Functional Kan Simplicial Sets: Non-Constructivity of Exponentiation

Erik Parmann

Institutt for Informatikk, Universitetet i Bergen,
Postboks 7803, 5020 Bergen, Norway
eparmann@gmail.com

Abstract

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 [TLCA 2015, v. 38 of LIPIcs]. 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.

1998 ACM Subject Classification F.4.1 Mathematical Logic and Formal Languages: Mathematical Logic

Keywords and phrases constructive logic, simplicial sets, semantics of simple types

Digital Object Identifier 10.4230/LIPIcs.TYPES.2015.8

1 Introduction

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

▶ **Theorem 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 [2], 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 4.

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 [5].

© Erik Parmann;
licensed under Creative Commons License CC-BY
21st International Conference on Types for Proofs and Programs (TYPES 2015).
Editor: Tarmo Uustalu; Article No. 8; pp. 8:1–8:25

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Theorem 1 is true classically, even without requiring that B is Kan (cf. [11, Appendix A, Theorem 3] or [10, 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 1 cannot be constructively proven is to show that the following constructive consequence of it cannot be constructively proven.

▶ **Theorem 2** (classical). *If B and A are functional Kan simplicial sets, then any edge in A^B can be reversed.*

In [2] we gave a Kripke counterexample to the constructive provability of Theorem 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 2 on the resulting simplicial set, we got that the graphs have a particular feature we can call *function-space 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 [2]), ensuring congruence turns out to be quite involved. To validate correctness, we have encoded and verified the model in the Coq [4] proof assistant.

The simplicial set A^B in Theorem 1 is not claimed to be functional Kan. This makes Theorem 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 [2].

In [3] 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 [3].

The first section of [2] 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) [8] with the homotopy theoretic interpretation of equality, and in this construction, Theorem 1 is important for the interpretation of function types. The results in [3] 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 [2] 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 [1]. 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 2, we introduce simplicial sets and provide several examples of simplicial sets which will be used later. In Section 3, we define hypergraphs which we can constructively interpret as simplicial sets. In Section 4, we use that interpretation, in combination with Theorem 1, to formulate a theorem about graphs. In Section 5, we provide a Kripke model rejecting the constructive provability of this theorem. In Section 6, we explain how we used the proof assistant Coq to verify the Kripke model, before concluding in Section 7.

2 Simplicial Sets

We start by recalling the formal definition of simplicial sets and Kan simplicial sets from [2]. We also introduce functional Kan simplicial sets, before we provide a more intuitive explanation intended for those new to simplicial sets.

▶ **Definition 3** (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:

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

$$d_i s_j = s_{j-i} d_i \qquad \text{if } i < j \tag{2}$$

$$d_i s_j = id \qquad \text{for } i = j, j+1 \tag{3}$$

$$d_i s_j = s_j d_{i-1} \qquad \text{if } i > j+1 \tag{4}$$

$$s_i s_j = s_j s_{i-1} \qquad \text{if } i > j \tag{5}$$

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 [10, 7, 6].

▶ **Definition 4** (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* $\text{fill}^{n-1}_k : Y[n-1] \times \ldots Y[n-1] \to Y[n]$ giving the required n-simplex, then we say that Y is a *functional Kan simplicial set*.

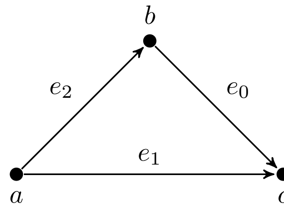


Figure 1 A single triangle.

A similar notion to functional Kan simplicial sets has been introduced by Thomas Nikolaus [12] 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 5.

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 – $d^1_0 : A[1] \to A[0]$ and $d^1_1 : A[1] \to A[0]$ – and we say that d^1_1 gives the startpoint and d^1_0 gives the endpoint of an edge. Likewise, there are three functions from triangles to edges, $d^2_0, d^2_1, d^2_2 : A[2] \to A[1]$ (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 1 shows a triangle t consisting of three edges, e_0, e_1 and e_2, with $d^2_i(t) = e_i$. 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 1.

In addition to the face maps d^n_j, there are the degeneracy maps $s^n_j : A[n] \to A[n+1]$. 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^1_0 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 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 2-4, while simplicial identity 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^1_0 of this edge, both provide the same degenerate triangle (with all three faces degenerate.)

