# Automating Reasoning in Cubical Type Theory

**Abstract.** Cubical type theory uses Kan cubical sets to give a principled account of proof-relevant equality. Properties of Kan cubical sets are treated as logical rules, this paper studies these rules from the point of view of automated reasoning. We give an efficient algorithm for finding cells in a cubical set; establish that the problem of finding cells in a Kan cubical set is undecidable; and explore several sub-problems which are (semi-)decidable using the language of constraint satisfaction programming. Our findings have been incorporated in a tactic for Cubical Agda which solves many commonly encountered proof goals.

**Keywords:** Homotopy type theory · Proof-relevant identity · Automated reasoning · Kan cubical sets

TODO Write introduction tailored to CADE crowd. Motivate with Cubical Agda snippet. Change other examples in this introduction to fit with the Cubical Agda code

*Proof goals in Cubical Agda* Cubical Agda implements the idea of higher inductive types (HIT) by treating them as freely generated cubical sets. When mapping into a HIT, one has to respect the coherences introduced by its path constructors, which means that one has to show that certain cells in the corresponding cubical set exist. For example, we have a very simple HIT $\mathsf{Int}$ which has three constructors: two term constructors $\mathsf{zero}$ and $\mathsf{one}$, and a 1-dimensional path constructor $\mathsf{seg}$ connecting both points. A proof goal one could face when mapping into $\mathsf{Int}$ is that one has to construct a square spanned by the $\mathsf{seg}$ path and the path which is constantly $\mathsf{one}$ which we can depict as follows:

$$\begin{tikzcd}
{} \arrow[r, "\mathsf{one}\langle s^1\rangle"] & {} \\
{} \arrow[u, "\mathsf{seg}"] \arrow[r, "\mathsf{seg}"'] & {} \arrow[u, "\mathsf{one}\langle s^1\rangle"']
\end{tikzcd}$$

We read the square as follows: Starting from the bottom left corner, which is the 0-dimensional cell $\mathsf{zero}$, we go with the $\mathsf{seg}$ path both to the top left and the bottom right corner, which is then the right boundary of $\mathsf{seg}$, which is $\mathsf{one}$. We then have as the top and right face the cell $\mathsf{one}$ as a degenerate 1-cell, which we denote by $\mathsf{one}\langle s^1\rangle$ . Coming from our topological intuition, we would expect that the boundary described by the above square has a filler as it just stretches the seg path into a certain square. Indeed, the cubical set described by the HIT Int contains a 2-dimensional cell with this boundary, let's this cell $\phi$. In Cubical Agda, $\phi$ is represented by the term $\lambda ij.\mathsf{seg}(i \vee j)$. By introducing two dimensional variables $i$ and $j$, this term represents a path in two dimensions, i.e., a square. We can apply one argument to the 1-dimensional path seg, for which we take $i \vee j$. This is a formula in the theory of bounded distributive lattices over the interval variables in the context.[1] Applying this formula to a path is commonly called an interval substitution.

Not all cells in Cubical Agda arise as interval substitutions. Going back to Kan [3], cubical sets need to satisfy one property to be a model for homotopy types: if we have a box, i.e., a well-defined boundary of a cube with only one side missing, then the cubical set must contain a cell which fills the missing side. This cell is then called the Kan composition of the box. For example, we can use Kan composition to derive that a path can be reversed. Treating the path seg as a logical manifestation of a function out of the interval, we would expect that we can come up with a function which travels along the points of seg in the other direction, i.e., that we have a path starting at one going back to zero. To show that such a path exists we use Kan composition on the following box:

$$\begin{tikzcd}
{} \arrow[r, "\mathsf{one}\langle s^1\rangle"] & {} \\
{} \arrow[u, "\mathsf{one}\langle s^1\rangle"] \arrow[r, dashed, "“\mathsf{seg}^{-1}”"'] & {} \arrow[u, "\mathsf{seg}"']
\end{tikzcd}$$

On the left and top side, the square is constantly one. On the right hand side, we have the seg path. Then the bottom left corner of the square is the 0-cell one and the bottom right corner is zero. Kan composition of this open box gives us a cell with one at its left and zero at its right boundary, which we could aptly call $\mathsf{seg}^{-1}$.

TODO Int is probably not the best motivating example. Can we find an example which feels more "logical" and is not directly related to homotopy theory?

*Poset maps as interval substitutions* The above examples are common proof obligations in Cubical Agda. Searching for interval substitutions automatically is difficult since it is not possible to construct a formula of the distributive lattice by looking at the boundary which needs to be filled (at least according to the author's knowledge). We will have to take a different perspective in the following, using a correspondence between $n$-tuples of formulas in the free bounded

[1] In Cubical Agda, one can use more general formulas, namely those of a De Morgan algebra. [5] has shown that a distributive lattices suffices. We will discuss the relationship between these two versions of cubical sets in more detail in **??**

distributive lattice over $m$ elements and monotone functions between the posets $\{0 < 1\}^m$ and $\{0 < 1\}^n$.

For example, the formula $i \vee j$ corresponds to the poset map $\{0 < 1\}^2 \to \{0 < 1\}^1$ which sends 00 to 0 and everything else to 1. In our setting, a cubical set is a presheaf on the full subcategory of posets of the form $\{0 < 1\}^n$, which means that this poset map will be sent contravariantly to a map from the 1-cells to the 2-cells of the cubical set. By filling in the definition of `seg` for the 1-cell, we can depict the action of the poset map on the 1-cell `seg` as follows:

$$\begin{tikzcd}
& 00 \arrow[dl, no head] \arrow[dr, no head] \arrow[rr, mapsto] & & \mathsf{zero} \arrow[dd, no head, "\mathsf{seg}"] \\
01 \arrow[dr, no head] \arrow[rrrd, mapsto] & & 10 \arrow[dl, no head] \arrow[rd, mapsto] & \\
& 11 \arrow[rr, mapsto] & & \mathsf{one}
\end{tikzcd}$$

This depiction suggests a geometrical intuition for how the boundary of $\phi$ comes about: the faces $00 - 01$ and $00 - 10$ are the `seg` cell as 00 is send to the left endpoint of `seg` and 01 and 10, respectively, to the right endpoint of `seg`. The faces $01 - 11$ and $10 - 11$ are both constantly `one` as both of its endpoints are send to the same 0-cell. This geometric intuition will allow us to find poset maps efficiently, allowing us to construct cells that would have been forbidding to find by brute-force.

