Formalizing Abstract Simplicial Complexes & Stellar Subdivisions in Lean

Garett Cunningham
University of Connecticut, Storrs, CT, USA

Daniel Zach
Universität Regensburg, Germany

Stefan Friedl
Universität Regensburg, Germany

Abstract

The theory of simplicial complexes is a cornerstone of topology, offering a sophisticated tool for computing invariants. We present a formalization of abstract simplicial complexes and stellar subdivisions in the Lean proof assistant. We adopt a purely combinatorial framework in order to provide a cohesive foundation for studying the theory of stellar subdivisions as seen in many contexts of combinatorial topology. In particular, we provide formalizations of morphisms between abstract simplicial complexes; several crucial constructions and operations on complexes, such as links and joins; and perform a comprehensive study of how stellar subdivisions interact with these operations. We state and prove a number of identities commonly used in the study of triangulated manifolds, such as deriving equivalences between links in an abstract simplicial complex K and in a stellar subdivision $\sigma_s K$, including results with no references in the standard literature. To our knowledge, this is the first formalization of stellar subdivisions in any proof assistant.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Type theory

Keywords and phrases Lean, mathlib, simplicial complex, stellar subdivision, combinatorial topology

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.21

Supplementary Material *Software (Source Code)*: `https://github.com/not-gary/pachner` archived at `swh:1:dir:0fb08ba626d8ef0048c8f25ee8c9d40ba60a5ff4`

Funding The authors acknowledge support from the German-U.S. Fulbright Commission and CRC 1085 "Higher Invariants" funded by the SFB.

Declaration about GenAI/LLM use All code associated with this work was developed without the aid of generative AI tools.

1 Introduction

The theory of simplicial complexes finds its roots in some of the cornerstone results of premodern geometry, beginning with the study of polyhedra in 3-dimensional Euclidean geometry. Perhaps the first topological result is the Euler characteristic and its independence from the choice of triangulation for a surface. The notion of a simplicial complex generalizes and subsumes this study, having grown to become an indispensable tool for algebraic topology, largely due to its quality of being especially amenable to computational techniques. The result is an array of methods for defining and computing topological invariants, such as simplicial homology and genus in the most elementary cases. Moreover, simplicial structure can be heavily leveraged to study the homotopy type of a space by gaining access to discretized versions of standard tools – for example, discrete Morse theory and simplicial collapse algorithms [21, 22]. Likewise, a specialization of spectral sequences computes higher homotopy groups in the simply connected case [35], as implemented in Kenzo [36].

Classically, the field of combinatorial topology has made a rigorous program of extending and refining this toolkit. The resulting efforts have received wide applications, especially in the modern era of computers. Topological data analysis often leverages representations of data as simplicial complexes to compute useful metrics or invariants. For example, persistent homology [11] provides information regarding the structure of a dataset that is independent of certain biases in sampling. The software package Regina [6] is designed for explicit computations in low-dimensional topology based on a triangulation of a space (e.g., a 3-manifold or knot), allowing preprocessing by simplifying a triangulation.

Concretely, this is done using *bistellar* and *stellar subdivisions*. These have the advantage of being especially pleasant for algorithms or combinatorial description. Stellar subdivisions have classically been a common choice for a "standard" subdivision [14, 20, 31, 34]. As such, they have become an essential tool in combinatorial topology – for example, in the classical proof of Newman's theorem [9]. Theorems of Alexander [1] and Pachner [29] show that (triangulations of) manifolds are homeomorphic if and only if they are related by a sequence of stellar or bistellar moves, respectively. A direct combinatorial proof of the equivalence was given in a thesis of Hannes [13].

The traditional view defines simplicial complexes as structures embedded in Euclidean space, $\mathbb{R}^n$, via barycentric coordinates. This has the immediate benefit of strengthening ties to topological spaces. Moreover, this construction leads to a straightforward theory of general subdivisions, of which stellar subdivisions are a core representative. Indeed, to study general subdivisions, it suffices to study stellar subdivisions [14, 20]. However, reliance on the ambient space adds additional data that makes computations increasingly untenable. As a result, one wishes to study the more abstract, combinatorial properties of simplicial complexes free of an ambient space, often called an *abstract simplicial complex*. With this view, stellar subdivisions are equally easy (if not easier) to study.

Abstract simplicial complexes have received attention in other proof assistants, with existing formalizations in Rocq [19] and ACL2 [4]. These projects were directed toward explicit computational applications – persistent homology in the former case [16, 17] and Alexander duals in the latter [3]. We highlight this as a rather strange gap in Lean's development, given the long-standing interest in other communities. Indeed, `mathlib` [39] contains many formalizations that relate to simplicial complexes, but with no interconnection between them. We discuss how this motivated our choices in design throughout.[1]

The combinatorial structure lends itself as a natural candidate for formalization. However, the highly geometric nature of the subject leads to much of the canonical literature appealing to intuition-focused arguments. One is given trust that the computations can be done and will succeed if checked. This has the pedagogical advantage of maintaining clarity of ideas, but leads to obvious gaps in logic. Certain identities with highly technical proofs may go increasingly unchecked by future generations, shifting toward the status of folklore instead of theorem. We see this work as a step toward cataloging results in a centralized and formally verified manner so that the details have been sufficiently worked out for the reader. Additionally, the task of patching published proofs led to new theorems that have, to our knowledge, no existing references in the standard literature.

[1] Since the submission of this manuscript, `mathlib` accepted independent implementations of `PreAbstractSimplicialComplex` and `AbstractSimplicialComplex`. The former is identical to our formalization presented here and the latter includes an additional totality axiom. We refer the reader to Remark 4 and Section 4 for a more thorough comparison to our work presented here.

Given the ubiquity of abstract simplicial complexes for computational applications, we present initial legwork toward formalizing them in Lean 4 [27] – in particular, some foundational theory regarding the combinatorial properties, constructions of subcomplexes, and initial work on stellar subdivisions. Our selection of theorems for formalization are informed by future work to formalize combinatorial manifolds, in particular toward a future formalization of Alexander and Pachners' theorems. Our contributions are as follows:

- A formalization of abstract simplicial complexes; including morphisms, plus certain operations and constructions thereon.
- A formalization of stellar subdivisions, including identities relating how they interact with objects defined above.
- Formal proofs of new theorems that, as far as the authors are aware, are not found in the standard literature.

To our knowledge this is the first formalization of stellar subdivisions in any proof assistant. Throughout, we emphasize a design philosophy that prioritizes preserving computability and compatibility with existing `mathlib` content as far as reasonably possible. The former is inspired by our interests in simplicial complexes as a computational tool with direct algorithmic applications. We detour to clarify the latter and preview some of the main issues faced in development. The guiding principle is to choose the fewest assumptions necessary, so that later efforts to connect components in `mathlib` may be as streamlined as possible.

Some minor attention has been given to the direct study of simplicial complexes via `SimplicialComplex`. However, its state of development is minimal. We chose to isolate the combinatorial aspects of `SimplicialComplex` as `AbstractSimplicialComplex`. As our goals are purely combinatorial in nature, this decision removed many roadblocks and considerably reduced the overhead in formalization. The existing `mathlib` definition is as follows:[2]

```
variable (𝕜 E) [Ring 𝕜] [PartialOrder 𝕜] [AddCommGroup E] [Module 𝕜 E]
structure SimplicialComplex where
  faces : Set (Finset E)
  empty_notMem : ∅ ∉ faces
  indep : ∀ {s}, s ∈ faces → AffineIndependent 𝕜 ((↑) : s → E)
  down_closed : ∀ {s t}, s ∈ faces → t ⊆ s → t ≠ ∅ → t ∈ faces
  inter_subset_convexHull : ∀ {s t}, s ∈ faces → t ∈ faces →
    convexHull 𝕜 ↑s ∩ convexHull 𝕜 ↑t ⊆ convexHull 𝕜 (s ∩ t : Set E)
```

Notably, the definition depends on the types E and $\mathbb{k}$, where E is a module over the partially ordered ring $\mathbb{k}$. This gives a minimal setting needed to define a simplicial complex as existing in an ambient $\mathbb{k}$-affine space. Formally, a simplicial complex is defined to be the convex hulls of its faces glued together nicely (`inter_subset_convexHull`) with no degeneracies (`indep`). For example, by setting $\mathbb{k} := \mathbb{R}$ and $E := \mathbb{R}^n$, one recovers the usual embedding in n-dimensional Euclidean space. We chose to drop these geometric conditions and highlight where potential overhead is generated as relevant, with a full discussion reserved for Section 4.