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

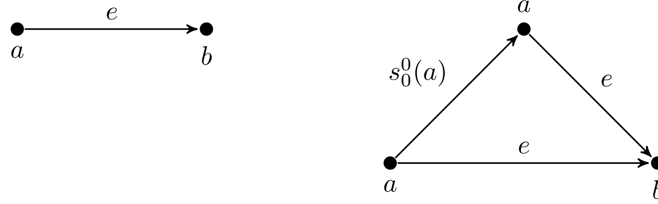


Figure 2 An edge e and the degenerate triangle $s_0^1(e)$.

Figure 3 An example of two compatible edges getting filled.

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 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 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 4 express precisely that we lack exactly one element in $A[n]$ to have all the faces of an element in $A[n+1]$.

▶ **Notation 5.** *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*

$$t : \begin{array}{c} \overset{e_2}{\nearrow} \quad \overset{e_0}{\searrow} \\ \xrightarrow[e_1]{} \end{array}$$

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 $t : \begin{array}{c} \overset{e_2}{\nearrow} \quad \overset{e_0}{\searrow} \\ \xrightarrow[e_1]{} \end{array}$, 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

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

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** (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 $d_1^1(f) = d_0^1(e)$ and $d_0^1(f) = d_1^1(e)$. 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 7.** *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

$$f = d_0^2(\mathrm{fill}_0^1(s_0^0(d_1^1(e)), e)).$$

If $e : a \to b$, then $s_0^0(d_1^1(e)) : a \to a$, and

$$\mathrm{fill}_0^1(s_0^0(d_1^1(e)), e) : \quad \begin{array}{ccc} & b & \\ {}^{e}\nearrow & & \searrow \\ a & \xrightarrow[s_0^0(d_1^1(e))]{} & a \end{array} ,$$

so $d_0^2(\mathrm{fill}_0^1(s_0^0(d_1^1(e)), e)) : b \to a$. ◀

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

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 [2].

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

2.1.2 The k-Horns Λ_j^k

Λ_j^k is the j'th horn of the standard k-simplex Δ^k, and defined by $\Lambda_j^k[n] = \{f \in \Delta^k[n] \mid [k] - \{j\} \not\subseteq Im(f)\}$. Alternatively, it is $\Delta^k[n]$ except every element must avoid some element not equal to j. For example, $\Lambda_0^2[1] = \{00, 01, 02, 11, \not{12}, 22\} = \Delta^2[1] - \{12\}$ (excluding 12, since 12 does not avoid any element not equal to 0). We also have:

$$\Lambda_0^2[2] = \{000, 001, 002, 011, \not{012}, 022, 111, \not{112}, \not{122}, 222\}.$$

The functional Kan extension condition from Definition 4 for a simplicial set Y can also be formulated as: we have a dependent function $\mathrm{fill}(k, j, F)$ such that $\mathrm{fill}(k, j, F) : \Delta^k \to Y$ extends $F : \Lambda_j^k \to Y$, for any k, j, F.

2.1.3 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 $d_i((a, b)) = (d_i^A(a), d_i^B(b))$. 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.)

2.1.4 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 $(s_k F)[i](a, x) = F[i](s_k^* a, x)$.

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 11; these are graphs which can be constructively interpreted as (functional Kan) simplicial sets.

▶ **Definition 8** (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 5 for reflexive hypergraphs as well. We require that all triangles $t \in C_2$ satisfy that, if $t : \overset{e_2 \nearrow \;\; \searrow e_0}{\xrightarrow[e_1]{}}$, 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

$$t : \; a \overset{e_2 \nearrow \; b \; \searrow e_0}{\xrightarrow[e_1]{\hspace{2em}}} c$$

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 $e : a \to b$, $s_0(e) : \; a \overset{s(a) \nearrow \; a \; \searrow e}{\xrightarrow[e]{\hspace{2em}}} b$ and $s_1(e) : \; a \overset{e \nearrow \; b \; \searrow s(b)}{\xrightarrow[e]{\hspace{2em}}} b$, and that for all v, $s_0(s(v)) = s_1(s(v)) : \; v \overset{s(v) \nearrow \; v \; \searrow s(v)}{\xrightarrow[s(v)]{\hspace{2em}}} v$.

The definition above is not much more than a specialization of Definition 3 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 [2] and [3], but extended in such a way that triangles are not necessarily equal when they have equal faces.