Moreover, the poset map perspective allows us to gradually build open box to find cells as Kan compositions.

TODO MORNING derivation space perspective

*Contributions and Structure* The present paper is the first exploration of cubical sets from the point of view of automated reasoning. We explore a presentation of cubical sets in terms of presheaves on a certain category of posets and monotone maps and give a notion of normal forms for cells of cubical sets in section 1. We introduce the notion of potential poset maps as a useful data structure for efficient search of poset maps in section 2 and give an efficient algorithm for finding poset maps for a given boundary. We then extend our considerations to Kan cubical sets and establish why searching for cells in Kan cubical sets is in general undecidable. We present efficient heuristics for finding Kan compositions in many practical cases in section 3. The findings of this paper have been used to implement a tactic for Cubical Agda which we will discuss alongside a case study deriving the Eckmann-Hilton argument in Cubical Agda TODO HOPEFULLY! in section 4 before concluding with some remarks in section 5.

## 1 Cubical Sets and Contortions

In this section we will introduce the universe of discourse for our considerations. We will define cubical sets and introduce some running examples in subsection 1.1. For proof search we require a finitary representation of cubical sets, which mirror higher inductive types in Cubical Agda and will be introduced in subsection 1.2. We need a unique description for each cell of a cubical set and introduce certain normal forms for this purpose in subsection 1.3.

## 1.1 Cubical sets

The Dedekind cube category $\square_{\wedge\vee}$ is the full subcategory of the category of posets and monotone maps with objects $\mathbf{1}^n$ for $n \geq 0$, where $\mathbf{1} = \{0 < 1\}$. Therefore morphisms in $\square_{\wedge\vee}$ are of the form $\mathbf{1}^m \to \mathbf{1}^n$. The cardinality of $\mathrm{Hom}(\mathbf{1}^m, \mathbf{1})$ is the $m$-th Dedekind number, which explains the name of $\square_{\wedge\vee}$.

We denote elements of $\mathbf{1}^n$ with $(e_1 \dots e_n)$ where $e_i \in \{0, 1\}$ for $1 \leq i \leq n$ (we sometimes may omit the brackets). For an element $x = (e_1 \dots e_n)$ write $x_i := e_i$. The poset $\mathbf{1}^0$ has one element $()$. We will write $\mathbf{1}^n_{i=e} := \{x \in \mathbf{1}^n \mid x_i = e\}$, which is a subposet of $\mathbf{1}^n$.

**Definition 1.** *Cubical sets are objects of* $\mathbf{Set}^{\square_{\wedge\vee}{}^{op}}$*, i.e., presheaves over the Dedekind cube category.*

For a cubical set $X$ write $X_n := X(\mathbf{1}^n)$, which are called the $n$-cells of $X$. Given an element $p$ of $X_n$, call $\mathsf{dim}(p) := n$ the dimension of $p$.

Given an $n$-cell $p$, any poset map $\sigma : \mathbf{1}^m \to \mathbf{1}^n$ gives rise to an $m$-cell $X(\sigma)(p)$. We will write $p\langle\sigma\rangle := X(\sigma)(p)$ and call $p\langle\sigma\rangle$ an *$m$-contortion* of $p$. We call $p$ the *origin* of $p\langle\sigma\rangle$ and refer to $n$ as the *original dimension* of $p\langle\sigma\rangle$. When restricting a function $\sigma$ to $\mathbf{1}^m_{i=e}$, we may freely use $\mathbf{1}^m$ or $\{(x_1 ... x_{i-1} x_i ... x_n) \mid x \in \mathbf{1}^m\}$ as a domain of $\sigma|_{\mathbf{1}^n_{i=e}}$.

Certain poset maps of interest are:

$$s^i : \mathbf{1}^n \to \mathbf{1}^{n-1}, (e_1 \dots e_n) \mapsto (e_1 \dots e_{i-1} e_{i+1} \dots e_n) \text{ for } 1 \leq i \leq n$$
$$d^{(i,e)} : \mathbf{1}^{n-1} \to \mathbf{1}^n, (e_1 \dots e_{n-1}) \mapsto (e_1 \dots e_{i-1} e e_{i+1} \dots e_{n-1}) \text{ for } 1 \leq i \leq n, e \in \{0, 1\}$$

Given an $n$-cell $p$, $p\langle s^{n+1}\rangle$ is called a *degeneracy* of $p$. Given an $m > n$, we will write $s^m$ for the composite $s^n \circ s^{n+1} \circ \dots \circ s^m$. This gives the shorthand $p\langle s^m\rangle$ for $p$ considered as a degenerate $m$-cell.

Conversely, the cell $p\langle d^{(i,e)}\rangle$ is called the $(i, e)$-th face of $p$. We will call the list of all faces of an $n$-cell $p$ its boundary $\partial(p)$:

$$\partial(p) := [p\langle d^{(1,0)}\rangle, p\langle d^{(1,1)}\rangle, \dots, p\langle d^{(n,0)}\rangle, p\langle d^{(n,1)}\rangle]$$

Contravariance in our cubical set means that for an $n$-cell $p$ and poset maps $\sigma : \mathbf{1}^m \to \mathbf{1}^n$ and $\tau : \mathbf{1}^l \to \mathbf{1}^m$ we have $p\langle\sigma\rangle\langle\tau\rangle := (p\langle\sigma\rangle)\langle\tau\rangle = p\langle\sigma \circ \tau\rangle$. For instance, we can use this to compute the $(i, e)$-th face of a contorted cell with $p\langle\sigma\rangle\langle d^{(i,e)}\rangle = p\langle\sigma|_{\mathbf{1}^n_{i=e}}\rangle$.