One final thematic difficulty is the dependence of definitions on some type. In particular, a simplicial complex (whether abstract or geometric) implicitly comes with the attached data of a labeling of its vertices. This is, primarily, a fragment of how Lean chooses to define sets as formulas $E \to$ `Prop` for some type E. Therefore, we pay particular attention to how this data is affected by operations and maps, leading to a necessary theory of proper typecasting between simplicial complexes. Joins become the first roadblock motivating this level of care.

[2] `mathlib` has since changed the downward closure condition to use `IsRelLowerSet`. We plan to refactor our code to match this in a future update.

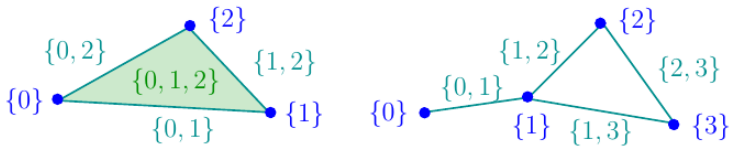


Figure 1 Examples of simplicial complexes with set-based encoding shown.

Throughout this paper, we do not assume familiarity with simplicial complexes or their applications. Indeed, we begin with the basic definition and work from the ground up. We refer the intrigued reader to some of the many excellent introductory resources on algebraic topology [12, 28, 37, 38] and combinatorial topology [14, 20, 21, 31] for a deeper study. Definitions are presented in their Lean form and theorems are presented in their natural language form. All claims are formalized with the respective Lean names attached. Our code is publically available via our GitHub repository,[3] currently totaling ~14,000 lines of code.

Section 2 presents a zoo of definitions and theorems describing the properties of abstract simplicial complexes and simplicial maps, as well as important subcomplexes for consideration; such as links, joins, and boundaries. Section 3 extends the discussion to stellar subdivisions. In particular, we focus discussion on formalizing the notion of *stellar equivalence* of simplicial complexes and prove a series of identities primarily describing the interaction of stellar subdivisions with the subcomplexes defined in Section 2. Throughout, we take cursory detours to highlight design choices and complications encountered during development. Section 4 is dedicated to a more thorough discussion thereof, with a view toward potential future work and an analysis of related contemporary work.

2 Simplicial Complexes

The notion of simplicial complexes can be seen as stemming from the process of triangulating topological spaces. Informally, a simplicial complex is composed of n-dimensional faces (possibly none), inductively closed under its $(n-1)$-dimensional edges for all $n \geq 1$. A point, or 0-face, is called a *vertex*. One can view simplicial complexes as a higher dimensional analog of graphs or polyhedra, which are special cases. In this more general setting, we are allowed to glue pieces of differing dimensions together. Figure 1 shows a visual example.

One method to encode the downward closure property is to label the vertices of a simplicial complex. An n-dimensional face is then defined as the set of $n+1$ vertices that it contains (cf. Fig. 1). In this manner, a simplicial complex is formally a collection of finite sets (faces) that is closed under the subset relation. To distinguish this from the geometric formulation, we call this an *abstract* simplicial complex.

```
structure AbstractSimplicialComplex (E : Type _) where
  faces : Set (Finset E)
  empty_notMem : ∅ ∉ faces
  down_closed : ∀ {s t}, s ∈ faces → t ⊆ s → t ≠ ∅ → t ∈ faces
```

In the sequel, we drop the adjective "abstract," so long as the context is clear, and use $s \in K$ to mean "s is a face of K". We use $V(K)$ to denote the set of vertices of K. One may define this in a few equivalent ways.

[3] https://github.com/not-gary/pachner.

▶ **Definition 1** (`AbstractSimplicialComplex.vertices`).

$V(K) := \{x \mid \{x\} \in K\}$.

▶ **Proposition 2** (`vertices_eq`, `vertices_setOf`).

$V(K) = \bigcup_{s \in K} s = \{x \mid \exists s \in K,\ x \in s\}$.

In our definition of `AbstractSimplicialComplex`, we retain the axioms of `SimplicialComplex` that relate to the combinatorial data, namely `face`, `empty_notMem` and `down_closed`, and cast out the geometric conditions `indep` and `inter_subset_convexHull`. As a consequence, we no longer need the additional type $\mathbb{k}$ and assumptions in the type signature. Therefore, we define an `AbstractSimplicialComplex` for just any base type E. If some ring $\mathbb{k}$ is provided such that E is a module over $\mathbb{k}$, then one can retrieve an instance of `SimplicialComplex`.

```
variable (𝕜 E) [Ring 𝕜] [PartialOrder 𝕜] [AddCommGroup E] [Module 𝕜 E]
def AbstractSimplicialComplex.ofGeometric (K : SimplicialComplex 𝕜 E) :
  AbstractSimplicialComplex E := ⟨K.faces, K.empty_notMem, K.down_closed⟩

def SimplicialComplex.ofAbstract
    (K : AbstractSimplicialComplex E)
    (indep : ∀ {s}, s ∈ K.faces → AffineIndependent 𝕜 ((↑) : s → E))
    (inter_subset_convexHull : ∀ {s t}, s ∈ K.faces → t ∈ K.faces →
        convexHull 𝕜 ↑s ∩ convexHull 𝕜 ↑t ⊆ convexHull 𝕜 (s ∩ t : Set E)) :
  SimplicialComplex 𝕜 E :=
    ⟨K.faces, K.empty_notMem, indep, K.down_closed, inter_subset_convexHull⟩
```

Hence, any results that hold for `AbstractSimplicialComplex` will lift to `SimplicialComplex`. Converting back and forth yields the original complex (e.g., `ofAbstract_ofGeometric_id`).

We carry over the same constructions and proofs from `SimplicialComplex` as applicable. However, there are a large number of basic notions missing that we develop. First, `mathlib` lacks definitions for unions and intersections of complexes.

```
def SimplicialUnion (K L : AbstractSimplicialComplex E) :=
  ⟨K.faces ∪ L.faces, _, _⟩
def SimplicialInter (K L : AbstractSimplicialComplex E) :=
  ⟨K.faces ∩ L.faces, _, _⟩
```

This induces instances of `instHasUnion` and `instHasInter` respectively to take advantage of the usual set-theoretic notation for unions and intersections.

Second, we define a notion of *dimension* for both a face (via its cardinality) and, by extension, a simplicial complex.

```
def face_dim (s : Finset E) : ℤ := Finset.card s - 1
def AbstractSimplicialComplex.dim
    (K : AbstractSimplicialComplex E) [Fintype K.faces] : ℤ :=
  Finset.max' ((Finset.image face_dim K.faces.toFinset) ∪ {-1}) _
```

We highlight two qualities of this definition. First, the usage of `Finset.max'` returns an integer rather than an element of type `Option` $\mathbb{Z}$, so long as the set is nonempty. We assign the empty complex $\bot$ dimension -1 as convention. Second, this requires that K be finite in order to extract a `Finset` for calculation. In practice, one is often concerned with finite (i.e., compact) simplicial complexes and explicit calculations with them. We found this useful for some dimension preservation results.

2.1 Simplicial Maps

We first define a *simplicial map* from K to L as a function $f : E \to F$ such that every face in K is mapped to a face in L:

```
structure SimplicialMap
    (K : AbstractSimplicialComplex E) (L : AbstractSimplicialComplex F)
  where
    map : E → F
    is_simplicial : ∀ s, s ∈ K.faces → Finset.image map s ∈ L.faces
```

In particular, all values of `map` outside the faces of K are treated as "junk values", since `is_simplicial` imposes no condition on them. Note that the property of being a simplicial map depends on the complexes used for the domain and codomain. However, there are circumstances in which a map stays simplicial after we change either parameter. For example, restriction to a subcomplex yields a simplicial map (`SimplicialMap.restrict`). Moreover, any function applied to a simplicial complex is simplicial onto its image, including restricting the codomain to the image (`SimplicialImage`, `SimplicialMap.ontoImage`). We informally denote the image of a simplicial map f on K by $f(K)$. Simplicial maps are also closed under composition. That is, if $f : E \to F$ is simplicial from K to L and $g : F \to G$ is simplicial from L to Z, then $g \circ f : E \to G$ is a simplicial map from K to Z (`SimplicialMap.comp`).