▶ **Definition 9** ($S(C)$)**.** 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 $(u_0, \ldots, u_n; \ldots, e_{ij}, \ldots; \ldots, t_{ijl}, \ldots)$ such that

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

$$t_{ijl} : \quad \begin{array}{ccc} & u_j & \\ {}^{e_{ij}}\nearrow & & \searrow^{e_{jl}} \\ u_i & \xrightarrow[e_{il}]{} & u_l \end{array} \quad \text{in } C_2 \text{ for all } 0 \leq i < j < l \leq n.$$

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

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

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 $d^3_k(q) = t_{012}$.

The maps s^n_k in $S(C)$ for $n > 3$ are defined by duplicating the point u_k in the tuple $(u_0, \ldots, u_n; \ldots, e_{ij}, \ldots; \ldots, t_{ijl}, \ldots)$, 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 $t : \begin{array}{ccc} & b & \\ {}^{e_2}\nearrow & & \searrow^{e_0} \\ a & \xrightarrow[e_1]{} & c \end{array}$, 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 4 to the first three dimensions, giving us a first-order structure we can extend to a functional Kan simplicial set.

▶ **Definition 10** (Kan fill-hypergraph)**.** A *Kan fill-hypergraph* is a reflexive hypergraph where we have functions $\text{fill}_{1,i} : C_1 \times C_1 \to C_2$ for $0 \leq i \leq 2$ and $\text{fill}_{2,i} : C_2 \times C_2 \times C_2 \to C_2$ for $0 \leq i \leq 3$, satisfying the following requirements.

For all $e_1, e_2 \in C_1$, $\text{fill}_{1,i} : C_1 \times C_1 \to C_2$ must satisfy:

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

If $e_0 : b \to c$ and $e_2 : a \to b$, then $\text{fill}_{1,1}(e_0, e_2) : \begin{array}{ccc} & b & \\ {}^{e_2}\nearrow & & \searrow^{e_0} \\ a & \xrightarrow{} & c \end{array}$.

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

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

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

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

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

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

Note that the above requirements 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:

$$\forall e_1, e_2 \in C_1, d^1_1(e_1) = d^1_1(e_2) \to d^2_1(\text{fill}_{1,0}(e_1, e_2)) = e_1 \wedge d^2_2(\text{fill}_{1,0}(e_1, e_2)) = e_2.$$

▶ **Lemma 11.** *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 4.

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 $d^2_i r = t_i$ 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 : $\begin{array}{ccc} & c & \\ e_0 \nearrow & & \searrow e_5 \\ b & \xrightarrow[e_4]{} & d \end{array}$, t_2 : $\begin{array}{ccc} & b & \\ e_2 \nearrow & & \searrow e_4 \\ a & \xrightarrow[e_3]{} & d \end{array}$,

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

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 9 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. ◀

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 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 ith dimension $Hom_S(\Delta^i \times A, B)$, so its edges are elements of $Hom_S(\Delta^1 \times A, B)$.

▶ **Definition 12** ($\Delta^i|_m$)**.** 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 13.** 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 2.1.3.

▶ **Lemma 14.** *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} and t'_{ijl} are given by $e'_{ij} = F(gt_a(i)gt_a(j), e_{ij})$ and $t'_{ijl} = F(gt_a(i)gt_a(j)gt_a(l), t_{ijl})$.

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 a 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 1 – which is essentially Theorem 2, reformulated as a property of Kan fill-hypergraphs.

▶ **Theorem 15** (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*

$$F^-(0, p) = F(1, p) \qquad F^-(1, p) = F(0, p)$$
$$F^-(00, l) = F(11, l) \qquad F^-(11, l) = F(00, l)$$
$$F^-(000, t) = F(111, t) \qquad F^-(111, t) = F(000, t)$$

Proof. We first extend F to an edge in $S(Y)^{S(X)}$ by Lemma 14 and note that, by Lemma 11, $S(Y)$ is a functional Kan simplicial set. We then apply the classical Theorem 2, giving that $S(Y)^{S(X)}$ is a Kan simplicial set, enabling us to reverse the edge by Lemma 7, 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 2.1.4 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 15 was the application of Theorem 2.

