k^* be the strictly increasing function on natural numbers such that $d_k^*(n) = n$ if $n < k$ and $d_k^*(n) = n + 1$ otherwise (d_k^* 'jumps' over k). Given $F \in Hom_S(\Delta^i \times X, Y)$, define $(d_k F)[i](a, x) = F[i](d_k^* a, x)$. For the degeneracy maps, let s_k^* be the weakly increasing function on natural numbers such that $s_k^*(n) = n$ if $n \leq k$ and $s_k^*(n) = n - 1$ otherwise (s_k^* 'duplicates' k). Then define $(s_k F)[i](a, x) = F[i](s_k^* a, x)$.

5.3.5 The simplicial set defined by a reflexive multigraph

The following definition from [BC15] gives the general construction of a simplicial set from a reflexive multigraph. It is important to note that, even if the reflexive multigraph is transitive, its simplicial set is not the same as the nerve [GJ09, Example 1.4] of the category defined by the multigraph. The difference is subtle: if we have edges $f : x \to y,\ g : y \to z,\ h, h' : x \to z$, where the composition $gf = h$, then the nerve does not contain the 2-simplex with f, g, h', in contrast to below.

Definition 5.4. Given a reflexive multigraph C we define the simplicial set $S(C)$ as follows. $S(C)[0] = C_0$, $S(C)[1] = C_1$ and $S(C)[n]$, for $n \geq 2$, consisting of all tuples of the form $(u_0, \dots, u_n; \dots, e_{ij}, \dots)$ such that

$$e_{ij} : u_i \to u_j \text{ in } C_1 \text{ for all } 0 \leq i < j \leq n.$$

The maps d_k in $S(C)$ are defined by removing from $(u_0, \dots, u_n; \dots, e_{ij}, \dots)$ the point u_k and all edges e_{ik} and e_{kj}. The maps s_k in $S(C)$ are defined by duplicating the point u_k in $(u_0, \dots, u_n; \dots, e_{ij}, \dots)$, adding an edge $e_{k(k+1)} = s(u_k)$, and duplicating edges and incrementing indices of edges as appropriate. This completes the construction of the simplicial set $S(C)$.

We now see why Kan graphs are named as they are: the S construction above turns them into Kan simplicial sets.

Lemma 5.5. *$S(Y)$ is a Kan simplicial set whenever Y is a Kan graph.*

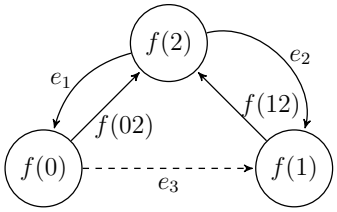
Proof. Consider Λ_k^n for some $n \geq 1$ and $0 \leq k \leq n$ and let $f : \Lambda_k^n \to S(Y)$. We have to define a lifting $h : \Delta^n \to S(Y)$. Δ^n consists of elements in every dimension,

but we only need to specify h for every element in $\Delta^n[n]$. This since both the higher and lower dimensional objects are the (possibly repeated) s_i or d_j images of objects in $\Delta^n[n]$, and h must commute with both s_i and d_j, which determines h.

If $n = 1$, note that that Λ^1_k only consists of one point, and degenerations of that point in the higher dimensions. E.g., if $k = 0$ then $\Lambda^1_0[0] = \{0\}$, $\Lambda^1_0[1] = \{00\}$ etc. In that case we extend f to $h : \Delta^1 \to S(Y)$ by mapping $h(1) = h(0) = f(0)$, which determines h in higher dimensions.

If $n = 2$ we use the fact that Y is a Kan graph, so for any two edges $f : a \to b$ and $g : a \to c$ there is an edge from b to c. The 2-horn gives two edges in the graph with at least one common point, and the fact that the graph is both reflexive, symmetric and transitive (because of the Kan property) enables us to find a third edge with compatible endpoints. The procedure depends on the value of k. We will here give the procedure for $k = 2$; $k = 0, 1$ are just simple adaptations.

Given $f : \Lambda^2_2 \to S(Y)$ we have edges $f(02) : f(0) \to f(2)$ and edges $f(12) : f(1) \to f(2)$, and we need to find a value for $h(01) : f(0) \to f(1)$ such that $d_1 h(01) = d_1 f(02)$ and $d_0 h(01) = d_1 f(12)$. In other words, we need to find that the dotted edge in the diagram actually exists (self-loops are not displayed).



Recall that $S(Y)[1] = Y[1]$, so both $f(01)$ and $f(12)$ are actual edges in Y. By applying the Kan property on $s(f(0))$ and $f(02)$ we get an edge $e_1 : f(2) \to f(0)$. Similarly we get an edge $e_2 : f(2) \to f(1)$. Now, by using the Kan property on e_1 and e_2 we get an edge $e_3 : f(0) \to f(1)$, and we put $h(01) = e_3$.

Finally, if $n \geq 3$ we observe that the horn Λ^n_k contains all points and edges of Δ^n, and we define the lifting by

$$h(q) = (f_{[0]}(q(0)), \ldots, f_{[0]}(q(m)); \ldots, f_{[1]}(e_{ij}), \ldots).$$

Here $q : [m] \to [n]$ is order-preserving and e_{ij} is the edge from $q(i)$ to $q(j)$ in $\Delta^n[1] = \Lambda^n_k[1]$. □

5.4 Edge reversal

In this section we give the classical proof of Theorem 5.2 and show that there is no constructive proof.

5.4.1 Edge reversal, definition and classical proof

Definition 5.6 (Edge reversal)**.** A simplicial set Y is said to have *edge reversal* when for every edge $e \in Y[1]$ there exists an edge $f \in Y[1]$ with $d_1(f) = d_0(e)$ and $d_0(f) = d_1(e)$.

Lemma 5.7. *Kan simplicial sets have edge reversal.*

Proof. Given an arbitrary Kan simplicial set Y and an edge $e \in Y[1]$ we can make a map $G : \Lambda^2_0 \to Y$ by letting $G(0) = G(2) = d_1(e)$, $G(1) = d_0(e)$, $G(01) = e$ and $G(02) = s(d_1(e))$. Since Y is Kan we can extend G to $G : \Delta^2 \to Y$, giving us a value for $G(12) \in Y[1]$, which must be an edge between $G(1)$ and $G(2) = G(0)$, giving the reverse edge. □

We introduce some convenient ad-hoc terminology for later use.

Definition 5.8 (Y^X*-good*)**.** Let X and Y are reflexive multigraphs and $F_{01} : X[1] \to Y[1]$. Define $F_0 = d_1 F_{01} s : X[0] \to Y[0]$ and $F_1 = d_0 F_{01} s : X[0] \to Y[0]$. We say that F_{01} is Y^X*-good* when the following two requirements hold for $i = 0, 1$:

- For all $e, e' \in X[1]$, if $d_i(e) = d_i(e')$ then $d_i F_{01}(e) = d_i F_{01}(e')$;
- For all $e \in X[1]$, $F_i d_0(e) = F_i d_1(e)$.

The first requirement expresses that F_{01}, F_0, F_1 respect endpoints, that is, if $e : a \to b$ in $X[1]$, then $F_{01}(e) : F_0(a) \to F_1(b)$ in $Y[1]$. The second requirement ensures that F_0 and F_1 are constant on each weakly connected component of X. (Notice that $F_{01}s(y)$ for $y \in Y[0]$ does not need to map to a degenerate edge, so F_0 and F_1 are not necessarily identical.)

Lemma 5.9. *If X and Y are reflexive multigraphs and $F_{01} : X[1] \to Y[1]$ is Y^X-good, then we can extend F_{01} to a 1-simplex in $S(Y)^{S(X)}$.*

Proof. To be a map in $S(Y)^{S(X)}$ we need to extend $F_{01} : X[1] \to Y[1]$ to a family of maps $F'_{01}[n] : (\Delta^1 \times S(X))[n] \to S(Y)[n]$ which commute with d_i and s_j. Recall the definitions $F_0 = d_1 F_{01} s$ and $F_1 = d_0 F_{01} s$. We define $F'_{01}[n]$ depending on n. If $n = 0$ then the input will have the form (i, x) where $0 \le i \le 1$ and $x \in X[0]$, and we put $F'_{01}[0](i, x) = F_i(x)$. If $n = 1$ the input will have the form (ij, e) where $0 \le i \le j \le 1$ and $e \in X[1]$. If $i = j$ we put $F'_{01}(ij, e) = sF_i(d_0(e))$. Note that since F_{01} is Y^X-good, we know that $F_i(d_0(e)) = F_i(d_1(e))$, justifying our choice of the degenerate edge as the output. If $i < j$ we let $F'_{01}(01, e) = F_{01}(e)$. If $n > 1$ any input to $F'_{01}[n]$ will have the form $(0^a 1^b, (x_0, \dots, x_n; \dots e_{ij}, \dots))$ such that $a + b = n + 1$. We let $F'_{01}[n]$ map this element to the tuple

$$(F_0(x_0), \dots, F_0(x_{a-1}), F_1(x_a) \dots, F_1(x_{a+b-1}); \dots e'_{ij}, \dots),$$

where $e'_{ij} = s(F_0(x_a))$ if $i < j < a$, $e'_{ij} = F_{01}(e_{ij})$ if $i < a \leq j$, and $e'_{ij} = s(F_1(x_a))$ if $a \leq i < j$. That is, the $F'_{01}[n]$ images are sequences of a number of F_0 images followed by b number of F_1 images, with all edges being degenerate, except the bridges between the two nodes. Since each of the derived F_i functions are constant on each connected component, and the input consists exactly of sequences of nodes in the same connected component, all of the elements $F_0(x_0), \ldots, F_0(x_{a-1})$ are the same element in $Y[0]$, and likewise for $F_1(x_a) \ldots, F_1(x_{a+b-1})$. This justifies our choice of e'_{ij} as the degenerate edges.

It should be clear that this map does indeed commute with d_i and s_j, completing the proof. □

Lemma 5.10 (classical)**.** *For all Kan graphs Y and X, if $F_{01} : X[1] \to Y[1]$ is Y^X-good, then there is an $F_{10} : X[1] \to Y[1]$ such that $d_0F_{01} = d_1F_{10}$ and $d_1F_{01} = d_0F_{10}$.*

Proof. Let X and Y be Kan graphs. The $S(Y)$ and $S(X)$ are Kan simplicial sets by Lemma 5.5. By applying the classical Theorem 5.2 we get that $S(Y)^{S(X)}$ has edge reversal. Since F_{01} is Y^X-good we extend F_{01} to an edge $F'_{01} \in S(Y)^{S(X)}[1]$ as defined in the proof of Lemma 5.9. By edge reversal in $S(Y)^{S(X)}$ we get an $F'_{10} \in S(Y)^{S(X)}[1]$ satisfying $d_1(F'_{10}) = d_0(F'_{01})$ and $d_0(F'_{10}) = d_1(F'_{01})$. We put $F_{10}(x) = F'_{10}(01, x)$. By expanding the definition of d_k from Section 5.3.4, we get the following properties: $F'_{10}(00, e) = F'_{01}(11, e)$ and $F'_{10}(11, e) = F'_{01}(00, e)$, giving $F'_{10}(0, d_i(e)) = F'_{01}(1, d_i(e))$ and $F'_{10}(1, d_i(e)) = F'_{01}(0, d_i(e))$. We calculate $d_0F_{01}(e) = d_0F'_{01}(01, e) = F'_{01}(1, d_0(e)) = F_1d_0(e)$. Since F_{01} is Y^X-good (2nd requirement) we have $F_1d_0(e) = F_1d_1(e)$. We continue the calculation: $F_1d_1(e) = F'_{01}(1, d_1(e)) = F'_{10}(0, d_1(e))$ where the last step is justified above. We continue: $F'_{10}(0, d_1(e)) = d_1F'_{10}(01, e) = d_1F_{10}(e)$. In total we have proved $d_0F_{01}(e) = d_1F_{10}(e)$ for all $e \in X[1]$. Hence $d_0F_{01} = d_1F_{10}$. The other equation is proved symmetrically. □

Kripke [Kri65] showed that constructive logic is sound for Kripke models, so the existence of a Kripke countermodel of a statement gives the non-existence of a constructive proof of that statement. We will now, by the means of a Kripke model, see that Lemma 5.10 does not hold constructively.

5.4.2 Edge reversal, the Kripke countermodel

We describe a Kripke model containing a Y^X-good F_{01} such that there cannot be a function $F_{10} : X[1] \to Y[1]$ with $d_0F_{01} = d_1F_{10}$ and $d_1F_{01} = d_0F_{10}$, even though X and Y are Kan graphs.

	Day 1
X_0	$\{x, x'\}$
X_1	$\{s(x), s(x')\}$
Y_0	$\{y_0, y_1, y'_0, y'_1\}$
Y_1	$\{s(y_0), s(y_1), s(y'_0), s(y'_1), y_0y_1 : y_0 \to y_1, y'_0y'_1 : y'_0 \to y'_1, a : y_1 \to y_0, b : y'_1 \to y'_0\}$

	Day 2
X_0	$\{x{=}x'\}$
X_1	$\{s(x){=}s(x')\}$
Y_0	$\{y_0 = y'_0, y_1 = y'_1\}$
Y_1	$\{s(y_0) = s(y'_0), s(y_1) = s(y'_1), y_0y_1 = y'_0y'_1, a, b\}$

	Input	Output
F_0	x	y_0
F_0	x'	y'_0
F_1	x	y_1
F_1	x'	y'_1
F_{01}	$s(x)$	y_0y_1
F_{01}	$s(x')$	$y'_0y'_1$

Table 5.1: Kripke (counter)model for edge reversal.

For clarity, the functions $F_0 = d_1F_{01}s$ and $F_1 = d_0F_{01}s$ as defined in Definition 5.8 are also made explicit in this model. Face maps are part of the model, but not made explicit.

The model consists of two days, with an X and a Y part each. On day 1 both X and Y consist of two separate components, which get merged on day 2. We give the model both in Table 5.1 and, graphically, in Figure 5.1 and 5.2.

It is easy to see that both X and Y are Kan graphs by simply observing that each of their two components are strongly connected. It is also clear that we cannot define a consistent F_{10}. In day 1 we would have to set $F_{10}(s(x)) = a$ and $F_{10}(s(x')) = b$ to satisfy the requirement that $d_0F_{01} = d_1F_{10}$ and $d_1F_{01} = d_0F_{10}$. The problem occurs in day 2, where we have that $s(x) = s(x')$, but $a \neq b$, making it impossible for F_{10} to respect equality. Note that all other functions, F_0, F_1, F_{01}, s, d_0, and d_1 remain consistent after collapsing, that is, they still map equal elements to equal elements.

5.5 Edge composition

In this section we give the classical proof of Theorem 5.3 and show that there is no constructive proof.

Definition 5.11 (Edge composition)**.** A simplicial set Y is said to have *edge composition* when for every edge $e_1, e_2 \in Y[1]$, if $d_0(e_1) = d_1(e_2)$ then there exists an edge $f \in Y[1]$ with $d_1(f) = d_1(e_1)$ and $d_0(f) = d_0(e_2)$.

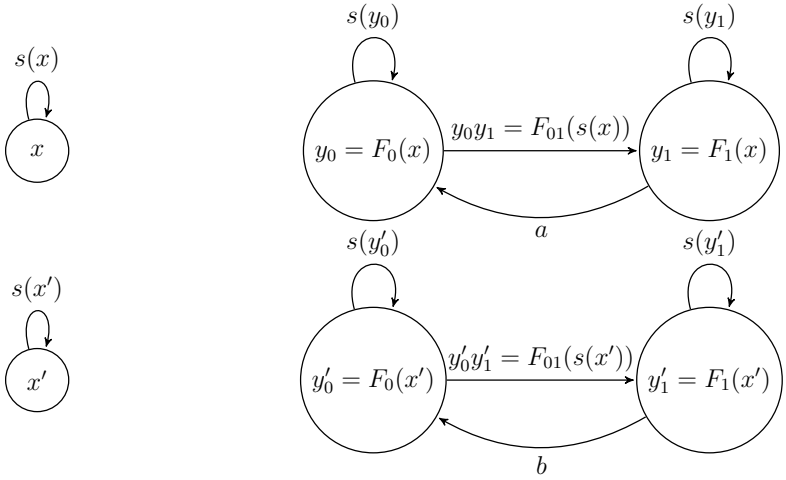


Figure 5.1: Kripke (counter)model for edge reversal, day 1.