▶ Remark 3. This is sufficient to define the category of abstract simplicial complexes with morphisms given by simplicial maps. Indeed, the identity map $E \to E$ is always a simplicial map (`idSimplicialMap`). We mark this as useful future work, wherein a subset of the authors have initiated active plans toward further development. Moreover, we note challenges to deriving an equivalent definition for `SimplicialComplex` in Section 4.

2.1.1 Simplicial Isomorphisms

Geometrically, an isomorphism of simplicial complexes, $K \cong L$, will preserve the simplicial structure, but potentially relabel the vertices. Formally stated:

```
def IsSimplicialIso
    {K : AbstractSimplicialComplex E} {L : AbstractSimplicialComplex F}
    (f : SimplicialMap K L) : Prop :=
  ∃ g : SimplicialMap L K,
    (K.vertices).restrict (g ∘ f).map = (K.vertices).restrict id ∧
    (L.vertices).restrict (f ∘ g).map = (L.vertices).restrict id
```

The above definition is propositional and is agnostic to the particular choice of inverse $g : F \to E$. We also supply a bundled version that includes a specific choice of inverse simplicial map with proofs that they are mutually inverses on the vertex sets.

```
structure SimplicialIso
    (K : AbstractSimplicialComplex E) (L : AbstractSimplicialComplex F) where
  toFun : SimplicialMap K L
  invFun : SimplicialMap L K
  left_inv : ∀ {x : E}, x ∈ K.vertices → invFun.map (toFun.map x) = x
  right_inv : ∀ {x : F}, x ∈ L.vertices → toFun.map (invFun.map x) = x
```

In line with Remark 3, `CategoryTheory.Iso` uses bundled versions of isomorphisms. We provide both definitions to allow some freedom of choice for users.

▶ Remark 4. Note that we only stipulate the bijectivity condition for the vertices of the simplicial complex. What the actual maps f and g do outside the set of vertices does not matter. One might define $f : E \to F$ to be a bijection of types. However, there arise examples where constructing an explicit inverse is tricky. For example, consider $E = F = \mathbb{N}$. Take $K := \mathcal{P}(\{1, \ldots, n\}) \setminus \{\varnothing\}$ and $f(x) := x \dot{-} 1$, denoting the monus operation. Then, $f(K) = \mathcal{P}(\{0, \ldots, n-1\}) \setminus \{\varnothing\} \cong K$. However, the monus operation is not a bijection, since

$$0 \dot{-} 1 = 1 \dot{-} 1 = 0.$$

Restricting to the set of vertices is easier in practice, since one only needs to care about a function's behavior localized to the set of vertices.

Simplicial complexes are also sometimes defined with an additional axiom stipulating that they are *total*. In this context, one would add the condition

```
total : ∀ x : E, {x} ∈ K.faces
```

This avoids the above pathological example, but introduces a much more severe complication for subdivisions. Stellar subdivisions, for example, introduce a new vertex. If K is total, then one either needs to change the underlying type to add a new label for this vertex, or shift the labels to make room for it. In either case, a monumental amount of overhead is generated to handle typecasting. In particular, many identities will only hold up to isomorphism, whereas our current formalization gives true equalities.

2.1.2 Simplicial Coercions

Isomorphisms give a useful tool for attempting to change the underlying type of a simplicial complex without loss of data. However, the definition relies on specifying a simplicial complex as the codomain and an inverse morphism. For practical use cases, it is more convenient to just provide a well-behaved function $\varphi : E \to F$ for typecasting. Moreover, one desires that this typecasting is lossless, essentially behaving as a relabeling of the vertices.

```
structure SimplicialCoe (K : AbstractSimplicialComplex E) (F : Type _) where
  coe : E → F
  Injective : Set.InjOn coe (K.vertices)
```

The core idea is that we lift typecasting functions $\varphi : E \to F$ to simplicial maps. Note, φ need not be injective everywhere. To have that φ is an isomorphism of K onto its image, it suffices to know that φ is injective on the set of vertices of K.

Some operations on simplicial complexes will change the underlying type E. Thus, a reasonable method for typecasting will be indispensable for ensuring complexes have the "correct" type at any given time, particularly in the context of joins. In this instance, we find that having a specific construct for typecasting on simplicial complexes to be handy, since we can omit details about codomains and proofs that φ is simplicial. Rather, we can just supply a function that is injective on the vertex set.

This localized injectivity condition allows us to prove some useful utility lemmas. For example, simplicial isomorphisms are coercions (`SimplicialMap.coe`). Moreover, they are preserved under restriction to subcomplexes (`SimplicialCoe.restrict`) and composition with isomorphisms (`SimplicialCoeOnImage`). They also distribute over set operations like unions.

▶ **Proposition 5** (`simplicialCoe_union`). *Let $\varphi : E \to F$ be a simplicial coercion on $K \cup L$. Then, $\varphi(K \cup L) = \varphi(K) \cup \varphi(L)$.*

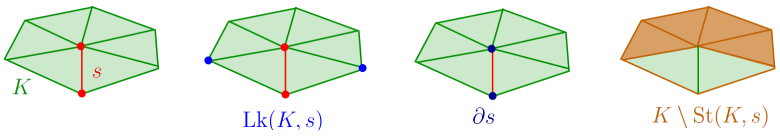


Figure 2 Examples of subcomplexes of K formed by operations on the face s. From left to right: (a) the link $\mathrm{Lk}(K, s)$, (b) the face boundary ∂s, (c) the star complement $K \setminus \mathrm{St}(K, s)$.

Most importantly, coercions yield nice interactions with the constructions in Sections 2.2 and 3, allowing us to prove useful identities to simplify typecasting steps in proofs. We present some of these utility lemmas as they naturally arise.

2.2 Subcomplexes & Operations

We explore some notable subcomplexes of and operations on a simplicial complex. For brevity, we will focus on those that will be essential for defining and studying stellar subdivisions in Section 3. A number of additional constructions; such as stars, cones, and balls around a face; are formalized, with definitions and some simple results available.

The first and foremost construction is the *link* of a face s:

```
def Link (K : AbstractSimplicialComplex E) (s : Finset E) :=
  ⟨{t ∈ K.faces | s ∪ t ∈ K.faces ∧ s ∩ t = ∅}, _, _⟩
```

Given the n-face s in K, the link $\mathrm{Lk}(K, s)$ is the collection of all faces in K that are disjoint from s and that form a face of K when combined with s. Figure 2a shows an example of a link. Pictorially, $\mathrm{Lk}(K, s)$ represents all the faces in K that "support" s via some "linking" face. They enjoy nice properties on their own, including interactions with isomorphims (`link_simplicialIso`) and set operations (`link_simplicialUnion`, `link_simplicialInter`). We may also derive some useful relationships between links of subfaces.

▶ **Proposition 6** (`link_of_face_complement`). *If $t \subseteq s$, then*

$$Lk(K, s) = Lk(Lk(K, t), s \setminus t).$$

The *boundary* of an n-face s, denoted ∂s, is the simplicial complex formed by all the $(n-1)$-faces of s. Figure 2b shows an example geometrically. Described formally,

```
def FaceBoundary (s : Finset E) := ⟨(Finset.powerset s \ {s, ∅}), _, _⟩
```

The boundary has useful qualities for studying individual faces and their subfaces. Results on monotonicity properties and simplicial maps are straightforward to obtain, such as preservation under isomorphisms and commuting with coercions.

▶ **Proposition 7** (`faceBoundary_mem_iff_subset`, `faceBoundary_subface_subcomplex`).

