A formalization of CW complexes in Lean4

Floris van Doorn
University of Bonn

Hannah Scholz
University of Bonn

2nd October, 2025

Contents

1 Introduction

This is the introduction. It could explain the following:

- Mathematical relevance of CW complexes
- What is Lean and mathlib and why is it relevant
- Any related work (do CW complexes exist in other proof assistants?)

I think Floris was maybe going to write this?

2 Preliminaries

2.1 Lean and `Mathlib`

Lean and `Mathlib` make use of *typeclasses* to provide definitions on various types with potentially different behaviour. For example, `Mathlib` has a general notion of a topological space on an arbitrary type. One can then provide specific *instances* of a typeclass, for example the metric topology on the reals, or the subspace topology on a subtype of `X`, assuming that `X` has a topology. We write `[TopologicalSpace X]` to assume that a space `X` has a topology. Additionally, there are forgetful instances, e.g. every metric space is a topological space. Lean uses *typeclass inference* to search the graph of instances to find the required instances. More about typeclasses in Lean can be found in [SUM20] and [Baa25].

By default, Lean's typeclass inference algorithm requires all parameters to be known before it searches for an instance. This behaviour can be modified by marking certain parameters with `outParam`, which means that typeclass inference will search for an instance even if the out-parameters are not yet known. Furthermore, typeclass inference will not consider the values of `outParam` parameters, and will search for the first instance where all parameters that are not marked as `outParam` unify (up to definitional equality). Therefore, an argument should only be marked as an `outParam` if for every combination of the parameters not marked as `outParam` there is at most one instance. For example, the class `Membership α β` states that for `a : α` and `b : β` we have specified the notation `a ∈ b`. The first parameter is marked as an `outParam`, since it is uniquely determined by the second parameter. Typical instances include `Membership α (List α)` and `Membership α (Set α)`.

Another technical detail of Lean that we will manipulate is reducibility. The reducibility of a definition describes what processes are allowed to unfold it. Definitions in `Mathlib` are by default `semireducible`, meaning they are unfolded for basic checks like definitional equality but not for more time intensive operations like typeclass inference. However, sometimes we do want definitions to be unfolded in typeclass inference. To achieve this behaviour, we need to change the reducibility setting to `reducible` which is done by writing `abbrev` instead of `def`. `Mathlib`, for example has the abbreviation `unitInterval` for the interval `Set.Icc 0 1 : Set ℝ`.

Better example? I think this one only exists to be able to provide notation? Would Path.Homotopy be better?

More on `outParam` and `abbrev` can be found in [Dev25].

Like a lot of programming languages, Lean makes use of namespaces to sort its declarations and reduce the length of names when namespaces are opened. One can prevent this shortening of names by specifying a declaration to be `protected`. This ensures that the declaration is still available but makes using it less convenient. The modifier is frequently used to prevent naming conflicts in namespaces that might be opened together and to discourage the use of certain declarations in favour of more idiomatic ones.

2.2 Preliminary Mathematics in Lean

In `Mathlib`, we write `TopologicalSpace X` to say that a type `X` has a topology. `IsOpen A` and `IsClosed A` for a set `A` in `X` assert openness and closedness. For a map `f : X → Y` between topological spaces continuity is described as `Continuous f` or `ContinuousOn f A` for continuity on a set. To specify that a topological space `X` is Hausdorff one writes `T2Space X`.

A partial bijection between two types `X` and `Y` has type `PartialEquiv X Y` and is made up of two total functions `X → Y` and `Y → X`, a set in `X` called the *source*, a set in `Y` called the *target* and proofs that that the target is mapped to the source and vice versa and that the two maps are inverse to each other on both the source and target.

A *completely distributive lattice* is a complete lattice in which even infinite joins and meets distribute over each other. This differentiates it from complete distributive lattices. In `Mathlib`, one writes `CompletelyDistribLattice α` to provide a completely distributive lattice structure on α. We furthermore write `x ⊔ y` for the binary and `⨆ (i : ι), x i` for the indexed join, and `x ⊓ y` for the binary and `⨅ (i : ι), x` for the indexed meet.

Explain the definition in more detail? Here or below?

3 Definition of CW complexes

An *(absolute) CW complex* is a topological space that can be constructed by glueing images of closed discs of different dimensions together along the images of their boundaries. The image of an n-dimensional open disc in the CW complex is called an *n-cell*. The cells up to and including dimension n make up what is called the *n-skeleton*. In a *relative CW complex* these discs can additionally be attached to a specified base set.