The central idea of this paper is the following: given some boundary, we want to find a cell with this boundary. An $n$-dimensional boundary $T$ is a list of $2n$ cells $[p_{(1,0)}, p_{(1,1)}, ..., p_{(n,0)}, p_{(n,1)}]$. We will write $T_{(i=e)} := p_{(i,e)}$.

TODO Define valid boundary, i.e., state how cells in a boundary must be related

*Example 1.* The cubical set $\mathsf{Int}$ is generated by the following data: $\mathsf{Int}_0 = \{\mathsf{zero}, \mathsf{one}\}$ and $\mathsf{seg} \in \mathsf{Int}_1$ with $\mathsf{seg}\langle d^{(1,0)}\rangle = \mathsf{zero}$ and $\mathsf{seg}\langle d^{(1,1)}\rangle = \mathsf{one}$, which we could have written more concisely with $\partial(\mathsf{seg}) = [\mathsf{zero}, \mathsf{one}]$.

$\mathsf{Int}$ contains all necessary contortions, for instance $\mathsf{zero}\langle s^1\rangle$ is a well-defined 1-cell which is degenerately $\mathsf{zero}$. The identity map gives an $n$-contortion for every $n$-cell which is equal to the original cell, for instance $\mathsf{seg}\langle \substack{0\mapsto 0\\ 1\mapsto 1}\rangle$ is just the cell $\mathsf{seg}$.

*Example 2.* We define the cubical set $\mathsf{Sphere}$ with the following data: We have a single 0-cell, so $\mathsf{Sphere}_0 = \{\mathsf{a}\}$. There is one non-degenerate 2-cell $\alpha \in \mathsf{Sphere}_2$, which has a degenerate $\mathsf{a}$ for each of its faces, i.e., $\partial(\alpha) = [\mathsf{a}\langle s^1\rangle, \mathsf{a}\langle s^1\rangle, \mathsf{a}\langle s^1\rangle, \mathsf{a}\langle s^1\rangle]$.

*Example 3.* We define the cubical set $\mathsf{Triangle}$ as generated by the following data: The 0-cells are $\mathsf{Triangle}_0 = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$. The non-degenerate 1-cells are $\{\mathsf{p}, \mathsf{q}, \mathsf{r}\} \subseteq \mathsf{Triangle}_1$ with boundaries $\partial(\mathsf{p}) = [\mathsf{x}, \mathsf{y}]$ $\partial(\mathsf{q}) = [\mathsf{y}, \mathsf{z}]$ $\partial(\mathsf{r}) = [\mathsf{x}, \mathsf{z}]$. We have one non-degenerate 2-cell $\phi \in \mathsf{Triangle}_2$ with $\partial(\phi) = [\mathsf{p}, \mathsf{r}, \mathsf{x}\langle s^1\rangle, \mathsf{q}]$.

## 1.2 Higher inductive types

Cubical sets as introduced in the previous section are infinite objects: as soon as there is a single cell in a cubical set $X$, we have faces or degeneracies in all dimensions, and more generally all sorts of cells induced by the poset maps. In order to mechanically search for cells in a cubical set, we need a finitary representation of cubical sets. The examples above suggest that we only require a bit of generating data from which the other cells and maps can be inferred in a unique way.

**Definition 2.** *A context $\Gamma$ is a list of declarations $[p_1 : T_1, \dots, p_k : T_k]$ where each $T_j$ is a valid boundary using only contortions of cells $p_i$ with $i < j$. The cubical set $X$ generated by a context $\Gamma$ has cells $p_i$ with boundaries $\partial(p_i) = T_i$ for $1 \leq i \leq k$ and all necessary other cells. We call $|\Gamma| := k$ the length of the context.*

**Example 1** (continued)**.** *The following context generates* $\mathsf{Int}$*:*
$[\mathsf{zero} : [], \mathsf{one} : [], \mathsf{seg} : [\mathsf{zero}, \mathsf{one}]]$

**Example 2** (continued)**.** *The following context generates* $\mathsf{Sphere}$*:*
$[\mathsf{a} : [], \alpha : [\mathsf{a}\langle s^1\rangle, \mathsf{a}\langle s^1\rangle, \mathsf{a}\langle s^1\rangle, \mathsf{a}\langle s^1\rangle]]$

**Example 3** (continued)**.** *The following context generates* $\mathsf{Triangle}$*:*
$[\mathsf{x} : [], \mathsf{y} : [], \mathsf{z} : [], \mathsf{p} : [\mathsf{x}, \mathsf{y}], , \mathsf{q} : [\mathsf{y}, \mathsf{z}], \mathsf{r} : [\mathsf{x}, \mathsf{z}], \phi : [\mathsf{p}, \mathsf{r}, \mathsf{x}\langle s^1\rangle, \mathsf{q}]]$

In the following we will assume a valid context $\Gamma$ generating a cubical set $X$.

### 1.3 Normal forms of contortions

There are different ways of describing a cell in a cubical set, for instance , both $\mathsf{zero}\langle s^1\rangle$ and $\mathsf{seg}\langle\substack{0\mapsto 0\\1\mapsto 0}\rangle$ in Example 1 describe the same cell, namely the degenerate 1-cell which is constantly $\mathsf{zero}$. For our considerations we will want a unique representation for every cell to allow for literal comparisons between cells.

**Definition 3.** *A contortion $p\langle\sigma\rangle$ is in normal form if there is no $q\langle\tau\rangle$ such that $p\langle\sigma\rangle = q\langle\tau\rangle$ and $\mathsf{dim}(q) < \mathsf{dim}(p)$.*

We can efficiently compute the normal form of a contortion with Algorithm 1. Given a contortion $p\langle\sigma\rangle$, we check whether $\sigma$ sends all elements of $\mathbf{1}^m$ to a subposet $\mathbf{1}^n_{i=e}$ of $\mathbf{1}^n$, which means that the contortion in fact only talks about a face of $p$. If this is not the case we already have been given a normal form. Otherwise, we retrieve the $(i,e)$-th face of $p$, which we call $q$, and normalize $q$ with the composed contortion.

