Formal specification and proofs for the topology and classification of combinatorial surfaces

Christophe Dehlinger and Jean-François Dufourd

Laboratoire ICUBE, Université de Strasbourg, CNRS,
Pôle d'Innovation Technologique, boulevard S. Brant,
BP10413, 67400 Illkirch, France
email: jfd@unistra.fr, christophedehlinger@gmail.com

Abstract

We describe one of the first attempts at using modern specification techniques in the field of geometric modelling and computational geometry. Using the Coq system, we developed a formal multi-level specification of combinatorial maps, used to represent subdivisions of geometric manifolds, and then exploited it to formally prove fundamental theorems. In particular, we outline here an original and constructive proof of a combinatorial part of the famous Surface Classification Theorem, based on a set of so-called "conservative" elementary operations on subdivisions.

Key words: combinatorial surfaces, classification, generalized maps, formal specification, assisted proof, Coq system

1 Introduction

Thanks to their recent rise in quality, efficiency and user-friendliness, formal specification and proof tools are starting to be used in all mathematical fields, including geometric modelling and computational geometry. In this paper, we explain how we built a specification of the topology of geometric manifolds in a higher-order logic framework with the Coq theorem proving system, then use this specification to formally prove a combinatorial part of the famous and non-trivial Surface Classification Theorem, restricted to compact surfaces with boundary.

Topology is here described by *generalized maps* [33] [34], a combinatorial model of subdivisions of manifolds. We specify *gmaps* (short for *generalized maps*), in the *Calculus of Inductive Constructions* (CIC) [9] [40], a higher-order logic that allows manipulation of types and objects as well as propositions and

proofs. We then express in this calculus the properties that we wish to show, and prove them by building corresponding proof terms from the axiom terms, using the inference rules of the calculus. In practice, all the specification, proof discovery and proof checking work is done with help of the Coq proof assistant [9] [1]. Some of the theorems we thus proved are very difficult, which testifies for the power of these tools.

Our specification features a 3-level hierarchy of increasingly fitting but harder-to-manipulate types to represent gmaps, a large number of constructions, exploration and modification operations on gmaps, and over a thousand formally expressed and proved theorems, including in particular a topological version of a part of the famous Surface Classification Theorem. These theorems were proved using a wide range of techniques, such as structural induction or Nœtherian induction.

The long-term objective of this work is to provide a very solid theoretical basis for the development of geometric modellers, i.e. programs which allow to build and manipulate combinatorial surfaces and even combinatorial manifolds of higher dimensions. Specification and proof techniques allow firstly to check the validity and relevance of models and implementations by proving theoretical results related to them. This is the case here for the Surface Classification Theorem. Secondly, they allow to formally check the correctness and termination of algorithms and operations. In our eyes, these two goals are of equal importance, and justify the need to develop and improve these already powerful formal techniques, so that they can be used to tackle difficult problems of geometric modelling and computational geometry. For instance, these tools could be very helpful in the study of the definition and manipulation of discrete surfaces, which are a very important link between computer graphics, geometric modelling and imaging.

The first part of this research is detailed in [13] and [14], which are mainly intended to the formal specification and automated proof community. In the present paper, we give a complete panorama of our work including the second part, while being less exhaustive on formal aspects and more focused on geometric modelling features. However, the entire Coq source of the development can be downloaded [12].

The Surface Classification Theorem is one of the most deep and exciting result in algebraic topology of dimension 2. In [26], Gallier and Xu offer a substantial history of the discovery and of the numerous attempts to prove this result. Indeed, a rigorous proof always needs many definitions and lemmas around topology, algebra and surfaces. So, the theorem says that, despite the fact that surfaces appear in many diverse forms, they can be classified, which means that every (compact) surface is equivalent to *exactly one* representative surface, also called a surface in *normal form*.

Of course, to make this statement rigourous, it is necessary to precise: *(i)* what is a surface, *(ii)* what is the equivalence of surfaces, *(iii)* what are normal forms of surfaces. This is the subject of entire textbooks [37,23,46,29,26], in which the Surface Classification Theorem is the major result. However, the basic mathematical tools, the normal forms of surfaces, and the proof techniques, can differ significantly. In the following, to address the problem, we will provide a way which is based on combinatorial tools and formalisms well-adapted to the help of an up-to-date interactive proof assistant.

After this introduction, we briefly present some related work in Sect. 2. In Sect. 3, we state precisely what we mean by surfaces, emphasizing the link between surface subdivisions and generalized maps. In Sect. 4, we describe intuitively and then formally the model of generalized maps, as well as important related basic notions. In Sect. 5, we introduce the Coq notation to describe the basic types involved. In Sect. 6, we start focusing on the classification theorem by listing a set of operations on generalized maps that we call conservative, and use them to define a notion of topological equivalence. In Sect. 7, we describe the easier second half, called the *trading theorem*, of the part of the classification theorem we deal with. In Sect. 8, we deal with the more complex (and meaningful) first half, called the *normalization theorem*, and we conclude in Sect. 9.

2 Related work

There are several approaches to the representation and building of geometrical objects: using equations, viewing them as Boolean combination of basic objects, or describing their boundaries. In the early 80s, Requicha [45] designed a framework to compare the different solid representation methods. In the late 80s, Lienhardt developed a powerful tool for boundary representation, the generalized maps, or gmaps [33] [34]. This model is an extension of Cori's hypermaps [10], which were themselves inspired by Jacques and Tutte's combinatorial maps [30] [47]. The generalized maps have the same modeling power as other models, for instance the ordered structures of Brisson [4], in the sense that they can model the mesh topology of any surface, open (i.e. with boundaries) or not, orientable or not. Their advantages are a precise mathematical description in terms of algebraic combinatorial structure, an interesting saving of concepts making implementations easier, and a great extension power allowing to describe manifolds of any dimension [34,35].

Despite the efforts to improve them, the topological models used for geometric modelling, including gmaps, remain quite complicated, often spawning very complex algorithms. However, quite little work has been done to formally specify them in order to reduce the odds of implementation faults. Several

specification techniques have been experimented, such as set-based models, algebraic specifications, term rewriting, functional programming. For references, see for instance [18]. Dufourd developed a generic algebraic specification for subdivisions [16] [17] on which an interactive 3D topology-based modeller Topofil designed by Bertrand was based [3]. Finally, an attempt of functional specification and programming in OCaml of classical computational geometry problems with combinatorial maps is related in [22]. One of the main shortcomings of these specification experiments is their lack of using supports for formal and computer-verifiable proofs. But even those fall short, as their proof facilities are outdated. Hence our motivation to work with a state-of-the-art specification and proof assistant.

Theorem provers have indeed gone a long way. Among the first, one can cite Automath by de Bruijn, based on a typed lambda-calculus, and Prolog by Colmerauer and Kowalski, implementing reasoning by resolution in classical first-order logic. Both date back to the sixties. Unlike these two, modern provers are based on higher-order logics, allowing quantification over sets, types and functions. There are currently two classes of proof assistants: *theorem provers*, that are dedicated to one or several fields and work by running specialized proof search algorithms against the sought goal, for example PVS, and *tactic provers*, generic proof systems that work by refining a goal using simple commands called tactics, for instance Isabelle. The Coq system [8], based on the Calculus of Inductive Constructions, is one of the most powerful of these, and has already been successfully used in many different fields.

Geometry, but not geometric modelling, has always been popular for automated deduction. One of the very first automatic theorem provers was dedicated to plane geometry [27], reasoning with the resolution rule from well-chosen *ad hoc* axioms. Heyting's intuitionistic plane affine geometry axiomatic system was used by von Plato [48] and Kahn [31] to develop and test in Coq their own constructive theory of ordered affine geometry. We used Coq to study Hilbert's axiomatics in a constructive framework [15]. Besides these "pure" geometry approaches, algebraic resolution methods were also implemented in an *ad hoc* way, most notably in [7]. Pichardie and Bertot formalized in Coq the development of proved convex hull algorithms from an axiomatic system by Knuth [32] [41], while Meickle and Fleuriot tackled similar problems in Isabelle [38].

