Searching in cubes

Maximilian Doré

November 14, 2022

As observed by [**?**], do not require a full De Morgan structure. In particular, it suffices to consider only $\vee$ and $\wedge$ forming a bounded free distributive lattice.

[**?**]: we shall think geometrically, and prove algebraically!

1 Cubical sets and their boundaries

1.1 Background

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 $\mathbf{1}^n$. Given $\mathbf{1}^m \to \mathbf{1}^n$ and a poset $P \subseteq \mathbf{1}^m$, we will denote with $\sigma|_P : P \to \mathbf{1}^n$ the restriction of σ to P. In particular, if $P = \mathbf{1}^m_{i=e}$, we might denote TODO INDEX REMOVED FROM IT

Certain poset maps of interest are:

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

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

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.

The cell $p\langle s^i\rangle$ is called a *degeneracy* of the cell p. If the dimension n is understood, we will sometimes denote with s^m the composite $s^n \circ s^{n+1} \circ \ldots \circ s^m$. This gives the shorthand $p\langle s^m\rangle$ for an n-cell p considered a degenerate m-cell for $m > n$.

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

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

The functor laws of X ensure that the faces of p have overlapping boundaries. TODO GENERAL FORM

Functoriality 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

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

This means for instance

$$p\langle\sigma\rangle\langle d^{(i,e)}\rangle = p\langle\sigma|_{\mathbf{1}^n_{i=e}}\rangle$$

TODO EXPLAIN WHY Thereby we have a specified boundary.

In the next section we will see how this consideration is turned up side down: Given a boundary, we want to come up with poset maps such as σ to attain the cell under question. 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 $T_{(i=e)} := p_{(i,e)}$.

Kan cubical sets TODO Kan condition This is not all! Kan condition

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 $d^{(1,0)}(\mathsf{seg}) = \mathsf{zero}$ and $d^{(1,1)}(\mathsf{seg}) = \mathsf{one}$.

Int contains all necessary contortions, for instance $\mathsf{zero}\langle^{0\mapsto\ ()}_{1\mapsto\ ()}\rangle$ is a well-defined 1-cell which is the degenerately zero (which is the same as $s^1(\mathsf{zero})$).

The identity map gives an n-contortion for every n-cell which is equal to the original cell, for instance $\mathsf{seg}\langle^{0\mapsto\ 0}_{1\mapsto\ 1}\rangle$ is just the cell seg.

Assuming that Int is Kan, we can obtain the reversal of seg as follows: TODO Kan inversion example

Example 2. We define the cubical set $\mathsf{S2}$ with the following data: We have a single 0-cell, so $\mathsf{S2}_0 = \{\mathsf{a}\}$. There is one non-degenerate 2-cell $\alpha \in \mathsf{S2}_2$, which has as its faces a as a degenerate 1-cell, i.e., $d^{(2,0)}(\alpha) = d^{(2,1)}(\alpha) = d^{(1,0)}(\alpha) = d^{(1,1)}(\alpha) = s^1(\mathsf{a})$.

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 $\{\mathsf{p}, \mathsf{q}, \mathsf{r}\} \subseteq \mathsf{Triangle}_1$ on which the face maps are defined by $d^{(1,0)}(\mathsf{p}) = \mathsf{x}$ and $d^{(1,1)}(\mathsf{p}) = \mathsf{y}$, $d^{(1,0)}(\mathsf{q}) = \mathsf{y}$ and $d^{(1,1)}(\mathsf{q}) = \mathsf{z}$, $d^{(1,0)}(\mathsf{r}) = \mathsf{x}$ and $d^{(1,1)}(\mathsf{r}) = \mathsf{z}$. We have one non-degenerate 2-cell $\phi \in \mathsf{Triangle}_2$ with $d^{(1,0)}(\phi) = \mathsf{p}$, $d^{(1,1)}(\phi) = \mathsf{r}$, $d^{(2,0)}(\phi) = s^1(\mathsf{x})$ and $d^{(2,1)}(\phi) = \mathsf{q}$.

1.2 Cubical sets algorithmically

(Kan) 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. TODO BACKGROUND This is what HITs are for In [**?**, Sect. 6.4], strict groupoid turned into cubical set with general construction Higher inductive types are constructed as generalized pushout construction (free ω-groupoid construction)

Furthermore, we will introduce a term language to talk about the objects generated from this finitary representation, with normal forms. Also we consider complexity of boundary computation

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 valid context and describe algorithm checking validity Checking that a context is well-formed can be done in $\mathcal{O}(?k)$ and is implemented in Algorithm 8 and Algorithm 9.