The reasoning in our constructive proofs can be formalized in IZF (Intuitionistic Zermelo-Fraenkel set theory), so IZF proves that Theorem 1 implies Theorem 2, and that Theorem 2 implies Theorem 15. We also have that if IZF proves Theorem 15, then Theorem 15 holds in any Kripke model. This result has been further elaborated in Section 6 of [2]. For this, it is vital that Theorem 15 can be expressed in first-order logic. So finally, by giving a Kripke model falsifying Theorem 15, we show that IZF cannot prove Theorem 1, and we will provide exactly such a model in the next section.

5 Kripke Countermodel

In this section we present a Kripke model falsifying the first-order sentence representing Theorem 15. 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 [9].

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 10, 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.

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 f in the figure from a to b represents an edge f in the model with $d_0^1(f) = b$ and $d_1^1(f) = a$.

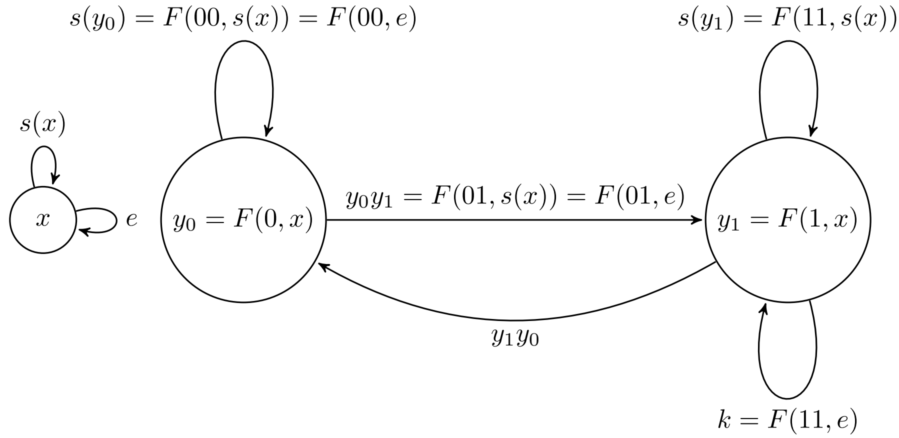


Figure 4 Kripke (counter)model for edge reversal, day 1.

5.1.1 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 concatenation of d_i^2 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:

$$s_0^1(s(x)) = s_1^1(s(x)) = sss$$

$$s_0^1(e) = ees : \quad s(x) \nearrow \searrow e, \ \xrightarrow{e}$$

$$s_1^1(e) = see : \quad e \nearrow \searrow s(x), \ \xrightarrow{e}$$

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.

5.1.2 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 $e_0_e_1_e_2 : \begin{smallmatrix} & & b & & \\ & e_2 \nearrow & & \searrow e_0 & \\ a & & \xrightarrow[e_1]{} & & c \end{smallmatrix}$ 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 1 in Appendix B.

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

$$s_0^1(y_0y_1) = y_0y_1_y_0y_1_s(y_0) : \begin{smallmatrix} & & y_0 & & \\ & s(y_0) \nearrow & & \searrow y_0y_1 & \\ y_0 & & \xrightarrow[y_0y_1]{} & & y_1 \end{smallmatrix}$$

$$s_1^1(y_0y_1) = s(y_1)_y_0y_1_y_0y_1 : \begin{smallmatrix} & & y_1 & & \\ & y_0y_1 \nearrow & & \searrow s(y_1) & \\ y_0 & & \xrightarrow[y_0y_1]{} & & y_1 \end{smallmatrix}$$

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 2 in Appendix B.

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 $\text{fill}_{1,i}$ to be total, we map every non-compatible pair of edges to $y_0y_0_y_0y_0_y_0y_0$.

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

Given three triangles t_1, t_2, t_3, when there is a triangle with the signature $\begin{smallmatrix} & & c & & \\ & e_1 \nearrow & & \searrow e_5 & \\ a & & \xrightarrow[e_3]{} & & d \end{smallmatrix}$ –