Combinatorial hypermaps have been formalized in Coq/SSReflect by Gonthier et al. to prove the very difficult Four Color Theorem [28]. Finally, the only geometric modelling-related automated deduction experiments with combinatorial maps that we are aware of is Puitg and Dufourd's proofs of a planarity criterion and the Euler-Poincaré formula [42,19] [44], leading to the formal proof of a discrete Jordan curve theorem in orientable closed surfaces [20]. Very recently, correctness proofs of algorithms in the context of hypermaps

were conducted for convex hull problems [5], and Dufourd and Bertot have achieved in Coq a proof of total correctness of a Delaunay 2D algorithm [21].

Our original idea to prove the classification theorem was to adapt in a formal context one of the classical proofs. Moreover, following Griffiths [29], we wanted to avoid classical mathematical treatments with topological spaces, continuity, homeomorphisms and homology, in favor of a direct proof with combinatorial arguments using the intuitive notion of generalized map. We then decided to build a new proof adapting the attractive approach of Griffiths with "paper surfaces" – also called "polyhedral surfaces" or "rubber sheet surfaces" by Seifert and Threlfall [46] – to the gmap notions. So, we fully use notions such as *panel*, panel *addition*, reasoning by structural and Nœtherian induction as well. However, to use the Coq system, we had to be extremely rigorous in the capture of these notions, to avoid axioms given by the writing of *"agreements"* [29], and to clearly separate what is geometry, topology and combinatorics. Although we excluded approaches based on word rewriting which seemed too far from this goal, we were inspired by Fomenko's work [25], to define some conservative operations able to prove what is called the *trading theorem* and the *normalization theorem*.

3 Surfaces and subdivisions

As it is the basis of algebraic topology in dimension 2, the approach that we follow in this section is common to most textbooks on surfaces [37,23,46,29,26], but our terminology is mainly borrowed from Griffiths [29]. Classically, a surface (or 2-dimensional manifold) is defined as a Hausdorff space where each point's neighborhood is homeomorphic to either $\mathbb{R}^2$ or the half-space $y \geq 0$. This definition is fine for mathematical study, but not for actual computer use: it is too abstract to realistically base a modeller on it. This kind of continuous object being too hard to manipulate, surfaces are often modelled instead by using combinatorial and modular models, which allow easier storing and handling of surfaces. A very common approach is to *subdivide* a surface into simple elementary surfaces and see how these elementary bricks are connected to one another to form the original surface.

Depending on the model, the elementary surfaces may be triangles, or, in our case, *panels*. A panel is a surface that is homeomorphic to a disc, i.e. that is the image of a disc by a bicontinuous bijection to it. Figure 1 shows a few panels: (*a*) a disc, (*b*) a polygon, (*c*) a slit ring, and (*d*) a cap. Panels have a single boundary which is a Jordan curve decomposed into a sequence of Jordan arcs, called *edges*, bounded by points, called *vertices*. For us, a surface is a patchwork of panels sewn along their edges. Some edges may be unsewn, they correspond to the boundaries of the subdivided surface. Each boundary

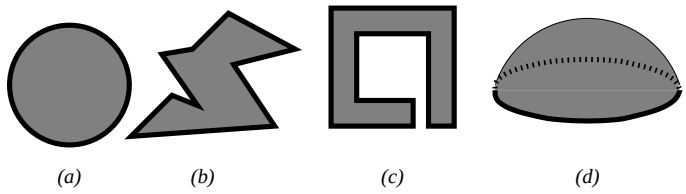


Fig. 1. Examples of panels

is a Jordan curve determining a *hole* in the surface.

Figure 2 shows a few examples of subdivided surfaces. Surface (a) is a triangle,

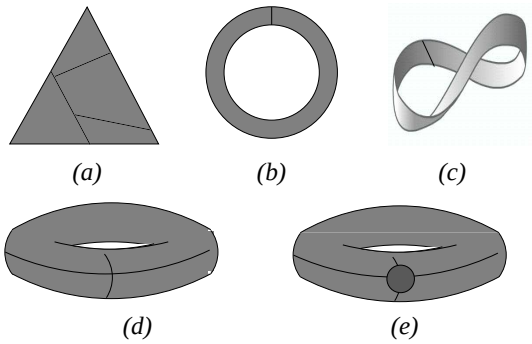


Fig. 2. Examples of surface subdivisions

made of four panels. It has a single boundary. Surface (b) is a ring, made of a single panel in which two parts of the boundary are sewn in order to close the ring. It has two boundaries (the "inside" and "outside" boundaries). Surface (c) is a Mœbius strip, a ring in which the panel is twisted before its ends are sewn. It has a single boundary. Surface (d) is a torus, made of a single bent rectangular panel the opposite sides of which are sewn to each other. It has no boundary. Finally, surface (e) is a torus that has been punctured, giving it a boundary. This approach allows to model open (i.e. with finitely many boundaries) or closed (i.e. without boundary) compact surfaces as well as orientable or non-orientable compact surfaces, like the Mœbius strip or the Klein bottle.

The theorem of classification deals with the classification of compact surfaces according to their topology. Two surfaces belong to the same class if they are homeomorphic, i.e. if there exists a bicontinuous bijection from one onto the other. In textbooks, the theorem has different equivalent formulations. The one that we retain here states:

Theorem 1 *(i) Any connected compact surface with boundaries belongs to one class, the members of which are homeomorphic to each other. Each class is characterized by a triplet of natural numbers (p, q, r) with $r \leq 2$.*
(ii) For $r \leq 2$, $r' \leq 2$ and $(p, q, r) \neq (p', q', r')$, the classes represented by (p, q, r) and (p', q', r') are distinct.

Of course, other formulations are possible when one retains other normal forms, i.e. representatives of the classes [37,23,46,29,26]. Moreover, although the theorem also holds for closed surfaces − and is often first proved for them −, which are obtained from open surfaces by adding *lids* on their holes, we remind the reader that we focus on open surfaces in this paper.

The values of p, q and r are very meaningful regarding the surfaces in the corresponding class: each surface in class (p, q, r) features p *punctures* (in addition to the outer boundary), q *handles* and r *twists*. Thus, surfaces in Figure 2 belong in order to classes $(0, 0, 0)$ for (a), $(1, 0, 0)$ for (b), $(0, 0, 1)$ for (c) and $(0, 1, 0)$ for (e). Surface (d) belongs to none as it is a closed surface, and as such cannot be applied the theorem we consider.

The surfaces from Figure 2 are actually very useful. Generally, in mathematics, it is often convenient for any classification theorem to exhibit a *canonical* (or *normal* element) for each class, a representative element of this class. For this theorem, following Griffiths [29], we use only sewn panels to build the normal surfaces, which we call *plans*. Thus, the plan for class *(p,q,r)*, denoted $P_{p,q,r}$, is built by:

- starting with a disc (a simple panel with a boundary);
- sewing p rings to its boundary (thus adding p punctures);
- sewing q punctured tori to the outer boundary (the tori are sewn along part of their puncture, thus adding q handles);
- sewing r Mœbius strips to the boundary (thus adding r twists).

Figure 3 shows the normal surface $P_{1,1,1}$ for class $(1, 1, 1)$, a surface with two

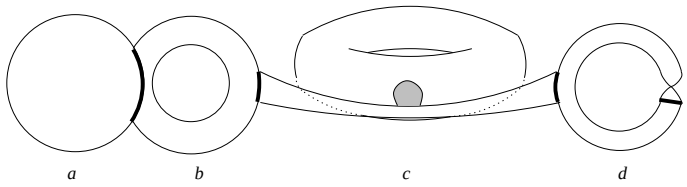


Fig. 3. Surface $P_{1,1,1}$

boundaries (the outer boundary and a puncture), one handle and one twist (with simplified drawings). Thus, part *(i)* of the classification theorem states that any open surface is homeomorphic to a surface like that in Figure 3, with $r \leq 2$.

To recover *closed* surfaces, it is enough to glue *lids* along the boundaries:

- the unique disc is closed into a *sphere*;
- the p rings become p *discs*;
- the q tori become q *handles*;
- if $r = 1$, the unique Mœbius strip becomes one *cross-cap* (or *projective*

plane), else ($r = 2$) the two Mœbius strips are glued together into one *Klein bottle*.

So, the result complies with the classification theorem for *closed* surfaces [37,23,46,26].