**Algorithm 1** Normalizing a contortion

```
Input: $p\langle\sigma\rangle$ where $p : T \in \Gamma$ is an $n$-cell and $\sigma : \mathbf{1}^m \to \mathbf{1}^n$
Output: Normal form of $p\langle\sigma\rangle$
procedure NORMALIZE($p\langle\sigma\rangle$)
    if $\exists 1 \leq i \leq n, e \in \{0,1\} : \mathsf{Img}(\sigma) \subseteq \mathbf{1}^n_{i=e}$ then
        $q\langle\tau\rangle \leftarrow p\langle d^{(i,e)}\rangle$
        return NORMALIZE($q\langle\tau \circ s^i \circ \sigma\rangle$)
    else
        return $p\langle\sigma\rangle$
```

The algorithm is correct by the following reasoning: For a poset map $\sigma : \mathbf{1}^m \to \mathbf{1}^n$ which satisfies for some $1 \leq i \leq n$, $e \in \{0,1\}$ that $\sigma(x)_i = e$ for all $x \in \mathbf{1}^m$, we naturally have $\sigma = d^{(i,e)} \circ s^i \circ \sigma$. But then we have by functoriality for $q\langle\tau\rangle := p\langle d^{(i,e)}\rangle$:

$$p\langle\sigma\rangle = p\langle d^{(i,e)} \circ s^i \circ \sigma\rangle = p\langle d^{(i,e)}\rangle\langle s^i \circ \sigma\rangle = q\langle\tau\rangle\langle s^i \circ \sigma\rangle = q\langle\tau \circ s^i \circ \sigma\rangle$$

If we compute $\mathsf{Img}(\sigma)$ as a multiset, checking whether $\mathsf{Img}(\sigma)$ only contains elements of $\mathbf{1}^n_{i=e}$ for some $1 \leq i \leq n$ and $e \in \{0,1\}$ takes $\mathcal{O}(2^m n)$ (go through all $n$ indices of all $2^m$ elements of $\mathsf{Img}(\mathbf{1}^m)$ and return the first index at which all elements are the same, if there is such an index). Since the dimension of the contortion under question is decreased by 1 in each normalization step, we need to run the algorithm at most $n$ times and therefore have a total complexity of $\mathcal{O}(2^m n^2)$ for contortion normalization.

**Example 3** (continued)**.** *The contortion $\phi\langle\substack{00\mapsto 00\\01\mapsto 00\\10\mapsto 01\\11\mapsto 01}\rangle$ has normal form $p\langle\substack{00\mapsto 0\\01\mapsto 0\\10\mapsto 1\\11\mapsto 1}\rangle$.*

It is worth pointing out that contortions do not necessarily have unique boundaries.

**Example 2** (continued)**.** *Consider* $T = [\mathsf{a}\langle s^1\rangle, \mathsf{a}\langle s^1\rangle, \mathsf{a}\langle s^1\rangle, \mathsf{p}\langle\begin{smallmatrix}0 \mapsto 00 \\ 1 \mapsto 11\end{smallmatrix}\rangle]$. *Both* $\mathsf{p}\langle\begin{smallmatrix}00 \mapsto 00 \\ 01 \mapsto 00 \\ 10 \mapsto 01 \\ 11 \mapsto 11\end{smallmatrix}\rangle$ *and* $\mathsf{p}\langle\begin{smallmatrix}00 \mapsto 00 \\ 01 \mapsto 00 \\ 10 \mapsto 10 \\ 11 \mapsto 11\end{smallmatrix}\rangle$ *are cells with boundary* $T$*.*

## 2 Searching for Contortions

In this section, we will study the following search problem: given a cubical set and a certain boundary, we want to decide whether there is a cell in this cubical set with that boundary. We will set out this problem in more detail in subsection 2.1. For our algorithms we introduce the notion of potential poset maps which are introduced in subsection 2.2 and use them to devise an efficient algorithm for computing contortions in subsection 2.3.

### 2.1 The problem CubicalCell

Using the finitary representation of cubical sets introduced in subsection 1.2, we can phrase the problem of finding a cell with a given boundary as follows:

**Definition 4.** *Given a context* $\Gamma$ *and a boundary* $T$*, the problem* CubicalCell*(*$\Gamma$*,*$T$*) is to find a cell* $p\langle\sigma\rangle$ *in the cubical set generated by* $\Gamma$ *such that* $\partial(p\langle\sigma\rangle) = T$*.*

The problem CubicalCell is clearly decidable since there are only finitely many cells in a context which all have finitely many contortions in a given dimension. Hence, given an $m$-dimensional boundary $T$, we can go through all generators $p$ of the context and check whether a poset map $\mathbf{1}^m \to \mathbf{1}^{\mathsf{dim}(p)}$ gives rise to $T$. However, the number of such poset maps grows super-exponentially: suppose we want to find an $m$-dimensional contortion of a 1-cell, i.e., we are looking for a poset map $\mathbf{1}^m \to \mathbf{1}$. The number of such monotone maps is the $m$-th Dedekind number, which we will denote $D_m$. For $m = 6$, we already have $D_6 = 7828354$ poset maps, and for 9 the exact number of possible poset maps is unknown [2]. The number of such poset maps grows furthermore exponentially with the dimension of original cell, i.e., we have $D_m^n$ many monotone maps $\mathbf{1}^m \to \mathbf{1}^n$. Therefore searching for contortions by brute-force gets quickly infeasible and we require something more clever to explore the search space.