In the following we will assume a valid context Γ generating a cubical set X.

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 $\mathsf{zero}\langle^{0\mapsto\,()}_{1\mapsto\,()}\rangle$ and $\mathsf{seg}\langle^{0\mapsto\,0}_{1\mapsto\,0}\rangle$ describe the same cell, namely the degenerate 1-cell which is constantly zero. Since the former is more basic, 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 σ' such that $p\langle\sigma\rangle = q\langle\sigma'\rangle$.

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 poset map $\sigma' \circ s^i \circ \sigma$. This map comes about as follows: we know that $\sigma(x)_i$ is the same for all $x \in \mathbf{1}^m$, so we can obtain $p\langle\sigma\rangle$ also as a contortion of q by first applying σ, then forgetting about index i with s^i and finally applying the contortion of the face, σ'.

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⟨σ⟩)
    if ∃1 ≤ i ≤ n, e ∈ {0,1} : Img(σ) ⊆ 1^n_{i=e} then
        q⟨σ'⟩ ← T_{i=e}
        return NORMALIZE(q⟨σ' ∘ s^i ∘ σ⟩)
    else
        return p⟨σ⟩
```

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 $1 \le i \le n$ 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 $\mathcal{O}(2^m n^2)$.

TODO where to describe Algorithm 7

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

Example ?? (continued)**.** The following context generates S2:
$[\mathsf{a} : [], \alpha : [\mathsf{a}\langle{}^{0\mapsto\,()}_{1\mapsto\,()}\rangle, \mathsf{a}\langle{}^{0\mapsto\,()}_{1\mapsto\,()}\rangle, \mathsf{a}\langle{}^{0\mapsto\,()}_{1\mapsto\,()}\rangle, \mathsf{a}\langle{}^{0\mapsto\,()}_{1\mapsto\,()}\rangle]]$

Example 3 (continued)**.** The following context generates Triangle:

$[\mathsf{x} : [], \mathsf{y} : [], \mathsf{z} : [], \mathsf{p} : [\mathsf{x}, \mathsf{y}]_{\substack{00\mapsto 00}}, , \mathsf{q} : [\mathsf{y}, \mathsf{z}], \mathsf{r} : [\mathsf{x}, \mathsf{z}], \phi : [\mathsf{p}, \mathsf{r}, \mathsf{x}\langle^{0\mapsto ()}_{1\mapsto ()}\rangle, \mathsf{q}]_{\substack{00\mapsto 0}}]$
The contortion $(\phi, \substack{01\mapsto 00\\10\mapsto 01\\11\mapsto 01})$ has normal form $(p, \substack{01\mapsto 0\\10\mapsto 1\\11\mapsto 1})$.

1.3 Searching for cells

The search problem is the following: given a cubical set and a certain boundary, is there a cell with that boundary. Using the finitary representation introduced above, we can phrase it as follows:

Definition 3. Given a context Γ and a (valid) boundary T, the problem CubicalCell(X,T) is to find a term $t \in Tm_{\mathsf{dim}(T)}(\Gamma)$ such that $\partial(t) = T$.

(More generally, we can encode all word problems from universal algebra in our setting.)

Theorem 1. CubicalCell is undecidable.

Proof. Given a set of generators $\{a, ...\}$ of a group G, define Group with a single 0-cell $\mathsf{Group}_0 = \{\star\}$, 1-cells $\{\mathsf{a}, ...\} \subseteq \mathsf{Group}_1$

(there are more 2-cells that need to be added as explained in [**?**, Sect. 6.3])

□

If it can be solved by a contortion, it is clearly decidable since there are only finitely many substitutions for each cell in the context, and the search space is thereby finite. However, the search space grows super-exponentially: Suppose we want to check if there is an m-dimensional contortion of a 1-cell satisfying a goal, i.e., we are looking for a monotone map $\mathbf{1}^m \to \mathbf{1}$,. For $m = 6$, there are 7828354 substitutions that we would have to check and for 9 the exact number of possible substitutions has not even been computed yet `http://oeis.org/A00372`. Therefore we need something more clever to explore the search space.

2 Decidable CubicalCell / Searching for contortions

A certainly decidable subproblem of CubicalCell is to decide whether there exists a cell simply by means of contorting the cells given in the context: for a given goal boundary T and a generating cell p, there are finitely many

poset maps $\mathbf{1}^{\mathsf{dim}(T)} \to \mathbf{1}^{\mathsf{dim}(p)}$, so we can naïvely check whether one of those maps gives rise to a fitting contortion.

We can try this for all elements in the context, thereby have run time $\mathcal{O}(|\Gamma|)$

2.1 General complexity considerations

Revert characterization

$$d^{(i,e)}(X(\sigma)(p)) = X(\sigma|_{\mathbf{1}^n_{i=e}})(p)$$

2.2 Potential substitutions

For an effective algorithm, we turn to following notion.

We introduce potential substitution, which keep track of all the potential values of $\mathbf{1}^n$ a poset map σ might assign to an element of $x \in \mathbf{1}^m$.

They allow to represent all maps $\mathbf{1}^m \to \mathbf{1}^n$ at once. Solves memory problem. We need to explore the search space cleverly.

Definition 4. A *potential poset map* (*ppm*) is a map $\Sigma : \mathbf{1}^m \to \mathcal{P}(\mathbf{1}^n)$ such that for all $x \leq y$: for all $u \in \Sigma(x)$ exists $v \in \Sigma(y)$ such that $u \leq v$. TODO MORE

The condition on poset maps means that

For effectively computing with poset maps we will consider 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.

A non-empty potential map 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.

Algorithm 11 Therefore we have a runtime of $\mathcal{O}(2^m 2^n)$

From a potential poset map we can generate all poset maps. This has a worst-case runtime of Dedekind for the full one.

Algorithm 12 $\mathcal{O}(D^n_m)$

2.3 Simple solver

We want to try and see

Algorithm 2 Simple solver

Input: Γ context, $p \in \Gamma$, T goal boundary ($n := \mathsf{dim}(p), m := \mathsf{dim}(T)$)
Output: σ s.t. $\partial(p\langle\sigma\rangle) = T$ if there is such a σ, `Unsolvable` otherwise