4 Generalized maps

4.1 Basic definitions and properties

Generalized maps (gmaps, in short) are a combinatorial model that may be used to represent the topology of manifolds. Gmaps are most importantly characterized by their *dimension*, which is also the dimension of the manifolds that they can represent. Gmaps of dimension -1, or -1-gmaps, represent isolated vertices, 0-gmaps isolated edges, 1-gmaps simple curves, 2-gmaps surfaces, 3-gmaps volumes, etc.

A gmap is a collection of basic abstract elements called *darts*, intuitively half-edges, that are connected by involutions α_k, k being a *dimension*, in order to form cells. We denote the (infinite) type of darts by *dart*. While they are a completely abstract type in our specification, darts are usually implemented as integers or pointers. Although our Coq specification encompasses all dimensions, we focus here on 2-gmaps in order to make definitions simpler. A common mathematical definition of such an object is the following

Definition 2 *A* generalized map of dimension 2*, or* 2-gmap*, is a quadruplet* $(D, \alpha_0, \alpha_1, \alpha_2)$*, where* D *is a finite subset of* dart *and where the* α_k *are involutions on* D*, such that* α_0 *and* α_1 *have no fixpoint and that* $\alpha_0 \circ \alpha_2$ *is also an involution.*

Thus, with $D \subset dart$, D finite and $\alpha_0, \alpha_1, \alpha_2 : D \to D$, $M = (D, \alpha_0, \alpha_1, \alpha_2)$ is a 2-gmap if:

- $\forall x \in D, \forall k \leq 2, \alpha_k^2(x) = x$;
- $\forall x \in D, \forall k \leq 1, \alpha_k(x) \neq x$;
- $\forall x \in D, (\alpha_0 \circ \alpha_2)^2(x) = x$.

A dart x is said to be *sewn at dimension* k, or k*-sewn*, to dart y if $\alpha_k(x) = y$. In this case, as the α_k are involutive, y is k-sewn to x as well. Dart y is also said to be the k*-neighbor* of dart x if $\alpha_k(x) = y$. Each of the α_k has a different purpose: α_0 is used to make up edges, α_1 simple curves, and α_2 surfaces. In general, for any given k, α_k is used to make up cells of dimension k.

The conditions imposed on the α_k enforce the consistency and completeness of cells: involutivity of the α_k guarantees that each dart has exactly one k-neighbor for each k. The lack of fixpoints for α_0 and α_1 prevents the presence of dangling darts. As we will see later, fixpoints of α_2 are the darts that belong to a boundary. Finally, forcing $\alpha_0 \circ \alpha_2$ to be involutive ensures that whenever a dart is 2-sewn to another, their respective 0-neighbors are also 2-sewn. Thus, only whole edges are 2-sewn.

Figure 4 shows the standard representation for darts and sewings (Left),

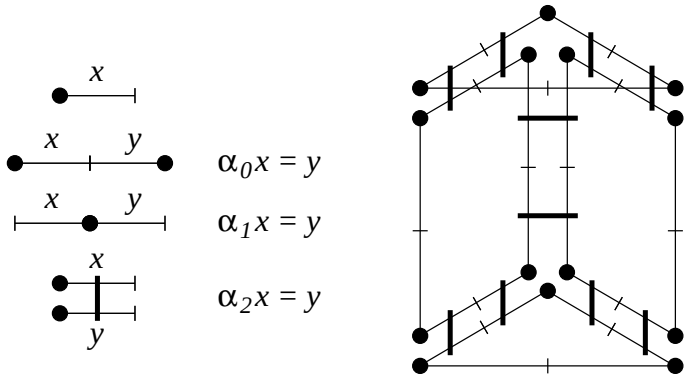


Fig. 4. Standard graphic representation of darts, sample 2-gmap

as well as a sample 2-gmap subdividing a prism with a triangular base and lacking the front face (Right). In this figure, x and y are darts. A dart pictured without k-neighbor is implicitely k-sewn to itself.

4.2 Cells and invariants

With the above definitions, we can introduce usual notions of topology, most of them being described as *orbits*:

Definition 3 *Let D be a set and $f_1, f_2, \ldots, f_n$ functions on D. For any $x \in D$, the* orbit of $f_1, f_2, \ldots, f_n$ at x *is defined to be the smallest subset of D containing x and stable by all functions f_i. It is denoted $< f_1, f_2, \ldots, f_n > (x)$.*

Let $M = (D, \alpha_0, \alpha_1, \alpha_2)$ be any 2-gmap. With orbits, connected components and cells of M are easy to define:

Definition 4 *The* connected component *of M* incident to dart $x \in D$ *is the 2-gmap $M' = (D', \alpha'_0, \alpha'_1, \alpha'_2)$ satisfying:*

- $D' =< \alpha_0, \alpha_1, \alpha_2 > (x)$;
- $\forall k \mid 0 \leq k \leq 2$, α'_k *is the restriction of* α_k *to* D'.

Definition 5 *For any $x \in D$ and any $i, j, k \leq 2$ pairwise distinct, we call orbit $< \alpha_i, \alpha_j > (x)$ the* k-cell of M incident to x. *A* 0*-cell is also called a* vertex*, a* 1*-cell an* edge *and a* 2*-cell a* face. *We define the* map of k-cells *of M to be the algebraic structure regrouping all the k-cells of M, obtained from M by ripping all its k-sewings. It is denoted M_k.*

Definition 6 *For any $x \in D$ and any $k \leq 2$ we call orbit $< \alpha_0, \alpha_1, \ldots, \alpha_{k-1} > (x)$ the* simple k-cell of M incident to x. *Like we did for cells, we define the* map of simple k-cells *as the algebraic structure regrouping all the simple k-cells of M. It is denoted M_k^S. A simple cell of dimension 0, 1 and 2 is respectively a single dart, an open edge and a cycle of alternately 0- and 1-sewn darts.*

In this definition, M_2 is a 2-gmap where every dart is an α_2-fixpoint, M_0 and M_1 are two 1-gmap (with 2 involutions) when α_0 and α_1, which may have fixpoints, are renumbered conveniently. Actually, M_0^S and M_1^S are respectively a 0-gmap and a 1-gmap. Also note that $M_2 = M_2^S$. Figure 5 shows an example

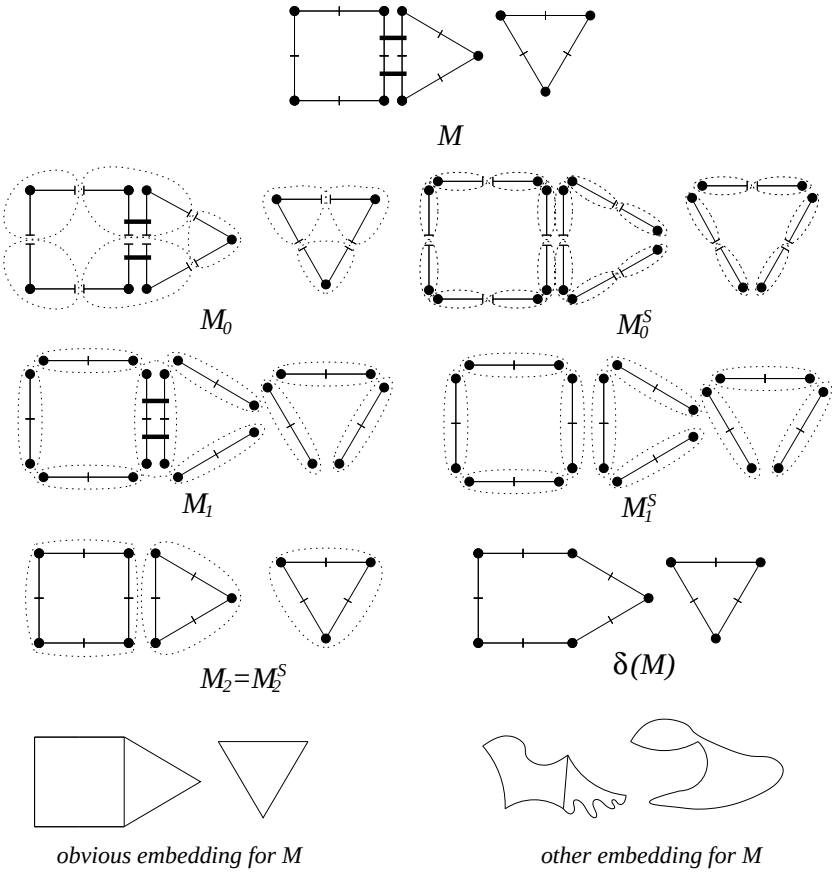


Fig. 5. An example of a 2-gmap and its maps of cells, 2-gmap of boundaries and standard embedding