Definitions of CW complexes present in the literature can be broadly categorized into two approaches: firstly, there is the "classical" approach that sticks closely in style to Whiteheads original definition in [Whi18]. This definition assumes that the cells all lie in one topological space and describes how they interact with each other and the topology. The second approach is more categorical in nature. In this version, the $(n+1)$-skeleton is defined as a pushout involving the n-skeleton. The CW complex is then defined as the colimit of the skeletons.

At the start of this project neither of the approaches had been formalized in Lean. We chose to proceed with the former approach because it avoids having to deal with different topological spaces and inclusions between them. As the other approach has been formalized by Jiazhen Xia and Elliot Dean Young and refactored by Joël Riou, both are now formalized and part of `Mathlib`. Our version is called `Topology.RelCWComplex`, while the other one is `TopCat.RelativeCWComplex`.

The definition chosen for formalization is the following, where D^n is the closed unit n-disc:

Definition 1. Let X be a Hausdorff space and $D \subseteq X$ be a subset of X. A *(relative) CW complex* on X with *base* D consists of a family of indexing sets $(I_n)_{n\in\mathbb{N}}$ and a family of continuous maps $(Q_i^n : D^n \to X)_{n\in\mathbb{N}, i\in I_n}$ called *characteristic maps* with the following properties:

(i) $Q_i^n|_{\mathrm{int}(D^n)} : \mathrm{int}(D^n) \to Q_i^n(\mathrm{int}(D^n))$ is a homeomorphism for every $n \in \mathbb{N}$ and $i \in I_n$. We call $e_i^n := Q_i^n(\mathrm{int}(D^n))$ an *(open) n-cell* and $\overline{e}_i^n := Q_i^n(D^n)$ a *closed n-cell*.

(ii) Two different open cells are disjoint.

(iii) Every open cell is disjoint with D.

(iv) For each $n \in \mathbb{N}$ and $i \in I_n$ the *cell frontier* $\partial e_i^n := Q_i^n(\partial D^n)$ is contained in the union of D with a finite number of closed cells of a lower dimension. We say that X has *closure finiteness*.

(v) A set $A \subseteq X$ is closed if the intersections $A \cap D$ and $A \cap \overline{e}_i^n$ are closed for all $n \in \mathbb{N}$ and $i \in I_n$. We say that X has *weak topology* with respect to the base and the closed cells.

(vi) D is closed.

(vii) The union of D and all closed cells is X.

It is important to notice that an open cell is not necessarily open and that the cell frontier is not necessarily the frontier of the corresponding cell.

The translation of this definition in `Mathlib` can be found in Figure 1.

One obvious change in the Lean definition is that instead of talking about the topological space X being a CW complex, it talks about a set C being a CW complex in the ambient space X. This allows us to treat spaces that are naturally defined as subspaces of a given space as a CW complex without taking subtypes. Additionally, for constructions such as the disjoint union of two CW complexes, it avoids dealing with constructed topologies. It is however derivable from the definition that C is closed in X. So while a closed interval in the real line can be considered as a CW complex in its natural ambient space, the open interval cannot and needs to be considered as a CW complex in itself. This approach is inspired by [Gon+13], where the authors notice that it is helpful to consider subsets of an ambient group to avoid having to work with different group operations and similar issues.

Even though the behaviour of a CW complex depends strongly on its data and there can be different "non-equivalent"[1] CW structures on the same space, we have chosen to make it a `class`, effectively treating it more like a property than a structure. This is to be able to make use of Lean's typeclass inference (see Section 2.1).

We don't require X to be a Hausdorff space, so to properly state that C is a CW complex with base D, we write `[RelCWComplex C D] [T2Space X]`.

The base D is an `outParam` (see Section 2.1). This is because lemma statements about CW complexes typically refer to just the underlying set `C` without mentioning the base `D`. Normally, for typeclass inference to run the user would have to go out of their way to specify `D`. We disable this requirement by adding the `outParam` specification.

[1] in the sense of cellular isomorphism

Figure 1: Definition of relative CW complexes in Mathlib

```
class RelCWComplex.{u} {X : Type u} [TopologicalSpace X] (C : Set X)
        (D : outParam (Set X)) where
  cell (n : ℕ) : Type u
  map (n : ℕ) (i : cell n) : PartialEquiv (Fin n → ℝ) X
  source_eq (n : ℕ) (i : cell n) : (map n i).source = ball 0 1
  continuousOn (n : ℕ) (i : cell n) : ContinuousOn (map n i) (closedBall 0 1)
  continuousOn_symm (n : ℕ) (i : cell n) : ContinuousOn (map n i).symm
    (map n i).target
  pairwiseDisjoint' :
    (univ : Set (Σ n, cell n)).PairwiseDisjoint
    (fun ni ↦ map ni.1 ni.2 '' ball 0 1)
  disjointBase' (n : ℕ) (i : cell n) : Disjoint (map n i '' ball 0 1) D
  mapsTo (n : ℕ) (i : cell n) : ∃ I : Π m, Finset (cell m),
    MapsTo (map n i) (sphere 0 1)
    (D ∪ ⋃ (m < n) (j ∈ I m), map m j '' closedBall 0 1)
  closed' (A : Set X) (hAC : A ⊆ C) :
    ((∀ n j, IsClosed (A ∩ map n j '' closedBall 0 1)) ∧ IsClosed (A ∩ D)) →
    IsClosed A
  isClosedBase : IsClosed D
  union' : D ∪ ⋃ (n : ℕ) (j : cell n), map n j '' closedBall 0 1 = C
```