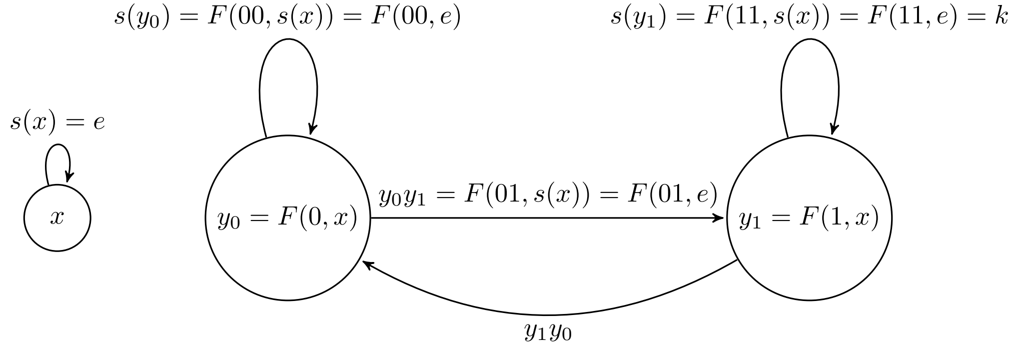


Figure 5 Kripke (counter)model for edge reversal, day 2.

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 $y_0y_0_y_0y_0_y_0y_0$.

5.1.3 *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 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.

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 10 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 5. In $X[2]$, we equate every triangle, leaving us with only 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-equation 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 3; the list of the remaining, non-equated triangles can be found in Table 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 $\text{fill}_{1/2,i}$), 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 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 10 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.

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 15, such that for all $p \in X[0]$, $l \in X[1]$ and $t \in X[2]$ we have:

$$F^-(0, p) = F(1, p) \qquad F^-(1, p) = F(0, p)$$
$$F^-(00, l) = F(11, l) \qquad F^-(11, l) = F(00, l)$$
$$F^-(000, t) = F(111, t) \qquad F^-(111, t) = F(000, t).$$

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.

Since $sss = s_0^1(s(x))$ 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 Formal Verification of the Kripke Model

The Kripke model from Section 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.[1] 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;

[1] See `https://github.com/epa095/funKanPowCounterModel-coq` for the Coq script.

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.

Appendix A contains the complete list of Theorems (sans proofs) and definitions.

6.1 Encoding the Kripke Model

The encoding of the Kripke model from Section 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
⋮
| 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, $\mathrm{fill}_{1,i}$: `EdgeY` × `EdgeY` → `TriangleY` and $\mathrm{fill}_{2,i}$: `TriangleY` × `TriangleY` × `TriangleY` → `TriangleY`. We implement them according to Section 5.1.2, making sure that e.g $\mathrm{fill}_{1,i}(y1y0, y1y1) = fi$ instead of de, and similarly for the other inputs. For $\mathrm{fill}_{2,i}$, 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 $\mathrm{fill}_{2,i}$ consists of just picking out the right edges from its argument, as exemplified by $\mathrm{fill}_{2,0}$:

```
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 5.1.3 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) :=
```

6.2 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 ternary 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 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 13, 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.

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.

References

1. M. Bezem, T. Coquand, and S. Huber. A model of type theory in cubical sets. In R. Matthes and A. Schubert, editors, *Proc. of 19th Int. Conf. on Types for Proofs and Programs, TYPES 2013*, volume 26 of *Leibniz Int. Proc. in Inf.*, pages 107–128. Dagstuhl Publishing, 2014. doi:10.4230/lipics.types.2013.107.
2. M. Bezem, T. Coquand, and E. Parmann. Non-constructivity in Kan simplicial sets. In T. Altenkirch, editor, *Proc. of 13th Int. Conf. on Typed Lambda Calculi and Applications, TLCA 2015*, volume 38 of *Leibniz Int. Proc. in Inf.*, pages 92–106. Dagstuhl Publishing, 2015. doi:10.4230/lipics.tlca.2015.92.
3. Marc Bezem and Thierry Coquand. A Kripke model for simplicial sets. *Theor. Comput. Sci.*, 574:86–91, 2015. doi:10.1016/j.tcs.2015.01.035.

4 The Coq Development Team. The Coq reference manual, version 8.4, 2012. URL: `https://coq.inria.fr/distrib/8.4/refman/`.

5 R. Diaconescu. Axiom of choice and complementation. *Proc. Amer. Math. Soc.*, 51(1):176–178, 1975. `doi:10.1090/s0002-9939-1975-0373893-x`.

6 G. Friedman. An elementary illustrated introduction to simplicial sets. arXiv preprint 0809.4221, 2008. URL: `https://arxiv.org/abs/0809.4221`.