Lemma 5.12. *Kan simplicial sets have edge composition.*

Proof. Given an arbitrary Kan simplicial set Y and edges $e_1, e_2 \in Y[1]$ with $d_0(e_1) = d_1(e_2)$, we can make a map $G : \Lambda^2_1 \to Y$ by putting $G(0) = d_1(e_1)$, $G(1) = d_0(e_1)$, $G(2) = d_0(e_2)$ $G(01) = e_1$ and $G(12) = e_2$. Since Y is Kan we can extend G to $G : \Delta^2 \to Y$, giving us a simplex $G(02) : G(0) \to G(2)$ in $Y[1]$, the composition of e_1 and e_2. □

By a proof essentially identical to the proof of Lemma 5.10 we get the following lemma

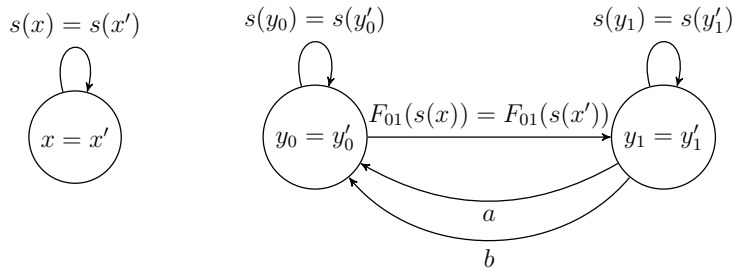


Figure 5.2: Kripke (counter)model for edge reversal, day 2.

Lemma 5.13 (classical)**.** *For all Kan graphs Y and X, if $F_{01} : X[1] \to Y[1]$ and $F_{12} : X[1] \to Y[1]$ are Y^X-good maps satisfying $d_0F_{01} = d_1F_{12}$, then there is an $F_{02} : X[1] \to Y[1]$ such that $d_0F_{01} = d_0F_{02}$ and $d_1F_{12} = d_1F_{02}$.*

In Figure 5.3 and 5.4 we see that Lemma 5.13 is not constructively provable. We have two Y^X-good functions F_{01} and F_{12}, satisfying the requirement, and both X and Y are Kan graphs. If $S(Y)^{S(X)}$ had edge composition we would get a function F_{02} that $d_1F_{01} = d_1F_{02}$ and $d_0F_{12} = d_0F_{02}$. However, such a function is not definable in the Kripke model. The reason is analogous to the case of edge-reversal: from day 1 to day 2 we have equated objects in the domain of F_{02} while keeping the images distinct. Specifically, on day 1 we are forced to set $F_{02}(s(x)) = a$ and $F_{02}(s(x')) = b$, but on day 2 we have $s(x) = s(x')$, but $a \neq b$.

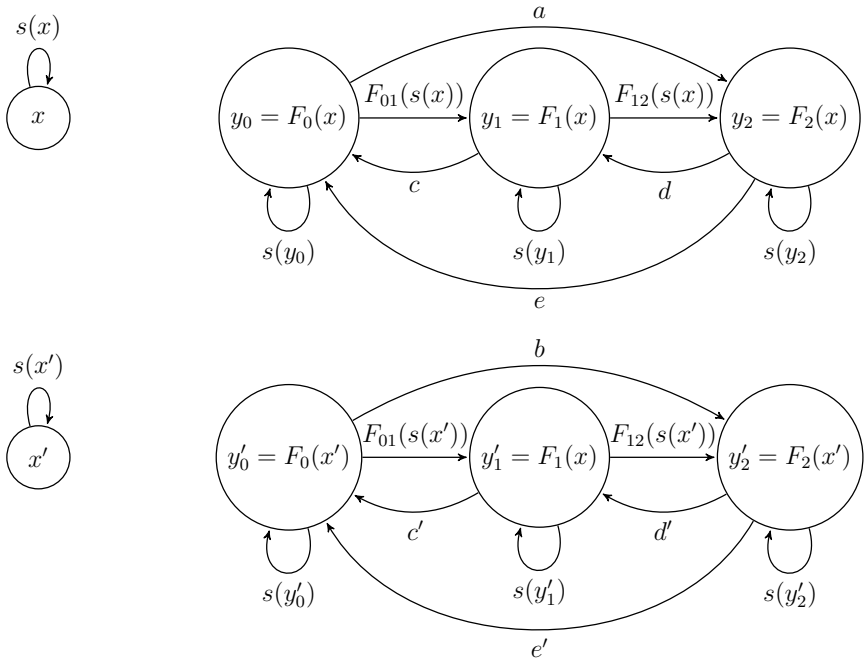


Figure 5.3: Kripke (counter)model for edge composition, day 1.

5.6 Evaluation of the results

The results up to now are summarized in Figure 5.5. Having concrete, finite Kripke

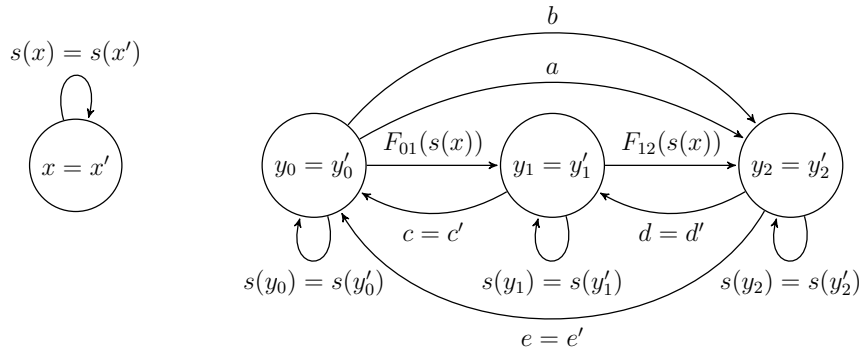


Figure 5.4: Kripke (counter)model for edge composition, day 2.

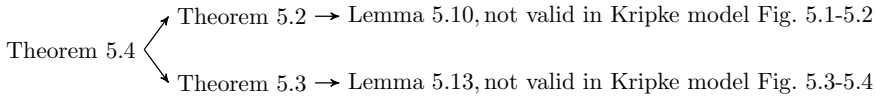


Figure 5.5: Summary of results, all implications constructive.

countermodels against Lemma 5.10 and 5.13 allows for a further simplification: everything remains valid under the condition that X has at most two points. Likewise, explicit bounds read off from the Kripke models can be imposed on the number of points of Y and on the number of edges in X and in Y. The simplified results are denoted by postfixing the number of the result by a 'b' for bounded, so Lemma 5.10b is the bounded version of Lemma 5.10.

With explicit bounds on the size of the domain, functions are completely determined by a finite number of function values. For example, if we have $\forall z \in X.\ (z = x \vee z = x')$ for $x, x' \in X$, then the binary predicate $\mathrm{fun}(y, y') \equiv (x = x' \to y = y')$ on Y completely describes all functions $X \to Y$, in evidence $x \mapsto y,\ x' \mapsto y'$. With this in mind it is not difficult to express Lemma 5.10b as a first-order classical tautology Φ that is not true in all Kripke models.

Now fix a constructive framework that is sufficiently expressive for the results in Figure 5.5. For example, IZF (Zermelo-Fraenkel set theory in IPL, intuitionistic predicate logic) will do. Let $[|\Phi|]$ be the Tarski interpretation of Φ expressed in IZF. The following fundamental property of IZF could be called the semantic conservativity of IZF over IPL:

If $[|\Phi|]$ is provable in IZF, then Φ is true in all Kripke models.

Lubarsky [Lub15] and McCarty [McC15] independently provided constructive

proofs of the above conservativity property of IZF. We gratefully acknowledge their prompt answers to our question.[1]

Empowered by the proofs of Lubarsky and McCarty we can now conclude that Lemma 5.10b cannot be proved in IZF. The same is true for Lemma 5.13b, and for all other results in Figure 5.5, as well as for their bounded versions.

5.7 Kan graphs with explicit filler functions

Let us first give an intuitive explanation of our countermodels. They actually exploit the undecidability of equality: on day 1 we don't know what will be equal on day 2. (This is different from the decidability of degeneracy, but the two are related: for example, an edge e is degenerate iff $e = s_0(d_1(e))$.) In Figure 5.1 and 5.2, the point is that $y_0 \neq y_0'$ on day 1, so one cannot put $F_{10}(s(x)) = F_{10}(s(x')) = a$ since this conflicts with $d_0F_{10} = d_1F_{01}$. One is thus forced to a choice that turns out to be wrong on day 2.

One attempt to deal with this lack of information is to give Kan simplicial sets more structure. One could for example change Definition 5.2 of a Kan simplicial set into one where we not only know that the required n-simplex exists, but actually have *functions* producing them. In the formulation using horns as in Section 5.3.2 this would amount to a dependent function $\mathrm{fill}(k, j, F)$ such that $\mathrm{fill}(k, j, F) : \Delta^k \to Y$ extends $F : \Lambda^k_j \to Y$, for any k, j, F. This form of Kan simplicial set has been introduced by Nikolaus in [Nik11] under the name of *algebraic Kan complex*. The definition with explicit fill-functions has certain advantages, both classically and constructively, as we will see below. However, one should be careful in defining Y^X: morphisms in the category of algebraic Kan complexes are required to map chosen fillers in X to chosen fillers in Y. As a consequence, there are less maps from X to Y as algebraic Kan complexes than as just simplicial sets. What we propose could be called a *functional* Kan simplicial set, with explicit fill-functions but with maps as for ordinary simplicial sets. As a consequence the exponential Y^X of simplicial sets can be used.

To be able to prove an analogue of Lemma 5.5 we have to strengthen the notion of Kan graph to also include such *filler functions*, cf. [BC15].

Definition 5.14 (Kan fill-graph)**.** A *Kan fill-graph* is a reflexive multigraph with a partial function $\mathrm{fill} : Y[1] \times Y[1] \to Y[1]$ such that for all $e_1, e_2 \in Y[1]$, if $e_1 : a \to b$ and $e_2 : a \to c$, then $\mathrm{fill}(e_1, e_2) : b \to c$.

[1] Strengthening the semantic conservativity to syntactic conservativity, that is, concluding that Φ is provable in intuitionistic predicate logic, by using the completeness of the Kripke semantics implicates some classical logic. Although not needed for this paper, we think there is some general interest in a constructive proof that IPL $\vdash \Psi$ whenever IZF $\vdash [|\Psi|]$, for any first-order sentence Ψ.

As noted earlier, the Kan property together with reflexivity implies symmetry and transitivity. We can now define the corresponding functions.

Definition 5.15 (Edge reversal)**.** For all $e \in Y[1]$ where Y is a Kan fill-graph let

$$e^{-1} = \text{fill}(e, sd_1(e)).$$

If $e : a \to b$, then $sd_1(e) : a \to a$, and $\text{fill}(e, sd_1(e)) : b \to a$.

Note that we in general don't have $(e^{-1})^{-1} = e$, but we do have that $d_i((e^{-1})^{-1}) = d_i(e)$.

Definition 5.16 (Edge composition)**.** Using the inverse for edges in Y we define the composition of two edges $e_1 : a \to b$ and $e_2 : b \to c$ as

$$\text{trans}(e_1, e_2) = \text{fill}(e_1^{-1}, e_2).$$

Again we are in no way guaranteed that $\text{trans}(e_1, s(b)) = e_1$ or $\text{trans}(s(x), s(x)) = s(x)$.

We immediately see that the addition of explicit functions adds power, as we can now prove constructively and trivially an analogue of Lemma 5.10.

Lemma 5.17. *For all Kan fill-graphs Y, X and for every $F : X[1] \to Y[1]$, the function $F^{-1} : X[1] \to Y[1]$ defined by $F^{-1}(e) = F(e)^{-1}$ satisfies $d_0F = d_1F^{-1}$ and $d_1F = d_0F^{-1}$.*

Note how using explicit functions rules out the Kripke counter-example we gave of Lemma 5.10. If $s(x) = s(x')$ on day 2, then we immediately get $a = F_{01}^{-1}(s(x)) = F_{01}^{-1}(s(x')) = b$ since equality has to be preserved.

We can even use the above fact to show that:

Lemma 5.18. *For any reflexive multigraph X and Kan fill-graph Y, $S(Y)^{S(X)}$ has edge reversal.*

Proof. Assume an edge $F \in S(Y)^{S(X)}[1]$, we proceed to define F^{-1} such that $d_0(F) = d_1(F^{-1})$ and $d_1(F) = d_0(F^{-1})$. As $F \in S(Y)^{S(X)}[1]$ we have $F[n] : \Delta^1[n] \times X[n] \to Y[n]$. We start with $n = 0$, defining $F^{-1}[0] : (\Delta^1 \times X)[0] \to Y[0]$ by letting $F^{-1}[0](0, x) = F[0](1, x)$ and $F^{-1}[0](1, x) = F[0](0, x)$. Likewise for $n = 1$ we define $F^{-1}(00, e) = F(11, e)$ and $F^{-1}(11, e) = F(00, e)$, these are directly enforced by $d_0(F) = d_1(F^{-1})$ and $d_1(F) = d_0(F^{-1})$. For the case of $F^{-1}(01, e)$ we need to find an edge $F^{-1}(01, e) : F^{-1}(0, d_1e) \to F^{-1}(1, d_0e)$, which from the way we defined $F^{-1}[0]$ is the same as an edge

$$F^{-1}(01, e) : F(1, d_1e) \to F(0, d_0e).$$

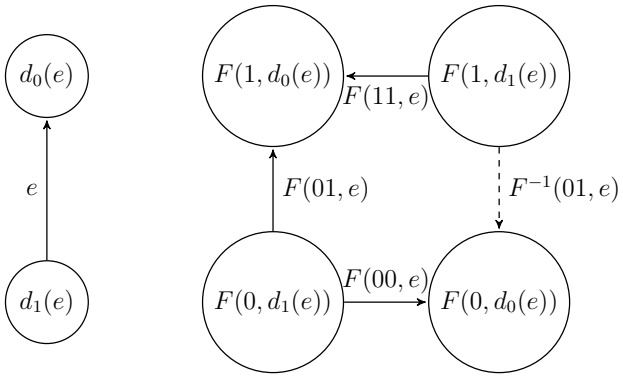


Figure 5.6: Reversing F.

The diagram in Figure 5.6 shows $e \in S(X)[1]$ with its endpoints on the left, and the nodes and edges we have directly reachable in $S(Y)$ using only F on the right. Reading off the figure we can define $F^{-1}(01, e)$ as follows:

$$F^{-1}(01, e) = \text{trans}(F(11, e)), \text{fill}(F(01, e), F(00, e)))$$

Note that F^{-1} is well-defined since the functions involved in the definition are. Moreover, F^{-1} commutes with s_0, d_0, d_1 by construction.

Having defined F^{-1} for dimension 0 and 1, F^{-1} is also determined in higher dimensions, because of the truncation in $S(X), S(Y)$. In the case of $n > 1$ any input to $F^{-1}[n]$ will have the form

$$F^{-1}(0^a 1^b, (x_0, \dots, x_n; \dots e_{ij}, \dots))$$

where $a + b = n + 1$. We let $F^{-1}[n]$ map this element to the tuple

$$(F^{-1}(0, x_0), \dots, F^{-1}(0, x_{a-1}), F^{-1}(1, x_a) \dots, F^{-1}(1, x_{a+b-1}); \dots e'_{ij}, \dots),$$

where $e'_{ij} = F^{-1}(00, e_{ij})$ if $i < j < a$, $e'_{ij} = F^{-1}(01, e_{ij})$ if $i < a \leq j$, and $e'_{ij} = F^{-1}(11, e_{ij})$ if $a \leq i < j$. This commutes with face and degeneracy maps. □

Using the same techniques we can constructively prove the following variant of Lemma 5.13.