In topology, most CW complexes that are considered have empty base and often the term "CW complex" refers to this type of complex. Those CW complexes are called *absolute CW complexes*.

Most naturally one would simply define absolute CW complexes in Lean in the same way: as a relative CW complex with empty base. However, this leads to two issues: firstly, when defining an absolute CW complex there are now trivial proofs that need to be provided and some simplifications that need to be performed for every new instance and definition. This produces a lot of duplicate code or requires a separate definition that is used as a replacement constructor. Secondly, with absolute CW complexes we have encountered instances on the same set with provably but not definitionally equal base sets. The product of two CW complexes `RelCWComplex C D` and `RelCWComplex E F` is of type `RelCWComplex (C ×ˢ E) (D ×ˢ E ∪ C ×ˢ F)` where `×ˢ` is the binary product of sets. For absolute CW complexes we get `RelCWComplex (C ×ˢ E) (∅ ×ˢ E ∪ C ×ˢ ∅)` which is not definitionally equal to `RelCWComplex (C ×ˢ E) ∅`. For this reason, we define an instance specifically for absolute CW complexes and want this to be inferred over the relative version. But since D is an `outParam`, we cannot specify typeclass inference to be looking for a base that is definitionally equal to the empty set.

The solution is to have absolute CW complexes be their own class that agrees with relative CW complexes except for the empty base, trivial proofs and simplifications. The type of absolute CW complexes on the set C in Lean is `CWComplex C`. We then provide an instance stating that absolute CW complexes are relative CW complexes and a definition in the other direction for relative CW complexes with empty base. The latter cannot be an instance as this would create an instance loop. Additionally, it would enable typeclass inference to also consider `RelCWComplex` instances when looking for a `CWComplex`

Reference or argument as to why this is bad?

instance, which is exactly what we wanted to avoid. To avoid having duplicate notions `CWComplex.cell` and `RelCWComplex.cell` and `CWComplex.map` and `RelCWComplex.map`, we mark the version for absolute CW complexes as `protected` strongly encouraging the user to only use the version for relative CW complexes which is also available for absolute ones through the instance (see Section 2.1).

Talk about attribute when it is done

As in Definition 1, we define the notions of open cells, closed cells and cell frontiers. We define them only for relative CW complexes but, as for the indexing types and characteristic maps, these notions can be used for absolute ones because of the instance mentioned above.

We then define subcomplexes as closed sets that are unions of open cells of the complex.

```
structure Subcomplex (C : Set X) {D : Set X} [RelCWComplex C D] where
  carrier : Set X
  I : Π n, Set (cell C n)
  closed' : IsClosed carrier
  union' : D ∪ ⋃ (n : ℕ) (j : I n), openCell (C := C) n j = carrier
```

We provide additional definitions for other ways of describing them: firstly, as a union of open cells where the closure of every cell is already contained in the union and secondly, as a union of open cells that is also a CW complex. Here is the former as we will need it below:

```
def RelCWComplex.Subcomplex.mk' [T2Space X] (C : Set X) {D : Set X}
    [RelCWComplex C D] (E : Set X) (I : Π n, Set (cell C n))
    (closedCell_subset : ∀ (n : ℕ) (i : I n), closedCell (C := C) n i ⊆ E)
    (union : D ∪ ⋃ (n : ℕ) (j : I n), openCell (C := C) n j = E) :
    Subcomplex C where
  carrier := E
  I := I
  closed' := /- Proof omitted-/
  union' := union
```

We show that subcomplexes are again CW complexes and that the type of subcomplexes of a specific CW complex has the structure of a `CompletelyDistribLattice` (see Section 2.2).

Defining subcomplexes allows us to talk about the skeletons of a CW complex. The typical definition of the n-skeleton in the following:

Definition 2. The n-skeleton of a CW complex C is defined as $C_n \coloneqq \bigcup_{m<n+1} \bigcup_{i \in I_m} \overline{e}_i^m$ where $-1 \leq n \leq \infty$.