### 2.2 Potential poset maps

For the following algorithms we introduce the notion of a potential poset map. In essence, a potential poset map keeps track of all the values that elements of the domain of the poset map could take, while making sure that there are no superfluous values that cannot be part of any poset map as they would otherwise violate monotonicity. Thereby we can represent all maps $\mathbf{1}^m \to \mathbf{1}^n$ at once with little data.

[2] `http://oeis.org/A00372`

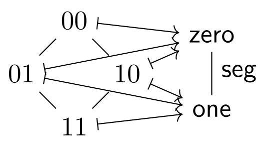
**Definition 5.** *A potential poset map (ppm) is a map $\varSigma : \mathbf{1}^m \to \mathcal{P}(\mathbf{1}^n)$ such that $\forall x \leq y$:*

- $\forall u \in \varSigma(y) : \exists v \in \varSigma(x) : v \leq u$.
- $\forall v \in \varSigma(x) : \exists u \in \varSigma(y) : v \leq u$.

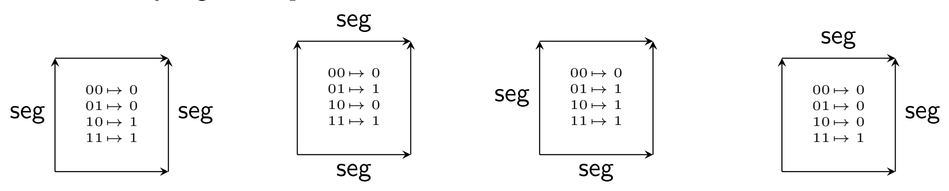
The two conditions above mean that any possible value in a ppm will give rise to at least one poset map. TODO EXPLAIN WHY TODO TOTAL PPM TODO NON-EMPTY PPM

We will extend the notion of a contortion as a poset map applied to a cell to also incorporate potential poset maps, leading to the notion of potential contortions.

**Example 1** (continued). *Let us consider the potential contortion $\mathsf{seg}\left\langle\begin{smallmatrix}00\mapsto\{0\}\\01\mapsto\{0,1\}\\10\mapsto\{0,1\}\\11\mapsto\{1\}\end{smallmatrix}\right\rangle$. To understand its action in the cubical set* Int *we again fill in the definition of* seg *into the poset map as follows:*



*The ppm can be unfolded into four poset maps as we can choose the values of* 01 *and* 10 *independently. The squares resulting from using these poset maps for contortions of* seg *are depicted below:*



*All these contortions have in common that the bottom left corner is* zero *and the top right corner is* one*, which we could already read off from the potential contortion.*

Thereby ppms are a useful notion to capture a number of poset maps with little data. For example, representing all $D_n$ poset maps $\mathbf{1}^n \to \mathbf{1}$ requires only $2^n$ entries of $\{0, 1\}$. Our goal in the following will be to restrict a ppm until only few poset maps remain. We can unfold a ppm $\varSigma$ into the set of poset maps that it is representing with the procedure UNFOLDPPM($\varSigma$) given in Algorithm 6 in the appendix. If we want to narrow down a ppm to take values $vs \subseteq \varSigma(x)$ for some $x$, we can update it in $\mathcal{O}(2^m 2^n)$ to still satisfy the conditions in Definition 5 with the algorithm Algorithm 5.

## 2.3 Contortion solver

Using the notion of potential poset maps, we develop in this section an efficient algorithm for deciding whether a cell $p$ can be contorted into a given boundary $T$. The central idea is that we start with a total ppm and gradually narrow it down by looking at the faces of $T$ in descending original dimension. The algorithm is given in Algorithm 2.

**Algorithm 2** Contortion solver

```
Input: $\Gamma$ context, $p \in \Gamma$, $T$ goal boundary with $n := \mathsf{dim}(p), m := \mathsf{dim}(T)$
Output: $\sigma : \mathbf{1}^m \to \mathbf{1}^n$ s.t. $\partial(p\langle\sigma\rangle) = T$ if there is such a $\sigma$, Unsolvable otherwise
1  procedure ContortionSolve($p$)
2      $\Sigma \leftarrow \{x \mapsto \mathbf{1}^n \mid x \in \mathbf{1}^m\}$
3      for $q\langle\tau\rangle := T_{(i=e)}$ with $i \leftarrow \{1, ..., m\}, e \leftarrow \{0, 1\}$ such that $\mathsf{dim}(q)$ desc. do
4          if $p = q$ then
5              $\Theta \leftarrow \{x \mapsto \tau(x) \mid x \in \mathbf{1}^m_{i=e}\}$
6          else
7              $\Theta \leftarrow \{x \mapsto \emptyset \mid x \in \mathbf{1}^m_{i=e}\}$
8              for $\sigma \leftarrow$ UnfoldPPM($\Sigma|_{\mathbf{1}^m_{i=e}}$) do
9                  if $T_{i=e} =$ Normalize($p\langle\sigma\rangle$) then
10                     for $x \in \mathbf{1}^m_{i=e}$ do
11                         $\Theta(x) \leftarrow \Theta(x) \cup \{\sigma(x)\}$
12         for $x \in \mathbf{1}^m_{i=e}$ do
13             UpdatePPM($\Sigma, x, \Theta(x)$)
14     if $\exists \sigma \in$ UnfoldPPM($\Sigma$) : $\partial(p\langle\sigma\rangle) = T$ then
15         return $\sigma$
16     else
17         return Unsolvable
```

We first generate the most general ppm $\Sigma$ in line 2. We then go through each face $q\langle\tau\rangle$ of the goal boundary $T$ and use it to restrict $\Sigma$. Crucially, we order the faces by their original dimensions as higher-dimensional cells in the boundary will restrict $\Sigma$ more than lower-dimensional faces.

Given a face $q\langle\tau\rangle$, we proceed as follows: in case the face has the same original cell as the cell we are contorting, we can give unique values for all element of the subposet corresponding to the current face as done in line 5. Otherwise, we generate all possible faces that could be generated from $\Sigma$ at the subposet and only keep values that contributed to some valid face. We finally propagate our findings through the whole poset map $\Sigma$ in line 13.