7 P. Goerss and J. F. Jardine. *Simplicial Homotopy Theory*, volume 174 of *Progress in Mathematics*. Birkhäuser, 1999. Reprinted in Modern Birkhäuser Classics, 2009. `doi:10.1007/978-3-0348-8707-6`.

8 C. Kapulkin, P. LeFanu Lumsdaine, and V. Voevodsky. The simplicial model of univalent foundations. arXiv preprint 1211.2851, 2012. URL: `https://arxiv.org/abs/1211.2851`.

9 S. Kripke. Semantical analysis of intuitionistic logic I. In M. Dummett and J.N. Crossley, editors, *Formal Systems and Recursive Functions*, volume 40 of *Studies in Logic and the Foundations of Mathematics*, pages 92–130. North Holland, 1965. `doi:10.1016/s0049-237x(08)71685-9`.

10 J. P. May. *Simplicial Objects in Algebraic Topology*. Chicago Lectures in Mathematics. University of Chicago Press, 2nd edition, 1993.

11 J. C. Moore. Algebraic homotopy theory. Lectures at Princeton, 1956. URL: `http://faculty.tcu.edu/gfriedman/notes/aht1.pdf`.

12 T. Nikolaus. Algebraic models for higher categories. *Indag. Math.*, 21(1–2):52–75, 2011. `doi:10.1016/j.indag.2010.12.004`.