2-gmap as well as its maps of cells. In this example, M has two connected components. The 2-gmap $\delta(M)$ is the 2*-gmap of boundaries* of M: its darts

are the darts of M that are incident to a boundary, each of them being 1-sewn in $\delta(M)$ to its boundary-neighbor in M, the notions of boundary and boundary neighborhood being defined as:

Definition 7 *A dart $x \in D$ is said to be* incident to a boundary *of M if $\alpha_2(x) = x$. A dart incident to a boundary is called* external*, it is called* internal *otherwise.*

Definition 8 *A gmap without boundary is said to be* closed*, and* open *otherwise. An edge whose darts are sewn to themselves is said to be* open *(it has 2 darts), and* closed *otherwise (it has 4 darts). A vertex containing a dart incident to a boundary is said to be* open*, and* closed *otherwise.*

Definition 9 *For any dart x, dart $y_k = (\alpha_2 \circ \alpha_1)^k(x)$ is called the* boundary-neighbor *of x if k is the smallest natural number such that y_k is incident to a boundary (i.e. $\alpha_2(y_k) = y_k$) and $y_k \neq x$. All the external darts have a boundary-neighbor, what is not necessarily the case for the other darts.*

Definition 10 *A* path *is a finite sequence of dimensions. Following a path $n_0, n_1, \ldots, n_k$ from dart x yields dart $\alpha_{n_k} \circ \cdots \circ \alpha_{n_1} \circ \alpha_{n_0}(x)$.*

Now that we have defined darts, vertices, edges, faces and connected components of a 2-gmap, we note their respective numbers by d, v, e, f and c.

When the gmap is connected, i.e. $c = 1$, it represents the topology of a surface subdivision, and, conversely, the topology of any surface subdivision is represented by a connected gmap. Then, we can compute the *Euler-Poincaré characteristic* χ of the underlying surface by the famous formula (slightly generalized):

$$\chi = v + e + f - d.$$

Let us go back to the former characteristics, (p, q, r), in order to better understand them. Fistly, p is about one unity the *number of boundaries* b. Secondly, when it is reduced to 0, 1 or 2, r is called the *orientability factor* of the surface. Thirdly, in this case, q is another classical topological invariant of the surface which is called its *genus*. It can be interpreted either as the *number of handles*, sometimes called *tunnels*, of the surface, or as the greatest number of (closed) Jordan curves that can be drawn on the surface without disconnecting it. Note that the tunnels must not be confused with holes or punctures.

These numbers are linked by well-known formulae [33]:

$$b = p + 1,$$

$$q = 1 - (r + b + \chi)/2.$$

Thus $(r+b+\chi)$ is necessarily *even*, and, q being a natural number, we always have:

$$r + b + \chi \leq 2.$$

Finally, in [33,34], it is shown how all these surface characteristics can be computed from a corresponding 2-gmap, which is implemented in Topofil [3].

4.3 Embedding and conservative operations

The complete link between subdivisions like 2-gmaps and actual surfaces is provided by the notion of *embedding*, i.e. the projection of topological objects into a representation space. We are interested here in *continuous embeddings*, embeddings that have "good properties" which make them intuitively valid. For instance, all darts from the same topological vertex must be embedded into the same geometrical vertex. Similarly, all darts from the same topological edge must be embedded into the same geometric edge and all darts from the same topological face must be embedded into the same geometric face. Incident topological objects must be embedded into incident geometrical objects.

Briant and Singerman [6], and Lienhardt [35], have shown that gmaps (with all the above constraints) *exactly* model all the subdivisions of compact surfaces, open or closed, orientable or not. However, as we had decided to focus on combinatorial aspects, we did not formally specify embeddings, as it would have required extensive algebra, continuous topology and geometry developments, probably making the specification at least twice as large. But, forgoing embeddings has a dramatic consequence. Indeed, the theorem of classification of surfaces applies to surfaces, i.e. embedded topological objects, while all we have available is a model of the topology of surfaces. Thus, the theorem cannot be directly expressed in our specification. The first step in dealing with this theorem was to find a way to adapt the theorem to the combinatorial topology world.

Our solution was to introduce what we called *conservative operations*, simple, local operations that, when applied to a suitable 2-gmap, are strongly believed not to alter the set of surfaces that this 2-gmap subdivides. Proving so would require specifying embeddings. We stress that while conservative operations were inspired by local surface deformations, they only deal with subdivisions and not surfaces. These operations are listed in Sect. 6. This allows us to *classify subdivisions*: two 2-gmaps are considered *topologically equivalent* if one can be obtained from the other by applying conservative operations. Thus, two 2-gmaps are equivalent only if they subdivide the same surfaces. Then, we proved the first part of the following full theorem of classification:

Theorem 11 *(i) Any open 2-gmap is topologically equivalent to a 2-gmap*

$N_{p,q,r}$ that subdivides one of the $P_{p,q,r}$ surfaces, with $r \leq 2$.
(ii) When $r \leq 2$, $r' \leq 2$ and $(p,q,r) \neq (p',q',r')$, $N_{p,q,r}$ and $N_{p',q',r'}$ are not topologically equivalent.

Thus, we shifted from a classification of surfaces to an analoguous classification of subdivisions, for which we proved part *(i)*. In order to get back to surfaces, we need to make the following fundamental classical hypothesis:

Hypothesis. Two surfaces that are subdivided by the same 2-gmap are homeomorphic.

Indeed, for any surface S there is (by our definition of surfaces) a 2-gmap M that subdivides it. According to the theorem, M also subdivides one of the $P_{p,q,r}$. Because of the hypothesis, S is homeomorphic to $P_{p,q,r}$. Thus any surface S is homeomorphic to one of the $P_{p,q,r}$, which proves part (i) of the theorem of classification of surfaces. Were the hypothesis not to hold, we would be left with a (still interesting) theorem of classification of subdivisions according to what they can subdivide.

5 Coq specification

Now that we have introduced the mathematical notions that appear in our work, let us see how they are expressed in Gallina, Coq's specification language that allows to write terms of the Calculus of Inductive Constructions [1].

5.1 Binary relations

Binary relations are only used to model the α_k, but, in order to be as modular as possible, they are specified separately. Actually, this specification of relations is an extension of [42] [44]. It can only deal with relations on objects of the same type.

Binary relations may be seen as two-place predicates. Thus, in Gallina, we specify the type of binary relations on a type as an *alias* for the type of two-place predicates on this type, with command `Definition`.

> DEFINITION 1 (BINARY RELATION) : Let E be a set. A *binary relation* on E is a two-place predicate on E, i.e. a function that associates a proposition to any two elements of E:

[1] We used the Coq version 7, the syntax has evolved in later versions

```
Definition relation : Set → Type
   := λE:Set E → E → Prop
```

As evidenced by the λ, the Gallina syntax is quite close to that of functional languages such as ML, Caml or Haskell. We have slightly simplified this language here by adding usual mathematical symbols in order to make terms more readable to neophytes.

This command declares symbol `relation` as an abbreviation for higher-order lambda-term `λE:Set E → E → Prop`. The notation `x:T` is a type judgement meaning "term `x` is of type `T`".

`Prop` is the builtin type for propositions. Propositions are not booleans; booleans are a two-value concrete type whose values are often associated to propositions according to their truth or falsehood. Symbol $\rightarrow$ is (from an intuitive point of view) overloaded, it represents both the connector to build function types and logical implication. Builtin type `Set` is that of concrete types, the types of objects that we want to build and use, such as numbers, lists, maps, darts, etc. Builtin `Type` is the type of `Set`, of `Prop`, and of functions into either of these types.