After restricting $\Sigma$ according to all faces we generate a valid solution and return it in line 14.

*Correctness* The algorithm is cleary correct since we only return a poset map $\sigma$ if it is indeed contorting $p$ to the correct boundary. Note that we have to do this step as there might be multiple solutions. Since our ppms captures all solutions at once, we need to make sure that we do not return assignments of different solutions. TODO MAKE CLEARER, PERHAPS WITH EXAMPLE

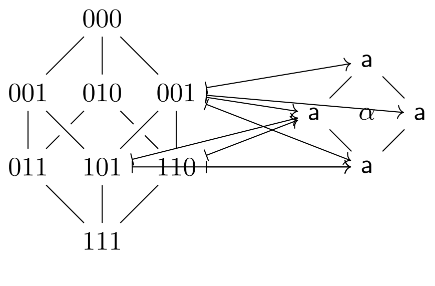
*Completeness* Supposing that a cell exists with a boundary, we want to show that we find it. This follows immediately since any value in the ppm contributing to some solution has been kept. TODO ELABORATE

*Runtime* The most expensive step in the algorithm is unfolding all poset maps of the subposet in line 8. In the worst case, we have $D^n_{m-1}$ many poset maps that we need to check, and as we are doing this for all $2m$ faces of $T$ we have a factor of $2mD^n_{m-1}$ in the asymptotic runtime. After performing all restrictions, $\Sigma$ is usually unique or contains very few poset maps, which is why the unfolding step in line 14 is not expensive.

The algorithm constitutes an improvement over naïvely checking all $D^n_m$ poset maps already in the worst case as the Dedekind numbers grow quicker than any operation in our algorithm, and hence our algorithm has a runtime which is one Dedekind number quicker than the naïve algorithm. In many practical cases, the original face of the contortion appears in its boundary, which significantly allows to reduce the search space before any ppm is unfolded. This means that we can compute many contortions that would have been impossible to find by brute-force.

**Example 2** (continued)**.** *We want to find $\sigma : \mathbf{1}^3 \to \mathbf{1}^2$ such that $\alpha\langle\sigma\rangle$ has boundary $T = [\alpha\langle\substack{00\mapsto 00\\01\mapsto 01\\10\mapsto 01\\11\mapsto 11}\rangle, \mathsf{a}\langle s^2\rangle, \mathsf{a}\langle s^2\rangle, \mathsf{a}\langle s^2\rangle, \mathsf{a}\langle s^2\rangle, \mathsf{a}\langle s^2\rangle]$.*

*In the beginning, the total ppm $\Sigma$ is initialized with $\{x \mapsto \mathbf{1}^2 \mid x \in \mathbf{1}^3\}$. We then go through all faces of the goal boundary and use them to restrict $\Sigma$. The face of the boundary with the largest original dimension is a contortion of $\alpha$. Since $\alpha$ is also the cell that we are contorting, the subposet $\mathbf{1}^3_{1=0}$ of the domain of $\Sigma$ is mapped in a unique way to the elements of $\mathbf{1}^2$. The monotonicity restrictions on ppms further restrict $\Sigma$: since 011 is mapped to the bottom element 00, also 111 will need to be mapped to 00. The elements 101 and 110 lie below elements which are mapped to 01, which mean that there are only two possible values $\{01, 11\}$ for each of these elements. Only the element 001 has the full poset $\mathbf{1}^2$ as potential values. In summary, we have the following potential contortion after the first restriction of $\Sigma$:*



*We then check for each of the faces which are degenerate* a *whether $\Sigma$ gives rise to such a face. Crucially, this will be done quickly as there are only 10 poset maps in the ppm $\Sigma$ remaining after the restricting $\Sigma$ according to the first face of $T$.*

## 3 Composition solver

Cubical type theory is based on a certain class of cubical sets, namely those that can model spaces. Kan [3] introduced a single condition for this purpose, which essentially states that any open cube can be filled in a coherent way. By turning cubical sets into a logic, cubical type theory turns the Kan condition into a logical rule.

TODO STRUCTURE OF SECTION

### 3.1 Kan cubical sets

In essence, Kan composition establishes that if one constructs a valid cell with only one face missing, then this face exists as well. For example, we can invert the path seg in Example 1 by constructing an open square as follows:

$$\begin{tikzcd}
{} \arrow[r, dotted, "\mathsf{p}^{-1}"] & {} \\
{} \arrow[u, "\mathsf{seg}"] \arrow[r, "\mathsf{zero}\langle s^1\rangle"'] & {} \arrow[u, "\mathsf{zero}\langle s^1\rangle"']
\end{tikzcd}$$

TODO give p q composition

Let us make this more precise. We have the following notion of open cube, which we call *box*: an $(n+1)$-dimensional box $U$ is a collection of $2 \cdot n + 1$ cells, which we will write $[p_1, q_2, ..., p_n, q_n]r$. We call $r$ the back of $U$, $p_i$ its $(i, 0)$-th side and $q_i$ its $(i, 1)$-th side.

**Definition 6.** *A Kan cubical set $X$ is cubical set which furthermore satisfies the following condition: given a box $U := [p_1, q_2, ..., p_n, q_n]r$, $X$ has the following cells:*

- $\mathsf{Comp}(U)$ *with boundary* $[p_1\langle d^{(n,1)}\rangle, q_1\langle d^{(n,1)}\rangle, ..., p_n\langle d^{(n,1)}\rangle, q_n\langle d^{(n,1)}\rangle]$, *called the composition of $U$.*
- $\mathsf{Fill}(U)$ *with boundary* $[p_1, q_1, ..., p_n, q_n, r, \mathsf{Comp}(U)]$, *called the filler of $U$.*

Note that cubical type theory as presented in [2] uses a more general Kan condition. We will not need this principle in this generality and will use the above notion for the sake of concreteness. TODO DISCUSS RELATION IN A BIT MORE DETAIL?