- *$t \in \partial s \iff t \subset s \wedge t \neq \emptyset$.*
- *If $s \subseteq t$, then $\partial s \subseteq \partial t$.*

▶ **Proposition 8** (`faceBoundary_simplicialIso`, `faceBoundary_simplicialCoe_image`).

- *If $s \cong t$, then $\partial s \cong \partial t$.*
- *If φ is injective on s, then $\partial(\varphi(s)) = \varphi(\partial s)$.*

Figure 3 Example of the join of two simplicial complexes, K and L.

The *star complement* of K with respect to s is usually defined as the complement of the interior of the *star* around s. Intuitively, the star around s represents a "neighborhood" with all faces that contain s. Thus, the star complement takes K and removes the "interior" of the star, denoted by $K \setminus \mathrm{St}(K, s)$. Figure 2c shows a visual example. For convenience, we can define the star complement directly as a set.

```
def StarComplement (K : AbstractSimplicialComplex E) (s : Finset E) :=
  ⟨{t ∈ K.faces | ¬s ⊆ t}, _, _⟩
```

Finally, we dedicate considerable attention to the *join* of two simplicial complexes. Geometrically, we consider the join of K and L, denoted $K \star L$, as the simplicial complex obtained by fully connecting the faces of K with those in L (cf. Figure 3). Described as a set, we consider all pairs $s \in K$ and $t \in L$, then take their disjoint unions. Hence, we first describe an appropriate notion of disjoint union.

While `mathlib` does include a definition of disjoint union for `Finset`, it requires a proof that s and t are disjoint, then returns $s \cup t$. However, it is reasonable to expect that faces of K and L are not always disjoint. We use the common trick of creating disjoint sets of pairs. Stated formally in Lean:

```
variable (𝕜 E) [DecidableEq 𝕜] [DecidableEq E]
variable [Semiring 𝕜] [Nontrivial 𝕜]
def FaceDisjointUnion (s t : Finset E) : Finset (E × 𝕜) :=
  s ×ˢ {0} ∪ t ×ˢ {1}
infixl:65 " ⊔ₛ " => FaceDisjointUnion

def SimplicialJoin (K L : AbstractSimplicialComplex E) :=
  ⟨{(s ⊔ₛ t) | (s ∈ K.faces ∪ {∅}) (t ∈ L.faces ∪ {∅})} \ {∅}, _, _⟩
```

▶ Remark 9. The definition deserves some explanation. First, the set description is designed to ensure that $s \sqcup_s \varnothing$ and $\varnothing \sqcup_s t$ are faces, so that K and L naturally embed into the join. Then, we remove $\varnothing \sqcup_s \varnothing = \varnothing$ to satisfy `empty_notMem`.

More striking is the choice of assumptions on $\Bbbk$. The basic function of $\Bbbk$ is for indexing which side of the disjoint union a face is in. If $\Bbbk$ is a nontrivial semiring, then there is a natural choice using 0 and 1 notation, mimicking the usual indexing by $\mathbb{N}$. This choice is designed with `SimplicialComplex` and `convexJoin` in mind, since if $\Bbbk$ is a ring and E is a $\Bbbk$-module, then $E \times \Bbbk$ will have a natural $\Bbbk$-module structure and use identical notation. Hence, this choice sets up future paths to lifting this definition to the geometric case.

One might alternatively choose to use the sum type $E \oplus E$. This has the advantage of more naturally capturing the disjoint union of types and directly encodes properties of the join like commutativity and associativity. For example, there is a natural equivalence $(E \oplus E) \oplus E \simeq E \oplus (E \oplus E)$. However, the sum type of $\Bbbk$-modules need not have a $\Bbbk$-module structure itself.

Another choice that does carry a natural $\Bbbk$-module structure is the direct sum of a module with itself, which is equivalent to $E \times E$. However, this presents an issue when attempting to capture disjoint unions. For example, consider $K \star K$. One might desire that each copy

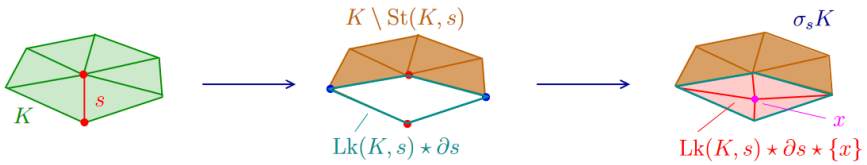


Figure 4 Example of a stellar subdivision, computed in steps. The interior of the star around s is removed and the barycenter x is placed inside, which is then fully connected to the complex via a join to $\mathrm{Lk}(K,s) \star \partial s$.

of K embeds into $E \times \{0\}$ and $\{0\} \times E$, respectively. However, if 0 is a vertex of K, then both inclusions give the point $(0,0)$ in $E \times E$ rather than distinguishing them as two distinct copies. Therefore, another trick is required to avoid such collisions. We find using $E \times \Bbbk$ to be a somewhat parsimonious solution, since it also carries a natural $\Bbbk$-module structure and captures the idea of disjoint unions using the standard trick of indexing by 0 and 1.

Joins are associative (`simplicialJoin_assoc`) and commutative (`simplicialJoin_comm`) up to isomorphism, and moreover preserve isomorphism type (`simplicialJoin_simplicialIso`). The empty complex also acts as the identity (e.g., `simplicialJoin_bot_iso_id`).

The join obeys a distributive law over unions (`simplicialJoin_simplicialUnion`) and intersections (`simplicialJoin_simplicialInter`). This is in fact true on the level of equality, not just up to isomorphism. These proofs make heavy use of the structure of faces under disjoint unions. Therefore, we provide numerous utility lemmas detailing properties about our definition of disjoint union above.

Since the complex $K \star L$ has the underlying type $E \times \Bbbk$, repeated applications of the join operation require typecasting $E \times \Bbbk \to E$. In practice, we can get away with simple maps that will completely cover our use cases in Section 3. For example, we defined disjoint unions in the manner above to handle the case where $V(K) \cap V(L) \neq \varnothing$. If they are disjoint, then the projection $p_1 : E \times \Bbbk \to E$ to the first coordinate will be injective on the vertex sets, hence can be used as a coercion. The faces then have nicer forms to work with, enabling much simpler proofs through some basic rewrites. For example,

▶ **Proposition 10** (`simplicialJoinProj_mem`, `simplicialJoinProj_mem_vertices`).
- $s \in p_1(K \star L) \iff \exists t \in K \cup \{\varnothing\},\ \exists u \in L \cup \{\varnothing\},\ s = t \cup u \wedge t \cup u \neq \varnothing$.
- $x \in V(p_1(K \star L)) \iff x \in V(K) \vee x \in V(L)$.

Finally, we observe that joins interact nicely with some of the other objects previously defined. For example, joins enjoy distributive properties with links and star complements. Note that these are true equalities.

▶ **Proposition 11** (`simplicialJoin_link`, `simplicialJoin_starComplement_left`).
- $Lk(K \star L, s \sqcup_s t) = Lk(K,s) \star Lk(L,t)$.
- $(K \setminus St(K,s)) \star L = (K \star L) \setminus St(K \star L, s \sqcup_s \varnothing)$.

3 Stellar Subdivisions

Stellar subdivisions subdivide a "neighborhood," or *star*, around a face in a manner that preserves the homeomorphism type. Following Figure 4, we excise out the interior of the star, introduce a new point x (called the *barycenter*), then use joins to connect the barycenter to the star's boundary and fill in the gaps. Informally stated:

$$\sigma_s K := K \setminus \mathrm{St}(K,s) \cup \mathrm{Lk}(K,s) \star \partial s \star \{x\}, \quad \text{where } x \notin V(K).$$

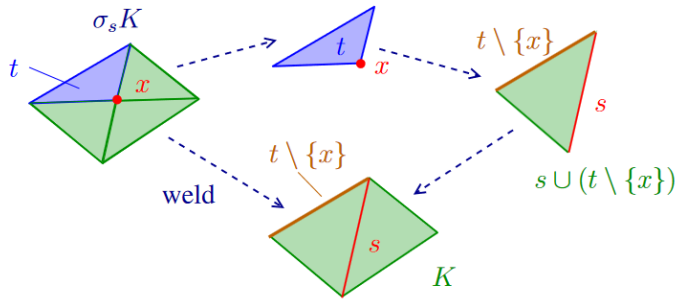


Figure 5 Example of a stellar weld, computing how the face $t \in \sigma_s K$ changes.