Lemma 5.19. *For any Kan graph X and Kan fill-graph Y, if $F_{01} : X[1] \to Y[1]$ and $F_{12} : X[1] \to Y[1]$ satisfy $d_0 F_{01} = d_1 F_{12}$, then there is a $F_{02} : X[1] \to Y[1]$ such that $d_1 F_{01} = d_1 F_{02}$ and $d_0 F_{12} = d_0 F_{02}$.*

Lemma 5.20. *For any reflexive multigraph X and Kan fill-graph Y, $S(Y)^{S(X)}$ has edge composition.*

Proof. Assume edges $F_{01} \in S(Y)^{S(X)}$, $F_{12} \in S(Y)^{S(X)}$ such that $d_0(F_{01}) = d_1(F_{12})$, and we proceed to define $F_{02} \in S(Y)^{S(X)}$ such that $d_1(F_{02}) = d_1(F_{01})$ and $d_0(F_{02}) = d_0(F_{12})$.

As was the case in the proof of Lemma 5.18, we are forced on $F_{02}(0, x) = F_{01}(0, x)$, $F_{02}(1, x) = F_{12}(1, x)$, $F_{02}(00, e) = F_{01}(00, e)$, and $F_{02}(11, e) = F_{12}(11, e)$.

For the case of $F_{02}(01, e)$ we need to find an edge $F_{02}(01, e) : F_{02}(0, d_1 e) \to F_{02}(1, d_0 e)$, which from the way we defined $F_{02}[0]$ is the same as an edge

$$F_{02}(01, e) : F_{01}(0, d_1 e) \to F_{12}(1, d_0 e).$$

We note that $d_0(F_{01}) = d_1(F_{12})$ enforces $F_{01}(11, e) = F_{12}(00, e)$, which again enforces $F_{01}(1, d_i(e)) = F_{12}(0, d_i(e))$. This gives the diagram in Figure 5.7, enabling us to read off:

$$F_{02}(01, e) = \text{fill}(\text{trans}(F_{01}(11, e), F_{01}(01, e)^{-1}), F_{12}(01, e)).$$

□

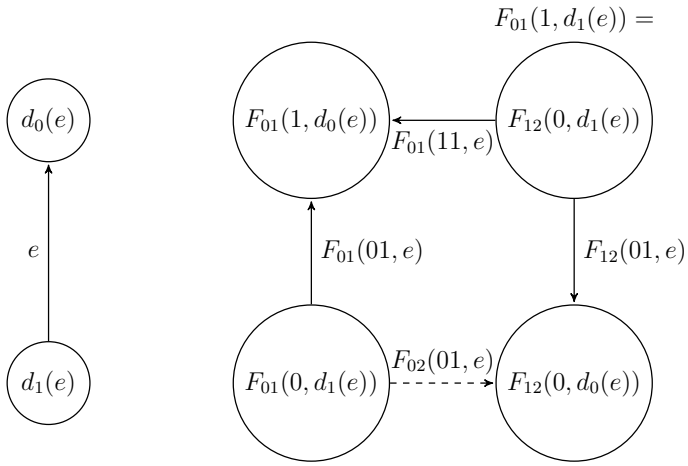


Figure 5.7: Filling the horn Λ^2_1.

5.8 Conclusions and Future Research

We have given a thorough analysis of the non-constructivity of the basic result that the Kan extension property is preserved under the usual operation of exponentiation

of simplicial sets. An important step in this analysis, also employed in [BC15], is the truncation of simplicial sets to dimension 1. This allows us to study the basic result in the simplified situation of Kan graphs. Once one has shown the constructive unprovability of the basic result in the situation of Kan graphs, one obtains *a fortiori* its unprovability for Kan simplicial sets.

The much simpler notion of Kan graph (as compared to Kan simplicial set) invites to further thought experiments. One of those is the study of simple, constructive consequences of the Kan extension property, such as edge reversal and edge composition. It turns out that already these consequences cannot be proven constructively.

Another experiment is to strengthen the Kan extension property from existence of an n-simplex as in Definition 5.2 to having a function, called a *filler*, yielding these n-simplices. This makes quite a difference. None of the Kripke models we have introduced is able to deal with such fillers, since equating objects in X and Y implies that filler-values such as a and b in Figure 5.1 also have to be equal. The question arises whether this is necessary so, or just coincidental in the particular Kripke model. This question is answered in Section 5.7, where we prove constructively that, if X is a graph and Y a Kan-fill graph, then $S(Y)^{S(X)}$ has edge reversal and edge composition. This result may be of independent interest. It suggests that showing the (expected) constructive unprovability of Theorem 5.4 for *algebraic Kan complexes* as in [Nik11] will require more complicated structures than graphs. The above expectation is based on an analysis of filling a 2-horn in Y^X, which requires defining $F(001, t)$. As F has to commute with s_0, one must know whether the 2-simplex t is an s_0-image or not. This can in general only be decided by an appeal to classical logic. We have to leave this to future research.

Functional Kan Simplicial Sets: Non-Constructivity of Exponentiation

Erik Parmann

Functional Kan simplicial sets are simplicial sets in which the horn-fillers required by the Kan extension condition are given explicitly by functions. We show the non-constructivity of the following basic result: if B and A are functional Kan simplicial sets, then A^B is a Kan simplicial set. This strengthens a similar result for the case of non-functional Kan simplicial sets shown by Bezem, Coquand and Parmann [BCP15]. Our result shows that—from a constructive point of view—functional Kan simplicial sets are, as it stands, unsatisfactory as a model of even simply typed lambda calculus. Our proof is based on a rather involved Kripke countermodel which has been encoded and verified in the Coq proof assistant.

6.1 Introduction

In this paper, we show that the following theorem cannot be constructively proven in Intuitionistic Zermelo-Fraenkel (IZF) set theory.

Theorem 6.1 (classical)**.** *If B and A are* functional *Kan simplicial sets, then A^B is a Kan simplicial set.*

We showed a similar result in [BCP15][1], but for non-functional Kan simplicial sets. We will introduce (functional Kan) simplicial sets properly in the next section; for now, we will explain what is needed to characterize the crucial difference between functional and non-functional Kan simplicial sets.

A simplicial set consist of a family of sets $A[i], i \in \mathbb{N}$ with certain functions going between them, such that these functions satisfy the so-called *simplicial identities*. A Kan simplicial set is a simplicial set which is, in some sense, "full": it satisfies that, for every compatible n-tuple of elements in $A[n-1]$, there exists a compatible element in $A[n]$, using the meaning of "compatible" given in Definition 6.2.

Functional and non-functional Kan simplicial sets differ only in that the expression "for every... there exists..." is given a constructive interpretation. Although classical mathematics easily passes—by applying the axiom of choice—from elements existing to functions giving those elements, constructive mathematics does not take this so lightly. Constructively, all functional Kan simplicial sets are Kan simplicial sets, but the converse does not hold unless we adopt the axiom of choice, which—depending on the context—makes the logic classical [Dia75].

Theorem 6.1 is true classically, even without requiring that B is Kan (cf. [Moo56, Appendix A, Theorem 3] or [May93, Theorem 6.9] for a more modern approach), and plays an important role when using Kan simplicial sets as a model of type theory. The way we prove that Theorem 6.1 cannot be constructively proven is to show that the following constructive consequence of it cannot be constructively proven.

Theorem 6.2 (classical)**.** *If B and A are functional Kan simplicial sets, then any edge in A^B can be reversed.*

In [BCP15] we gave a Kripke counterexample to the constructive provability of Theorem 6.1 for *non-functional* Kan simplicial sets, showing that the appeal to classical logic in the proofs is essential. We did this by showing that certain graph-like, first-order structures can be constructively extended to Kan simplicial sets; and by using the corresponding version of Theorem 6.2 on the resulting simplicial set, we got that the graphs have a particular feature we can call *function-space*

[1] Recall that [BCP15] occurs as Chapter 5 in this thesis.

edge reversal. We then showed that the same class of structures does not have function-space edge reversal constructively.

Unfortunately, the countermodel only yields a *non-functional* Kan simplicial set, and we showed that if we assume explicit filler functions, then the simplicial sets induced by graphs always have function-space edge reversal. This shows that a simple tweak of the model is not sufficient; we need structures other than simple graphs. More precisely, we conjectured that a countermodel must be a hypergraph containing at least three dimensions of a simplicial set—not only points and edges, but also triangles—and this might significantly increase the complexity.

The present paper provides such a Kripke countermodel. In addition to the extra complexity of the new dimension, it also contains explicit filler functions respecting equality (the equality relation must be a congruence). Since this Kripke model equates elements (as the one in [BCP15]), ensuring congruence turns out to be quite involved. To validate correctness, we have encoded and verified the model in the Coq [CDT12] proof assistant.

The simplicial set A^B in Theorem 6.1 is not claimed to functional Kan. This makes Theorem 6.1 weaker than if we had required A^B to be functional Kan, strengthening the non-provability result in this paper. It also means that the present paper properly generalizes [BCP15].

In [BC15] it was shown that the homotopy equivalence of the fibers of a functional Kan fibration over a connected base cannot be proved constructively. The techniques used in the present paper are strongly inspired by [BC15].

The first section of [BCP15] provides a introduction as to why Kan simplicial sets are interesting from a type-theoretical perspective. In short, Kan simplicial sets can be used to build a model of Martin-Löf Type Theory(MLTT)[KLV12] with the homotopy theoretic interpretation of equality, and in this construction, Theorem 6.1 is important for the interpretation of function types. The results in [BC15] show that this construction, with equality interpreted as homotopy equivalences, is fundamentally non-constructive. This result closes one of the possible paths to finding a computational interpretation of the Univalence Axiom.

In [BCP15] we showed that an even more fundamental part of the Kan simplicial set model of Type Theory—the interpretation of function types—is fundamentally non-constructive for *non-functional* Kan simplicial sets. The present paper shows the same for *functional* Kan simplicial sets. As a result the Kan simplicial set model is shown to presently be, from a constructive perspective, unsatisfactory as a model of even simply typed lambda calculus.

An alternative to interpreting type theory in Kan simplicial sets is to use cubical sets with a uniform Kan condition, as in [BCH14]. The results of the present paper suggests that using cubical sets with the uniform Kan condition is more promising than using Kan simplicial sets.

In addition to its type-theoretical implications, we think the result in this paper is valuable in its own right: we prove that a basic result in homotopy theory is not constructively provable.

The rest of the paper is organized as follows. In Section 6.2, we introduce simplicial sets and provide several examples of simplicial sets which will be used later. In Section 6.3, we define hypergraphs which we can constructively interpret as simplicial sets. In Section 6.4, we use that interpretation, in combination with Theorem 6.1, to formulate a theorem about graphs. In Section 6.5, we provide a Kripke model rejecting the constructive provability of this theorem. In Section 6.6, we explain how we used the proof assistant Coq to verify the Kripke model, before concluding in Section 6.7.

6.2 Simplicial sets

We start by recalling the formal definition of simplicial sets and Kan simplicial sets from [BCP15]. We also introduce functional Kan simplicial sets, before we provide a more intuitive explanation intended for those new to simplicial sets.

Definition 6.1 (Simplicial set)**.** A *simplicial set* A is a collection of sets $A[i]$ for $i \in \mathbb{N}$ such that, for every $0 < n$ and $j \leq n$, we have a function (*face map*) $d^n_j : A[n] \to A[n-1]$, and for every $0 \leq n$ and $j \leq n$, we have a function (*degeneracy map*) $s^n_j : A[n] \to A[n+1]$, satisfying the following *simplicial identities* for all suitable superscripts, which we happily omit:

$$d_i d_j = d_{j-1} d_i \qquad \text{if } i < j \tag{6.1}$$
$$d_i s_j = s_{j-i} d_i \qquad \text{if } i < j \tag{6.2}$$
$$d_i s_j = id \qquad \text{for } i = j, j+1 \tag{6.3}$$
$$d_i s_j = s_j d_{i-1} \qquad \text{if } i > j+1 \tag{6.4}$$
$$s_i s_j = s_j s_{i-1} \qquad \text{if } i > j \tag{6.5}$$

An element of $A[i]$ is called an *i-simplex*. A *degenerate* element is any element $a \in A[i+1]$ in the image of a degeneracy map.

Note that a simplicial identity, such as, $d^n_i d^{n+1}_j = d^n_{j-1} d^{n+1}_i$, actually means

$$\forall x \in A[n+1].\ d^n_i(d^{n+1}_j(x)) = d^n_{j-1}(d^{n+1}_i(x)).$$

Simplicial sets form a category. For two simplicial sets A and B, $Hom_S(A, B)$ is the set of all natural transformations from A to B. A natural transformation is a collection of maps $g[n] : A[n] \to B[n]$ commuting with the face and degeneracy maps of A and B: $g[n]s_i = s_i g[n-1]$ for all $0 \leq i < n$ and $g[n+1]d_i = d_i g[n]$ for all

$0 \leq i \leq n+1$. We freely omit the dimension $[n]$ when it can be inferred from the other arguments. For more information on simplicial sets, see [May93, GJ09, Fri08].

Definition 6.2 (Functional Kan simplicial set)**.** A simplicial set Y satisfies the Kan condition if for any collection of simplices $y_0, \ldots, y_{k-1}, y_{k+1}, \ldots, y_n$ in $Y[n-1]$ such that $d_i y_j = d_{j-1} y_i$ for any $i < j$ with $i \neq k$ and $j \neq k$, there is an n-simplex y in Y such that $d_i y = y_i$ for all $i \neq k$. The Kan condition is also called the Kan extension property, and a simplicial set is called a *Kan simplicial set* if it satisfies the Kan condition.

If we have *functions* $\mathrm{fill}_k^{n-1} : Y[n-1] \times \ldots Y[n-1] \to Y[n]$ giving the required n-simplex, then we say that Y is a *functional Kan simplicial set*.

A similar notion to functional Kan simplicial sets has been introduced by Thomas Nikolaus [Nik11] as algebraic Kan complexes (AlgKan). The difference between AlgKan and functional Kan simplicial sets lies in the notion of morphisms (maps) in the corresponding category. For functional Kan simplicial sets, the morphisms are the same as between simplicial sets—they are natural transformations commuting with face and degeneracy maps—while for AlgKan, the morphisms must also send fillings to fillings. While the category of simplicial sets have a well-behaved exponential object, there is, to the author's knowledge, no good notion of exponentiation for AlgKan. Exponentiation is used to interpret the function type.

We will now provide some intuition of Kan simplicial sets by inspecting how they work in the lower dimensions. Readers who are already familiar with simplicial sets can skip ahead to Notation 1.

A simplicial set is an algebraic model of a topological space. It can also be seen as generalizations of reflexive directed multigraphs with countably infinite many dimensions. The first four dimensions of a simplicial set A can be viewed as points, edges, triangles and tetrahedrons. There are two functions going from edges to points—$d_0^1 : A[1] \to A[0]$ and $d_1^1 : A[1] \to A[0]$—and we say that d_1^1 gives the startpoint and d_0^1 gives the endpoint of an edge. Likewise, there are three functions from triangles to edges, $d_0^2, d_1^2, d_2^2 : A[2] \to A[1]$ (giving the three edges a triangle consists of); and there are four similarly-named functions from tetrahedrons to triangles (giving the four triangles of the tetrahedron.)

It is important to note that elements are *not* necessarily equal when their components are; there can be several different edges going from one point to another, there can be several triangles having the exact same edges components, and so on.

Figure 6.1 shows a triangle t consisting of three edges, e_0, e_1 and e_2, with $d_i^2(t) = e_i$. Since the triangle is built up by three edges, we expect certain relations between the endpoints of those edges; for example, that the endpoint of e_2 matches the startpoint of e_0. This is precisely what is enforced by simplicial identity 6.1.

