Proof Search in Kan Cubical Sets

Maximilian Doré

No Institute Given

Abstract.

Keywords: One · Two

The realization that identity in constructive type theory is best understood in analogy to homotopies as studied in algebraic topology [1] has sparked the research field of homotopy type theory [7]. Recent research has been focused on devising a computationally well-behaved type theory making this analogy precise. One such type theory is cubical type theory, which takes one combinatorial model of homotopy types, cubical sets, as inspiration [2]. One major contribution of cubical type theory is that it gives computational meaning to the univalence axiom [4]. In doing so, it is the first type theory with explicit path arithmetic allowing to reason about homotopy types in a synthetic way. Cubical type theory has been implemented as an extension to the interactive theorem prover Agda [8]. Thereby it is a prime candidate for a constructive logic for higher identities. This paper will treat it as such and investigate how we can automate its reasoning principles.

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 Int which has three constructors: two term constructors zero and one, and a 1-dimensional path constructor seg connecting both points. A proof goal one could face when mapping into Int is that one has to construct a square spanned by the seg path and the path which is constantly one which we can depict as follows:

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

TODO Give concrete Agda code leading to this proof goal, for instance when constructing a function $\mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$

We read the square as follows: Starting from the bottom left corner, which is the 0-dimensional cell zero, we go with the seg path both to the top left and the bottom right corner, which is then the right boundary of seg, which is one. We then have as the top and right face the cell one considered 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. In Cubical Agda, this cell 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:

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

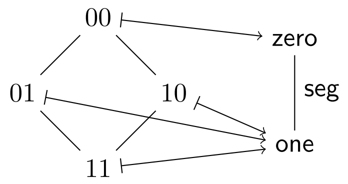
On the left and top side, the square is constantly one. On the right hand side, we have the seg path. Then the bottom left corner of the square is the 0-cell one and the bottom right corner is zero. Kan composition of this open box gives us a cell with one at its left and zero at its right boundary,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

TODO Int is probably not the best motivating example. Can we find an example which feels very “logicy” and is not directly related to homotopy theory?

[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 subsection 4.3

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 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:



This depiction suggests a geometrical intuition for how the boundary of $\lambda ij.\mathsf{seg}(i \vee j)$ 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, thereby solving the problem of finding interval substitutions in TODO RUNTIME. This is a significant improvement over naïve search for interval substitutions, as the number of n-dimensional interval substitutions of a 1-dimensional path is the n-th Dedekind number.

Moreover, the poset map perspective allows us to gradually build open box to find cells as Kan compositions. The problem of whether a cell can be filled by Kan composition is in general undecidable as we will show in subsection 3.2. We will explore different semi-decidable procedures and heuristics to still come up with Kan compositions in many practical examples.

Contributions and Structure The present paper is the first exploration of cubical sets from the point of view of automated reasoning. We give 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 algorithm for finding poset maps for a given boundary with a run time of TODO RUNTIME. 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 some case studies 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 we will call higher inductive types and introduce 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 $\mathbf{1} = \{\mathsf{0} < \mathsf{1}\}$. 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 $e_i \in \{\mathsf{0}, \mathsf{1}\}$ 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 $\mathbf{1}^n$.

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, we 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.

Certain poset maps of interest are:

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

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 $s^n \circ s^{n+1} \circ \ldots \circ s^m$. 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 $\partial(p)$:

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

Contravariance means that for an n-cell p, $\sigma : \mathbf{1}^m \to \mathbf{1}^n$ and $\tau : \mathbf{1}^l \to \mathbf{1}^m$ we have

$$p\langle\sigma\rangle\langle\tau\rangle = (p\langle\sigma\rangle)\langle\tau\rangle = p\langle\sigma \circ \tau\rangle$$

In this paper we will explore how one can come up with a cell given some boundary. For this we will introduce some notation. 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 $T_{(i=e)} := p_{(i,e)}$.

Example 1. The cubical set 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 $\partial(\mathsf{seg}) = [\mathsf{zero}, \mathsf{one}]$.

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

Example 2. We define the cubical set 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 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 1. *A context Γ is a list of declarations $[p_1 : T_1, \ldots, p_k : T_k]$. The cubical set X generated by a context Γ 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.*

TODO Define validity of boundary and context descriptions and give algorithm checking validity.

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

Example 2 2 (continued) *The following context generates* 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 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 Γ generating a cubical set X.