```
procedure SimpleSolve(p)
    Σ ← {x ↦ 1^n | x ∈ 1^m}
    for i ← 1 ≤ i ≤ m, e ← {0,1} do
        Θ ← (x : 1^m_{i=e}) ↦ ∅
        for σ ← UnfoldPPM(Σ|_{1^m_{i=e}}) do
            if T_{i=e} = Normalize(p⟨σ⟩) then
                for x ∈ 1^m_{i=e} do
                    Θ(x) ← Θ(x) ∪ {σ(x)}
            for x ∈ 1^m_{i=e} do
                UpdatePPM(Σ, x, Θ(x))
    if ∃σ ∈ UnfoldPPM(Σ) : ∂(p⟨σ⟩) = T then
        return σ
    else
        return Unsolvable
```

Instead of having to check all D^n_m poset maps, we have to check in the worst case $2mD^n_{m-1}$ poset mapos.

Algorithm 3 Simple solver recursively

Input: Γ context, $p \in \Gamma$, T goal boundary ($n := \mathsf{dim}(p), m := \mathsf{dim}(T)$)
Output: All σ s.t. $\partial(p\langle\sigma\rangle) = T$

```
procedure SimpleSolve(p)
    Σ ← {x ↦ 1^n | x ∈ 1^m}
    for i ← 1 ≤ i ≤ m, e ← {0,1} do
        Θ ← (x : 1^m_{i=e}) ↦ ∅
        for σ ← SimpleSolve(∂(T_{i=e})) do
            for x ∈ 1^m_{i=e} do
                Θ(x) ← Θ(x) ∪ {σ(x)}
        for x ∈ 1^m_{i=e} do
            UpdatePPM(Σ, x, Θ(x))
    return {σ | σ ∈ UnfoldPPM(Σ) : ∂(p⟨σ⟩) = T}
```

More than that, the potential substitutions allow us to narrow down the search space in other ways.

Try to reduce the search space going from the bottom.

COMPLETENESS??? It's complete. But not correct – overapproximat-

ing results

3 Composition solver

We use Kan composition

4 Negation

How to encode search for reversals

5 Relation to different notions of cubical sets

Since the generating data is the non-degenerate cells and the face maps on them, we can adopt the following notation to describe cubical sets, which go under higher inductive types in Cubical Agda:

TODO also boundary description with poset maps. These correspond to specifying the dmaps: One index is constant, rest is given by given poset map

5.0.1 Relation to face formulas

The morphisms in $\square_{\wedge\vee}$ have a succinct description as tuples of elements of a free distributive lattices, which we will call telescopes. Cubical Agda uses telescopes to describe this. However, these are hard to construct and have no geometric intuition. This is why we had used poset maps. We can go back and forth between both representations easily.

Given $s : \mathbf{1}^m \to \mathbf{1}^n$, 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.

Algorithm 4 TODO

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

Output: ϕ n-tuple of elements of free distributive lattice over m variables