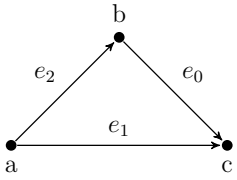


Figure 6.1: A single triangle.

In addition to the face maps d_j^n, there are the degeneracy maps $s_j^n : A[n] \to A[n+1]$. These give degenerate elements: elements which are, so to say, constructed solely from a lower-dimensional object. For points, s_0^0 gives a reflexive edge on that point, and this edge is called the degenerate self-loop of the point. For edges, s_0^1 gives a triangle where two of the sides are the original edge and the third side is the degenerate self-loop of the startpoint of the edge, as shown in Figure 6.2. The function s_1^1 gives a triangle with the degenerate edge built on the endpoint. These properties are enforced by the simplicial identities 6.2-6.4, while simplicial identity 6.5 enforces the natural constraint that certain different ways of degenerating lead to the same degenerate element. The latter is most easily exemplified by first taking the degenerate edge on a point, and then either s_1^1 or s_0^1 of this edge, both provide the same degenerate triangle (with all three faces degenerate.)

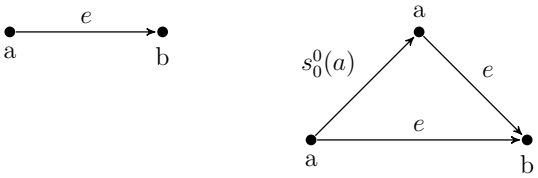


Figure 6.2: An edge e and the degenerate triangle $s_0^1(e)$.

Kan simplicial sets are special insofar as they are guaranteed to contain certain elements. One intuition is that they are simplicial sets satisfying the following condition: Whenever we have $n+1$ elements in $A[n]$ such that we lack exactly one element in $A[n]$ to have all the faces of an element in $A[n+1]$, then we have this extra element in $A[n]$, and we have the element in $A[n+1]$ containing them all. For example, if we have two edges as in the left of Figure 6.3 where we lack only one edge to have all the edges of a triangle, then that edge exists (the edge g in the figure), *and* there is a triangle such that its faces are exactly e, f and g. For triangles, the Kan condition ensures that if we have three triangles

Figure 6.3: An example of two compatible edges getting filled.

such that we only lack a fourth to form a tetrahedron, then we have both that triangle and the tetrahedron. The relatively unwieldy condition on the sequence of elements $y_0, \ldots, y_{k-1}, y_{k+1}, \ldots, y_n$ given in Definition 6.2 express precisely that we lack exactly one element in $A[n]$ to have all the faces of an element in $A[n+1]$.

Notation 1. We introduce some notation for describing elements in the lower dimensions of a simplicial set A. We write $e : a \to b$ if $e \in A[1]$, $d_1^1(e) = a$, and $d_0^1(e) = b$ (note the direction). We write

$$t : \begin{array}{ccc} & e_2 & \nearrow \searrow & e_0 \\ \bullet & & \longrightarrow & \bullet \\ & & e_1 & \end{array}$$

for a triangle $t \in A[2]$ with $d_i^2(t) = e_i$. We say that a triangle t contains an edge e if $d_i^2 t = e$ for some $0 \leq i \leq 2$. The simplicial identities enforce that all triangles $t \in A[2]$ satisfy that, if $t : \begin{array}{ccc} & e_2 & \nearrow \searrow & e_0 \\ \bullet & & \longrightarrow & \bullet \\ & & e_1 & \end{array}$, then $d_0^1 e_2 = d_1^1 e_0$, $d_1^1 e_2 = d_1^1 e_1$ and $d_0^1 e_0 = d_0^1 e_1$. This justifies writing

$$t : \begin{array}{ccccc} & & \mathrm{b} & & \\ & e_2 & \nearrow & \searrow & e_0 \\ \mathrm{a} & & \longrightarrow & & \mathrm{c} \\ & & e_1 & & \end{array}$$

for a triangle t with $d_i^1 t = e_i$ and $e_2 : a \to b$, $e_0 : b \to c$, and $e_1 : a \to c$.

Before moving on to some examples of simplicial sets, we show a property of Kan simplicial sets which will be important later: they have edge reversal.

Definition 6.3 (Edge reversal)**.** A simplicial set Y is said to have *edge reversal* when, for every edge $e \in Y[1]$, there exists an edge $f \in Y[1]$ with $d_1(f) = d_0(e)$ and $d_0(f) = d_1(e)$. We say that a simplicial set has *functional edge reversal* when we have a *function* giving, for every edge $e \in Y[1]$, the edge $f \in Y[1]$ as above.

Lemma 6.4. *Functional Kan simplicial sets have functional edge reversal, and Kan simplicial sets have edge reversal.*

Proof of functional edge reversal. For all $e \in Y[1]$ where Y is a functional Kan fill-graph, let

$$f = d_0(\text{fill}_{1,0}(s(d_1(e)), e)).$$

If $e : a \to b$, then $s(d_1(e)) : a \to a$, and

$$\text{fill}_{1,0}(s(d_1(e)), e) : \quad \begin{array}{ccccc} & e & & b & \\ & \nearrow & & \searrow & \\ a & & \xrightarrow[s(d_1(e))]{} & & a \end{array},$$

so $d_0(\text{fill}_{1,0}(s(d_1(e)), e)) : b \to a$. □

Proof of non-functional version. As above, but instead of using the fill function we can only claim that the edge exists. □

6.2.1 Examples of simplicial sets

In this section we give some examples of simplicial sets which will be useful later. This section contains standard definitions, and is taken from [BCP15].

Standard simplicial k-simplex Δ^k

Δ^k is the simplicial set with $\Delta^k[j]$ consisting of all non-decreasing sequences of numbers $0, \ldots, k$ of length $j+1$. Equivalently, $\Delta^k[j]$ is the set of order-preserving functions $[j] \to [k]$, where $[i]$ denotes $0, \ldots, i$ with the natural ordering. Examples are $\Delta^1[0] = \{0, 1\}$, $\Delta^1[1] = \{00, 01, 11\}$, $\Delta^2[1] = \{00, 01, 02, 11, 12, 22\}$ and

$$\Delta^2[2] = \{000, 001, 002, 011, 012, 022, 111, 112, 122, 222\}.$$

The degeneracy map $s^j_k : \Delta^i[j] \to \Delta^i[j+1]$ duplicates the k-th element in its input. So, $s^j_k(x_0 \ldots x_k \ldots x_{j+1}) = x_0 \ldots x_k x_k \ldots x_{j+1}$. The face map $d^j_k : \Delta^i[j] \to \Delta^i[j-1]$ deletes the k-th element. So, $d^j_k(x_0 \ldots x_j) = x_0 \ldots x_{k-1} x_{k+1} \ldots x_j$.

The k-horns Λ^k_j

Λ^k_j is the j'th horn of the standard k-simplex Δ^k, and defined by $\Lambda^k_j[n] = \{f \in \Delta^k[n] \mid [k] - \{j\} \not\subseteq Im(f)\}$. Alternatively, it is $\Delta^k[n]$ except every element must avoid some element not equal to j. For example, $\Lambda^2_0[1] = \{00, 01, 02, 11, \cancel{12}, 22\} = \Delta^2[1] - \{12\}$ (excluding 12, since 12 does not avoid any element not equal to 0). We also have:

$$\Lambda^2_0[2] = \{000, 001, 002, 011, \cancel{012}, 022, 111, \cancel{112}, \cancel{122}, 222\}.$$

The functional Kan extension condition from Definition 6.2 for a simplicial set Y can also be formulated as: we have a dependent function $\text{fill}(k, j, F)$ such that $\text{fill}(k, j, F) : \Delta^k \to Y$ extends $F : \Lambda^k_j \to Y$, for any k, j, F.

Cartesian products

For two simplicial sets A and B, $A \times B$ is the simplicial set given by $(A \times B)[i] = A[i] \times B[i]$, and the structural maps d and s use d^A and d^B component-wise (and likewise for s^A and s^B). So if $a \in A[i]$ and $b \in B[i]$ then $(a, b) \in (A \times B)[i]$, and $d_i((a, b)) = (d_i^A(a), d_i^B(b))$. In particular, the degenerate simplices of $A \times B$ are pairs $(s_j^A(a), s_j^B(b)) \in (A \times B)[i + 1]$. (Caveat: this is stronger than both components being degenerate.)

Function spaces

Y^X is the simplicial set given by $Y^X[i] = Hom_S(\Delta^i \times X, Y)$, where Hom_S denotes morphisms (natural transformations) of simplicial sets, and structural maps as follows. The face map $d_k[i] : Y^X[i] \to Y^X[i-1]$ need to map elements of $Hom_S(\Delta^i \times X, Y)$ to $Hom_S(\Delta^{i-1} \times X, Y)$ and the degeneracy maps vice versa. For their definition it is convenient to view a k-simplex in Δ^i as a non-decreasing function $a : [k] \to [i]$. Let d_k^* be the strictly increasing function on natural numbers such that $d_k^*(n) = n$ if $n < k$ and $d_k^*(n) = n + 1$ otherwise (d_k^* 'jumps' over k). Given $F \in Hom_S(\Delta^i \times X, Y)$, define $(d_k F)[j](a_0 \dots a_j, x) = F[j](d_k^* a_0 \dots d_k^* a_j, x)$. For the degeneracy maps, let s_k^* be the weakly increasing function on natural numbers such that $s_k^*(n) = n$ if $n \leq k$ and $s_k^*(n) = n - 1$ otherwise (s_k^* 'duplicates' k). Then we define $(s_k F)[i](a, x) = F[i](s_k^* a, x)$.

6.3 Hypergraphs as simplicial sets

We now define graph classes corresponding to (functional Kan) simplicial sets. The meaning of "corresponding" is made precise in Lemma 6.8; these are graphs which can be constructively interpreted as (functional Kan) simplicial sets.

Definition 6.5 (Reflexive hypergraph)**.** A *reflexive hypergraph* consists of C_2, C_1, C_0, d_0^1, d_1^1, d_0^2, d_1^2, d_2^2, s, s_0, s_1 where C_0 is a set of points, C_1 a set of edges and C_2 a set of triangles. For $d_i^1 : C_1 \to C_0$, d_1^1 is the *source* and d_0^1 the *target* function, and $s(c)$ is a degenerate self-loop. Each $d_i^2 : C_2 \to C_1$ is an *edge* function, giving an edge of a triangle; and each $s_i : C_1 \to C_2$ is a function mapping an edge to a degenerate triangle. These are all subject to different restrictions, which are given below.

We will use the notation introduced in Notation 1 for reflexive hypergraphs as well. We require that all triangles $t \in C_2$ satisfy that, if t : 

, then

$d_0^2 e_2 = d_1^2 e_0$, $d_1^2 e_2 = d_1^2 e_1$ and $d_0^2 e_0 = d_0^2 e_1$, justifying writing

$$t : \begin{array}{ccccc} & & \mathrm{b} & & \\ & e_2 \nearrow & & \searrow e_0 & \\ \mathrm{a} & & \xrightarrow[e_1]{} & & \mathrm{c} \end{array}$$

for a triangle t with $d_i^2 t = e_i$, $e_2 : a \to b$, $e_0 : b \to c$, and $e_1 : a \to c$.

We require $d_i^1(s(c)) = c$ for all $c \in C_0$, and we require that the functions s_0 and s_1 satisfy the following: for all $e : a \to b$, $s_0(e) :$ $\begin{array}{ccccc} & & \mathrm{s(a)} & & \\ & \nearrow & & \searrow & \mathrm{e} \\ \mathrm{a} & & \xrightarrow[\mathrm{e}]{} & & \mathrm{b} \end{array}$ and $s_1(e) :$ $\begin{array}{ccccc} & & \mathrm{e} & & \\ & \nearrow & & \searrow & \mathrm{s(b)} \\ \mathrm{a} & & \xrightarrow[\mathrm{e}]{} & & \mathrm{b} \end{array}$, and that for all v, $s_0(s(v)) = s_1(s(v)) :$ $\begin{array}{ccccc} & & \mathrm{s(v)} & & \\ & \nearrow & & \searrow & \mathrm{s(v)} \\ \mathrm{v} & & \xrightarrow[\mathrm{s(v)}]{} & & \mathrm{v} \end{array}$.

The definition above is not much more than a specialization of Definition 6.1 to the first three dimensions, so it is not particularly surprising that we can extend any reflexive hypergraph to a simplicial set. The method is a natural extension of the one used in [BCP15] and [BC15], but extended in such a way that triangles are not necessarily equal when they have equal faces.

Definition 6.6 ($S(C)$)**.** Given a reflexive hypergraph C, we can construct a simplicial set $S(C)$ in the following way:

$S(C)[0] = C_0$, $S(C)[1] = C_1$, $S(C)[2] = C_2$ and $S(C)[n]$, for $n \geq 3$, consisting of all tuples of the form $(u_0, \dots, u_n; \dots, e_{ij}, \dots; \dots, t_{ijl}, \dots)$ such that

$$e_{ij} : u_i \to u_j \text{ in } C[1] \text{ for all } 0 \leq i < j \leq n, \text{ and}$$

$$t_{ijl} : \begin{array}{ccccc} & & u_j & & \\ & e_{ij} \nearrow & & \searrow e_{jl} & \\ u_i & & \xrightarrow[e_{il}]{} & & u_l \end{array} \text{ in } C[2] \text{ for all } 0 \leq i < j < l \leq n.$$

The maps d_k^n in $S(C)$ are defined for $n > 3$ by removing from the input tuple

$$(u_0, \dots, u_n; \dots, e_{ij}, \dots; \dots, t_{ijl}, \dots)$$

the point u_k, all edges e_{ik} and e_{kj}, and all triangles containing either of those edges. For $n = 3$, if we do this on an element q in $S(C)[3]$, the result is a tuple $(u_0, u_1, u_2; e_{01}, e_{02}, e_{12}; t_{012})$ containing only one triangle t_{012}, and we let $d_k^3(q) = t_{012}$.

The maps s_k^n in $S(C)$ for $n > 3$ are defined by duplicating the point u_k in the tuple $(u_0, \dots, u_n; \dots, e_{ij}, \dots; \dots, t_{ijl}, \dots)$, adding an edge $e_{k(k+1)} = s(u_k)$, and

duplicating edges and incrementing indices of edges as appropriate. In addition, we add $t_{k(k+1)j} = s_0(e_{kj})$ for every e_{kj} and $t_{ik(k+1)} = s_1(e_{ik})$ for every e_{ik}, and duplicating triangles and incrementing indices as needed. For $n = 3$, we are given a triangle t : $\begin{array}{ccccc} & & \mathrm{b} & & \\ & e_2 \nearrow & & \searrow e_0 & \\ \mathrm{a} & & \xrightarrow[e_1]{} & & \mathrm{c} \end{array}$, and we perform the above construction on the tuple $(a, b, c; e_2, e_1, e_0; t)$.

This completes the construction of the simplicial set $S(C)$, and it is fairly easy to see that it satisfies the simplicial identities.

Similarly, we can specialize Definition 6.2 to the first three dimensions, giving us a first-order structure we can extend to a functional Kan simplicial set.

Definition 6.7 (Kan fill-hypergraph)**.** A *Kan fill-hypergraph* is a reflexive hypergraph where we have functions $\mathrm{fill}_{1,i} : C[1] \times C[1] \to C[2]$ for $0 \leq i \leq 2$ and $\mathrm{fill}_{2,i} : C_2 \times C_2 \times C_2 \to C_2$ for $0 \leq i \leq 3$, satisfying the following requirements.

For all $e_1, e_2 \in C[1]$, $\mathrm{fill}_{1,i} : C[1] \times C[1] \to C[2]$ must satisfy:

If $e_1 : a \to c$ and $e_2 : a \to b$, then $\mathrm{fill}_{1,0}(e_1, e_2)$: $\begin{array}{ccccc} & & b & & \\ & e_2 \nearrow & & \searrow & \\ a & & \xrightarrow[e_1]{} & & c \end{array}$.