Note that we require iterated joins and a union for the definition. For the expression to typecheck in Lean, this means we must typecast $E \times \mathbb{k} \to E$. However, the components of the join are pairwise disjoint. Thus, we can apply p_1. Stated formally in Lean:

```
def StellarSubdivision (K : AbstractSimplicialComplex E)
    (s : Finset E) (s_in_K : s ∈ K.faces)
    (x : E) (x_nin_K : x ∉ K.vertices) :
  AbstractSimplicialComplex E :=
    ⟨K\St(K, s) ∪ p₁ ''ˢ (p₁ ''ˢ (Lk(K, s) ⋆ ∂s) ⋆ {x}), _, _⟩
```

We can leverage simple rewrite lemmas (as in Section 2.2) that allow us to act as if these typecasting steps never occurred. We call going from a stellar subdivision $\sigma_s K$ back to K a *stellar weld*. A *stellar move* is then a stellar subdivision, weld, or an isomorphism. One can concretely describe a stellar weld as a set of faces.

▶ **Proposition 12** (`stellarWeld_faces`).

$$\begin{aligned} K &= \{t \mid t \in \sigma_s K,\ x \notin t\} \cup \{s \cup (t \setminus \{x\}) \mid t \in \sigma_s K,\ x \in t\} \\ &= ((\sigma_s K) \setminus St(\sigma_s K, \{x\})) \cup \{s \cup (t \setminus \{x\}) \mid t \in \sigma_s K,\ x \in t\}. \end{aligned}$$

As far as the authors are aware, there is no standard reference for this description, so we provide the intuition behind the construction, with details formalized in our code.

Proof. There are two cases for any face $t \in \sigma_s K$. Either $x \notin t$, in which case $t \in K \setminus \mathrm{St}(K, s)$, or $x \in t$ and hence t is in the subdivided star. In the first case, $t \in K$, since $K \setminus \mathrm{St}(K, s)$ is a subcomplex of K. In the second case (cf. Fig. 5), we can excise the barycenter point x and glue the face s back in its place. Note that s is recovered by $t = \{x\}$. The second equality follows since $x \in t$ iff $\{x\} \subseteq t$. ◀

3.1 Stellar Equivalence

We can consider sequences of stellar moves and generate an equivalence relation. We say that K and L are *stellar equivalent* if there exists a finite sequence of stellar moves from K to L, denoted $K \cong_{\mathrm{st}} L$. Formally, this is the transitive closure of the stellar move relation.

```
def StellarEquiv :
  AbstractSimplicialComplex E → AbstractSimplicialComplex E → Prop :=
    Relation.ReflTransGen StellarMove
```

This has the advantage that proofs can be straightforwardly done via transitivity or induction. We moreover provide support for quality-of-life tactics, such as `calc`.

One challenge is to consider when typecasting from E to F preserves stellar equivalence. Suppose K, L are simplicial complexes over E and Z, W over F. If $K \cong Z$, $L \cong W$, and $K \cong_{\text{st}} L$, can we say $Z \cong_{\text{st}} W$? The answer is: not always. The reason is that one needs to derive a sequence of stellar moves from Z to W given those from K to L.

Recall that the definition of a stellar subdivision requires the existence of an unused value $x : E$ to act as the barycenter. Then, for example, if F is a finite type such that there are more values $x_1, \ldots, x_n : E$ used in the relation $K \cong_{\text{st}} L$ than available in F, it may be impossible to directly derive $Z \cong_{\text{st}} W$. Thus, any result showing that stellar equivalence is preserved after typecasting must have an additional assumption to ensure that this is done without losing information about the barycenters.

▶ **Theorem 13** (`stellarEquiv_iso`). *Let K, L be simplicial complexes over E and Z, W over F such that $K \cong Z$ and $L \cong W$. Let $f : E \to F$ be injective. Then,*

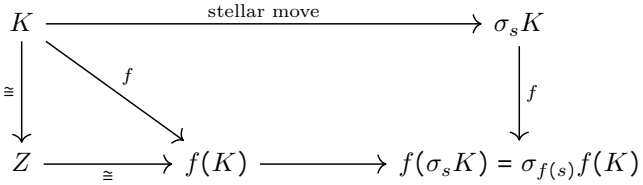
$$K \cong_{st} L \implies Z \cong_{st} W.$$

Equivalently, stipulating the existence of an injective $f : E \to F$ means that $|E| \leq |F|$ on the level of cardinalities. This is to ensure that there is always "room" in F to support any barycenters associated with the assumption $K \cong_{\text{st}} L$.

Proof. We desire that a sequence of stellar moves over type E induces a sequence of stellar moves over type F. If we show the case for a single stellar subdivision or stellar weld, the claim follows by induction. Therefore, it suffices to prove the following lemma. ◀

▶ **Lemma 14** (`stellarMove_exists_iso`). *Under the same assumptions, if $K \cong Z$ and there is a stellar move from K to L, then there exists W such that $L \cong W$ and $Z \cong_{st} W$.*

Proof. We sketch the argument in the stellar subdivision case. The stellar weld case is similar. The proof can be expressed as a diagram:



Since f is injective, $K \cong f(K)$. Hence, $f(K) \cong Z$ by transitivity. Likewise, $\sigma_s K \cong f(\sigma_s K)$ by injectivity. It suffices to show that $f(\sigma_s K) = \sigma_{f(s)} f(K)$. Then, $Z \cong_{\text{st}} \sigma_{f(s)} f(K)$, a stellar subdivision. Setting $W := \sigma_{f(s)} f(K)$ gives the desired result.

The needed equality comes by carefully running through the definitions of each simplicial complex and showing both inclusions. The argument is elementary, but lengthy. Details are formalized in `stellarSubdivision_injective_image`. In fact, this equality can be strengthened to simplicial coercions. ◀

▶ **Proposition 15** (`stellarCoe_stellarSubdivision`). *Let $\varphi : E \to F$ be a simplicial coercion on K. Then,*

$$\varphi(\sigma_s K) = \sigma_{\varphi(s)} \varphi(K).$$

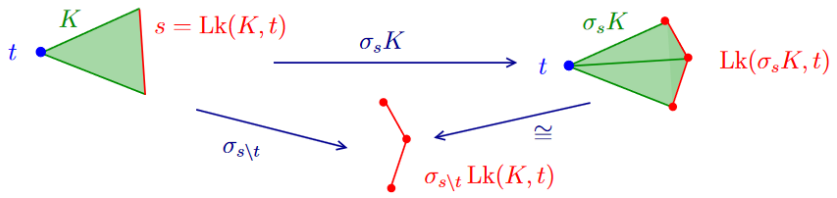


Figure 6 Geometric example of the identity stated in `stellarSubdivision_anticomm_link`.

3.2 Operations on Stellar Subdivisions

Finally, we provide a study of how stellar subdivisions interact with some of the operations defined in Section 2.2. The first result is about how the join of a subdivision can be rewritten as a subdivision of the join. This allows us to obtain results on stellar equivalences and joins.

▶ **Theorem 16** (`simplicialJoin_stellarSubdivision_left`, `simplicialJoin_stellarEquiv`).
- $(\sigma_s K) \star L = \sigma_{s \sqcup_s \varnothing}(K \star L)$.
- *If* $K \cong_{st} L$ *and* $Z \cong_{st} W$, *then* $K \star Z \cong_{st} L \star W$.