1.3 Normal forms of contortions

There are different ways of describing a cell in a cubical set. For our considerations we will want a unique representation for every cell to allow for literal comparisons between cells. For instance in Example 1, both $\mathsf{zero}\langle s^1\rangle$ and $\mathsf{seg}\langle\begin{smallmatrix}0\mapsto 0\\1\mapsto 0\end{smallmatrix}\rangle$ describe the same cell, namely the degenerate 1-cell which is constantly zero. Since the former has a lower dimension, we will regard it as the normal form of this cell. In general:

Definition 2. *A contortion $p\langle\sigma\rangle$ is in normal form if there is no face q of p and map τ such that $p\langle\sigma\rangle = q\langle\tau\rangle$.*

We can efficiently compute the normal form of a contortion with Algorithm 1. Given a contortion $p\langle\sigma\rangle$, we check whether σ 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 4 (continued) *The contortion $\phi\langle\begin{smallmatrix}00\mapsto 00\\01\mapsto 00\\10\mapsto 01\\11\mapsto 01\end{smallmatrix}\rangle$ has normal form $p\langle\begin{smallmatrix}00\mapsto 0\\01\mapsto 0\\10\mapsto 1\\11\mapsto 1\end{smallmatrix}\rangle$.*

More results: for normal forms we do not have to worry about poset maps

Proposition 1. *Uniqueness of contortions: if $\partial(p\langle\sigma\rangle) = \partial(p\langle\tau\rangle)$ TODO ABOVE WITHOUT boundary, then $\sigma = \tau$.*

Proof.

> *TODO As soon as poset map differs in one point, then completely different boundary??? Proof by contradiction? Or counting argument?*

This lemma will be relevant when searching for contortions:

Lemma 1. *If $p\langle\sigma\rangle$ is in normal form for $\mathsf{dim}(p) = n$ and $\sigma : \mathbf{1}^m \to \mathbf{1}^n$ with $m > n$, then there is a face $p\langle\tau\rangle$ of $p\langle\sigma\rangle$ (with $\tau : \mathbf{1}^{m-1} \to \mathbf{1}^n$).*

Proposition 2.

> *TODO For $m = n$ this does not hold: fgfg square filled with triangle. For $m > n$ I wasn't able to construct a cube containing f and g but not the triangle. Check with brute force this really does not hold?*

This means that if we are given a boundary of a cell, we do not have to check all faces: todo which not.

Furthermore, we can use this cell to reduce the search space.

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 introduced above, we can phrase the problem of finding a cell with a given boundary as follows:

Definition 3. *Given a context Γ and a boundary T, the problem* CUBICALCELL*(Γ,T) is to find a contortion $p\langle\sigma\rangle$ such that $\partial(p\langle\sigma\rangle) = T$.*

The problem CUBICALCELL is clearly decidable since there are only finitely many contortions of a cell in a specific dimension. However, the search space 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 the path that is contorted, i.e., we have D_m^n many monotone maps $\mathbf{1}^m \to \mathbf{1}^n$. Therefore bruteforcing the search for contortions 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 potential values that elements of the domain of the poset map might have, 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.

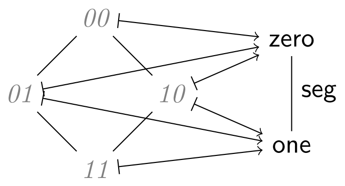
Definition 4. *A potential poset map (ppm) is a map $\Sigma : \mathbf{1}^m \to \mathcal{P}(\mathbf{1}^n)$ such that $\forall x \leq y$:*

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

The two conditions above mean that any possible value in a ppm will give rise to at least one poset map. We will represent ppms simply with their graphs. We also list the vertices such that if $x \leq y$ then x is earlier than y in the list. This allows us to traverse through the graph and know that if an element x is at a certain position in the list, all elements below x will be seen at a certain point.

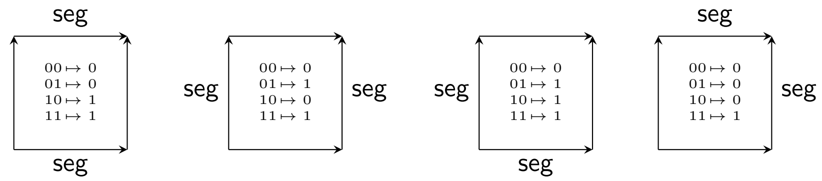
We will extend the notion of a contortion as a poset map applied to a cell to also incorporate potential poset maps, leading to the notion of potential contortions.

Example 1 5 (continued) *We have the following 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:*



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

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 very 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 into the set of poset maps that it is representing with Algorithm 6 given in the appendix. We can restrict the values of a ppm with Algorithm 5.