If $e_0 : b \to c$ and $e_2 : a \to b$, then $\mathrm{fill}_{1,1}(e_0, e_2)$: $\begin{array}{ccccc} & & b & & \\ & e_2 \nearrow & & \searrow e_0 & \\ a & & \longrightarrow & & c \end{array}$.

If $e_0 : b \to c$ and $e_1 : a \to c$, then $\mathrm{fill}_{1,2}(e_0, e_1)$: $\begin{array}{ccccc} & & b & & \\ & \nearrow & & \searrow e_0 & \\ a & & \xrightarrow[e_1]{} & & c \end{array}$.

For all $t_1, t_2, t_3 \in C[2]$, $\mathrm{fill}_{2,i} : C_2 \times C_2 \times C_2 \to C_2$ must satisfy:

If t_1: $\begin{array}{ccccc} & e_1 \nearrow & c & \searrow e_5 & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$, t_2: $\begin{array}{ccccc} & e_2 \nearrow & b & \searrow e_4 & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$ and t_3: $\begin{array}{ccccc} & e_2 \nearrow & b & \searrow e_0 & \\ a & & \xrightarrow[e_1]{} & & c \end{array}$, then $\mathrm{fill}_{2,0}(t_1, t_2, t_3)$: $\begin{array}{ccccc} & e_0 \nearrow & c & \searrow e_5 & \\ b & & \xrightarrow[e_4]{} & & d \end{array}$.

If t_1: $\begin{array}{ccccc} & e_0 \nearrow & c & \searrow e_5 & \\ b & & \xrightarrow[e_4]{} & & d \end{array}$, t_2: $\begin{array}{ccccc} & e_2 \nearrow & b & \searrow e_4 & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$ and t_3: $\begin{array}{ccccc} & e_2 \nearrow & b & \searrow e_0 & \\ a & & \xrightarrow[e_1]{} & & c \end{array}$, then $\mathrm{fill}_{2,1}(t_1, t_2, t_3)$: $\begin{array}{ccccc} & e_1 \nearrow & c & \searrow e_5 & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$.

If t_1: $\begin{array}{ccccc} & e_0 \nearrow & c & \searrow e_5 & \\ b & & \xrightarrow[e_4]{} & & d \end{array}$, t_2: $\begin{array}{ccccc} & e_1 \nearrow & c & \searrow e_5 & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$ and t_3: $\begin{array}{ccccc} & e_2 \nearrow & b & \searrow e_0 & \\ a & & \xrightarrow[e_1]{} & & c \end{array}$, then $\mathrm{fill}_{2,2}(t_1, t_2, t_3)$: $\begin{array}{ccccc} & e_2 \nearrow & b & \searrow e_4 & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$.

If t_1: $\begin{array}{ccccc} & e_0 \nearrow & c & \searrow e_5 & \\ b & & \xrightarrow[e_4]{} & & d \end{array}$, t_2: $\begin{array}{ccccc} & e_1 \nearrow & c & \searrow e_5 & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$ and t_3: $\begin{array}{ccccc} & e_2 \nearrow & b & \searrow e_4 & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$, then $\mathrm{fill}_{2,3}(t_1, t_2, t_3)$: $\begin{array}{ccccc} & e_2 \nearrow & b & \searrow e_0 & \\ a & & \xrightarrow[e_1]{} & & c \end{array}$.

Note that the above requierments can be translated from the visual description

above into first-order logic, and this is the intended reading of the above definition. For example, the requirement for $\text{fill}_{1,0}$ is:

$$\forall e_1, e_2 \in C[1], d_1^1(e_1) = d_1^1(e_2) \to d_1^2(\text{fill}_{1,0}(e_1, e_2)) = e_1 \wedge d_2^2(\text{fill}_{1,0}(e_1, e_2)) = e_2.$$

Lemma 6.8. *If C is a Kan fill-hypergraph, we can extend $S(C)$ to a functional Kan simplicial set.*

Proof. We have to define the functions $\text{fill}_{n,k}$ in $S(C)$. We write $\text{fill}^S_{n,k}$ for the functions in $S(C)$ and $\text{fill}^C_{n,k}$ for the functions in C.

If $n = 0$, we simply let $\text{fill}^S_{0,i} : C_0 \to C_1$ be s. If $n = 1$ we put $\text{fill}^S_{1,i} = \text{fill}^C_{1,i}$, and it is easy to see that $\text{fill}^C_{1,i}$ satisfies the requirements given in Definition 6.2.

If $n = 2$, we use $\text{fill}^C_{2,k}$; but we cannot use it directly, as it provides an element of $S(C)[2]$, not an element in $S(C)[3]$ as needed. Instead, we use it to construct an element of $S(C)[3]$. We show the procedure for $k = 1$; the other cases proceed analogously. We are given $t_0, t_2, t_3 \in S(C)[2] = C[2]$, such that $d_i t_j = d_{j-1} t_i$ for any $i < j \leq 3$ with $i \neq 1$ and $j \neq 1$, and we proceed to construct an $r \in S(C)[3]$ such that $d_i^2 r = t_i$ for $i = 0, 2, 3$. Expanding this and naming the resulting edges gives the equations

$$\begin{aligned} d_0 t_2 &= d_1 t_0 = e_4 \\ d_0 t_3 &= d_2 t_0 = e_0 \\ d_2 t_3 &= d_2 t_2 = e_2 \end{aligned}$$

Naming the three remaining edges e_1, e_3 and e_5 we get that $t_0 : \begin{array}{ccc} & {\scriptstyle e_0}\nearrow\ c\ \searrow{\scriptstyle e_5} & \\ b & \xrightarrow[e_4]{} & d \end{array}$,

$t_2 : \begin{array}{ccc} & {\scriptstyle e_2}\nearrow\ b\ \searrow{\scriptstyle e_4} & \\ a & \xrightarrow[e_3]{} & d \end{array}$, and $t_3 : \begin{array}{ccc} & {\scriptstyle e_2}\nearrow\ b\ \searrow{\scriptstyle e_0} & \\ a & \xrightarrow[e_1]{} & c \end{array}$, giving $\text{fill}^C_{2,1}(t_1, t_2, t_3) : \begin{array}{ccc} & {\scriptstyle e_1}\nearrow\ c\ \searrow{\scriptstyle e_5} & \\ a & \xrightarrow[e_3]{} & d \end{array}$.

Observe that the tuple

$$r = (a, b, c, d\ ;\ e_2, e_1, e_3, e_0, e_4, e_5\ ;\ t_3, t_2, \text{fill}^C_{2,1}(t_0, t_2, t_3), t_0)$$

satisfies the form given in Definition 6.6 for elements in $S(C)[3]$, so $r \in S(C)[3]$, while also satisfying $d_i r = t_i$ for $i = 0, 2, 3$, giving us the value for $\text{fill}^C_{2,1}(t_0, t_2, t_3)$.

For higher values of n, we observe that any sequence of tuples applicable to $\text{fill}^S_{n,k}$ contains all the components of a satisfying element; it is just a matter of extracting the right triangles, edges and points from the arguments. □

6.4 Function spaces between hypergraphs

In this section, we identify a class of functions between reflexive hypergraphs which we can extend to edges in the function space between the corresponding simplicial sets. This enables us to formulate a constructive consequence of Theorem 6.1 which we will give a countermodel to in the next section.

Remember that the function space between two simplicial sets A and B has as the i^{th} dimension $Hom_S(\Delta^i \times A, B)$, so its edges are elements of $Hom_S(\Delta^1 \times A, B)$.

Definition 6.9 ($\Delta^i|_m$)**.** We define $\Delta^i|_m$ to be the family of m sets given by removing from Δ^i every dimension larger than m. The functions s^j and d^{j+1} for $0 \leq j < m$ are kept as is, while they are discarded for $j > m$.

Definition 6.10. For reflexive hypergraphs X and Y, we say that an $F : \Delta^i|_2 \times X \to Y$ is *commuting* when it commutes with d^n and s^m for $0 \leq m \leq 1 \leq n \leq 2$, where both work on the product $\Delta^i|_2[n] \times X[n]$ component-wise, similar to Cartesian products of simplicial sets as described in Section 6.2.1.

Lemma 6.11. *For reflexive hypergraphs X and Y, any commuting $F : \Delta^1|_2 \times X \to Y$ can be extended to an edge in $S(Y)^{S(X)}$.*

Proof. We need to extend F to an $F' \in Hom_S(\Delta^1 \times S(X), S(Y))$; that is, a family of functions $F'[n] : (\Delta^1 \times S(X))[n] \to S(Y)[n]$ for $n \in \mathbb{N}$ commuting with s^i_j and d^i_j. For $0 \leq n \leq 2$, we let $F'[n] = F$. For $n > 2$, any input to $F'[n]$ will have the form

$$(0^a 1^b, (x_0, \dots, x_n; \dots e_{ij}, \dots; \dots, t_{ijl}, \dots))$$

such that $a + b = n + 1$. We define the function $gt_a : \mathbb{N} \to \{0, 1\}$ as $gt_a(x) = 1$ if $x \geq a$ and $gt_a(x) = 0$ otherwise, and we then let $F'[n]$ map the input to the tuple

$$(F(0, x_0), \dots, F(0, x_{a-1}), F(1, x_a) \dots, F(1, x_{a+b-1}); \dots e'_{ij}, \dots; \dots t'_{ijl}, \dots),$$

where e'_{ij} are given by $e'_{ij} = F(gt_a(i)gt_a(j), e_{ij})$, and t'_{ijl} are given by $t'_{ijl} = F(gt_a(i)gt_a(j)gt_a(l), t_{ijl})$.

It should be clear that this map does indeed commute with d_i and s_j. It holds in the lower dimensions since we assume F to be commuting. It also holds in the higher dimensions since we apply F uniformly to every element in the tuple; if we remove an particular element and then apply F, we get the same result as if we first apply F to all elements in the tuple and then remove the particular element. □

Now we are ready to formulate the constructive consequence of Theorem 6.1—which is essentially Theorem 6.2, reformulated as a property of Kan fill-hypergraphs.

Theorem 6.3 (Classical)**.** *For any Kan fill-hypergraphs X and Y, for any commuting $F : \Delta^1|_2 \times X \to Y$ we can find a commuting $F^- : \Delta^1|_2 \times X \to Y$ such that for all $p \in X[0]$, $l \in X[1]$ and $t \in X[2]$ we have*

$$\begin{aligned} F^-(0,p) &= F(1,p) & F^-(1,p) &= F(0,p) \\ F^-(00,l) &= F(11,l) & F^-(11,l) &= F(00,l) \\ F^-(000,t) &= F(111,t) & F^-(111,t) &= F(000,t) \end{aligned}$$

Proof. We first extend F to an edge in $S(Y)^{S(X)}$ by Lemma 6.11 and note that, by Lemma 6.8, $S(Y)$ is a functional Kan simplicial set. We then apply the classical Theorem 6.2, giving that $S(Y)^{S(X)}$ is a Kan simplicial set, enabling us to reverse the edge by Lemma 6.4, giving an F^- in $S(Y)^{S(X)}[1]$ satisfying $d_0(F) = d_1(F^-)$ and $d_1(F) = d_0(F^-)$. We discard every dimension of F^- above 2. Being an element of $Hom_S(\Delta^1 \times S(X), S(Y))$ means that F^- commutes, and expanding the definition of d_i from Section 6.2.1 we calculate:

$$F^-(0,p) = F^-(d_1^*(0),p) = d_1(F^-)(0,p) = d_0(F)(0,p) = F(d_0^*(0),p) = F(1,p),$$

$$F^-(1,p) = F^-(d_0^*(0),p) = d_0(F^-)(0,p) = d_1(F)(0,p) = F(d_1^*(0),p) = F(0,p)$$

The other dimensions go the same way, showing that F^- is as desired. □

Observe that the only non-constructive step in the proof of Theorem 6.3 was the application of Theorem 6.2.

The reasoning in our constructive proofs can be formalized in IZF (Intuitionistic Zermelo-Fraenkel set theory), so IZF proves that Theorem 6.1 implies Theorem 6.2, and that Theorem 6.2 implies Theorem 6.3. We also have that if IZF proves Theorem 6.3, then Theorem 6.3 holds in any Kripke model. This result has been further elaborated in Section 6 of [BCP15]. For this, it is vital that Theorem 6.3 can be expressed in first-order logic. So finally, by giving a Kripke model falsifying Theorem 6.3, we show that IZF cannot prove Theorem 6.1, and we will provide exactly such a model in the next section.

6.5 Kripke countermodel

In this section we present a Kripke model falsifying the first-order sentence representing Theorem 6.3. Recall that a Kripke model is a partially ordered set of classical models—often called states or days—where the domains and relations are monotone, and a formula holds in a state if it holds (classically) at that state and all of its successors. For further elaboration, see [Kri65].

We have two classical models in our Kripke model, and we will call them "day 1" and "day 2", with day 1 before day 2. Each consists of an X-part and a Y-part,

both satisfying the requirements on Kan fill-hypergraphs. In addition, our Kripke model contains a commuting family of functions $F : \Delta^1|_2[n] \times X[n] \to Y[n]$ for $0 \leq n \leq 2$.

The only change from day 1 to day 2 is the interpretation of equivalence; we equate more elements in day 2. As we equate elements, we have to ensure that all functions respect equivalence; that is, that they send equal elements to equal elements. In addition, by equating elements, more elements may satisfy the antecedents in Definition 6.7, and we need to ensure that these formulae remain satisfied for our X and Y-parts to be valid Kan fill-hypergraphs.

We first present X and Y in day 1. We then present the family of functions $F : \Delta^1|_2[n] \times X[n] \to Y[n]$ for $0 \leq n \leq 2$, before we present X and Y day 2.

6.5.1 Day 1

The two first dimensions of X and Y are both shown below, with triangles and fill functions further described below. Different edges in the figure are non-equated, and an edge e in the figure from a to b represents an edge e in the model with $d^1_0(e) = b$ and $d^1_1(e) = a$.

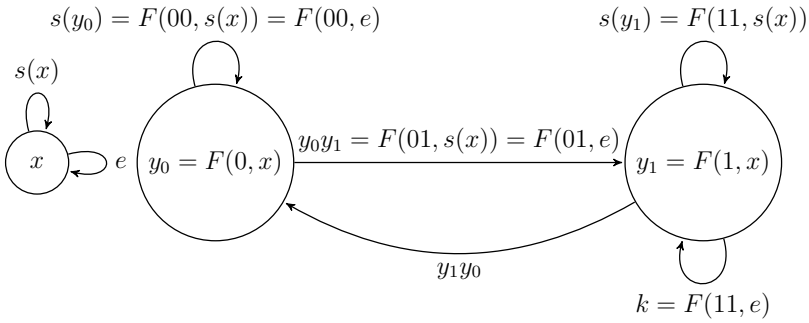


Figure 6.4: Kripke (counter)model for edge reversal, day 1.

Triangles in X, Day 1

$X[2]$ consists of exactly all eight combinations of $s(x)$ and e as faces. So we have the following triangles, where the names are given as the concatination of d^2_i of

the triangle for $0 \leq i \leq 2$.

sss : s(x) s(x) / s(x) sse : e s(x) / s(x)

ees : s(x) e / e eee : e e / e

The functions s_i^1 are forced by the simplicial identities to be:

$$s_0^1(s(x)) = s_1^1(s(x)) = sss$$

$s_0^1(e) = ees$: s(x) e / e

$s_1^1(e) = see$: e s(x) / e

Finally, we define the functions $\text{fill}_{1,i}^X : X[1] \times X[1] \to X[2]$ and $\text{fill}_{2,i}^X : X[2] \times X[2] \times X[2] \to X[2]$. For the former, we have a choice; the two arguments determine two of the edges in the resulting triangle, but the third edge can be either $s(x)$ or e. We chose, rather arbitrary, for it to always be $s(x)$, resulting in only one possible triangle. For $\text{fill}_{2,i}^X$, there are no options; its three arguments determine the three edges of the resulting triangle, describing it completely.