Since proofs about CW complexes frequently employ induction, we want to make using this proof technique as easy as possible. Starting an induction at -1 is unfortunately not very convenient in Lean. For this reason, we first define an auxiliary version of the skeletons where the dimensions are shifted by one:

```
def RelCWComplex.skeletonLT (C : Set X) {D : Set X} [RelCWComplex C D]
    (n : ℕ∞) : Subcomplex C :=
  Subcomplex.mk' _ (D ∪ ⋃ (m : ℕ) (_ : m < n) (j : cell C m), closedCell m j)
    (fun l ↦ {x : cell C l | l < n}) (/-Proof omitted-/) (/-Proof omitted-/)
```

We use this to define the usual skeleton:

```
abbrev RelCWComplex.skeleton (C : Set X) {D : Set X} [RelCWComplex C D]
    (n : ℕ∞) : Subcomplex C :=
  skeletonLT C (n + 1)
```

Since we expect proofs about `skeleton` to be short reductions of the claim to the corresponding statement about `skeletonLT`, we spare the user the manual unfolding of `skeleton` by marking it as an `abbrev` instead of a `def` (see Section 2.1). The definition `skeleton` exists mostly for completeness' sake. Both lemmata and definitions should use `skeletonLT` to make proofs easier and then possibly derive a version for `skeleton`.

Should subcomplexes and cellular maps go into a separate section? They don't really fit here but also think there isn't enough to say to put them in their own section.

We also want to introduce a sensible notion of structure preserving maps between CW complexes. A natural notion is a *cellular map*. A cellular map is a continuous map between two CW complexes X and Y that sends the n-skeleton of X to the n-skeleton of Y for every n. In Lean this definition translates to:

```
structure CellularMap (C : Set X) {D : Set X} [RelCWComplex C D] (E : Set Y)
    {F : Set Y} [RelCWComplex E F] where
  protected toFun : X → Y
  protected continuousOn_toFun : ContinuousOn toFun C
  image_skeletonLT_subset' (n : ℕ) : toFun '' (skeletonLT C n) ⊆ skeletonLT E n
```

We also introduce the notion of *cellular isomorphisms*:

Is there even a math name?

```
structure CellularEquiv (C : Set X) {D : Set X} [RelCWComplex C D] (E : Set Y)
    {F : Set Y} [RelCWComplex E F] extends PartialEquiv X Y where
  continuousOn_toPartialEquiv : ContinuousOn toPartialEquiv C
  image_toPartialEquiv_skeletonLT_subset' (n : ℕ) :
    toPartialEquiv '' (skeletonLT C n) ⊆ skeletonLT E n
  continuousOn_toPartialEquiv_symm : ContinuousOn toPartialEquiv.symm E
  image_topPartialEquiv_symm_skeletonLT_subset' (n : ℕ) :
    toPartialEquiv.symm '' (skeletonLT E n) ⊆ skeletonLT C n
  source_eq : toPartialEquiv.source = C
  target_eq : toPartialEquiv.target = E
```

Mention cellular approximation here?

4 Finiteness notions

Should this be a subsection in the definition section instead?

There are three important finiteness notions on CW complexes. We say that a CW complex is *of finite type* if there are only finitely many cells in each dimension. We call it *finite dimensional* if there is an n such that the complex equals its n-skeleton. Finally, it is said to be *finite* is it is both finite dimensional and of finite type. In Lean, these definitions take the following form:

```
class RelCWComplex.FiniteDimensional.{u} {X : Type u} [TopologicalSpace X]
```

```
    (C : Set X) {D : Set X} [RelCWComplex C D] : Prop where
  eventually_isEmpty_cell : ∀ᶠ n in Filter.atTop, IsEmpty (cell C n)

class RelCWComplex.FiniteType.{u} {X : Type u} [TopologicalSpace X] (C : Set X)
    {D : Set X} [RelCWComplex C D] : Prop where
  finite_cell (n : ℕ) : Finite (cell C n)

class RelCWComplex.Finite {X : Type*} [TopologicalSpace X] (C : Set X)
    {D : Set X} [RelCWComplex C D] extends FiniteDimensional C, FiniteType C
```

Here, "`∀ᶠ n in Filter.atTop, IsEmpty (cell C n)`" uses *filters* to state that, eventually, for large enough `n` all types `cell C n` are empty. Filters are used extensively throughout `Mathlib`. More on filters and their use in `Mathlib` can be found in ???.