A non-empty ppm is one which has a non-empty set of possible values for any element, which can be checked in 2^m steps. Given a poset $xs \subseteq \mathbf{1}^m$, we can restrict a ppm $\Sigma : \mathbf{1}^m \to \mathcal{P}(\mathbf{1}^n)$ to a new ppm $\Sigma|_{xs} : xs \to \mathcal{P}(\mathbf{1}^n)$ just like any function.

2.3 Contortion solver

Using the notion of potential poset maps, we give the following algorithm to generate a contortion for a given boundary.

Algorithm 2 Contortion solver

```
Input: $\Gamma$ context, $p \in \Gamma$, $T$ goal boundary with $n := \mathsf{dim}(p), m := \mathsf{dim}(T)$
Output: $\sigma$ s.t. $\partial(p\langle\sigma\rangle) = T$ if there is such a $\sigma$, Unsolvable otherwise
  procedure ContortionSolve($p$)
    $\Sigma \leftarrow \{x \mapsto \mathbf{1}^n \mid x \in \mathbf{1}^m\}$
    for $q\langle\tau\rangle := T_{i=e}$ with $i \leftarrow 1 \leq i \leq m, e \leftarrow \{0,1\}, \mathsf{dim}(q)$ desc. do
      if $p = q$ then
        $\Theta \leftarrow \{x \mapsto \tau(x) \mid x \in \mathbf{1}^m_{i=e}\}$
      else
        $\Theta \leftarrow \{x \mapsto \emptyset \mid x \in \mathbf{1}^m_{i=e}\}$
        for $\sigma \leftarrow$ UnfoldPPM($\Sigma|_{\mathbf{1}^m_{i=e}}$) do
          if $T_{i=e} =$ Normalize($p\langle\sigma\rangle$) then
            for $x \in \mathbf{1}^m_{i=e}$ do
              $\Theta(x) \leftarrow \Theta(x) \cup \{\sigma(x)\}$
      for $x \in \mathbf{1}^m_{i=e}$ do
        UpdatePPM($\Sigma, x, \Theta(x)$)
    if $\exists\sigma \in$ UnfoldPPM($\Sigma$) : $\partial(p\langle\sigma\rangle) = T$ then
      return $\sigma$
    else
      return Unsolvable
```

The idea of the algorithm is as follows: We first generate the most general ppm Σ. We then go through each face $q\langle\tau\rangle$ of the goal boundary T and use it to restrict Σ. Crucially, we order the faces by their dimensions as higher-dimensional faces in the boundary will restrict Σ more than lower-dimensional faces. In the most extreme case, q will have the same dimension as p, which allows us to give a unique value for each element of the face q, i.e., each element of $\mathbf{1}^m_{i=e}$ if $q\langle\tau\rangle$ was the (i,e)-th face. Otherwise, we will check for each element of $\mathbf{1}^m_{i=e}$ which values lead to some contortion which is equal to $q\langle\tau\rangle$. To do this we need to unfold Σ at this subposet and check for each resulting poset map if it gives rise to a valid contortion.

The algorithm is obviously correct since we only return a poset map σ if it is indeed contorting p to the correct boundary.

TODO It appears that we do not need this filter! Can we prove this?

TODO Prove completeness of Algorithm 2. This is not essentially difficult to see, but requires some reasoning.

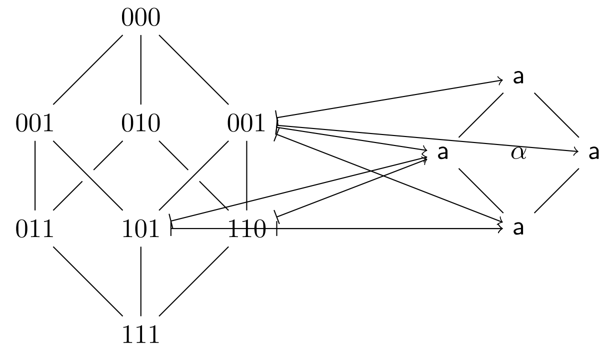
TODO Prove run time of Algorithm 2. I have not been able to come up with an argument for this. Potentially, unfolding the subposet corresponding to a face could lead to D^n_{m-1} many poset maps which need to be checked, hence we would only reduce the complexity to the previous Dedekind number. However, in all examples I have encountered we have one face q which is the same as p, therefore we can significantly reduce the size of Σ before we need to do any unfolding. I do not yet know if this is always the case and what this means for the worst-case complexity of the algorithm.