Triangles in Y, Day 1

We construct $Y[2]$ in two stages. First, it consists of all compatible triples of edges from $Y[1]$. That is, for all edges e_0, e_2, e_1 such that $e_0 : b \to c$, $e_1 : a \to c$, and $e_2 : a \to b$, we add exactly one triangle $e_0\_e_1\_e_2$: a $\xrightarrow[e_1]{e_2}$ b $\xrightarrow{e_0}$ c to $Y[2]$. This result in 18 triangles, and as with $X[2]$ we name then according to their faces. This means that we have a triangle

$$y_1y_0\_y_1y_0\_s(y_1) : (y_1, y_1, y_0; s(y_1), y_1y_0, y_1y_0) \in Y[1],$$

and we will call it T_{de}. The second stage of the construction is simply to add an additional triangle T_{fi} of the form

$$T_{\text{fi}} : (y_1, y_1, y_0; s(y_1), y_1y_0, y_1y_0)$$

to $Y[2]$. It is no coincidence that T_{fi} has identical faces to T_{de}; this enables us to use T_{fi} as a substitute of T_{de} in certain situations. All triangles of $Y[2]$ at day 1 are listed in Table B.1.

We define $s_i^1 : Y[1] \to Y[2]$ before concluding with the fill functions. In most cases, there is only one compatible triangle to which s_i^1 can map, forcing

$$s_0^1(y_0y_1) = y_1y_0\_y_1y_0\_s(y_0) : \begin{array}{ccccc} & & s(y_1) & y_0 & y_0y_1 \\ & \nearrow & & \searrow & \\ y_0 & & \xrightarrow[y_0y_1]{} & & y_1 \end{array}$$

$$s_1^1(y_0y_1) = s(y_0)\_y_1y_0\_y_1y_0 : \begin{array}{ccccc} & & y_0y_1 & y_1 & s(y_1) \\ & \nearrow & & \searrow & \\ y_0 & & \xrightarrow[y_0y_1]{} & & y_1 \end{array}$$

and similar for the other edges. The exception is $s_0^1(y_1y_0)$, which we can map to both T_{de} and T_{fi}. We set

$$s_0^1(y_1y_0) = T_{\text{de}},$$

and this concludes the definition of s_i^1. The complete listing of s_i^1 can be found in Table B.2.

Finally, we define the functions $\text{fill}_{1,i} : Y[1] \times Y[1] \to Y[2]$ and $\text{fill}_{2,i} : Y[2] \times Y[2] \times Y[2] \to Y[2]$. They are, in the same way as s_i^1, in most cases determined by the fact that there is exactly one compatible triangle. There are some exceptions. For $\text{fill}_{1,i}$, we have certain inputs where we can choose if the third edge in the resulting triangle is $s(y_1)$ or k, and in those cases we choose for it to be $s(y_1)$; and when there is a choice between mapping to T_{fi} and T_{de}, we map to T_{fi}. If we want $\text{fill}_{2,i}$ to be total, we map every non-compatible pair of edges to $y0y0\_y0y0\_y0y0$.

For $\text{fill}_{2,i}$, there are inputs where the output can be either T_{de} or T_{fi}, and in these cases we map to T_{fi}. In addition, we have the choice of how to map the triangles which do *not* satisfy the requirements in Definition 6.7. Equating elements in day 2 will have the effect of making more triangles satisfy the requirements for $\text{fill}_{2,i}$, so the choices we make now must be compatible with this. We show this for $\text{fill}_{2,1}$; the other cases are similar. Recall that the requirement for $\text{fill}_{2,1}$ is:

If t_1: $\begin{array}{ccccc} & & e_0 & c & e_5 \\ & \nearrow & & \searrow & \\ b & & \xrightarrow[e_4]{} & & d \end{array}$, t_2: $\begin{array}{ccccc} & & e_2 & b & e_4 \\ & \nearrow & & \searrow & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$ and t_3: $\begin{array}{ccccc} & & e_2 & b & e_0 \\ & \nearrow & & \searrow & \\ a & & \xrightarrow[e_1]{} & & c \end{array}$, then $\text{fill}_{2,1}(t_1, t_2, t_3)$: $\begin{array}{ccccc} & & e_1 & c & e_5 \\ & \nearrow & & \searrow & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$.

Given three triangles t_1, t_2, t_3, if there is a triangle with the signature $\begin{array}{ccccc} & & e_1 & c & e_5 \\ & \nearrow & & \searrow & \\ a & & \xrightarrow[e_3]{} & & d \end{array}$ — where $e_5 = d_0^2(t_1)$, $e_1 = d_1^2(t_3)$ and $e_3 = d_1^2(t_2)$—we map $\text{fill}_{2,1}(t_1, t_2, t_3)$ to it (and to T_{fi} if there is a choice between T_{fi} and T_{de}.) If there is no such triangle, we map $\text{fill}_{2,1}(t_1, t_2, t_3)$ to $y0y0\_y0y0\_y0y0$.

F

We define the commuting family $F : \Delta^1|_2[n] \times X[n] \to Y[n]$ for $0 \leq n \leq 2$. Both $F_0 : \Delta^1|_2[0] \times X[0] \to Y[0]$ and $F_1 : \Delta^1|_2[1] \times X[1] \to Y[1]$ are completely given in Figure 6.4, only $F_2 : \Delta^1|_2[2] \times X[2] \to Y[2]$ is in need of further description. But it is completely locked by the requirement that it should commute with d_i (since, besides T_{fi} and T_{de}, we have exactly one triangle per compatible triple of edges). Note that F_1 does not map anything to the edge y_1y_0, since this goes from $F(1, x)$ to $F(0, x)$; similarly, F_2 maps to neither T_{de} nor T_{fi}, relieving us from having to choose which of these to map to.

6.5.2 Day 2

Moving from day 1 to day 2, we equate a number of elements, but make no changes otherwise. This means that we only need to ensure that the defined functions still respect equality, and verify that the filling-conditions in Definition 6.7 remain satisfied.

First, we present the equating for the first two levels of both X and Y; following this we equate triangles. In $X[1]$, we set $s(x) = e$; and in $Y[1]$, we set $s(y_1) = y_1y_1 = k$. Other edges are as they were in day 1. The first two dimensions are shown in Figure 6.5. In $X[2]$, we equate every triangle, leaving us with only

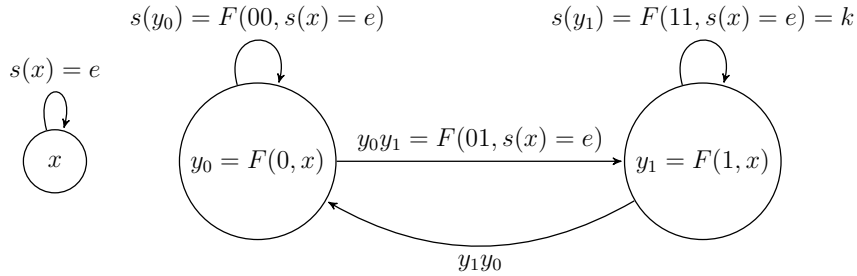


Figure 6.5: Kripke (counter)model for edge reversal, day 2.

one degenerate triangle. Having only one point, one edge and one triangle means that all functions with both domain and co-domain inside X trivially respect equality.

In $Y[2]$, we equate exactly those triangles which have identical faces after the equation of elements in $Y[1]$, except that we keep T_{de} distinct from any other triangle. Since the only edge-eqation in $Y[1]$ was $y_1y_1 = k$, we only equate triangles containing this edge. The complete list of equated triangles can be found in Table B.3; the list of the remaining, non-equated triangles can be found in

Table B.4. Note that we can keep T_{de} distinct from every other triangle, since T_{de} is only in the image of s_0^1 (it is, crucially, *not* in the image of $\text{fill}_{2/3,i}$), and then as the value of $s_0^1(y_1y_0)$. The edge y_1y_0 is not equated with any other edge, thus we are not forced through s_0^1 to equate T_{de} with any other triangle.

This clearly does not enforce any further equations in $Y[1]$. We also claim that this keeps all functions consistent. It should be clear from Figure 6.5 that both $F_0 : \Delta^1|_2[0] \times X[0] \to Y[0]$ and $F_1 : \Delta^1|_2[1] \times X[1] \to Y[1]$ remain consistent. $F_2 : \Delta^1|_2[2] \times X[2] \to Y[2]$ is consistent since F commutes with d_i and as $F_1 : \Delta^1|_2[1] \times X[1] \to Y[1]$ is consistent, so any two triangles mapped to in $Y[2]$ (with the same argument from $\Delta^1|_2[2]$) now must have identical faces, giving that they are also equal.

It is important for $\text{fill}_{1/2,i}$ that T_{de} is not in their image, and that all other triangles are equal exactly when they have equal faces. So if $e_1 = e_1'$ and $e_2 = e_2'$, then $\text{fill}_{1,i}(e_1, e_2)$ and $\text{fill}_{1,i}(e_1', e_2')$ have identical faces, giving $\text{fill}_{1,i}(e_1, e_2) = \text{fill}_{1,i}(e_1', e_2')$.

For $\text{fill}_{2,i}$, observe that the output is a triangle containing one edge from each of its inputs. So if we have $t_1 = t_1'$, $t_2 = t_2'$, and $t_3 = t_3'$ then $\text{fill}_{2,i}(t_1, t_2, t_3)$ will have identical faces to $\text{fill}_{2,i}(t_1', t_2', t_3')$, so they are also equal. Also observe that all requirements in Definition 6.7 remain satisfied. Three triangles which now satisfy one of the antecedents are already sent to a satisfying triangle by the way we defined $\text{fill}_{2,i}$.

Finally, we observe that all the simplicial identities are still satisfied, since we have not changed s_i/d_i, and the equivalence relation is monotone.

6.5.3 Non-existence of F^-

We will now see that we cannot consistently define the commuting reverse function $F^- : \Delta^1|_2 \times X \to Y$, prescribed by Theorem 6.3, such that for all $p \in X[0]$, $l \in X[1]$ and $t \in X[2]$ we have:

$$\begin{aligned} F^-(0,p) &= F(1,p) & F^-(1,p) &= F(0,p) \\ F^-(00,l) &= F(11,l) & F^-(11,l) &= F(00,l) \\ F^-(000,t) &= F(111,t) & F^-(111,t) &= F(000,t). \end{aligned}$$

Assume towards a contradiction that there is such an F^-. We will expand on the values of $F^-(001, eee)$ and $F^-(001, sss)$. Applying commutativity of the face maps in combination with the above requirements, we see that F^- would have to satisfy the following two requirements:

$$d_2^2(F^-(001,eee)) = F^-(d_2^2(001), d_2^2(eee)) = F^-(00,e) = F(11,e) = k$$

$$d_0^2 d_0^2(F^-(001,eee)) = F^-(d_0^2 d_0^2(001), d_0^2 d_0^2(eee)) = F^-(1,x) = F(0,x) = y_0.$$

This forces $F^-(001, eee) = y_1y_0\_y_1y_0\_k$, since this is the only triple in $Y[2]$ satisfying the above requirements.

Given that $sss = s_0^1(s(x))$ and $001 = s_0(01)$, commutativity with s_0 forces $F^-(001, sss) = s_0^1(F^-(01, s(x)))$. The only compatible edge for $F^-(01, s(x))$ is y_1y_0. Since $s_0^1(y_1y_0) = T_{\text{de}}$ we get $F^-(001, sss) = T_{\text{de}}$.

In day 2, we have that $eee = sss$; but we also have that $T_{\text{de}} \neq y_1y_0\_y_1y_0\_k$, showing that there can be no consistent F^- satisfying the desired requirements.

6.6 Formal verification of the Kripke model

The Kripke model from Section 6.5 is quite complex. Verifying that it has the properties claimed is not a trivial task; it is for this reason that we have formally verified it using the Coq proof assistant.[2] Using Coq in this way—essentially, as a model checker—is not very common, but it worked quite well. The reason is the nature of our model checking problem; we have a model with relatively few states and we want to prove many properties. Encoding the model in Coq makes it easy to read the statements of the theorems, verifying that they indeed prove what we need to prove.

The model checking is divided into two separate parts. The first part is the encoding of the Kripke model and the second part consists of the proofs that the encoding satisfies the desired properties.

Section A contains the complete list of Theorems (sans proofs) and definitions.

6.6.1 Encoding the Kripke model

The encoding of the Kripke model from Section 6.5 is quite direct. First, we define that there are two days:

```
Inductive Days := d1 | d2.
```

We then define each of the sorts in the Kripke model—points, edges and triangles—for both X and Y, as finite inductive types. Here we show it for Y; the definitions are similar for X.

```
Inductive VY := y0 | y1.
Inductive EdgeY := y0y0 | y0y1 | y1y0 | y1y1| k.

Inductive TriangleY :=
| y0y0_y0y0_y0y0
| y0y1_y0y1_y0y0
```

[2] See `https://github.com/epa095/funKanPowCounterModel-coq` for the Coq script.

```
⋮
| fi
| de.
```

The names of the triangles are given by $d_0\_d_1\_d_2$ of the triangle, so as an example $d_2(y0y1\_y0y1\_y0y0) = y0y0$. We then encode the functions $d_0^1, d_1^1, d_0^2, d_1^2, d_2^2, s, s_0, s_1$, which we name sY, d0Y, d1Y, se0Y, se1Y, dp0Y, dp1Y, and dp2Y. They are all defined explicitly for all possible inputs, as shown in the following example.

```
Function sY (v1 :VY) := match v1 with
                        | y0 ⇒ y0y0
                        | y1 ⇒ y1y1
                        end.
```

We are using an explicitly defined equivalence relation for each sort. In day 1, two elements are equal exactly when they have the same constructors. In day 2, the edge $y1y1$ is equated with k; otherwise, edges are equal when they have the same constructor. Two triangles are equal when their faces are equal, except de, which is only equal to itself:

```
Function eqTriangleY (day : Days)(t1 t2 :TriangleY) :=
 match day with
   | d1 ⇒ sameConstructorTriangleY t1 t2
   | d2 ⇒
     match t1,t2 with
       | de,de ⇒ true
       | de,_ | _,de ⇒ false
       | _,_ ⇒ andb (eqEdgeY d2 (dp0Y t1) (dp0Y t2))
              (andb (eqEdgeY d2 (dp1Y t1) (dp1Y t2))
                    (eqEdgeY d2 (dp2Y t1) (dp2Y t2)))
     end
 end.
```

Finally, we define the filler functions, $\text{fill}_{1,i}$: `EdgeY` × `EdgeY` → `TriangleY` and $\text{fill}_{2,i}$: `TriangleY` × `TriangleY` × `TriangleY` → `TriangleY`. We implement them according to Section 6.5.1, making sure that e.g $\text{fill}_{1,i}(y1y0, y1y1) = fi$ instead of de, and similarly for the other inputs. For $\text{fill}_{2,i}$, we make use of a function `edgesToTriangle` which maps three edges $d0, d1, d2$ to a triangle t, such that $dp0Y(t) = d0$, $dp1Y(t) = d1$ and $dp2Y(t) = d2$. There are two things to notice. First, `edgesToTriangle` maps to fi and not de when it has a choice; and since it needs to be total, it also maps every non-compatible triple of edges to the triangle $y0y0\_y0y0\_y0y0$. The implementation of $\text{fill}_{2,i}$ consists of just picking out the right edges from its argument, as exemplified by $\text{fill}_{2,0}$:

```
Function fill20Y (t1 t2 t3 :TriangleY) := (edgesToTriangle (dp0Y t1)
```

```
                                    (dp0Y t2)
                                    (dp0Y t3)).
```

This concludes the definition of Y and its associated functions. The encoding of X is slightly simpler, since it has exactly one triangle per compatible triple of edges, eliminating the fi vs de distinction. In addition, we encode $\Delta^1|_2[n]$, $0 \leq n \leq 2$ with face and degeneracy maps in the same explicit way, with `Delta10`, `Delta11` and `Delta12` being points, edges and triangles respectively. This lets us define the function $F : \Delta^1|_2[n] \times X[n] \to Y[n]$ for $0 \leq n \leq 2$ from Section 6.5.1 explicitly, ending the definition of the Kripke model.

```
Function Fv (delt:Delta10) (v:VX) := ...
Function Fe (delt:Delta11) (e:EdgeX) :=...
Function Ft (delt:Delta12) (t:TriangleX) :=
```

6.6.2 Verifying the encoded model

The encoding above is straightforward, but tedious to verify. In this section, we will explain how we used Coq to show that the encoded model has the properties desired.