Cite something

When defining a CW complex of finite type, we can add a condition stating that the type of cells in each dimension is finite and relax the condition `mapsTo` of Figure 1 to be

```
mapsTo : ∀ (n : ℕ) (i : cell n), MapsTo (map n i) (sphere 0 1) (D ∪ ⋃ (m < n)
    (j : cell m), map m j '' closedBall 0 1)
```

When constructing a finite CW complex, we can again add conditions stating that the type of cells in each dimension is finite and that starting at a large enough dimension it is empty. In exchange, we can drop the condition `closed'` of Figure 1 and modify the condition `mapsTo` in the way described above. We provide constructors for both of these situations.

We then show that a CW complex is finite if and only if it is compact and that a compact subset of a CW complex is contained in a finite subcomplex.

5 Basic constructions

I am not sure if this section should even exist. But I could briefly talk about:

(i) attaching cells

(ii) disjoint unions?

(iii) transporting along partial homeomorphisms?

We have formalized a handful of miscellaneous constructions that we present in this section.

Firstly, we provide a way to get CW complex instances for non-definitional equalities. In Lean, this is

```
def RelCWComplex.ofEq {X : Type*} [TopologicalSpace X] (C D : Set X)
    {E F : Set X} [RelCWComplex C D] (hCE : C = E) (hDF : D = F) :
    RelCWComplex E F :=
  /- Proof omitted-/
```

We also want to be able to transport CW-structures along homeomorphisms. Because of our local approach, the more general statement features "local homeomorphisms":

```
def RelCWComplex.ofPartialEquiv.{u} {X Y : Type u} [TopologicalSpace X]
```

```
    [T2Space X] [TopologicalSpace Y] (C : Set X) {D : Set X} (E : Set Y)
    {F : Set Y} [RelCWComplex C D] (hE : IsClosed E) (f : PartialEquiv X Y)
    (hfC1 : f.source = C) (hfE1 : f.target = E) (hDF : f '' D = F)
    (hfC2 : ContinuousOn f C) (hfE2 : ContinuousOn f.symm E) :
    RelCWComplex E F :=
  /- Proof omitted-/
```

We can then easily derive a version for the general "total" homeomorphisms.

Next, we show that the disjoint union of two CW complexes is again a CW complex. Here, we again simplify the problem by considering both CW complexes as subsets of the same topological space. The sufficient conditions for this to work are that the complexes are disjoint and their bases are separated by neighbourhoods. This translates to:

```
def RelCWComplex.disjointUnion [RelCWComplex C D] {E F : Set X}
    [RelCWComplex E F] (hCE : Disjoint C E) (hDF : SeparatedNhds D F) :
    RelCWComplex (C ∪ E) (D ∪ F) :=
  /- Proof omitted-/
```

Here, I would like to say something about this not being an instance and us therefore providing one but I have not managed to define this instance :(

We provide definitions that allow you to attach one cell at a time or attach a collection of cells of the same dimension at once. These are in convenient when constructing examples of CW complexes. Lastly, we make it possible to restrict and enlarge the ambient type around a CW complex.

6 Products

In general, the product of two CW complexes is not necessarily a CW complex because the weak topology of the CW complex might not match the product topology. A counterexample was first provided by Dowker in [Dow52].

In order to achieve the correct topology on the product we need to turn it into a compactly generated space which we will discuss in the next subsection.

6.1 Compactly generated spaces

The name "*compactly generated space*" (or sometimes "*k-space*") is used for different notions in the literature. Firstly, it can refer to a space with a topology that is coherent with its compact subsets, i.e. a set is closed if and only if its intersection with every compact subset is closed in that subset. Secondly, it can refer to a space with a topology determined by continuous maps from compact Hausdorff spaces, i.e. a set is closed if and only if its preimage under every continuous map from a compact Hausdorff space is closed. Thirdly, it can refer to a space with a topology coherent with its compact Hausdorff subspaces, i.e. a set is closed if and only if its intersection with every compact Hausdorff subspace is closed in that subspace. We believe that the classification into these three notions was first done by Wikipedia contributors in [Wik25].

While these three notions agree for Hausdorff spaces, in the general case, the first is the weakest and the third the strongest. When starting this formalization, the second version was already in `Mathlib` as `CompactlyGeneratedSpace`; the two other versions had not been formalized. We intended to follow the construction of the product presented in [Hat02] which uses the first version of compactly generated spaces. Since we assume our ambient space to be Hausdorff, we could have just translated the proof to use the version already in `Mathlib`. Instead, we decided to formalize the first version and named it `CompactlyCoherentSpace`. This name was suggested by Steven Clontz. We will also refer to the mathematical notion as a *compactly coherent space* going forward.