We also consider how stellar subdivisions interact with links. This is a core issue in the theory of manifolds, specifically if one wants to study subdivisions on the boundary of an n-dimensional combinatorial manifold M^n:

$$\partial M^n := \{s \in M^n \mid \forall k,\ \mathrm{Lk}(M^n, s) \not\cong_{\mathrm{st}} S^k\}.$$

Thus, given $t \in \sigma_s K$, we want to consider $\mathrm{Lk}(\sigma_s K, t)$ in terms of the original complex K when possible. In particular, when can we derive an equality, isomorphism, or stellar equivalence? The simplest case is if $t \in K \setminus \mathrm{St}(K, s)$ and $t \notin \mathrm{Lk}(K, s) \star \partial s \star \{x\}$. Geometrically, t is *outside* the neighborhood being subdivided. Thus, the link should be unaffected by a stellar subdivision around s.

▶ **Theorem 17** (`stellarSubdivision_link_of_starComplement`). *If* $t \in K \setminus St(K, s)$ *and* $t \notin Lk(K, s) \star \partial s$, *then*

$$Lk(\sigma_s K, t) = Lk(K, t).$$

The assumption $t \notin \mathrm{Lk}(K, s) \star \partial s$ is sufficient, since one can think of $\mathrm{Lk}(K, s) \star \partial s$ as the boundary of the star being subdivided (cf. Fig. 4). Equivalently,

▶ **Theorem 18** (`stellarSubdivision_link_of_barycenter`, `star_boundary_is_join`).

$$K \setminus St(K, s) \cap Lk(K, s) \star \partial s \star \{x\} = Lk(K, s) \star \partial s = Lk(\sigma_s K, \{x\}).$$

In the case where $t \in \mathrm{Lk}(K, s) \star \partial s$, we obtain a more complicated relation. In general, $\mathrm{Lk}(\sigma_s K, t)$ will contain x, thus any relationship to $\mathrm{Lk}(K, t)$ cannot be an equality. Indeed, it need not be an isomorphism either. However, we can derive a stellar equivalence, in which the link and stellar subdivision "anticommute" in a precise sense (cf. Figure 6).

▶ **Theorem 19** (`stellarSubdivision_anticomm_link`). *If* $t \in Lk(K, s) \star \partial s$, *then*

$$Lk(\sigma_s K, t) = \sigma_{s \setminus t}(Lk(K, t)).$$

In all cases, these identities are proven directly by unfolding the definitions of both sets in the equality. This results in rather lengthy arguments (the previous theorem alone constitutes ~2,000 lines of code), but yields direct, elementary proofs. This perhaps demonstrates why some of this theory has resolved into folklore status, since arguments can involve an immense amount of tedium. In fact, to our knowledge, the previous theorem has no standard reference in the literature. We run through a simple portion of the argument to demonstrate the general flow of these proofs and provide a more complete reference for the above.

Proof. We show the backwards inclusion. Let $u \in \sigma_{s\setminus t}(\mathrm{Lk}(K,t))$. By definition,

$$u \in \mathrm{Lk}(K,t) \setminus \mathrm{St}(\mathrm{Lk}(K,t), s \setminus t) \vee u \in \mathrm{Lk}(\mathrm{Lk}(K,t), s \setminus t) \star \partial(s \setminus t) \star \{x\}.$$

To obtain $u \in \mathrm{Lk}(\sigma_s K, t)$, we must show $u \in \sigma_s K$, $t \cup u \in \sigma_s K$, and $t \cap u = \varnothing$.

If $u \in \mathrm{Lk}(K,t) \setminus \mathrm{St}(\mathrm{Lk}(K,t), s \setminus t)$, then unfolding the definition gives u, $t \cup u \in K$; $t \cap u = \varnothing$; and $s \setminus t \notin u$. Note that u, $t \cup u \in K \setminus \mathrm{St}(K,s)$ since u, $t \cup u \in K$ and since $s \setminus t \notin u$ implies both $s \notin u$ and $s \notin t \cup u$. Hence, $u \in \mathrm{Lk}(\sigma_s K, t)$.

If $u \in \mathrm{Lk}(\mathrm{Lk}(K,t), s \setminus t) \star \partial(s \setminus t) \star \{x\}$, then $u = u_1 \cup u_2 \cup u_3$ mutually disjoint and not all empty where $u_1 \in \mathrm{Lk}(\mathrm{Lk}(K,t), s \setminus t)$, $u_2 \in \partial(s \setminus t)$, $u_3 \in \{x\}$. Likewise, we can split $t = t_1 \cup t_2$ where $t_1 \in \mathrm{Lk}(K,s)$ and $t_2 \in \partial s$. We can derive by unfolding the definitions:

1. $s \setminus t_1 = s$
2. $s \setminus t_2 \neq \varnothing$
3. If $u_1 = \varnothing$ or $u_2 \cup u_3 = \varnothing$, then $s \notin u$.

If $u_2 \cup u_3 = \varnothing$, then $u \in \mathrm{Lk}(\mathrm{Lk}(K,t), s \setminus t)$. The definitions and above properties yield u, $t \cup u \in K \setminus \mathrm{St}(K,s)$. $t \cap u = \varnothing$ follows since $u \in \mathrm{Lk}(K,t)$. Therefore, $u \in \mathrm{Lk}(\sigma_s K, t)$.

Otherwise, suppose $u_1 = \varnothing$. Then, $\partial(s \setminus t) \star \{x\} \subseteq \partial s \star \{x\}$ implies $u = u_2 \cup u_3 \in \sigma_s K$. Properties (1) and (2) give $t_2 \cup u_2 \in \partial s$, hence $t \cup u \in \sigma_s K$. $x \notin t$, so $t \cap u_3 = \varnothing$. Moreover, $u_2 \subset s \setminus t$ implies $t \cap u_2 = \varnothing$. Therefore, $t \cap u = \varnothing$ and $u \in \mathrm{Lk}(\sigma_s K, t)$.

If $u_1 \neq \varnothing$, then $u \in \sigma_s K$ follows by combining the above arguments. $t \cup u \in \sigma_s K$ follows by splitting $t \cup u = (t_1 \cup u_1) \cup (t_2 \cup u_2) \cup u_3$. Similar arguments give that $t_1 \cup u_1 \in \mathrm{Lk}(K,s)$, $t_2 \cup u_2 \in \partial s$, and $u_3 \in \{x\}$. Likewise, $t \cap u = \varnothing$ follows since $t \cap u_1 = \varnothing$ by definition then running the previous argument to get $t \cap u_2 = t \cap u_3 = \varnothing$. Therefore, $u \in \mathrm{Lk}(\sigma_s K, t)$. ◀

4 Discussion

4.1 Design & Challenges

This project stemmed from a desire to see a formalized proof of Alexander and Pachners' theorems on (bi)stellar subdivisions. In particular, to validate the fine details, primarily following the purely combinatorial proof by Hannes [13]. The first necessary step is a useful formalization of simplicial complexes and stellar subdivisions. Throughout, we were motivated by three main points: (a) applications to manifolds, (b) maintaining computability, and (c) maintaining generality. The first point motivated our choice of results on stellar subdivisions. The second point is motivated by a downstream desire to use subdivisions for concrete computations in applications. The third point arose by testing the assumptions and conclusions of Hannes' results. In many cases, we were able to remove conditions like finiteness or strengthen results to equality instead of isomorphism.

4.1.1 Simplicial Complexes

This project originally began before the addition of `SimplicialComplex` to `mathlib`. Initially, our goal was to reformulate our results, but eventually pivoted to isolating just abstract simplicial complexes, since supporting the extra geometric theory was too large of a detour for our purposes.

We highlight some differences in conventions between our original work and the current formalization. First, we relax the assumptions on the labeling type E. Since we live in the setting of purely combinatorial objects, the additional requirement of being a module over some ring $\mathbb{k}$ is unnecessary. We foresee this making work connecting `mathlib`'s formalization of the simplex category (defined over $\mathbb{N}$) and undirected graphs (defined over arbitrary vertex type V) as 1-dimensional complexes far more straightforward. Indeed, we experimented by defining the latter in the obvious way (`Graph.ofAbstract`). We set up future work by inducing instances of `SimplicialComplex`, carrying forward the results presented here.

