# Blah Blah

Blah Blah Blah
University of Bonn

27th June, 2025

## Contents

# 1 Introduction

This is the introduction. It could explain the following:

- Mathematical relevance of CW complexes
- What is Lean in 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. This is called *ad-hoc polymorphism*. For example, this means that there can be 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 assume that an instance exists for a declaration, for example assuming a topology on a type `X` by writing `[TopologicalSpace X]`. Instances can then be combined into new ones: `Mathlib` has an instance providing the subspace topology on a subtype of a type that has a `TopologicalSpace` instance and for two types with each a topology there is an instance providing the product topology. Additionally, instances can be used to construct another instance of a different typeclass: `Mathlib` for example provides an instance expressing that a uniform space is a topological space. This creates a complicated graph of typeclass instances that can then be searched by an algorithm to automatically infer instances. This *typeclass inference* ensures, for example, that when a type is assumed to be a metric space, lemmas about topological spaces can be used even though `Mathlib` does not have a declaration stating explicitly that a metric space has an associated topology. Instead, this fact is comprised of different instances that are automatically combined. More about typeclasses in Lean can be found in [SUM20].

This isn't really a good example, should I do Addition instead?

This feels way too long and clunky to read.

There are different ways to modify the behaviour of typeclass inference for specific typeclasses. One such way, which we make use of in the definition of CW complexes, is ascribing the property `outParam` to a parameter of the typeclass. An instance of a typeclass depends on its parameters: for `TopologicalSpace X` there is one parameter `X`; for `Membership α γ`, a typeclass stating that objects of type $\alpha$ can be considered as elements of objects of type $\gamma$, there are the two parameters $\alpha$ and $\gamma$. Looking for an instance when some of the parameters are not known would mean significant slowdowns and potentially unwanted results of the search, which is why Leans typeclass inference requires all parameters. However, sometimes this behaviour is not desired. As the typical examples for membership instances include things like `Membership α (List α)` and `Membership α (Set α)`, it makes sense to assume that the first parameter $\alpha$ can be inferred from the second or is at least uniquely determined by it. `Mathlib` therefore marks the first parameter as `outParam` enabling typeclass inference to run even when this parameter is not known. Labeling a parameter as `outParam` is an assurance to typeclass inference that for every combination of the parameters not marked as `outParam` there is at most one instance.

That means that typeclass inference stops searching for further matches when it finds one that matches the non-`outParam` parameters, even if it does not match the `outParam` parameter specified by the user. It is important to know that "matching a parameter" in this case means being definitionally equal. So even when the parameters are provably but not definitionally equal, Lean will treat corresponding instances as different.

Another technical detail of Lean that we will want to manipulate is reducibility. Definitions are not generally unfolded in Lean, meaning that Lean cannot use information about the components that make up an object. However, sometimes this behaviour would be desirable especially for processes like typeclass inference and definitional equality checks. To achieve this behaviour one can use the keyword `abbrev` instead of `def`.

Example?

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

## 2.2 Preliminary Mathematics in Lean

In `Mathlib`, a topological space is a type `X` together with a topology `TopologicalSpace X` on it. This then allows you to describe whether a set `A : Set X` is open or closed by writing `IsOpen A` and `IsClosed A`. A function `f : X → Y` between two topological spaces `X` and `Y` can be described as being continuous and as being continuous on a set `A : Set X` which is expressed by writing `Continuous f` and `ContinuousOn f A`. `Mathlib` also implements various separation axioms: to specify that a topological space `X` is Hausdorff one can write `T2Space X`.

A non-topological concept that we will need is `PartialEquiv` which is `Mathlib`'s version of a partial bijection. To define a `PartialEquiv X Y` for two types `X` and `Y` one needs to provide as data a total function on `X`, another total function on `Y` a set in `X` called the *source* and a set in `Y` called the *target*. Additionally, one needs to prove 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.

Explain CompletelyDistribLattice and its difference to CompleteDistribLattice

Explanation of PartialEquiv too detailed?

.

# 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. These images of closed discs in the CW complex are called *cells*. To specify that a cell is the image of an $n$-dimensional disc, one can call it 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.

The different 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 Whitehead's original definition in [Whi18]. This definition assumes the cells to all lie in one topological space and then describes how the cells interact with each other and the space. Secondly, there is a popular approach that is more categorical in nature. In this version the skeletons are regarded as different spaces and the definition describes how to construct the $(n+1)$-skeleton from 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 ??? , both are now formalized and part of `Mathlib`.