The definition explicitly states that the type of `relation` is `Set → Type`, the type of functions from `Set` into `Type`. Thus, an object of type `relation` is a function that, when applied to a concrete type `E` (the type itself, not an object of this type), yields the type `E → E → Prop` of two-place predicate on `E`.

For instance, `nat` being the predefined type for natural numbers, `relation nat` is a shortcut for the redex `λE:Set E → E → Prop nat`, which β-reduces to `nat → nat → Prop`, the type of binary predicates on `nat`.

Usual properties of relations are also specified with the definition mechanism:

DEFINITION 2 (INJECTIVITY) : Let `E` be a set and `R` a relation on `E`. `R` is *injective* if equality of images by `R` implies (syntactical) equality of arguments:

```
Definition injective : (∀E:Set) (relation E) → Prop
      := λE:Set λR:(relation E)
         (∀x,x',y:E) (R x y) → (R x' y) → x=x'.
```

DEFINITION 3 (INVOLUTIVITY) : Let `E` be a set and `R` a relation on `E`. `R` is *involutive* if for any element, an image of an image of this element is the element itself:

```
Definition involutive : (∀E:Set) (relation E) → Prop
      := λE:Set λR:(relation E)
         (∀x,x',y:E) (R x y) → (R y x') → x=x'.
```

In Coq, the usual ordering on natural numbers is named `lt`, its type is *nat* → *nat* → *Prop*. According to our definitions, the injectivity of `lt` should be represented by term `(injective nat lt)`. However, notice that *nat* is redundant: indeed, the first argument of `injective` is the type on which its second argument is a relation. Thus, as `lt` is a relation on natural numbers, the only possible first argument of `injective` is *nat* (or any equivalent type). Coq is able to infer this kind of redundant type arguments if asked to. With this facility, `(injective lt)` becomes a valid term. Be aware that all our further definitions use this facility.

In the same way, we define functionality (`functional`), irreflexivity (`irreflexive`), the proprety of being the identity (`identical`), as well as the composition of two binary relations (`composition`, of type *(∀E:*Set*) (relation E) → (relation E) → (relation E)*), partial surjectivity (`surjective`), the property of being a permutation (`permutation`) on a subset of *E*, and the property of being an involution (`involution`).

5.2 Darts

Little is said about darts in our previous sections, except that their type is *dart*. Thus, we declare them as an abstract type, in order to remain as generic as possible.

PARAMETER 1 (DART) : there exists a type of darts called `dart`:
`Parameter` *dart* : *Set*.

This declaration adds a new object, *dart* of type *Set* , into the environment. This differs from a *Definition* command, which simply creates an abbreviation to an already existing term. This declaration does not say much about *dart*. For our proofs, we need to make two assumptions on *dart* :

- *Darts may be compared.* More precisely, equality of darts should be *decidable*, i.e. given any two darts, we must assume that there is a way to find out whether they are equal or not. This is not the case for any imaginable set of darts, for instance if darts were represented by formulae over reals that use exponents: there is no algorithm that allows to check whether two formulae correspond to the same one real number.

  This assumption is necessary, as the CIC, being a constructive logic, does not allow reasoning by cases if not provided with a method to determine the relevant cases. Thus, in order to have special cases in our proofs when two darts are equal, we must assume that we are able to test their equality. This is an axiom of our specification:

  DEFINITION 2 (DECIDABILITY OF DART EQUALITY) :
  `Axiom EQ_DART_DEC : (∀x,y:`*dart*`) x=y∨¬x=y`

- *There are always unused darts available.* We will regularly need fresh darts in our algorithms, so we must assume that there is an infinite number of them, as well as have a way to reference them. To do so, we assume that there is an injection `idg` (for *injective dart generator*) from natural numbers into `dart`, thus ensuring that there is at least as many darts as there are naturals:

  PARAMETER 3 (DART GENERATOR) :

  ```
  Parameter idg : nat → dart.
  ```

  DEFINITION 4 (INJECTIVITY OF idg) :

  ```
  Axiom IDG_INJ : (∀n,n':nat) (idg n)=(idg n') → n=n'
  ```

5.3 Sewings

A sewing is a triplet made of a natural number (the dimension of the sewing, not bounded by 2 in the beginning) and the two darts that are sewn:

DEFINITION 4 (SEWINGS) :

```
Inductive sw : Set := c : nat → dart → dart → sw.
```

The type of sewings `sw` is defined as an *inductive type*, i.e. the smallest type that contains all the terms that can be built using only its *constructors*. Type `sw` only has a single constructor, a three-argument function that takes one `nat` and two `darts` to create a sewing. Thus, if for instance `s`, `t` and `u` are darts, then `(c 0 s t)`, `(c 3 s u)`, `(c 2 u u)` and `(c 2 t s)` are four sewings. Constructors are assumed to be *distinct* and *injective*, thus those four sewings are distinct as the arguments of `c` are distinct (unless `s=t=u`, in which case the last two sewings are equal). The fact that a sewing can be built only using `c` is exploited by our first lemma:

LEMMA 1 (INVERSION OF SEWINGS) : any sewing is an image of `c`:

```
Lemma SW_INV : (∀s:sw)
   (∃n:nat |(∃x,y:dart | s=(c n x y)))
```

This kind of lemma can be linked to the elementary selectors for a type in a traditional programming language. They will be used to manipulate `sw` while building proofs of propositions related to sewings.

5.4 Free maps

Free maps are the simplest type we use to represent gmaps. A free map is simply a finite collection of darts and sewings, defined much like an ML list:

DEFINITION 5 (FREE MAP) :

```
Inductive fmap : Set :=
    v : fmap
  | i : dart → fmap → fmap
  | l : sw → fmap → fmap
```

This is an inductive type definition, which means that the type of free maps is the smallest type of terms that can be built with constructors `v`, `i` and `l`. In other words, a free map is either the empty map `v`, a pair preceded by `i` of a dart and a free map (intuitively the insertion of a dart into a free map), or a pair preceded by `l` of a sewing and a free map (intuitively the addition of a sewing into the map). When an inductive type is defined, Coq automatically generates a *structural induction scheme* for it. Here, it allows to prove properties on free maps by showing that they are true for the empty map `v` and stable by dart insertion with `i` and sewing insertion with `l`. Most proofs on *fmap* use this principle.

We can then define two basic selectors as predicates on *fmap*, using a similar inductive construction:

DEFINITION 6 (SUCCESSORS) : Let `k` be a dimension, `x` and `y` two darts and `m` a free map. Dart `y` is a `k`-successor of `x` in map `(l (c k x y) m)`; this property is stable by dart and sewing insertions:

```
Inductive succ : nat → fmap → dart → dart → Prop
   := SUCC_L_X : (∀k:nat; ∀m:fmap; ∀x,y:dart)
         (succ k (l (c k x y) m) x y)
   | SUCC_I : (∀k:nat; ∀m:fmap; ∀x,y:dart) (∀d:dart)
         (succ k m x y)
               → (succ k (i d m) x y)
   | SUCC_L : (∀k:nat; ∀m:fmap; ∀x,y:dart) (∀s:sw)
         (succ k m x y)
               → (succ k (l s m) x y)
```

The other selector is `exd`, of type *dart → fmap → Prop*, which describes the existence of a dart in a free map. An important variation on `succ`, the term of which is too long to print here, is the following:

DEFINITION 7 (ALPHA) : `(alpha n x m)` is either a `k`-successor of `x`, or `x` itself if there is none:

```
Definition alpha : nat → dart → fmap → dart := ...
```

Its main purpose is to turn the predicate-based definition of succession into an easier-to-use functional one. Provided that a free map satisfies all the constraints given in the definition of gmaps, `alpha` has the same properties as the α_k. A number of other things related to free maps are also defined, including all the notions given in the previous section (boundaries, cells, paths),

but also an *observational equality relation*. Being defined like lists, free maps are incidentally ordered. But this order is artificial, as equality between free maps should be determined only by the darts and sewings in each map, not the order in which they were added, which is unavoidably taken into account by the builtin Coq equality, since it is only the syntactical equality between terms of the same type. Thus we introduce our own equality:

DEFINITION 8 (OBSERVATIONAL EQUALITY ON FREE MAPS) : two free maps are *observationally equal* if the behaviour of `alpha` and `exd` is the same on both maps

```
Definition ≅ : fmap → fmap → Prop :=
   λm,m':fmap
            ((∀x:dart) (exd x m) ↔ (exd x m'))
            ∧ ((∀k:nat; ∀x:dart)
                  (alpha k x m)=(alpha k x m'))
```

The definition of gmaps using free maps is heavily underconstrained, which allows many situations forbidden by the definition of gmaps. A dart may be sewn at any dimension to any dart, including itself or darts not previously inserted into the map, or to several darts at the same dimension. In order to capture precisely generalized maps, we need to introduce a new type, *gmap*.