Second, we originally allowed $\varnothing$ as a (-1)-dimensional face. This is simply a matter of convention – all of our results still hold regardless of which is used. Our original choice simply made much shorter proofs. For example, as seen in Theorem 19, faces in

$$\mathrm{Lk}(K, s) \star \partial s \star \{x\}$$

split into unions from each component of the join where at least one is nonempty, generating a fair amount of case work that $\varnothing$ can normally shortcut. We align our convention with `mathlib`'s `SimplicialComplex` for the purpose of compatibility, which excludes $\varnothing$ as a face.

4.1.2 Pre-abstract Simplicial Complexes

Since the preparation of this manuscript, `mathlib` has accepted independent formalizations of (pre-)abstract simplicial complexes in `AlgebraicTopology.SimplicialComplex`. Their definition of `PreAbstractSimplicialComplex` is identical to our formalization presented here. They provide instances of partial order and lattice structures, as well as the complex obtained as the image of a map $E \to F$. Additionally, they define `AbstractSimplicialComplex` as a `PreAbstractSimplicialComplex` with the additional totality axiom:

```
singleton_mem : ∀ v : E, {v} ∈ faces
```

We alluded to the issues from including a totality axiom in Remark 4. Formalizing subdivisions often requires obtaining fresh labels for new vertices generated in the process. This is true for barycentric, stellar, and chromatic subdivisions, for example. Therefore, formalizing subdivisions for both objects will require a radically different approach. We outline an approach that is applicable for `PreAbstractSimplicialComplex` in this paper, but highlight some challenges as applies to their `AbstractSimplicialComplex`.

Notably, if all values are currently used as labels, then one must find a source of fresh labels either within the existing type or from a new type. In the former case, one must impose a cardinality assumption that E is infinite and supply a map to shift labels so that a value is freed up. In the latter case, one can define the subdivision as being labeled in a new type that contains fresh values. However, these both conceivably generate an immense overhead in handling proper typecasting.

A downstream consequence is that many of the identities proven above are no longer equalities, but rather only hold up to isomorphism. Even if the base type E remains unchanged, this occurs with the added data of an injection $f : E \to E$ to obtain new labels. For example, consider $K \setminus \mathrm{St}(K, s)$ as a subcomplex of both K and $\sigma_s K$ where we use some injective f to label the barycenter. Under this formalization, these two objects are not equal, since f changes the labels of vertices. They are merely isomorphic.

4.1.3 Simplicial Maps

To suitably define simplicial maps for `SimplicialComplex`, one needs to carefully preserve the affine independence and gluing conditions. There are many equivalent definitions of simplicial maps, most of which either directly state it for embedded complexes or state it in the abstract case and extend to the embedded case. We believe following the latter would be the more successful approach, but leave some concerns for consideration.

Namely, one needs to extend a simplicial map $f : E \to F$ to be linear on the faces. Obviously, this is true if one simply requires $f : E \to_l[\mathbb{k}] F$ to be linear, but this misses many simplicial maps. Hence, a more refined implementation is needed to capture the entire category of simplicial complexes. We judged this complication as too far beyond our interests to resolve at the moment. Particularly, proving that maps are simplicial makes up a substantial portion of our code. Proving linearity conditions would at least double this work for no benefit. Hence, we find it better to leave the abstract definition in a form where one can later provide the correct extension to the geometric case.

Finally, we note that one could, in general, implement isomorphisms to utilize `Equiv` or `EquivLike` from `mathlib`. For example, one may then easily formalize the automorphism group of a simplicial complex. However, both expect the maps to be equivalences of the underlying types. This poses a problem, as we explicitly do not require simplicial isomorphisms to have inverses on the entire ambient type, but rather only on the vertex sets of the respective simplicial complexes. One might resolve this issue with `PartialEquiv`, which only requires maps to be inverses on specified source and target sets. This still presents a subtle issue. `PartialEquiv` (and, in fact, `Equiv`), expects a function between types. So, one would desire some $f : K \to L$ where K and L are fixed (abstract) simplicial complexes. This is doable if one can express something of the rough form

```
{E F K L : Type _} [AbstractSimplicialComplex E K] [AbstractSimplicialComplex F L]
(f : K → L) [SimplicialMap K L f]
```

similar to the syntax for rings and groups, for example.

However, this requires that `AbstractSimplicialComplex` be defined with the `class` keyword, rather than `structure`. We choose to stay compatible with `mathlib`'s implementations, which choose to define simplicial complexes via `structure`. In reality, we would desire some `PartialEquivLike` feature to fully resolve this issue, which currently does not exist.

4.1.4 Stellar Equivalence

Recall that our definition of stellar equivalence was defined between two simplicial complexes over type E, which resulted in a fair amount of work to support typecasting between stellar equivalences. In principle, one could define stellar equivalence between simplicial complexes of different underlying types to bake this property into the definition.

The trade-off is a lack of support for heterogenous equivalence relations in `mathlib`. Specifically, relations between different types are given by `Rel`, which generalizes the usual `Relation`. While we can prove analogs of reflexivity, symmetry, and transitivity, we lose access to any reasonable pre-implemented support for useful tools like induction. We briefly searched for alternative solutions and reached out to the `mathlib` community regarding this and the above issue of isomorphisms, but received no fruitful responses. So, we chose to limit our definitions to access the more complete toolbox of `Relation`.

4.2 Related Work

Within the `mathlib` community, significant progress has been made with the formalization of simplicial sets [8] and the simplex category, including the recent addition of simplicial homology defined for simplicial sets. There have been a number of impressive nontrivial results formalized using this theory, including work on group cohomology [24], homotopy type theory [10], higher category theory [7], and Quillen equivalences of model categories [30]. ACL2 formalized simplicial sets and objects in the mid-2000's [2] with subsequent work toward verifying the algorithms used in the Kenzo software [18, 26], implementing algorithms that arose from the field of constructive algebraic topology [32]. For example, computing homotopy groups of simply connected simplicial sets [35].

Generalizing to CW complexes, `mathlib` recently saw preliminary work formalizing the definition and basic properties [33]. Much earlier, CW complexes had been formalized in HOL Light [15] and Agda [5] with the primary intent of formalizing singular homology.

The most directly comparable work to our own is Löh's [25] expository text on Lean 3, which uses simplicial complexes as a case study. We found this work to be a useful first reference, but which required universal retooling. However, the text contains many great explicit examples of simplicial complexes and theorems not found in our current work, hence potentially good targets for further development. With the goal of formalizing persistent homology [16, 17], simplicial complexes have been formalized in the Rocq proof assistant [19]. More recently, they have also been formalized in Isabelle [4] for the purpose of studying Alexander duals and evasiveness properties [3]. These works are highly comparable to our own, focusing their attention on abstract simplicial complexes as an explicit computational tool.

4.3 Future Work

This work is an initial step toward a larger long-term goal of formalizing Alexander and Pachners' theorems. Much of the foundation for the work presented resulted from a push to define *combinatorial manifolds* [12, 13, 23] and to provide an equally comprehensive treatment of stellar subdivisions and homeomorphism type thereon. However, our work building up to and including stellar subdivisions became mature enough to be of its own general interest to share our experiences formalizing them in Lean and inspire further applications. We present a few potential routes inspired by classical use cases or experiences during development.

4.3.1 Algorithmic Applications

Our primary downstream consideration is the computational power of simplicial complexes and subdivisions. In particular, simplicial complexes naturally arise as a combinatorial tool for extracting properties of different spaces. Most recognizable are the nerve of a cover, the flag complex of a graph, or the Rips complex of a metric space. One may also consider determining the homotopy type of a space via collapsing algorithms. Discrete Morse theory and shelling orders of simplicial complexes are a key tool in this study. Simplicial homology and cohomology are, of course, natural candidates for future formalization. Kozlov [21, 22] provides an excellent overview of these topics and their applications.

4.3.2 Integration with `mathlib`

Given the fragmentary related work already conducted in `mathlib`, we placed some emphasis on attempting integration with these various constructions. In particular, isolating just the abstract necessities and type signatures that are as general as possible. Our hope is that our

formalization can be somewhat painlessly integrated with these formalizations, particularly with simplicial sets and inducing instances of a geometric `SimplicialComplex`, a CW complex, or a graph structure. Indeed, we have started work on resolving issues presented above toward the geometric case, and have initiated plans for integration with elements of the `mathlib` library.