A useful feature for doing this is Coq's type classes. These enables us to define a collection of properties—parameterized on types and functions—and give several instantiations of those types and functions, ensuring that each instance satisfies the properties specified in the type class.

We define two type classes: one for reflexive hypergraphs, and another for Kan fill-hypergraphs. We show that X, Y and Delta are instances of the first class, and that X and Y are instances of the second. In what follows, we will describe the properties encoded by these type classes.

We begin by defining what it means for a function `eqFun: Days → A → A → bool` to be an equivalence, before going on to define what it means for a unary, binary and tertary function to respect equality on its domain and co-domain.

```
Definition EquivalenceFun {A:Type}(eqFun: Days → A → A → bool):=
ReflexiveFun eqFun ∧ SymetricFun eqFun ∧ TransitiveFun eqFun.

Definition binaryFunctionRespectsEquality {Domain CoDomain:Type}
        (eqDomain: Days → Domain → Domain → bool)
        (eqCoDomain: Days → CoDomain → CoDomain → bool)
        (function: Domain→  Domain → CoDomain):= ...
```

We now define the class giving the basic properties of X, Y and Delta. It expresses that all of the face and degeneracy maps respect equality, that the equality functions are monotone equalities, and that the simplicial identities

hold between the face and degeneracy maps. The whole class can be found in Appendix A. Giving Y, X and Delta as instances leaves the properties that need to be shown as obligations, which are all pretty straightforward to close.

The next step consists of verifying the filling functions. We construct a type class once more, this time parameterized on a member of the previously defined type class and all seven fill functions. The type class specifies that all of the fill functions must respect equality, in addition to encoding each of the properties the fill functions must respect, as the following example for $\text{fill}_{1,0}$:

```
fill10Prop: forall (d:Days)(e1 e2:Edges),
                  ((eqV d (d1E e1) (d1E e2))=true)→
                      (eqEdges d (d1T (fill10 e1 e2)) e1)=true ∧
                      (eqEdges d (d2T (fill10 e1 e2)) e2)=true
```

Finally, we verify that our encoding of the family of functions F is correct, and that the argumentation from Section 6.5.3 holds. We start by formalizing the notion of a family of functions $F : \Delta^1|_2[n] \times X[n] \to Y[n]$ commuting with both the face and degeneracy maps in X, Delta and Y according to Definition 6.10, before showing that F, as encoded above, commutes.

We then encode that an inverse of F is a family $F^- : \Delta^1|_2 \times X \to Y$ such that, for all $p \in X[0]$, $l \in X[1]$ and $t \in X[2]$, we have

$$\begin{aligned} F^-(0,p) &= F(1,p) & F^-(1,p) &= F(0,p) \\ F^-(00,l) &= F(11,l) & F^-(11,l) &= F(00,l) \\ F^-(000,t) &= F(111,t) & F^-(111,t) &= F(000,t). \end{aligned}$$

We finish by showing that all commuting reverses of F must satisfy $F^-(001, eee) = y1y0\_y1y0\_k$ and $F^-(001, sx\_sx\_sx) = de$, and that these two images remain distinct in both days.

6.7 Conclusion

In this paper, we provided a model showing that we cannot constructively prove that the Kan property is preserved under exponentiation. This means, from a constructive perspective, that Kan simplicial sets are currently unsatisfactory as models of simply typed lambda calculus. The result was shown for a strong interpretation of Kan simplicial sets requiring explicit functions for the horn fillings, closing the gaps from previous work on a similar problem for Kan simplicial sets *without* explicit filler functions. The model has been encoded and verified using the Coq proof assistant.

Appendix A. Theorems proved in Coq

```
  Definition ReflexiveFun {A:Type}(eqFun: Days → A → A → bool):=
    forall (d:Days)(el:A), (eqFun d el el) = true.

  Definition SymetricFun {A:Type}(eqFun: Days → A → A → bool):=
    forall (d:Days)(elem1 elem2:A),
      ((eqFun d elem1 elem2)=true)→ (eqFun d elem1 elem2)=true.

  Definition TransitiveFun {A:Type}(eqFun: Days → A → A → bool):=
    forall (d:Days)(elem1 elem2 elem3:A),
      ((eqFun d elem1 elem2)=true ∧ (eqFun d elem2 elem3)=true) →
      (eqFun d elem1 elem3)=true.

  Definition EquivalenceFun {A:Type}(eqFun: Days → A → A → bool):=
    ReflexiveFun eqFun ∧ SymetricFun eqFun ∧ TransitiveFun eqFun.

Definition unaryFunctionRespectsEquality {Domain CoDomain:Type}
          (eqDomain: Days → Domain → Domain → bool)
          (eqCoDomain: Days → CoDomain → CoDomain → bool)
          (function: Domain→ CoDomain):=
  forall d:Days, forall (elem1 elem2: Domain),
    (eqDomain d elem1 elem2)=true →
      (eqCoDomain d (function elem1) (function elem2))=true.

Definition binaryFunctionRespectsEquality {Domain CoDomain:Type}
          (eqDomain: Days → Domain → Domain → bool)
          (eqCoDomain: Days → CoDomain → CoDomain → bool)
          (function: Domain→  Domain → CoDomain):=
  forall d:Days, forall (elem1 elem1p elem2 elem2p: Domain),
    (eqDomain d elem1 elem1p)=true ∧ (eqDomain d elem2 elem2p)=true
    → (eqCoDomain d (function elem1 elem2) (function elem1p elem2p))=true.

Definition tertaryFunctionRespectsEquality {Domain CoDomain:Type}
          (eqDomain: Days → Domain → Domain → bool)
```

```
          (eqCoDomain: Days → CoDomain → CoDomain → bool)
          (function: Domain→ Domain→ Domain → CoDomain):=
  forall d:Days, forall (elem1 elem2 elem3 elem1p elem2p elem3p: Domain),
    ((eqDomain d elem1 elem1p)=true ∧
     (eqDomain d elem2 elem2p)=true ∧
     (eqDomain d elem3 elem3p)=true)
    → (eqCoDomain d (function elem1 elem2 elem3)
                    (function elem1p elem2p elem3p))=true.

Definition EqFunctionMonotone {Domain:Type}
       (eq: Days → Domain → Domain → bool):=
       forall (elem1 elem2: Domain),
         (eq d1 elem1 elem2)=true → (eq d2 elem1 elem2)=true.

Class twoDayKripke (Points Edges Triangles :Type)
:= {
    sP : Points → Edges;
    d0E : Edges → Points;
    d1E : Edges → Points;
    s0E : Edges → Triangles;
    s1E : Edges → Triangles;
    d0T : Triangles → Edges;
    d1T : Triangles → Edges;
    d2T : Triangles → Edges;
    eqV : Days → Points → Points → bool;
    eqEdges : Days → Edges → Edges → bool;
    eqTriangles : Days → Triangles → Triangles → bool;
    eqVisEq : EquivalenceFun eqV;
    eqEdgessisEq : EquivalenceFun eqEdges;
    eqTrianglesisEq : EquivalenceFun eqTriangles;
    _ : EqFunctionMonotone eqV;
    _ : EqFunctionMonotone eqEdges;
    _ : EqFunctionMonotone eqTriangles;
    sPRespectsEq : unaryFunctionRespectsEquality eqV eqEdges sP;
    d0ERespectsEq : unaryFunctionRespectsEquality eqEdges eqV d0E;
    d1ERespectsEq : unaryFunctionRespectsEquality eqEdges eqV d1E;
    s0ERespectsEq : unaryFunctionRespectsEquality eqEdges eqTriangles s0E;
    s1ERespectsEq : unaryFunctionRespectsEquality eqEdges eqTriangles s1E;
    d0TRespectsEq : unaryFunctionRespectsEquality eqTriangles eqEdges d0T;
    d1TRespectsEq : unaryFunctionRespectsEquality eqTriangles eqEdges d1T;
    d2TRespectsEq : unaryFunctionRespectsEquality eqTriangles eqEdges d2T;
    (* Simplicial identity 1 *)
    _: forall t: Triangles, d0E(d1T(t)) = d0E(d0T(t));
    _: forall t: Triangles, d0E(d2T(t)) = d1E(d0T(t));
```

```
    _: forall t: Triangles, d1E(d2T(t)) = d1E(d1T(t));
    (* Simplicial identity 2 *)
    _: forall e: Edges, d0T(s1E(e)) = sP(d0E(e)) ;
    (* Simplicial identity 3 *)
    _: forall p: Points, d0E(sP(p)) = p;
    _: forall p: Points, d1E(sP(p)) = p;
    _: forall e: Edges,  d0T(s0E(e)) = e;
    _: forall e: Edges,  d1T(s0E(e)) = e;
    _: forall e: Edges,  d1T(s1E(e)) = e;
    _: forall e: Edges,  d2T(s1E(e)) = e;
    (* Simplicial identity 4 *)
    _ : forall e:Edges, d2T(s0E(e)) = sP(d1E(e));
    (* Simplicial identity 5 *)
    _: forall p: Points, s1E(sP(p)) = s0E(sP(p))
 }.

Class fillableModel {Points Edges Triangles:Type}
                    {m: twoDayKripke Points Edges Triangles} := {
fill10: Edges → Edges → Triangles;
fill11: Edges → Edges → Triangles;
fill12: Edges → Edges → Triangles;
fill20: Triangles → Triangles → Triangles → Triangles;
fill21: Triangles → Triangles → Triangles → Triangles;
fill22: Triangles → Triangles → Triangles → Triangles;
fill23: Triangles → Triangles → Triangles → Triangles;

fill10RespectEquality: binaryFunctionRespectsEquality eqEdges eqTriangles fill10;
fill11RespectEquality: binaryFunctionRespectsEquality eqEdges eqTriangles fill11;
fill12RespectEquality: binaryFunctionRespectsEquality eqEdges eqTriangles fill12;
fill20RespectsEqualty: tertaryFunctionRespectsEquality eqTriangles eqTriangles fill20;
fill21RespectsEqualty: tertaryFunctionRespectsEquality eqTriangles eqTriangles fill21;
fill22RespectsEqualty: tertaryFunctionRespectsEquality eqTriangles eqTriangles fill22;
fill23RespectsEqualty: tertaryFunctionRespectsEquality eqTriangles eqTriangles fill23;

fill10Prop: forall (d:Days)(e1 e2:Edges),
             ((eqV d (d1E e1) (d1E e2))=true)→
                (eqEdges d (d1T (fill10 e1 e2)) e1)=true ∧
                (eqEdges d (d2T (fill10 e1 e2)) e2)=true;

fill11Prop: forall (d:Days)(e1 e2:Edges),
             ((eqV d (d1E e1) (d0E e2))=true)→
                (eqEdges d (d0T (fill11 e1 e2)) e1)=true ∧
                (eqEdges d (d2T (fill11 e1 e2)) e2)=true;
```

```
fill12Prop: forall (d:Days)(e0 e1:Edges),
             ((eqV d (d0E e0) (d0E e1))=true)→
                (eqEdges d (d0T (fill12 e0 e1)) e0)=true ∧
                (eqEdges d (d1T (fill12 e0 e1)) e1)=true;

fill20Prop: forall (d:Days)(t1 t2 t3:Triangles),
             (eqEdges d (d1T t1) (d1T t2))=true ∧
             (eqEdges d (d2T t1) (d1T t3))=true ∧
             (eqEdges d (d2T t2) (d2T t3))=true →
               ((eqEdges d (d0T t1) (d0T (fill20 t1 t2 t3)))=true ∧
                (eqEdges d (d0T t2) (d1T (fill20 t1 t2 t3)))=true ∧
                (eqEdges d (d0T t3) (d2T (fill20 t1 t2 t3)))=true);

fill21Prop: forall (d:Days)(t1 t2 t3:Triangles),
             (eqEdges d (d1T t1) (d0T t2))=true ∧
             (eqEdges d (d2T t1) (d0T t3))=true ∧
             (eqEdges d (d2T t2) (d2T t3))=true →
               ((eqEdges d (d0T t1) (d0T (fill21 t1 t2 t3)))=true ∧
                (eqEdges d (d1T t2) (d1T (fill21 t1 t2 t3)))=true ∧
                (eqEdges d (d1T t3) (d2T (fill21 t1 t2 t3)))=true);

fill22Prop: forall (d:Days)(t1 t2 t3:Triangles),
             (eqEdges d (d0T t1) (d0T t2))=true ∧
             (eqEdges d (d2T t1) (d0T t3))=true ∧
             (eqEdges d (d2T t2) (d1T t3))=true →
               ((eqEdges d (d1T t1) (d0T (fill22 t1 t2 t3)))=true ∧
                (eqEdges d (d1T t2) (d1T (fill22 t1 t2 t3)))=true ∧
                (eqEdges d (d2T t3) (d2T (fill22 t1 t2 t3)))=true);

fill23Prop: forall (d:Days)(t1 t2 t3:Triangles),
             (eqEdges d (d0T t1) (d0T t2))=true ∧
             (eqEdges d (d1T t1) (d0T t3))=true ∧
             (eqEdges d (d1T t2) (d1T t3))=true →
               ((eqEdges d (d2T t1) (d0T (fill23 t1 t2 t3)))=true ∧
                (eqEdges d (d2T t2) (d1T (fill23 t1 t2 t3)))=true ∧
                (eqEdges d (d2T t3) (d2T (fill23 t1 t2 t3)))=true)

}.

Definition FCommutesS (Fpoint: Delta10 → VX → VY)
          (FEdge: Delta11 → EdgeX → EdgeY)
           (FTriangle: Delta12 → TriangleX → TriangleY):=
  (forall (deltp:Delta10) (p:VX), FEdge (s deltp) (sX p) = sY (Fpoint deltp p))
  ∧ (forall (d:Days) (delt:Delta11) (e:EdgeX),
```

```
      eqTriangleY d (FTriangle (s0 delt) (se0X e))
                    (se0Y (FEdge delt e)) = true)
  ∧ (forall (d:Days) (delt:Delta11) (e:EdgeX),
      eqTriangleY d (FTriangle (s1 delt) (se1X e))
                    (se1Y (FEdge delt e)) = true).

Definition FCommutesD (Fpoint: Delta10 → VX → VY)
           (FEdge: Delta11 → EdgeX → EdgeY)
           (FTriangle: Delta12 → TriangleX → TriangleY) :=
  (forall (delt:Delta11) (e:EdgeX),
     Fpoint (dv0 delt) (d0X e) = d0Y (FEdge delt e)) ∧
  (forall (delt:Delta11) (e:EdgeX),
     Fpoint (dv1 delt) (d1X e) = d1Y (FEdge delt e)) ∧
  (forall (delt:Delta12) (e:TriangleX),
     FEdge (dp2 delt) (dp2X e) = dp2Y (FTriangle delt e)) ∧
  (forall (delt:Delta12) (e:TriangleX),
     FEdge (dp1 delt) (dp1X e) = dp1Y (FTriangle delt e)) ∧
  forall (delt:Delta12) (e:TriangleX),
     FEdge (dp0 delt) (dp0X e) = dp0Y (FTriangle delt e).

Definition FCommutes (Fpoint: Delta10 → VX → VY)
           (FEdge: Delta11 → EdgeX → EdgeY)
           (FTriangle: Delta12 → TriangleX → TriangleY) :=
  (FCommutesS Fpoint FEdge FTriangle) ∧ (FCommutesD Fpoint FEdge FTriangle).

Theorem FOrdCommutesS: FCommutesS Fv Fe Ft.
Theorem FOrdCommutesD: FCommutesD Fv Fe Ft.

Theorem FVRespectsEq: forall (delt:Delta10),
                        unaryFunctionRespectsEquality eqVX eqVY (Fv delt).
Theorem FERespectsEq: forall (delt:Delta11),
                      unaryFunctionRespectsEquality eqEdgeX eqEdgeY (Fe delt).
Theorem FTRespectsEq: forall (delt:Delta12),
            unaryFunctionRespectsEquality eqTriangleX eqTriangleY (Ft delt).

Theorem allFInverseInconsistentEEE:
  forall (FIpoint: Delta10 → VX → VY)
         (FIEdge: Delta11 → EdgeX → EdgeY)
         (FITriangle: Delta12 → TriangleX → TriangleY),
           Finverse FIpoint FIEdge FITriangle →
           FCommutesS FIpoint FIEdge FITriangle →
           FCommutesD FIpoint FIEdge FITriangle →
             FITriangle delta001 e_e_e = y1y0_y1y0_k.
```

```
Theorem allFInverseInconsistentsss:
  forall (FIpoint: Delta10 → VX → VY)
         (FIEdge: Delta11 → EdgeX → EdgeY)
         (FITriangle: Delta12 → TriangleX → TriangleY),
           Finverse FIpoint FIEdge FITriangle →
           FCommutesS FIpoint FIEdge FITriangle →
           FCommutesD FIpoint FIEdge FITriangle →
             FITriangle delta001 sx_sx_sx = de.

Theorem deNotEqualToThatOtherEdge: forall (d:Days),
                                     (eqTriangleY d y1y0_y1y0_k de)=false.
```