## 3.2 Undecidability of finding cells in a Kan cubical set

Since the cells of a box can itself be the result of composing or filling a box, the search space for Kan cubical cells grows infinitely large. Naturally, deciding whether there is a cell in a Kan cubical set with a specific boundary is undecidable, which we will demonstrate in this section with a reduction to string rewrite systems.

The problem of finding cells in a Kan cubical set is the following:

**Definition 7.** *Given a context $\varGamma$ and a boundary $T$, the problem* KanCubicalCell*($\varGamma$,$T$) is to find a cell $p$ in the Kan cubical set generated by $\varGamma$ such that $\partial(p) = T$.*

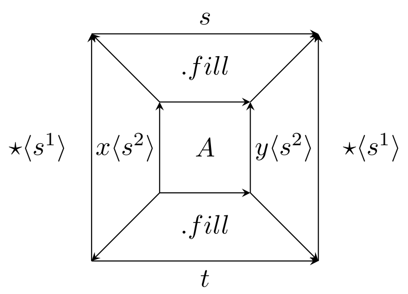
We can encode very many structures and problems with Kan cubical sets: groups, ... TODO GIVE OVERVIEW.

**Proposition 1.** KanCubicalCell *is undecidable.*

*Proof.* We show how the word problem for Thue systems Word reduces to KanCubicalCell. Given a Thue system $(\varSigma, R)$ consisting of an alphabet $\varSigma$ and a symmetric relation over words of $\varSigma$, $R \subseteq \varSigma^* \times \varSigma^*$. The rewriting relation is defined as follows: we have $s \to_R t$ if there are $x, y, u, v \in \varSigma^*$ such that $s = xuy$, $t = xvy$ and $(u, v) \in R$.

We construct the following cubical set. We have a single 0-cell $\star$, 1-cells $\varSigma$ and for any $(u, v) \in R$ a 2-cell $r_{u,v}$ with $\partial(r_{u,v}) = [u, v, \star\langle s^1\rangle, \star\langle s^1\rangle]$ where we use path composition TODO WHERE INTROCED AND SHOWN ASSOC for concatenating letters and $\star\langle s^1\rangle$ as the empty word.

The reflexive transitive closure of the rewriting relation, denoted $s \to_R^* t$, is mirrored in the Kan cubical set as follows: for any $s$, we have a proof of reflexivity $s\langle s^2\rangle$. Suppose we have a cell $A$ witnessing $u \to_R^* v$, then the following cell witnesses that $s = xuy \to_R^* xvy = t$:



TODO SPELL OUT CONNECTION IN MORE DETAIL

Therefore any rewriting sequence between two words $u$ and $v$ corresponds precisely to a 2-dimensional cell with boundary $[u, v, \star\langle s^1\rangle, \star\langle s^1\rangle]$. If it was decidable whether such a cell exists, we would have a decision procedure for Word. □

### 3.3 Finding Kan compositions using constraint satisfaction

In order to find Kan compositions, we can use the language of constraint satisfaction programming.

Given a HIT $X$

To construct an $n$-dimensional box whose composition has a given boundary $T$, we have the following CSP:

- Variables $X_{\text{Back}}$ and $X_{(i,e)}$ for $1 \leq i \leq n$, $e \in \{\mathsf{0},\mathsf{1}\}$ stand for the back and side faces of the cube.
- $D_{\text{Back}} = \{p\langle TODOtotalsigmafromn \to \mathsf{dim}(p)\rangle \mid p \in X\}$ and $D_{(i,e)} = \{p\langle TODOtotalsigmafromn \to \mathsf{dim}(p)\rangle \mid p \in X \text{ such that } p\langle d^{(1,n)}\rangle = T_{(i=e)}\}$
- Constraints are that all boundaries of the sides and back need to match.
  Back constraints: $X_{\text{Back}}\langle d^{(i,e)}\rangle = X_{(i,e)}\langle d^{(n,0)}\rangle$
  Side constraints: $X_{(i,e)}\langle d^{(i,e')}\rangle = X_{(i',e')}\langle d^{(i,e)}\rangle$ for $1 \leq i < j \leq n$ and $e, e' \in \{\mathsf{0},\mathsf{1}\}$.

When this does not work, we can create a larger CSP where one of the faces is itself the result of a Kan composition. We cannot pass over results from the 1-dimensional case. Also not clear in which direction we need to extend our cube. For the 1-dimensional cube we can however safely only extend in one direction.

TODO EXTEND TO NESTED CUBES. EXAMPLE WITH PATH COMPOSITION

## 4 Case study: TODO

We have reasoned about semantic objects in this paper: since syntax and semantics so closely linked, this gives us a syntactic solver as well. We just have to translate to the term language of Cubical Agda.

The above algorithms have been implemented in `https://github.com/maxdore/cubetac/`. In the following we will discuss how our approach relates to Cubical Agda and a case study using this tactic.

### 4.1 Connecting with Cubical Agda

The morphisms in $\square_{\wedge\vee}$ have a succinct description as tuples of formulas of a free distributive lattices, these formulas are also used in Cubical Agda. In order to use our algorithms in a tactic for Cubical Agda, we need to translate back and forth efficiently between both characterizations of $\square_{\wedge\vee}$.

To generate a lattice formula from a poset map we can use Algorithm 3.

**Algorithm 3** Poset map to lattice formula

**Input:** $\sigma : \mathbf{1}^m \to \mathbf{1}^n$

**Output:** $\phi$ $n$-tuple of elements of free distributive lattice over $m$ variables

```
procedure SUBST2TELE($\sigma$)
    for $i \leftarrow 1$ to $n$ do
        $\phi_i \leftarrow \{x \in \mathbf{1}^m \mid \sigma(x)_i = 1\}$        ▷ $\mathcal{O}(2^m)$ many elements in $\mathbf{1}^m$
    return $(\phi_1, ..., \phi_n)$
```

Given $\sigma : \mathbf{1}^m \to \mathbf{1}^n$, we compute an $n$-tuple of elements of the free distributive lattice over $m$ elements as follows: The $i$-th entry is $\{x \in \mathbf{1}^m \mid s(x)_i = 1\}$. An element $x$ can be seen as a clause if we regard it as indicator of which elements of the lattice are used, e.g., $(1, 0, 1)$ represents the clause $x \vee z$ if the three elements of the lattice are $x$, $y$ and $z$.