In `Mathlib`, the definition is the following:

```
class CompactlyCoherentSpace (X : Type*) [TopologicalSpace X] : Prop where
  isCoherentWith : IsCoherentWith (X := X) {K | IsCompact K}
```

which uses the already pre-existing structure `IsCoherentWith` that is defined as:

```
structure IsCoherentWith (S : Set (Set X)) : Prop where
  isOpen_of_forall_induced (u : Set X) :
    (∀ s ∈ S, IsOpen ((↑) ⁻¹' u : Set s)) → IsOpen u
```

Credit author of IsCoherentWith?

Here, `↑ : Set s → Set X` is the natural inclusion from `Set s` into `Set X` where `s : Set X`. Thus the condition `isOpen_of_forall_induced` states that for all subsets `u` of `X` and all elements `s` of the collections of subsets `S`, if the preimage of `u` under `↑ : Set s → Set X`, i.e. the intersection of `u` with `s`, is open in `s`, then `u` is open in `X`.

We first show that this definition is equivalent to the one characterizing closedness which we stated at the beginning of the subsection. `Mathlib` already had the proofs for two examples of compactly coherent spaces: sequential spaces (which include metric spaces) and weakly locally compact spaces. Lastly, we show that `CompactlyCoherentSpace` is a weaker notion of `CompactlyGeneratedSpace` but that the two agree assuming the space is Hausdorff.

Anatole Dedecker refactored a lot of this section (k-ification) for me in the review process. How do I credit that?

Next, we want to provide a way to turn any topological space into a compactly coherent space. This operation is typically referred to as *k-ification*. We will call it *compact coherentification* corresponding to our naming of compactly coherent spaces. Since we will be considering two different topologies on the same type, we need to define a type synonym in order for Lean to recognize which topology we are talking about. We set `def CompactCoherentification (X : Type*) := X` and abbreviate it to `k X`. This means that `X` and `k X` are definitionally equivalent but this equality should not be abused. Instead one should move between these topologies using a bijection:

Write why above? Cite something?

```
protected def mk (X : Type*) : X ≃ CompactCoherentification X := Equiv.refl _
```

Now, we can provide a topology on `k X` in the following way:

```
instance instTopologicalSpace : TopologicalSpace (k X) :=
  .coinduced (.mk X) (⨆ (K : Set X) (_ : IsCompact K), .coinduced (↑)
    (inferInstanceAs <| TopologicalSpace K))
```

Where we set our new topology two be coinduced by the disjoint union topology of all the compact subsets `K` of `X`. We prove that this definition implies that a set in our new topology is open if and only if its intersection with every compact set is open in the subspace topology of that compact set induced by the original topology. We show

I don't actually understand myself what is going on in this definition.

the equivalent statement for closed sets, prove that the new topology is finer than the original one and formalize conditions under which maps to, from and between compact coherentifications are continuous. Lastly, we show that the compact coherentification does indeed make an arbitrary topological space into a compactly coherent space.

6.2 Constructing the product

Give a fairly detailed mathematical proof of the product here (a little less detailed than in my thesis).

We want to use this subsection to not only discuss the implementations but also give a fairly detailed adaptation of the proof in [Hat02] to relative CW complexes. For the rest of the section let C be a CW complex with base D and E be a CW complex with base F. The respective families of characteristic maps are $(Q_i^n : D^n \to C)_{n\in\mathbb{N}, i\in I_n}$ and $(P_j^m : D^m \to E)_{m\in\mathbb{N}, j\in J_m}$. We will write the cells of C as e_i^n and the cells of E as f_j^m. The theorem we aim to prove is the following:

Theorem 3. *There is a CW structure on* $\mathrm{k}(C \times E)$ *with base* $(D \times E) \cup (C \times F)$ *and characteristic maps* $(Q_i^n \times P_j^m : D^n \times D^m \to \mathrm{k}(C \times E))_{n,m\in\mathbb{N}, i\in I_n, j\in J_m}$. *The indexing sets* $(K_l)_{l\in\mathbb{N}}$ *are given by* $K_l = \bigcup_{n+m=l} I_n \times J_m$ *for every* $l \in \mathbb{N}$ *and the cells are therefore of the form* $e_i^n \times f_j^m$ *for* $n, m \in \mathbb{N}$, $i \in I_n$ *and* $j \in J_m$.