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 SymmetricFun {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.
```

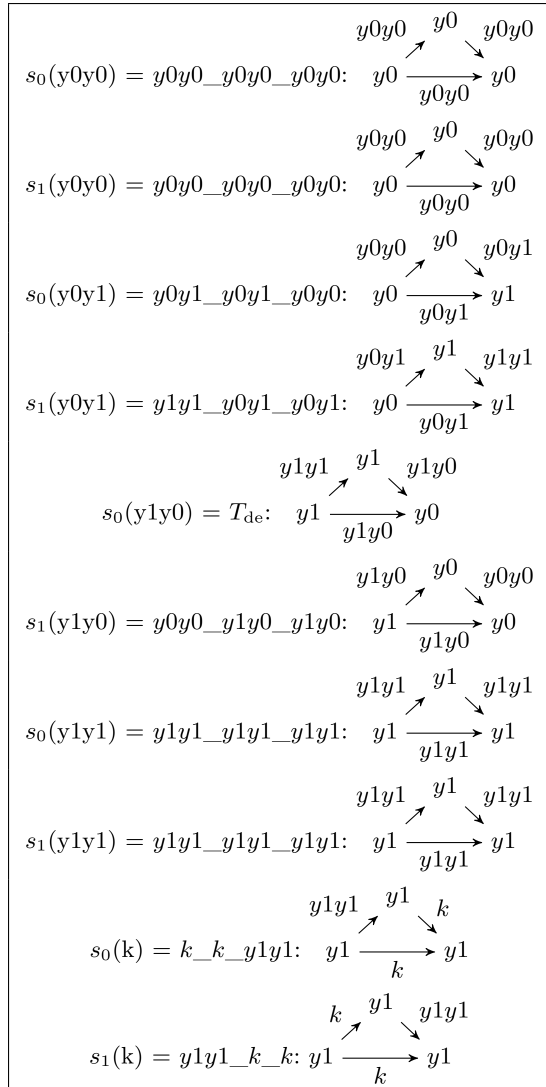
```
Theorem deNotEqualToThatOtherEdge: forall (d:Days), (eqTriangleY d y1y0_y1y0_k de)=false.
```

B Triangles in Y

Table 1 Triangles in Y.

$y0y0_y0y0_y0y0$: $y0 \xrightarrow{y0y0} y0 \xrightarrow{y0y0} y0$, $y0 \xrightarrow{y0y0} y0$	$y1y1_k_y1y1$: $y1 \xrightarrow{y1y1} y1 \xrightarrow{y1y1} y1$, $y1 \xrightarrow{k} y1$
$y0y1_y0y1_y0y0$: $y0 \xrightarrow{y0y0} y0 \xrightarrow{y0y1} y1$, $y0 \xrightarrow{y0y1} y1$	k_y1y1_y1y1: $y1 \xrightarrow{y1y1} y1 \xrightarrow{k} y1$, $y1 \xrightarrow{y1y1} y1$
$y1y0_y0y0_y0y1$: $y0 \xrightarrow{y0y1} y1 \xrightarrow{y1y0} y0$, $y0 \xrightarrow{y0y0} y0$	k_k_y1y1: $y1 \xrightarrow{y1y1} y1 \xrightarrow{k} y1$, $y1 \xrightarrow{k} y1$
$y1y1_y0y1_y0y1$: $y0 \xrightarrow{y0y1} y1 \xrightarrow{y1y1} y1$, $y0 \xrightarrow{y0y1} y1$	$y1y0_y1y0_k$: $y1 \xrightarrow{k} y1 \xrightarrow{y1y0} y0$, $y1 \xrightarrow{y1y0} y0$
k_y0y1_y0y1: $y0 \xrightarrow{y0y1} y1 \xrightarrow{k} y1$, $y0 \xrightarrow{y0y1} y1$	$y1y1_y1y1_k$: $y1 \xrightarrow{k} y1 \xrightarrow{y1y1} y1$, $y1 \xrightarrow{y1y1} y1$
$y0y0_y1y0_y1y0$: $y1 \xrightarrow{y1y0} y0 \xrightarrow{y0y0} y0$, $y1 \xrightarrow{y1y0} y0$	$y1y1_k_k$: $y1 \xrightarrow{k} y1 \xrightarrow{y1y1} y1$, $y1 \xrightarrow{k} y1$
$y0y1_y1y1_y1y0$: $y1 \xrightarrow{y1y0} y0 \xrightarrow{y0y1} y1$, $y1 \xrightarrow{y1y1} y1$	k_y1y1_k: $y1 \xrightarrow{k} y1 \xrightarrow{k} y1$, $y1 \xrightarrow{y1y1} y1$
$y0y1_k_y1y0$: $y1 \xrightarrow{y1y0} y0 \xrightarrow{y0y1} y1$, $y1 \xrightarrow{k} y1$	k_k_k: $y1 \xrightarrow{k} y1 \xrightarrow{k} y1$, $y1 \xrightarrow{k} y1$
T_{de}: $y1 \xrightarrow{y1y1} y1 \xrightarrow{y1y0} y0$, $y1 \xrightarrow{y1y0} y0$	T_{fi}: $y1 \xrightarrow{y1y1} y1 \xrightarrow{y1y0} y0$, $y1 \xrightarrow{y1y0} y0$
$y1y1_y1y1_y1y1$: $y1 \xrightarrow{y1y1} y1 \xrightarrow{y1y1} y1$, $y1 \xrightarrow{y1y1} y1$	

Table 2 Degenerate triangles in Y.



$s_0(\mathrm{y0y0}) = y0y0_y0y0_y0y0$: $y0 \xrightarrow{y0y0} y0$

$s_1(\mathrm{y0y0}) = y0y0_y0y0_y0y0$: $y0 \xrightarrow{y0y0} y0$

$s_0(\mathrm{y0y1}) = y0y1_y0y1_y0y0$: $y0 \xrightarrow{y0y1} y1$

$s_1(\mathrm{y0y1}) = y1y1_y0y1_y0y1$: $y0 \xrightarrow{y0y1} y1$

$s_0(\mathrm{y1y0}) = T_{\mathrm{de}}$: $y1 \xrightarrow{y1y0} y0$

$s_1(\mathrm{y1y0}) = y0y0_y1y0_y1y0$: $y1 \xrightarrow{y1y0} y0$

$s_0(\mathrm{y1y1}) = y1y1_y1y1_y1y1$: $y1 \xrightarrow{y1y1} y1$

$s_1(\mathrm{y1y1}) = y1y1_y1y1_y1y1$: $y1 \xrightarrow{y1y1} y1$

$s_0(\mathrm{k}) = k_k_y1y1$: $y1 \xrightarrow{k} y1$

$s_1(\mathrm{k}) = y1y1_k_k$: $y1 \xrightarrow{k} y1$

Table 3 Equated triangles in Y, day 2.