5.5 Generalized maps

An object `m` of type *gmap* is basically a free map that has been proved to be *well-formed*, i.e. to be satisfying the fundamental properties used in the definition of generalized maps of dimension 2 given in the previous section. The most natural approach in Gallina is to define *gmap* as the type of pairs of a free map (called the *support*) and a *formal proof* that the free map satisfies a *well-formedness predicate* that expresses said fundamental properties of 2-gmaps. This is a *dependent* pair, as the second element is a proof of a proposition parametrized by the first element. The well-formedness predicate is defined as:

DEFINITION 9 (WELL-FORMEDNESS OF FREE MAPS) : a free map `m` is well-formed if, for any `k`, relation `(succ k m)` is an involution, if for any `k`$\leq$`1` relation `(succ k m)` is irreflexive, if relation `(succ 0 m)`$\circ$`(succ 2 m)` is involutive, and if for any `k` such that `k`$\geq$`3` relation `(succ k m)` is the identity:

```
Definition wf : fmap → Prop
   := λm:fmap
      ((∀k:nat) (involution (succ k) m))
      ∧ ((∀k:nat) k<=1 → (irreflexive (succ k m)))
```

```
∧ (involutive (composition
                (succ 0 m) (succ 2 m)))
∧ ((∀k:nat) 3<=k → (identical (succ k m)))
```

The last condition is added to effectively limit the number of relevant α_k relations to $k = 0, 1$ or 2: at higher dimensions, all darts are considered to be sewn to themselves. Thus, the type of 2-gmaps is:

DEFINITION 10 (GMAP) : an object of type *gmap* is a pair made up of a free map `m` and a proof that `m` is well-formed, i.e. a term of type *(wf m)* :

```
Inductive gmap : Set
    := mkg : (∀m:fmap; ∀w:(wf m)) gmap
```

Also specified is a selector to get the first element of the pair, called `gsupport`: *gmap* → *fmap*.

Now we have a type that, by construction, exactly encompasses 2-gmaps. However, this type is harder to handle than free maps when proving theorems, as there is no (useful) structural induction scheme on this type; being deprived of induction really hurts, so we had to find some kind of induction scheme ourselves. To do so, we used another common definition of generalized maps.

5.6 Sewn-cell maps

Generalized maps are often defined in a recursive and incremental manner, using a simple-cell-sewing operation that we note `sm`, of type *nat* → *fmap* → *dart* → *dart* → *fmap* [33] [34]. The integer is the dimension of the sewings, and the dimension of the affected simple cells plus 1. For instance `(sm 2 m x y)` 2-sews each dart of free map `m` in the simple 1-cell incident to `x` to the corresponding dart in the simple 1-cell incident `y`. More precisely, `x` is sewn to `y`, and any dart `x'` in the simple 1-cell incident to `x` is sewn to the dart obtained by following from `y` one of the paths that lead from `x` to `x'`. Figure 6 shows what `sm` does on simple examples.

Free maps built using only `sm` are said to be *well-constructed*. More precisely, a well-constructed map is built by:

- starting with the empty map;
- adding all the darts;
- making all 0-sewings with `sm`;
- making all 1-sewings with `sm`;
- making all 2-sewings with `sm`.

Well-constructedness is expressed in a predicate. The type of sewn-cell maps is

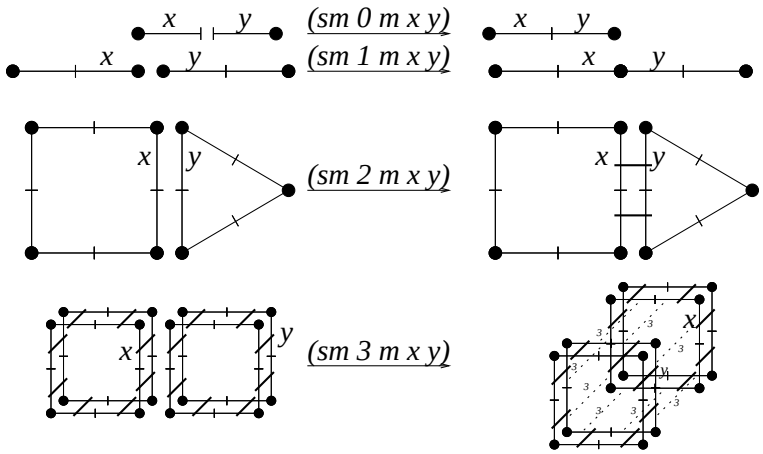


Fig. 6. Simple cell sewing at dim. 0,1,2,3

called *smap* . Like *gmap* , *smap* is the set of pairs composed of a support free map and a proof term of well-constructedness of the support.

The main advantage of *smap* is that such a map is built in an incremental manner, which allowed us to prove a useful induction scheme for this type. It roughly states that if a property is true for a map without sewings and is preserved by `sm`, then it is true for any *smap*.

Our actual Coq specification is more general than what is shown here, in that well-formedness and well-constructedness are parametrized by a dimension, allowing us to work with gmaps of any dimension. Our first major result in this specification is that any *gmap* support is *observationally equal* to a *smap* support (at any dimension), which means that well-constructedness amounts to well-formedness, order of insertions and sewings aside. Thus, we have formally proved that the two usual definitions of generalized maps are indeed equivalent.

From a practical point of view, it allows us to switch between the gmap types at will while proving a property, provided it is preserved by observational equality, which is the case for all geometrically meaningful properties. The main consequence is that we now can indirectly reason by structural induction on *gmap* , by switching to and then back from *smap*.

6 Conservative operations

From now on, we focus on surfaces and 2-gmaps. As we explained in Sect. 3, conservative operations are at the core of our proof of the classification theorem. Being conservative means that they are expected to preserve the set of subdivided surfaces. In order to maximize confidence in the conservativity

of these operations, they are kept as simple, local and constrained as possible.

Building operations on types *gmap* and *smap* are very hard to directly specify: they must turn a dependent pair into another pair, which in part consists in turning a proof term into another proof term, something that is quite difficult to perform directly. The technique we use is to specify the effect of an operation at the simple free map level, then prove a lemma stating that, provided preconditions are satisfied, the operation preserves well-formedness (resp. well-constructedness). The free map operation and lemma are then combined to form an operation on *gmap* or *smap*. Actually, the possibility of using this method to easily build operations on complex types was our main motivation behind the *fmap* type.

6.1 Stretching of an open vertex `stro`

We define an *open vertex* as a vertex incident to a boundary. The stretching operation splits such a vertex into two vertices connected by an open edge (see Figure 7). We start by giving the operation in the *fmap* universe. It makes

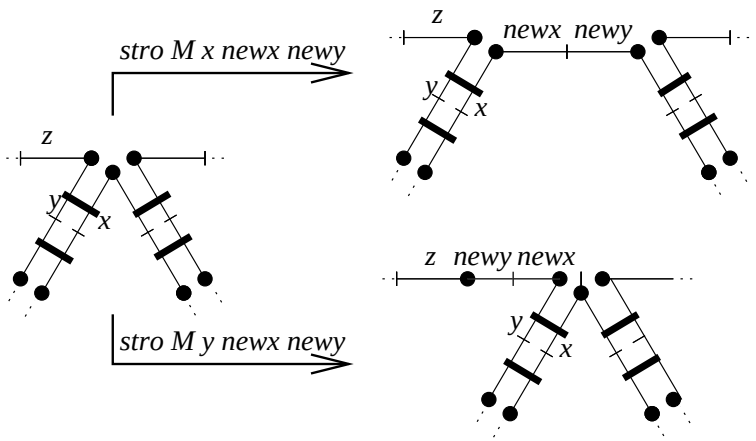


Fig. 7. Example of stretching of a vertex with an open edge

use of `unsm`, the reverse of `sm`.