In Lean, we make the following assumptions for the entire section:

```
variable {X : Type*} {Y : Type*} [TopologicalSpace X] [TopologicalSpace Y]
  {C D : Set X} {E F : Set Y}
```

The above theorem then translates to

```
instance RelCWComplex.ProductCompactCoherentification [RelCWComplex C D]
    [RelCWComplex E F] :
    RelCWComplex (X := k (X × Y)) (C ×ˢ E) (D ×ˢ E ∪ C ×ˢ F) :=
  sorry
```

where `sorry` is a way to state that we do not know the proof yet. We define the indexing type of the cells of the product as

```
structure RelCWComplex.prodCell (C : Set X) {D : Set X} (E : Set Y) {F : Set Y}
    [RelCWComplex C D] [RelCWComplex E F] (n : ℕ) where
  m : ℕ
  l : ℕ
  hml : m + l = n
  j : cell C m
  k : cell E l
```

and the characteristic maps as

```
def RelCWComplex.prodMap [RelCWComplex C D] [RelCWComplex E F] {n : ℕ}
    (e : prodCell C E n) : PartialEquiv (Fin n → ℝ) (X × Y) :=
  (prodIsometryEquiv e.hml).transPartialEquiv
  (PartialEquiv.prod (map e.m e.j) (map e.l e.k))
```

where `PartialEquiv.prod (map e.m e.j) (map e.l e.k)` is the product of the two relevant characteristic maps of `C` and `D` and `prodIsometryEquiv` is the natural isometric isomorphism between `Fin n → ℝ` and `(Fin m → ℝ) × (Fin l → ℝ)` when `m + l = n`.

We will focus on the two most important properties: weak topology and closure finiteness. First let us show that in the compact coherentification, the weak topology and the product topology agree.

Lemma 4. *$\mathrm{k}(C \times E)$ has weak topology, i.e. $A \subseteq \mathrm{k}(C \times E)$ is closed if $((D \times E) \cup (C \times F)) \cap A$ and $\overline{e}_i^n \times \overline{f}_j^m \cap A$ are closed for all $n, m \in \mathbb{N}$, $i \in I_n$ and $j \in J_m$.*

This proof should probably be a lot shorter

Proof. Let $A \subseteq C \times E$ be a set such that $((D \times E) \cup (C \times F)) \cap A$ and $\overline{e}_i^n \times \overline{f}_j^m \cap A$ are closed for all $n, m \in \mathbb{N}$, $i \in I_n$ and $j \in J_m$. We need to show that A is closed in $\mathrm{k}(C \times E)$. We know that the compact coherentification is a compactly coherent space and that A is closed if for every compact set $K \subseteq \mathrm{k}(C \times E)$, $A \cap K$ is closed in K. Take such a compact set K. The projections $\mathrm{pr}_1(K)$ and $\mathrm{pr}_2(K)$ are compact as images of a compact set. Thus, there are finite sets $G \subseteq \{e_i^n \mid n \in \mathbb{N}, i \in I_n\}$ and $H \subseteq \{f_j^m \mid m \in \mathbb{N}, j \in J_m\}$ such that $\mathrm{pr}_1(K) \subseteq D \cup \bigcup_{e \in G} e$ and $\mathrm{pr}_2(K) \subseteq F \cup \bigcup_{f \in H} f$. Then,

$$K \subseteq \mathrm{pr}_1(K) \times \mathrm{pr}_2(K) \subseteq \left(D \cup \bigcup_{e \in G} e\right) \times \left(F \cup \bigcup_{f \in H} f\right).$$

Claim 1: It suffices to show that $A \cap \left(D \cup \bigcup_{e \in G} e\right) \times \left(F \cup \bigcup_{f \in H} f\right)$ is closed.

Proof. Indeed, since

$$A \cap K \subseteq A \cap \left(D \cup \bigcup_{e \in G} e\right) \times \left(F \cup \bigcup_{f \in H} f\right),$$

we get

$$A \cap K = K \cap \left(A \cap \left(D \cup \bigcup_{e \in G} e\right) \times \left(F \cup \bigcup_{f \in H} f\right)\right)$$

which is closed as the intersection of two closed sets and therefore in particular closed in K.

Now observe that

$$\begin{aligned} A \cap &\left(D \cup \bigcup_{e \in G} e\right) \times \left(F \cup \bigcup_{f \in H} f\right) \\ &= \left(A \cap \left(D \times F \cup \bigcup_{e \in G} e \times F \cup \bigcup_{f \in H} D \times f\right)\right) \cup \left(A \cap \bigcup_{e \in G} \bigcup_{f \in G} e \times f\right). \end{aligned}$$

Claim 2: $A \cap \left(D \times F \cup \bigcup_{e \in G} e \times F \cup \bigcup_{f \in H} D \times f\right)$ is closed.