$y1y1_y1y1_y1y1$: $\begin{array}{ccc} & y1 & \\ \overset{y1y1}{\nearrow} & & \overset{y1y1}{\searrow} \\ y1 & \xrightarrow[y1y1]{} & y1 \end{array}$	$y1y1_y0y1_y0y1$: $\begin{array}{ccc} & y1 & \\ \overset{y0y1}{\nearrow} & & \overset{y1y1}{\searrow} \\ y0 & \xrightarrow[y0y1]{} & y1 \end{array}$
$y1y1_k_y1y1$: $\begin{array}{ccc} & y1 & \\ \overset{y1y1}{\nearrow} & & \overset{y1y1}{\searrow} \\ y1 & \xrightarrow[k]{} & y1 \end{array}$	k_y0y1_y0y1: $\begin{array}{ccc} & y1 & \\ \overset{y0y1}{\nearrow} & & \overset{k}{\searrow} \\ y0 & \xrightarrow[y0y1]{} & y1 \end{array}$
k_y1y1_y1y1: $\begin{array}{ccc} & y1 & \\ \overset{y1y1}{\nearrow} & & \overset{k}{\searrow} \\ y1 & \xrightarrow[y1y1]{} & y1 \end{array}$	$y0y1_y1y1_y1y0$: $\begin{array}{ccc} & y0 & \\ \overset{y1y0}{\nearrow} & & \overset{y0y1}{\searrow} \\ y1 & \xrightarrow[y1y1]{} & y1 \end{array}$
k_k_y1y1: $\begin{array}{ccc} & y1 & \\ \overset{y1y1}{\nearrow} & & \overset{k}{\searrow} \\ y1 & \xrightarrow[k]{} & y1 \end{array}$	$y0y1_k_y1y0$: $\begin{array}{ccc} & y0 & \\ \overset{y1y0}{\nearrow} & & \overset{y0y1}{\searrow} \\ y1 & \xrightarrow[k]{} & y1 \end{array}$
$y1y1_y1y1_k$: $\begin{array}{ccc} & y1 & \\ \overset{k}{\nearrow} & & \overset{y1y1}{\searrow} \\ y1 & \xrightarrow[y1y1]{} & y1 \end{array}$	$y1y0_y1y0_k$: $\begin{array}{ccc} & y1 & \\ \overset{k}{\nearrow} & & \overset{y1y0}{\searrow} \\ y1 & \xrightarrow[y1y0]{} & y0 \end{array}$
$y1y1_k_k$: $\begin{array}{ccc} & y1 & \\ \overset{k}{\nearrow} & & \overset{y1y1}{\searrow} \\ y1 & \xrightarrow[k]{} & y1 \end{array}$	T_{fi}: $\begin{array}{ccc} & y1 & \\ \overset{y1y1}{\nearrow} & & \overset{y1y0}{\searrow} \\ y1 & \xrightarrow[y1y0]{} & y0 \end{array}$
k_y1y1_k: $\begin{array}{ccc} & y1 & \\ \overset{k}{\nearrow} & & \overset{k}{\searrow} \\ y1 & \xrightarrow[y1y1]{} & y1 \end{array}$	
k_k_k: $\begin{array}{ccc} & y1 & \\ \overset{k}{\nearrow} & & \overset{k}{\searrow} \\ y1 & \xrightarrow[k]{} & y1 \end{array}$	

Table 4 Non-equated triangles in Y, day 2.

$y0y0_y0y0_y0y0$: $\begin{array}{ccc} & y0 & \\ \overset{y0y0}{\nearrow} & & \overset{y0y0}{\searrow} \\ y0 & \xrightarrow[y0y0]{} & y0 \end{array}$	$y0y1_y0y1_y0y0$: $\begin{array}{ccc} & y0 & \\ \overset{y0y0}{\nearrow} & & \overset{y0y1}{\searrow} \\ y0 & \xrightarrow[y0y1]{} & y1 \end{array}$
$y1y0_y0y0_y0y1$: $\begin{array}{ccc} & y1 & \\ \overset{y0y1}{\nearrow} & & \overset{y1y0}{\searrow} \\ y0 & \xrightarrow[y0y0]{} & y0 \end{array}$	$y0y0_y1y0_y1y0$: $\begin{array}{ccc} & y0 & \\ \overset{y1y0}{\nearrow} & & \overset{y0y0}{\searrow} \\ y1 & \xrightarrow[y1y0]{} & y0 \end{array}$
T_{de}: $\begin{array}{ccc} & y1 & \\ \overset{y1y1}{\nearrow} & & \overset{y1y0}{\searrow} \\ y1 & \xrightarrow[y1y0]{} & y0 \end{array}$	