For the converse we can use Algorithm 4

**Algorithm 4** Lattice formula to poset map

**Input:** $\phi$ $n$-tuple of elements of free distributive lattice over $m$ variables

**Output:** $\sigma : \mathbf{1}^m \to \mathbf{1}^n$

```
procedure TELE2SUBST($p$)
    for $x \leftarrow \mathbf{1}^m$ do
        for $i \leftarrow 1$ to $n$ do
            $\sigma(x)_i \leftarrow \phi_i @ x$
    return $\sigma$
```

From an $n$-tuple of formulas over $\phi$ we can read off a poset map $\sigma : \mathbf{1}^m \to \mathbf{1}^n$ as follows: Given an element $x \in \mathbf{1}^m$, $\sigma(x) = (e_1, ..., e_n)$ where $e_i = \phi_i @ x$. Here $\phi_i$ is the $i$-th element of the tuple $\phi$ and @ denotes evaluation of formulas: The indices of $x$ which are set to 1 mean that the corresponding variable of $\phi$ is “true”, then we check whether the formula evaluates in total to true by interpreting join and meet as logical operations.

TODO The above presentation is not very clean and should be streamlined.

Therefore going back and forth between formulas and poset map takes $\mathcal{O}(2^m n)$ many steps. This is linear in the number of elements of the data structures we are considering, so does not pose an issue for performance of our tactic.

TODO Show correctness of both algorithms and that they are mutually inverse.

Cubical Agda uses PathP types to represent boundaries of a cube. The PathP type has three constructors, where the first is again a PathP type, allowing for nesting paths and thereby creating higher cubes. If the second and third parameter are $n$-dimensional paths, then the first parameter might introduce $n$ interval variables.

**Example 2** (continued)**.** *The proof goal already introduced above:*

$$[\mathsf{p}\langle\begin{smallmatrix}00\mapsto 00\\01\mapsto 01\\10\mapsto 01\\11\mapsto 11\end{smallmatrix}\rangle, \mathsf{a}\langle s^2\rangle, \mathsf{a}\langle s^2\rangle, \mathsf{a}\langle s^2\rangle, \mathsf{a}\langle s^2\rangle, \mathsf{a}\langle s^2\rangle]$$

*has the following corresponding PathP type in Cubical Agda.*

$$\begin{aligned}&\mathsf{PathP}(\lambda i \to \mathsf{PathP}(\lambda j \to \mathsf{Path}\ (\mathsf{p}\ (i \wedge j)\ (i \vee j))\ \mathsf{a})\\&\qquad\qquad(\lambda j \to \mathsf{a})(\lambda j \to \mathsf{a}))(\lambda ij \to \mathsf{a})(\lambda ij \to \mathsf{a})\end{aligned}$$

TODO Translating between our notions and PathP types is straightforward, we should still explain it here

## 5 Conclusions

TODO

Check whether decidable subproblems: Only finitely many cells/ up to dim n/ ...?

## References

1. Steve Awodey and Michael A Warren. Homotopy theoretic models of identity types. In *Mathematical proceedings of the cambridge philosophical society*, volume 146, pages 45–55. Cambridge University Press, 2009.
2. Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *LIPIcs*, pages 5:1–5:34. Schloss Dagstuhl, 2018.
3. Daniel M. Kan. Abstract homotopy. i. *Proceedings of the National Academy of Sciences of the United States of America*, 41(12):1092–1096, 1955.
4. Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. Univalence in simplicial sets. *arXiv:1203.2553*, 2012.
5. Ian Orton and Andrew M. Pitts. Axioms for modelling cubical type theory in a topos. *arXiv:1712.04864*, 2017.
6. Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories. *Higher Structures*, 1(1):116–193, 05 2017.
7. The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

8. Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical Agda: a dependently typed programming language with univalence and higher inductive types. *Proceedings of the ACM on Programming Languages*, 3(ICFP):1–29, 2019.

**Algorithm 5** Update potential poset map

```
Input: $\Sigma : \mathbf{1}^m \to \mathcal{P}(\mathbf{1}^n)$ ppm, $x \in \mathbf{1}^m$, $vs \subseteq \Sigma(x)$
Output: Updated ppm $\Sigma' : \mathbf{1}^m \to \mathbf{1}^n$ with $\Sigma(x) = vs$.
  procedure UpdatePPM($\Sigma, x, vs$))
      for $y \leftarrow \mathbf{1}^m$ do
          if $x = y$ then
              $\Sigma'(x) \leftarrow vs$
          else if $y \leq x$ then
              $\Sigma'(y) \leftarrow \{u \mid u \in \Sigma(y)$ such that $\exists v \in vs : u \leq v\}$
          else if $x \leq y$ then
              $\Sigma'(y) \leftarrow \{u \mid u \in \Sigma(y)$ such that $\exists v \in vs : v \leq u\}$
      return $\Sigma'$
```

TODO EXPLAIN Worst case run-time of $\mathcal{O}(2^m 2^n)$

**Algorithm 6** Potential substitution to substitutions

```
Input: $\Sigma : \mathbf{1}^m \to \mathcal{P}(\mathbf{1}^n)$ potential poset map
Output: $\{\sigma : \mathbf{1}^m \to \mathbf{1}^n\}$, all poset maps represented by $\Sigma$
  procedure UnfoldPPM($\Sigma$)
      return $\{x \mapsto v, \text{UnfoldPPM}(\text{UpdatePPM}(\Sigma, x, v) - x) \mid v \in \Sigma(x)\}$
```

TODO EXPLAIN Worst case run-time of $\mathcal{O}(D^n_m)$