DEFINITION 11 (STRETCHING A VERTEX WITH AN OPEN VERTEX) : let `m` be a free map and `x`, `newx` and `newy` three darts. This function rips the sewings of `x` at dimension 1, then adds an edge made up of `newx` and `newy`, then 1-sews `newy` to `(alpha 1 x m)` and `newx` to `x`

```
Definition stro : fmap → dart → dart → dart → fmap
  λm:fmap λx,newx,newy:dart
    (sm 1 (sm 1 (sm 0 (i newx (i newy (unsm 1 m x)))
    newx newy) newy (alpha 1 x m)) newx x).
```

The preconditions have two purposes: to ensure first the preservation of well-formedness by the operation, and second the conservativity of the operation, by verifying that the operation is only used to do what it is intuitively supposed to. A precondition is expressed in one predicate.

DEFINITION 12 (PRECONDITION FOR STRETCHING OF AN OPEN VERTEX) : let `m` be a free map and `x, newx` and `newy` three darts. These parameters may be used for open vertex stretching if `m` is of dimension 2, if `x` belongs to `m` and is incident to an open vertex, and if `newx` and `newy` are distinct and do not belong to `m`:

```
Definition pre_stro: gmap → dart → dart → dart → Prop
    := λm:gmap λx,newx,newy:dart
       (exd x m) ∧ ¬(exd newx m) ∧ ¬(exd newy m)
       ∧ newx ≠ newy ∧ (openvertex m x)
```

Note that the first argument, of type *gmap,* is used with predicate `exd`, which expects type *fmap*. The typing problem is avoided because, using the *coercion* facility, we previously told Coq to automatically cast *gmap* into *fmap* when needed using `gsupport` (Sect. 5.5). Thus, in this formula, `(exd x m)` implicitly stands for `(exd x (gsupport m))`. This allows us to use free map operations on *gmap* in a very natural way. In the same way, *smap* is implicitly cast into *gmap,* and by transitivity into *fmap,* so we can apply *gmap* and *fmap* operations to *smap*. We can now prove with Coq that this predicate ensures preservation of well-formedness:

LEMMA 2 (PRESERVATION OF WELL-FORMEDNESS WITH STRO) : if `pre_stro` is satisfied for a set of arguments, using them for a stretching preserves well-formedness:

```
Lemma WF_STRO : (∀m:gmap; ∀x,newx,newy:dart)
          (pre_stro m x newx newy)
          → (wf 3 (stro m x newx newy))
```

For space concerns, it is absolutely impossible for us to present here a formal proof of any theorem, as they are either far too long if given in full detail, or impossible to follow if abridged.

As a CIC lemma is actually a function that transforms proof terms of the hypotheses into a proof term of the conclusion, we use the above lemma to build a *gmap* from another *gmap,* thus obtaining a version of `stro` that yields a *gmap*. Note that Coq automatically infers the first element of the pair from the second, which is thus the only one to provide:

DEFINITION 13 (STRETCHING AN OPEN VERTEX IN A GMAP) : if `pre_stro` is satisfied for a set of arguments, the lemma `WF_STRO` is used to build a *gmap* pair corresponding to the result of stretching:

```
Definition gstro : λm:gmap λx,newx,newy:dart
```

```
        (pre_stro m x newx newy) → gmap
:= λm:gmap λx,newx,newy:dart
   λp:(pre_stro m x newx newy) (mkg (WF_STRO p))
```

This is a little technical, and might be difficult to understand for readers not accustomed to this formalism. The point was simply to show how a lemma can be used to build a concrete object.

All other conservative operations are built with the same method: define the operation on *fmap*, put all preconditions into a predicate, prove the preservation of well-formedness and use this proof to extend the operation to *gmap*.

6.2 Removal of an open edge

This is the converse of `stro` (and as such is not pictured): it *removes* unneeded edges from boundaries. The precondition ensures that the edge itself is open and that there is another edge on its boundary, so that removing this edge will not remove the entire boundary.

6.3 Sliding along a boundary

This applies to places in the map where a section of the boundary has been connected to another edge by 2-sewing that edge to one of the edges of the boundary. This operation simply sews one of the edges right next to it instead, intuitively *sliding* the 2-sewing along the boundary sections (see Figure 8, where x is a dart of the edge that loses its 2-sewings in the process).

The precondition states that vertices on either side of the connecting edge should be open, and that both sides of the boundary section contain at least two darts, in order to make sure that no boundary gets suppressed in the process and that the configuration is really that of the figure.

6.4 Merging two faces

This operation *merges* two distinct faces that have at least one common edge by removing that edge and connecting what is left (see Figure 9, where x is a dart of the removed edge).

The precondition ensures that the two faces are indeed distinct and that both

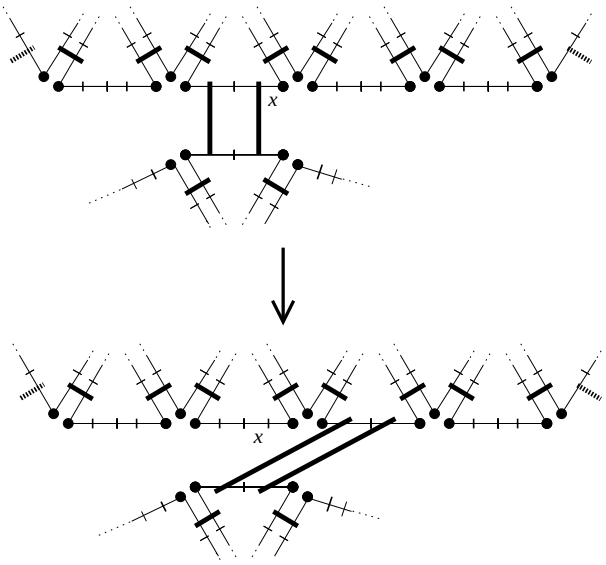


Fig. 8. Example of sliding

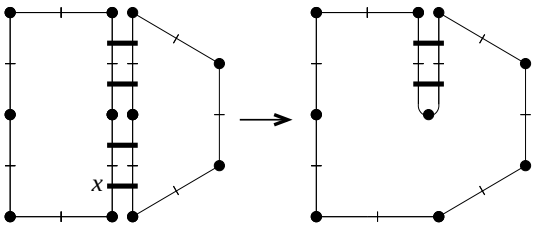


Fig. 9. Example of merging

of them contain more than one edge (one-edged faces do not have anything left to connect).

6.5 Absorption of a one-edge face

This operation is the previous one in the case where one (and only one) of the faces is made of a single edge. In that case, the face is simply removed, or *absorbed*, by the larger face (see Figure 10, which also shows bump removal,

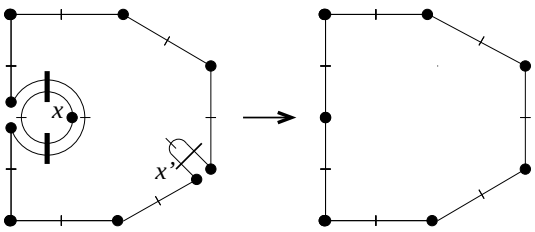


Fig. 10. Example of absorption and bump removal

and where x is a dart of the one-edged face).

The precondition ensures that the two faces are distinct, that one of them has a single edge and that the other has more.

6.6 Bump removal

This operation removes a *bump*, i.e. an edge closed on itself, which has no actual relevance from a topological point of view (see Figure 10, where x' is a dart of the bump). There are no additional precondition.

6.7 Dart renaming

That is the only global operation: it changes the name of all the darts in the map using a renaming function of type $\texttt{dart} \to \texttt{dart}$. At some point, the darts we manipulate will need to bear imposed names to be conform to a normalization. This function must be injective on the set of darts of the original map, otherwise some darts will be confused in the resulting map.