Example 2 6 (continued) *We will now sketch an example run of Algorithm 2. We want to find a $\sigma : \mathbf{1}^3 \to \mathbf{1}^2$ such that $\alpha\langle\sigma\rangle$ has this boundary:*

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

In the beginning, the ppm Σ is initialized totally 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 Σ. The first face of the boundary is a contortion of α, which is the highest-dimensional face, so we use it to restrict Σ. Since α is also the cell that we are contorting, the subposet $\mathbf{1}^3_{1=0}$ of the domain of Σ is mapped in a unique way to the elements of $\mathbf{1}^2$. The monotonicity restrictions on ppms further restrict Σ: 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 Σ:*



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

3 Composition solver

TODO This section only exists as a rough scaffolding.

3.1 Kan cubical sets

A cubical set X satisfies the Kan condition if for any open box $[t_1, s_1, \ldots, t_n, s_n]\ t$ we have a cell s such that $[t_1, s_1, \ldots, t_n, s_n, t, s]$ is a valid boundary. Call $\mathsf{Comp}([t_1, s_1, \ldots, t_n, s_n]\ t) := s$ the Kan composition of the box.

TODO Make more general to uniform Kan condition

Definition 5. *Given a context Γ and a (valid) boundary T, the problem* KanCubicalCell*(X,T) is to find an open box $[t_1, s_1, \ldots, t_n, s_n]\ t$ such that $\partial(\mathsf{Comp}([t_1, s_1, \ldots, t_n, s_n]\ t)) = T$.*

3.2 Undecidability of finding Kan compositions

Since Kan cubical sets are a model of groupoids and groupoids are a generalization of groups, it is no surprise that we can encode groups with cubical sets. Since also Kan composition is sufficient to generate concatenations of paths etc, we can encode word problems for groups, which can be used to show that .. is undecidable.

Theorem 1. KanCubicalCell *is undecidable.*

Proof. *We show how the word problem for groups* WordGroup *reduces to* KanCubicalCell.

Given a finitely generated group $G = \langle X \mid R \rangle$ with neutral element e and where X are its set of generators and R relations over X, i.e., $R \subseteq \{a_1 \ldots a_n \mid a_\_ \in X \cup X^{-1}\}$ such that for all $w \in R$, we have $w = e$ in G.

Construct the cubical set G with 0-cell $\star$, 1-cells X with $\partial(p) = [\star, \star]$ for any $p \in X$, and for each $w \ \in R$ a 2-cell r_w with $\partial(r_w) = [w, \star\langle s^1\rangle, \star\langle s^1\rangle, \star\langle s^1\rangle]$ where w is the composition of words

TODO We need to have introduced compositions as this point

TODO We need to have introduced inverses as this point

Given a word v in G, we now want to decide whether $v = e$ in G. Suppose that KanCubicalCell *was decidable. We show that we can then decide* WordGroup

Given s = Comp such that $\partial(s) = [v, ...]$
Suppose R infinite or something?

TODO Give reduction Word ¡ KanCubicalCell. For some specific group with undecidable word problem or Uniform word problem?

3.3 A semi-deciability procedure for 1-dimensional cubes

TODO Only going in one side direction will always yield a proof, if it exists

3.4 Decidability of finitely generated groups

TODO Carry over some decidability results from the word problem for groups over to our setting?

4 A Tactic for Cubical Agda

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 Translating to 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 ϕ 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 ϕ_i is the i-th element of the tuple ϕ and @ denotes evaluation of formulas: The indices of x which are set to 1 mean that the corresponding variable of ϕ 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.

4.2 Translating to PathP types

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 7 (continued) *The proof goal already introduced above:*

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

has the following corresponding PathP type in Cubical Agda.

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

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

TODO We might want to have a prover which can take more general boundary descriptions as proposed with extension types by [6].

4.3 Searching for involutions

TODO Cubical Agda allows formulas of a De Morgan algebra over the interval variables, meaning that we can also have inverted paths primitively. Since this makes many proofs shorter we will want to search for such proofs as well.

4.4 Using the tactic

TODO Give a case study using the tactic for a more involved proof goal

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.

A Algorithms

We will give pseudo-code for algorithms. We assume we have a context Γ generating a cubical set and a generating cell $p \in \Gamma$ with $\mathsf{dim}(p) = n$ in the following.

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 $\Sigma'$
```

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: $\{\sigma : \mathbf{1}^m \to \mathbf{1}^n\}$ poset maps
  procedure UnfoldPPM($\Sigma$)
      return $\{x \mapsto v, \text{UnfoldPPM}(\text{UpdatePPM}(\Sigma, x, v) - x) \mid v \in \Sigma(x)\}$
```

Worst case run-time of $\mathcal{O}(D_m^n)$