```
procedure SUBST2TELE(σ)
    for i ← 1 to n do
        φ_i ← {x ∈ 1^m | σ(x)_i = 1}        ▷ O(2^m) many elements in 1^m
    return (φ_1, ..., φ_n)
```

TODO mention set representation of DNF. Also mention normalization necessary – what's its runtime?

From an n-tuple of formulas over ϕ we can read off a poset map $s : \mathbf{1}^m \to \mathbf{1}^n$ as follows: Given an element $x \in \mathbf{1}^m$, $s(x) = (e_1, ..., e_n)$ where $e_i = \phi_i @ x$ TODO EVALUATION

Algorithm 5 TODO

Input: TODO

Output: TODO

```
procedure TELE2SUBST(p)
    for x ← 1^m do
        for i ← 1 to n do
            σ(x)_i ← φ_i@x
    return σ
```

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 pretty quick and no obstruction.

Example 4.

TODO PROOF THAT THESE ARE MUTUALLY INVERSE?

5.1 Nominal perspective

Have set $I = \{i, j, k, ...\}$ of names. Then give complete description for Cubical Agda.

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.

TODO definition retrieval?

Algorithm 6 Computing the face of a contortion

Input: Contortion $p\langle\sigma\rangle$ with $\sigma : \mathbf{1}^m \to \mathbf{1}^n$, (i, e) $1 \leq i \leq n$, $e \in \{0, 1\}$
Output: $d^{(i,e)}(p\langle\sigma\rangle)$
  procedure FACE($p\langle\sigma\rangle, (i, e)$)
    Do Something

Algorithm 7 Computing the boundary of a contortion

Input: Contortion $p\langle\sigma\rangle$ where p is an n-cell of X and $\sigma : \mathbf{1}^m \to \mathbf{1}^n$.
Output: $\partial(p\langle\sigma\rangle)$
  procedure BOUNDARY($p\langle\sigma\rangle$)
    Do Something

Algorithm 8 Well-formed boundary

Input: Γ context, T term
Output: `OK` if T well formed, `Error` otherwise
  procedure WELLFORMED(Γ, T)
    Do Something

Algorithm 9 Well-formed context

Input: Γ context
Output: `OK` if Γ well formed, `Error` otherwise
  procedure WELLFORMED(Γ)
    Gradually build up? Or is it ok to have mutually defined cells?

Algorithm 10 Faces

Input: m, n
Output: $\{xs\}$ where xs are the n-faces of $\mathbf{1}^m$

```
procedure FACES(m, n)
    if n = 0 then
        {{x} | x ∈ 1^m}
    else if m = n then
        {1^m}
    else
        {{0x | x ∈ xs}, {1x | x ∈ xs} | xs ∈ FACES(m − 1, n) }
        ∪TODO
```

A.1 Potential substitutions

Algorithm 11 Update potential poset map

Input: $\Sigma : \mathbf{1}^m \to \mathcal{P}(\mathbf{1}^n)$ ppm, $x \in \mathbf{1}^m$, $vs \in \mathcal{P}(\mathbf{1}^n) - \emptyset$
Output: Updated ppm $\Sigma' : \mathbf{1}^m \to \mathbf{1}^n$ with $\Sigma(x) = vs$.

```
procedure UPDATEPPM(Σ, (x, vs))
    for y ← 1^m do
        if x = y then
            Σ'(x) ← vs
        else if y ≤ x then
            Σ'(y) ← {u | u ∈ Σ(y) such that ∃v ∈ vs : u ≤ v}
        else if x ≤ y then
            Σ'(y) ← {u | u ∈ Σ(y) such that ∃v ∈ vs : v ≤ u}
    return Σ'
```

Algorithm 12 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(Σ)
    σ : x ↦ Σ(x) TODO
```