Current authorship is unclear to me

The definition chosen for formalization is the following:

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

(i) $Q_i^n|_{\mathrm{int}(D_i^n)} : \mathrm{int}(D_i^n) \to Q_i^n(\mathrm{int}(D_i^n))$ is a homeomorphism for every $n \in \mathbb{N}$ and $i \in I_n$. We call $e_i^n := Q_i^n(\mathrm{int}(D_i^n))$ an *(open) $n$-cell* and $\overline{e}_i^n := Q_i^n(D_i^n)$ 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* $\partial e_i^n := Q_i^n(\partial D_i^n)$ is contained in the union of $D$ with a finite number of closed cells of a lower dimension.

(v) A set $A \subseteq X$ is closed if $A \cap D$ and the intersections $A \cap \overline{e}_i^n$ are closed for all $n \in \mathbb{N}$ and $i \in I_n$.

(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.

I am not sure if I like the code being a float

One obvious change in the Lean definition is that instead of talking about the topological space $X$ being a CW complex, it talks about the set $C$ being a CW complex in the ambient space $X$. This eases working with constructions and examples of CW complexes. For constructions it allows you to avoid dealing with constructed topologies, for example the disjoint union topology, and for examples it allows you to use the possibly nicer topology of the ambient space that is often already naturally present. 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" 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. A short explanation typeclass inference can be found in section 2.1.

Is this too basic?

We omit the requirement for $X$ to be a Hausdorff space and instead naturally require it for most of the lemmata.

The base $D$ is an `outParam`. This is because lemma statements about CW complexes typically refer to just the underlying set `C` without mentioning the base `D`. Typically,

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
```

for typeclass inference to run the user would have to go out of their way to specify `D`. But this requirement of typeclass inference can be disabled by adding the `outParam` specification. The purpose and consequences of using `outParam` are discussed in more detail in Section 2.1. In particular this eases using typeclass inference but it can have unfortunate consequences when there are two CW complexes with the same underlying set but different bases.

While we do not expect there to be instances on the same set with a different base, we have encountered instances where the base is provably equal but not definitionally so. Typeclass inference can fail in these situations and the instances need to be provided manually. An example will be discussed later in ???.

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 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. Secondly, absolute CW complexes are precisely where we have encountered instances on the same set with probably but not definitionally equal base sets. Constructions for relative CW complexes, e.g. products, produce instances for which in the case of an absolute CW complex the base is provably equal to the empty set but not definitionally so. In that case 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 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. The latter cannot be an instance as this would create an instance loop. To avoid having duplicate notions `CWComplex.cell` and `RelCWComplex.cell` of the type of cells 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.

I think there was another reason?

Explain protected above?

Talk about the priority of the instance? I think Floris said no

Talk about the whole export mess?

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 unions of open cells of the complex.

```
structure Subcomplex (C : Set X) {D : Set X} [RelCWComplex C D] where
  carrier : Set X
  I : ∏ 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 union of open cells that is also a CW complex.

```
def RelCWComplex.Subcomplex.mk' [T2Space X] (C : Set X) {D : Set X}
    [RelCWComplex C D] (E : Set X) (I : ∏ 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

def RelCWComplex.Subcomplex.mk'' [T2Space X] (C : Set X) {D : Set X}
    [RelCWComplex C D] (E : Set X) (I : ∏ n, Set (cell C n)) [RelCWComplex E D]
    (union : D ∪ ⋃ (n : ℕ) (j : I n), openCell (C := C) n j = E) :
    Subcomplex C where
  carrier := E
  I := I
  closed' := isClosed
  union' := union
```

Is this helpful? mk' is used for skeleton

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 for an explanation of that structure.

Defining subcomplexes allows aus 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 that proof technique as easy as possible. Starting an induction at $-1$ is unfortunately not that convenient in Lean. For that 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-/)
```

then we can 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`. For an explanation on `abbrev` see Section 2.1. The definition `skeleton` exists mostly for completeness' sake. Both lemmas 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 seperate 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 are *cellular maps*. A cellular map is a continuous map between two CW complexes $X$ and $Y$ that send 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 equivalences*:

Is that the 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

talk about finiteness notions here, subsection in definition section?

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.{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)
  finite_cell (n : ℕ) : _root_.Finite (cell C n)
```

# 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?

# 6 Products

Give a fairly detailed mathematical proof of the product here (a little less detailed than in my thesis). I should probably start with a subsection on kspaces.

# 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?).

# References


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

[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.

[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] J. H. C. Whitehead. "Combinatorial homotopy. I". In: *Bulletin (new series) of the American Mathematical Society* 55.3 (2018), pp. 213–245. ISSN: 0273-0979.