Appendix B. Triangles in Y

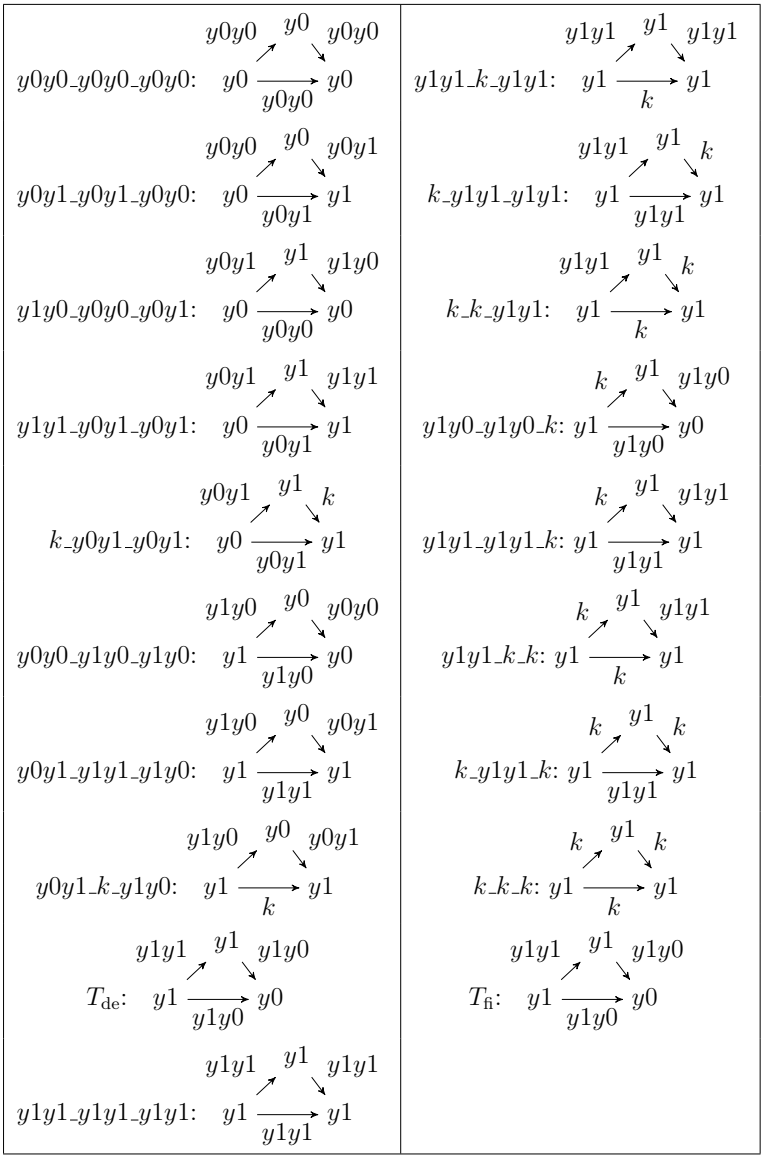


Table B.1: Triangles in Y.

$$s_0(\mathrm{y0y0}) = y0y0\_y0y0\_y0y0\colon \quad \begin{array}{ccc} & y0y0 \;\; y0 \;\; y0y0 & \\ y0 & \longrightarrow & y0 \\ & y0y0 & \end{array}$$

$$s_1(\mathrm{y0y0}) = y0y0\_y0y0\_y0y0\colon \quad \begin{array}{ccc} & y0y0 \;\; y0 \;\; y0y0 & \\ y0 & \longrightarrow & y0 \\ & y0y0 & \end{array}$$

$$s_0(\mathrm{y0y1}) = y0y1\_y0y1\_y0y0\colon \quad \begin{array}{ccc} & y0y0 \;\; y0 \;\; y0y1 & \\ y0 & \longrightarrow & y1 \\ & y0y1 & \end{array}$$

$$s_1(\mathrm{y0y1}) = y1y1\_y0y1\_y0y1\colon \quad \begin{array}{ccc} & y0y1 \;\; y1 \;\; y1y1 & \\ y0 & \longrightarrow & y1 \\ & y0y1 & \end{array}$$

$$s_0(\mathrm{y1y0}) = T_{\mathrm{de}}\colon \quad \begin{array}{ccc} & y1y1 \;\; y1 \;\; y1y0 & \\ y1 & \longrightarrow & y0 \\ & y1y0 & \end{array}$$

$$s_1(\mathrm{y1y0}) = y0y0\_y1y0\_y1y0\colon \quad \begin{array}{ccc} & y1y0 \;\; y0 \;\; y0y0 & \\ y1 & \longrightarrow & y0 \\ & y1y0 & \end{array}$$

$$s_0(\mathrm{y1y1}) = y1y1\_y1y1\_y1y1\colon \quad \begin{array}{ccc} & y1y1 \;\; y1 \;\; y1y1 & \\ y1 & \longrightarrow & y1 \\ & y1y1 & \end{array}$$

$$s_1(\mathrm{y1y1}) = y1y1\_y1y1\_y1y1\colon \quad \begin{array}{ccc} & y1y1 \;\; y1 \;\; y1y1 & \\ y1 & \longrightarrow & y1 \\ & y1y1 & \end{array}$$

$$s_0(\mathrm{k}) = k\_k\_y1y1\colon \quad \begin{array}{ccc} & y1y1 \;\; y1 \;\; k & \\ y1 & \longrightarrow & y1 \\ & k & \end{array}$$

$$s_1(\mathrm{k}) = y1y1\_k\_k\colon \; \begin{array}{ccc} & k \;\; y1 \;\; y1y1 & \\ y1 & \longrightarrow & y1 \\ & k & \end{array}$$

Table B.2: Degenerate triangles in Y.

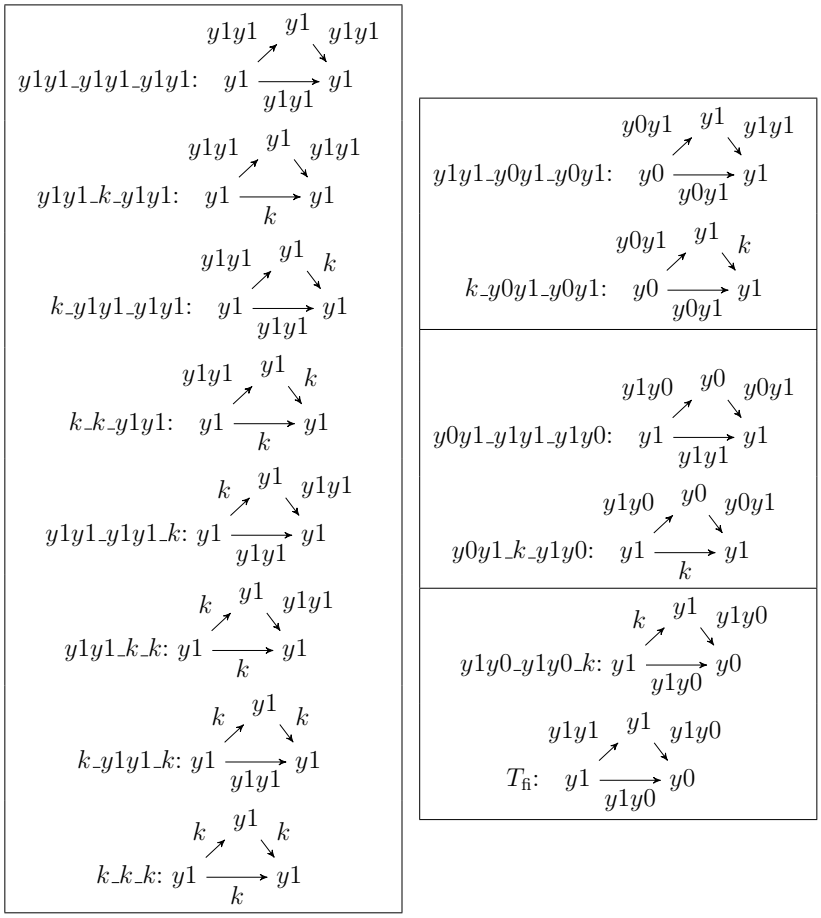


Table B.3: Equated triangles in Y day 2.

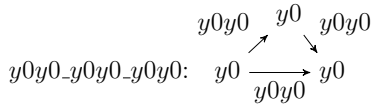
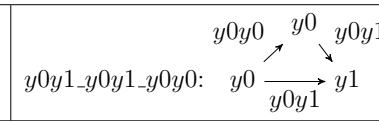
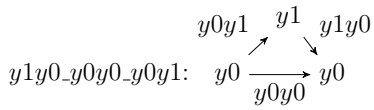
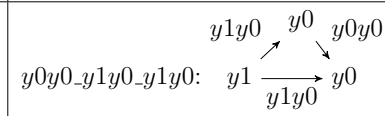
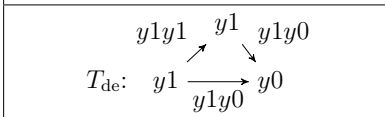
$y0y0\_y0y0\_y0y0$: 	$y0y1\_y0y1\_y0y0$: 
$y1y0\_y0y0\_y0y1$: 	$y0y0\_y1y0\_y1y0$: 
T_{de}: 	

Table B.4: Non-equated triangles in Y day 2.

Bibliography

[AW09] Steve Awodey and Michael Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146:45–55, 2009.

[BC15] Marc Bezem and Thierry Coquand. A kripke model for simplicial sets. *Theoretical Computer Science*, 574:86 – 91, 2015.

[BCH14] Marc Bezem, Thierry Coquand, and Simon Huber. A Model of Type Theory in Cubical Sets. In Ralph Matthes and Aleksy Schubert, editors, *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107–128, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BCNP] Marc Bezem, Thierry Coquand, Keiko Nakata, and Erik Parmann. A Realizability model for type theory. Work in progress.

[BCP15] Marc Bezem, Thierry Coquand, and Erik Parmann. Non-Constructivity in Kan Simplicial Sets. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 92–106, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal logic*. Cambridge University Press, New York, NY, USA, 2001.

[Bee79] Michael Beeson. Goodman's theorem and beyond. *Pacific Journal of Mathematics*, 84(1):1–16, 1979.

[BNU12] Marc Bezem, Keiko Nakata, and Tarmo Uustalu. On streams that are finitely red. *Logical Methods in Computer Science*, Volume 8, Issue 4, October 2012.

[BP13] Douglas Bridges and Erik Palmgren. Constructive mathematics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2013 edition, 2013.

[CDT12] The Coq Development Team. *The Coq Reference Manual, version 8.4*, August 2012. Available electronically at `http://coq.inria.fr/doc`.

[Coq15] Thierry Coquand. Type theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2015 edition, 2015.

[Cro15] Laura Crosilla. Set theory: Constructive and intuitionistic zf. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2015 edition, 2015.

[CS10] Thierry Coquand and Arnaud Spiwack. Constructively finite? In *Contribuciones científicas en honor de Mirian Andrés Gómez*, pages 217–230. Universidad de La Rioja, 2010.

[Dia75] Radu Diaconescu. Axiom of choice and complementation. *Proceedings of the American Mathematical Society*, 51(1):176–178, 1975.

[Fri08] Greg Friedman. An elementary illustrated introduction to simplicial sets. Preprint, `http://arxiv.org/abs/0809.4221`, 2008.

[GJ09] Paul G. Goerss and John F. Jardine. *Simplicial Homotopy Theory*. Modern Birkhäuser Classics. Birkhauser Verlag GmbH, 2009. Reprint of Vol. 174 of Progress in Mathematics, 1999.

[GZ67] Peter Gabriel and Michel Zisman. *Calculus of fractions and homotopy theory*. Springer, 1967.

[Hof97] Martin Hofmann. Syntax and semantics of dependent types. In Andrew M. Pitts and Peter Dybjer, editors, *Semantics and logics of computation*, volume 14 of *Publ. Newton Inst.*, pages 79–130. Cambridge University Press, Cambridge, 1997.

[Jar53] Dov Jarden. A simple proof that a power of an irrational number to an irrational exponent may be rational. *Scr. Math*, 19:229, 1953.

[KLV12] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. Preprint, `http://arxiv.org/abs/1211.2851`, 2012.

[Kri65] Saul Kripke. Semantical analysis of intuitionistic logic I. In M. Dummett and J.N. Crossley, editors, *Formal Systems and Recursive Functions*, pages 92–130. North–Holland, Amsterdam, 1965.

[Lub15] Bob Lubarsky. Personal communication, March 2015.

[May93] Jon Peter May. *Simplicial Objects in Algebraic Topology*. Chicago Lectures in Mathematics. University of Chicago Press, 2nd edition, 1993.

[McC15] Charles McCarty. Two questions about IZF and intuitionistic validity. Pdf file, personal communication, March 2015.

[ML75] Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

[MLS84] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory*, volume 17. Bibliopolis Naples, 1984.

[Moo56] John C. Moore. Algebraic homotopy theory. Lectures at Princeton, `http://faculty.tcu.edu/gfriedman/notes/aht1.pdf`, 1956.

[Nik11] Thomas Nikolaus. Algebraic models for higher categories. *Indagationes Mathematicae*, 21(1–2):52 – 75, 2011.

[Par15] Erik Parmann. Investigating Streamless Sets. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, volume 39 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 187–201, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Ram30] Frank P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, 2(1):264–286, 1930.

[Ric90] Fred Richman. Intuitionism as generalization. *Philosophia Math*, 5:124–128, 1990.

[RS93] Fred Richman and Gabriel Stolzenberg. Well quasi-ordered sets. *Advances in Mathematics*, 97(2):145 – 153, 1993.

[SU06] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism, Volume 149 (Studies in Logic and the Foundations of Mathematics)*. Elsevier Science Inc., New York, NY, USA, 2006.

[Tur36] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 43(230-265), 1936.

[TVD88] Anne Sjerp Troelstra and Dirk Van Dalen. *Constructivism in mathematics*, volume 2. Elsevier, 1988.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[VB93] Wim Veldman and Marc Bezem. Ramsey's theorem and the pigeonhole principle in intuitionistic mathematics. *Journal of the London Mathematical Society*, 2(2):193–211, 1993.

[VCW12] Dimitrios Vytiniotis, Thierry Coquand, and David Wahlstedt. Stop when you are almost-full. In *Interactive Theorem Proving*, pages 250–265. Springer, 2012.

[Voe09] Vladimir Voevodsky. Notes on type systems. `http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/expressions_current.pdf`, 2009.