Proof. We see that

$$D \times F \cup \bigcup_{e \in G} e \times F \cup \bigcup_{f \in H} D \times f \subseteq (D \times E) \cup (C \times F)$$

and thus

$$\begin{aligned} A \cap & \left(D \times F \cup \bigcup_{e \in G} e \times F \cup \bigcup_{f \in H} D \times f\right) \\ &= (A \cap ((D \times E) \cup (C \times F))) \cap \left(D \times F \cup \bigcup_{e \in G} e \times F \cup \bigcup_{f \in H} D \times f\right) \end{aligned}$$

which is closed as the intersection of two closed sets.

Claim 3: $A \cap \bigcup_{e \in G} \bigcup_{f \in G} e \times f$ is closed.

Proof. Since

$$A \cap \bigcup_{e \in G} \bigcup_{f \in G} e \times f = \bigcup_{e \in G} \bigcup_{f \in G} A \cap e \times f,$$

this set is closed as a finite union of closed sets.

We see that the last two claims yield the desired result. □

Now let us move onto closure finiteness:

Lemma 5. $\mathrm{k}(C \times E)$ *has closure finiteness, i.e. each frontier of a cell is contained in the union of* $(D \times E) \cup (C \times F)$ *with a finite union of closed cells of a lower dimension.*

Proof. We consider a cell $e_i^n \times f_j^m$ for $n, m \in \mathbb{N}$, $i \in I_n$ and $j \in J_m$. First observe that its frontier is $\partial e_i^n \times \overline{f}_j^m \cup \overline{e}_i^n \times \partial f_j^m$. We can verify the claim separately for $\partial e_i^n \times \overline{f}_j^m$ and $\overline{e}_i^n \times \partial f_j^m$. As both proofs are analogous, we will only do the former. Since C fulfils closure finiteness, there is a finite set G of cells of C of dimension less than n such that $\partial e_i^n \subseteq D \cup \bigcup_{e \in G} \overline{e}$. This gives us

$$\partial e_i^n \times \overline{f}_j^m \subseteq \left(D \cup \bigcup_{e \in G} \overline{e}\right) \times \overline{f}_j^m = D \times \overline{f}_j^m \cup \bigcup_{e \in G} \overline{e} \times \overline{f}_j^m \subseteq ((D \times E) \cup (C \times F)) \cup \bigcup_{e \in G} \overline{e} \times \overline{f}_j^m,$$

which is the union of the base with a finite union of closed cells of $\mathrm{k}(X \times Y)$ of dimension less than $n + m$. □

The rest of the proof of Theorem 3 is straightforward.
We can then derive an instance for absolute CW complexes

```
instance CWComplex.ProductCompactCoherentification [CWComplex C] [CWComplex E] :
    CWComplex (X := k (X × Y)) (C ×ˢ E) :=
  (RelCWComplex.ofEq (X := k (X × Y)) (C ×ˢ E) (∅ ×ˢ E ∪ C ×ˢ ∅) rfl
    (by simp)).toCWComplex
```

and prove finiteness properties about these instances.

7 Examples

Should I talk about examples? I think the spheres would be nice. But the code is far from being polished. . .

8 Conclusion

Write what an impact this has made (?). Describe further possible research (Celluar approximation theorem, cellular homology?). I think Floris was going to do this?

References

[Baa25] Anne Baanen. “Use and Abuse of Instance Parameters in the Lean Mathematical Library”. In: *Journal of Automated Reasoning* 69.1 (2025), pp. 1–30.

[Dev25] The Lean Developers. *The Lean Language Reference*. Accessed: 27.06.2025. 2025. URL: `https://lean-lang.org/doc/reference`.

[Dow52] Clifford Hugh Dowker. “Topology of Metric Complexes”. In: *American journal of mathematics* 74.3 (1952), pp. 555–577. ISSN: 0002-9327.

[Gon+13] Georges Gonthier et al. “A Machine-Checked Proof of the Odd Order Theorem”. In: *Interactive Theorem Proving*. Ed. by Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 163–179. ISBN: 978-3-642-39634-2.

[Hat02] Allen Hatcher. *Algebraic Topology*. Algebraic Topology. Cambridge University Press, 2002. ISBN: 9780521795401.

[SUM20] Daniel Selsam, Sebastian Ullrich, and Leonardo de Moura. *Tabled Typeclass Resolution*. 2020. arXiv: `2001.04301 [cs.PL]`. URL: `https://arxiv.org/abs/2001.04301`.

[Whi18] John Henry Constantine Whitehead. “Combinatorial homotopy. I”. In: *Bulletin (new series) of the American Mathematical Society* 55.3 (2018), pp. 213–245. ISSN: 0273-0979.

[Wik25] Wikipedia contributors. *Compactly generated space — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Compactly_generated_space&oldid=1286715539`. [Online; accessed 15-August-2025]. 2025.