`Analysis.Convex` also defines the `convexJoin` between two sets as the collection of line segments between them. In principle, one could use this as a connection to give a coherent notion of joins for `SimplicialComplex`. Formally, one would prove a theorem of the form

```
(K ⋆ L).space = convexJoin 𝕜 K.space L.space
```

Moreover, many of the proofs involved in this work are direct, burning through tedious checks that faces satisfy conditions for equalities (cf. Theorem 19) or showing that maps are simplicial. Many arguments run similar courses, which could be automated or refactored in more paradigmatic ways, perhaps with more cleverly structured arguments. Further work can be done to dramatically reduce much of the tedium and proof lengths.

4.4 Conclusion

This paper presents a summary of the core definitions and results from our efforts to formalize abstract simplicial complexes and stellar subdivisions. Our motivation and focus is toward computational downstream use cases, in particular the combinatorial theory of manifolds, and isolating only the necessities to achieve that goal, while encouraging future work toward integration with the wide array of similar developments in `mathlib`. We constrain ourselves to discussion of the more pertinent features. The available Lean code contains more constructions and results than what can be presented in a clear and concise manner here. While somewhat "classical," the tools of combinatorial topology are not only widely used in many fields of mathematics but also outside of mathematics. We are optimistic that a proper treatment of the subject will end with a useful tool for future formalizations in Lean, hopefully avoiding further slides into mathematical folklore.

References

1 James W Alexander. The combinatorial theory of complexes. *Annals of Mathematics*, 31(2):292–320, 1930.

2 Mirian Andrés, Laureano Lambán, Julio Rubio, and José Luis Ruiz-Reina. Formalizing simplicial topology in acl2. In *Proceedings of ACL2 Workshop*, volume 2007, pages 34–39, 2007.

3 Jesús Aransay, Laureano Lambán, Julius Michaelis, and Julio Rubio. Formalizing alexander duality through bdds. In *ISAIM*, 2022.

4 Jesús Aransay, Alejandro del Campo, and Julius Michaelis. Simplicial complexes and boolean functions. *Archive of Formal Proofs*, November 2021. Formal Proof Development. URL: `https://isa-afp.org/entries/Simplicial_complexes_and_boolean_functions.html`.

5 Ulrik Buchholtz and Kuen-Bang Hou Favonia. Cellular cohomology in homotopy type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 521–529, 2018. `doi:10.1145/3209108.3209188`.

6 Benjamin A. Burton, Ryan Budney, William Pettersson, et al. Regina: Software for low-dimensional topology. `http://regina-normal.github.io/`, 1999–2023.

7 Mario Carneiro and Emily Riehl. Formalizing colimits in Cat. *arXiv preprint*, 2025. `arXiv:2503.20704`.

8 Floris Cnossen. Simplicial sets in Lean. *Bachelor thesis*, 2021.

9 Marshall M Cohen. A proof of newman's theorem. *Mathematical Proceedings of the Cambridge Philosophical Society*, 64(4):961–963, 1968.

10 Kunhong Du. On the formalization of the simplicial model of hott. Master's thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn, 2025.

11 Herbert Edelsbrunner, John Harer, et al. Persistent homology – A survey. *Contemporary mathematics*, 453(26):257–282, 2008.

12 Stefan Friedl. Topology, 2023. URL: `https://www.ams.org/open-math-notes/omn-view-listing?listingId=111368`.

13 Stefan Friedl and Julian Hannes. Pachner's theorem. In *2021-2022 MATRIX Annals*, pages 777–832. Springer, 2024.

14 L.C. Glaser. *Geometrical Combinatorial Topology*. Number v. 1 in Geometrical Combinatorial Topology. Van Nostrand Reinhold Company, 1970.

15 John Harrison. A HOL light formalization of singular homology theory, 2018. URL: `https://docslib.org/doc/4817357/a-hol-light-formalization-of-singular-homology-theory`.

16 Jónathan Heras, Thierry Coquand, Anders Mörtberg, and Vincent Siles. Computing persistent homology within coq/ssreflect. *ACM Transactions on Computational Logic (TOCL)*, 14(4):1–16, 2013. `doi:10.1145/2528929`.

17 Jónathan Heras, Maxime Dénès, Gadea Mata, Anders Mörtberg, María Poza, and Vincent Siles. Towards a certified computation of homology groups for digital images. In *Computational Topology in Image Context: 4th International Workshop, CTIC 2012, Bertinoro, Italy, May 28-30, 2012. Proceedings*, pages 49–57. Springer, 2012. `doi:10.1007/978-3-642-30238-1_6`.

18 Jónathan Heras, Vico Pascual, and Julio Rubio. Proving with acl2 the correctness of simplicial sets in the kenzo system. In *International Symposium on Logic-Based Program Synthesis and Transformation*, pages 37–51. Springer, 2010. `doi:10.1007/978-3-642-20551-4_3`.

19 Jónathan Heras, María Poza, Maxime Dénès, and Laurence Rideau. Incidence simplicial matrices formalized in coq/ssreflect. In *International Conference on Intelligent Computer Mathematics*, pages 30–44. Springer, 2011. `doi:10.1007/978-3-642-22673-1_3`.

20 J. F. P. Hudson. *Piecewise Linear Topology*. W. A. Benjamin, Inc., 1969.

21 Dmitry Kozlov. *Combinatorial Algebraic Topology*. Springer, 2008. `doi:10.1007/978-3-540-71962-5`.

22 Dmitry Kozlov. *Organized Collapse: An Introduction to Discrete Morse Theory*, volume 207. American Mathematical Society, 2021.

23 William Bernard Raymond Lickorish. Simplicial moves on complexes and manifolds. *Geometry and Topology Monographs*, 2(299-320):314, 1999.

24 Amelia Livingston. Group cohomology in the Lean community library. In *14th International Conference on Interactive Theorem Proving (ITP 2023)*, pages 22:1–22:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPIcs.ITP.2023.22`.

25 Clara Löh. *Exploring Formalisation: A Primer in Human-readable Mathematics in Lean 3 with Examples from Simplicial Topology*, volume 11. Springer Nature, 2022.

26 Francisco-Jesus Martín-Mateos, Julio Rubio, and Jose-Luis Ruiz-Reina. Acl2 verification of simplicial degeneracy programs in the kenzo system. In *International Conference on Intelligent Computer Mathematics*, pages 106–121. Springer, 2009.

27 Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*, pages 625–635, Berlin, Heidelberg, 2021. Springer-Verlag. `doi:10.1007/978-3-030-79876-5_37`.

28 James R Munkres. *Elements of Algebraic Topology*. CRC Press, 2018.

29 Udo Pachner. Pl homeomorphic manifolds are equivalent by elementary shellings. *European journal of Combinatorics*, 12(2):129–145, 1991. `doi:10.1016/S0195-6698(13)80080-7`.

30 Joël Riou. topcat-model-category. `https://github.com/joelriou/topcat-model-category`, 2024.

31 Colin P Rourke and Brian Joseph Sanderson. *Introduction to Piecewise-Linear Topology*. Springer Science & Business Media, 2012.

32 Julio Rubio and Francis Sergeraert. Constructive algebraic topology. *Bulletin des Sciences Mathématiques*, 126(5):389–412, 2002.

33 Hannah Scholz. Formalisation of CW-complexes. Bachelor's thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn, 2024.

34 Herbert Seifert and William Threlfall. *Lehrbuch der Topologie*, volume 31. American Mathematical Society, 2004.

35 Francis Sergeraert. The computability problem in algebraic topology. *Advances in Mathematics*, 104(1):1–29, 1994.

36 Francis Sergeraert. Kenzo: A symbolic software for effective homology computation, 1999. URL: `https://www-fourier.univ-grenoble-alpes.fr/~sergerar/Kenzo/`.

37 RB Sher and RJ Daverman. *Handbook of Geometric Topology*. Elsevier, 2001.

38 Edwin H Spanier. *Algebraic Topology*. Springer Science & Business Media, 2012.

39 The mathlib Community. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, New Orleans, LA, USA, January 2020. ACM. `doi:10.1145/3372885.3373824`.