6.8 Slitting a vertex

The *slitting* operation removes extraneous 2-sewings. It rips the 2-sewings that tie together the two sides of a closed edge the end vertices of which are respectively open and closed (see Figure 11, where x belongs to the opened

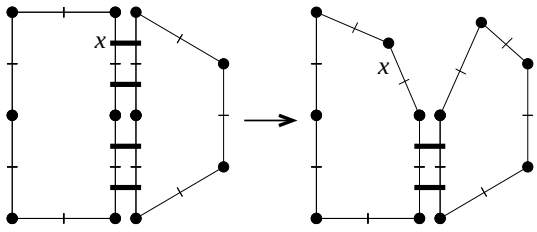


Fig. 11. Example of slitting

edge).

The precondition simply checks that one of the end vertices is open and the other closed.

7 Normal maps and the trading theorem

7.1 Normal maps

Now that we have the conservative operations, we know precisely our topological equivalence relation between 2-gmaps. Remember that two 2-gmaps are *topologically equivalent* provided one can be obtained from the other using only a combination of any of the eight conservative operations while always satisfying the preconditions. Topologically equivalent 2-gmaps subdivide the same surfaces.

The next step is to prove that any open connected 2-gmap is equivalent to one of the normal maps $N_{p,q,r}$. Now we need to precisely describe these maps. To find a good candidate for $N_{p,q,r}$, we start with a map that obviously subdivides a surface with p punctures, q handles and r twists. Then we apply as many conservative operations as possible to it so that it becomes as small as possible, making it easier to handle. In the end, we obtain the map on Figure 12. This

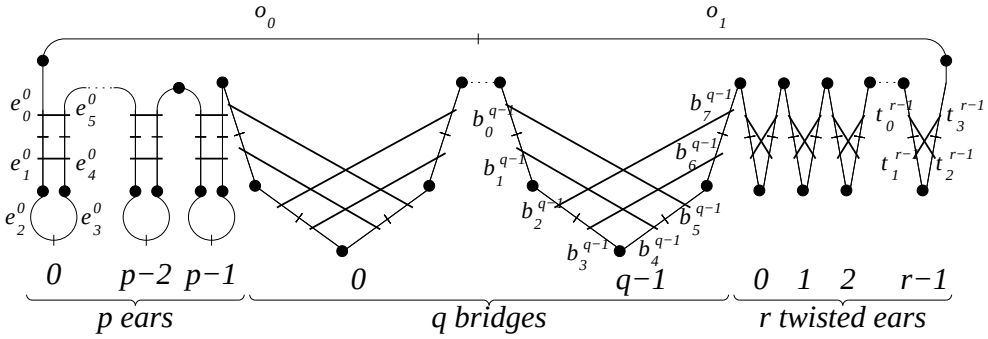


Fig. 12. General form of a normal map

map is made of one open edge (the horizontal upper edge on the figure) that represents the mandatory boundary of the map. Then, from left to right, there are p vial-shaped *ear* patterns (one for each puncture), q larger *bridge* patterns (one for each handle) and r *twisted ear* patterns (one for each twist). The names of the patterns are inspired by Griffiths [29].

A normal map is made of a single cycle of alternately 0- and 1-sewn darts, some of which have been 2-sewn to form the patterns. Each ear pattern is made of 6 consecutive darts. The darts of the k-th ear pattern are denoted e_i^k, with i between 0 and 5, according to the order they appear on the cycle. This means that in ear pattern k, e_0^k and e_5^k are 2-sewn, as well as e_1^k and e_4^k, and e_2^k and e_3^k are both 2-sewn to themselves.

In the same way, the 8 darts that make up bridge k are denoted b_i^k, with i between 0 and 7, and the 4 darts that make up twisted ear k are denoted t_i^k,

with i between 0 and 3. The two darts that form the mandatory boundary are denoted o_0 and o_1. Thus, map $N_{p,q,r}$ contains $2 + 6p + 8q + 4r$ darts.

7.2 Normal maps in Coq

Normal maps are entirely characterized by p, q and r. Thus, in Coq, we should be able to build a normal map, of type ***smap***, using only the three natural numbers. The darts in normal maps will be obtained using the *idg* dart generator. Thus, dart o_0 and o_1 are actually `(idg 0)` and `(idg 1)`, e_i^k is `(idg (2+6k+i))`, b_i^k is `(idg (2+6p+8k+i))`, and t_i^k is `(idg (2+6p+8q+4r+i))`.

To build the map, we write recursive Coq functions that deal with one step of the construction process of a normal map:

- `(dartmap n)` yields a the 0- ***smap*** only containing the darts `(idg 0)` to `(idg (n-1))`;
- `(edgemap n)` yields the 1- ***smap*** made by taking `(dartmap 2n)` and 0-sewing every `(idg 2k)` dart to `(idg (2k+1))`. This is a collection of n edges;
- `(cycle n)` yields the 2- ***smap*** made by taking `(edgemap n)` and 1-sewing every `(idg (2k+1))` to `(idg (2k+2))`, with a special case with `(idg (2n-1))` that is instead 1-sewn to `(idg 0)`. Thus, we obtain a cycle of alternately 0- and 1-sewn darts;
- `(addears p q r)` yields `(cycle (1+3p+4q+2r))` in which the edge incident to e_0^k is 2-sewn to e_5^k using `sm`, for each `k` under `p`. This takes care of the ear patterns;
- `(addbridges p q r)` yields `(addears p q r)` in which the edge incident to b_0^k is 2-sewn to b_5^k, and the edge incident to b_2^k is 2-sewn to b_7^k, for each `k` under `q`. This takes care of the bridge patterns;
- `(makenmap p q r)` yields `(addbridges p q r)` in which the edge incident to t_0^k is 2-sewn to t_1^k, for each `k` under `r`. This takes care of the twisted ear patterns, thus completing the normal map.

The normal maps are indeed determined only using the numbers p, q and r. Thus, in our Coq specification, the type ***nmap*** of normal maps is simply the type of *triplets of natural numbers,* with selectors named `nmp`, `nmq` and `nmr`. We then use the coercion facility to automatically cast such triplets into ***smap*** when needed using function ***makenmap***. Thus, we can transparently use ***nmap*** objects as free maps.

7.3 The trading theorem

As we mentioned in the introduction, part (i) of the classification theorem is split again into two parts. The first half (the *normalization theorem*) states that any map belongs to one of the classes (p, q, r), the second half (the *trading theorem*) identifies all classes to classes with $r \leq 2$. We first deal with the latter, which is much simpler. In our specification, the trading theorem states that:

Theorem 12 *Normal map $N_{p,q,r}$ is equivalent to $N_{p,q+1,r-2}$ if $r > 2$.*

Thus, this theorem actually allows to *trade* two twists in a normal map for an additional handle, provided there were at least three twists in the original map.

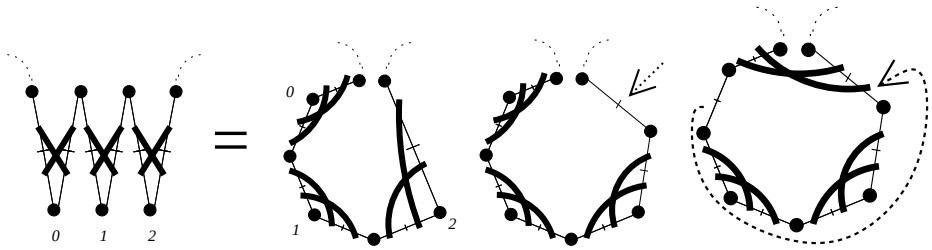
Our proof is very straightforward, it is in fact similar to the one in [25]. It simply consists in applying 10 conservative operations to $N_{p,q,r}$ and proving that the result is $N_{p,q+1,r-2}$. The operations are the following, with new darts called i_0, i_1, i_2 and i_3:

(1) stretch the vertex incident to t_3^2 with an open edge made of i_0 and i_1;
(2) slide the 2-sewing of t_3^0;
(3) stretch the vertex incident to t_1^2 with an open edge made of i_2 and i_3;
(4) slide the 2-sewing of t_1^0;
(5) slide the 2-sewing of t_3^1;
(6) slide the 2-sewing of t_3^2;
(7) slide the 2-sewing of t_3^0;
(8) remove the open edge incident to t_2^0;
(9) remove the open edge incident to t_2^1;
(10) rename the darts properly (as dart names in a normal map are imposed).

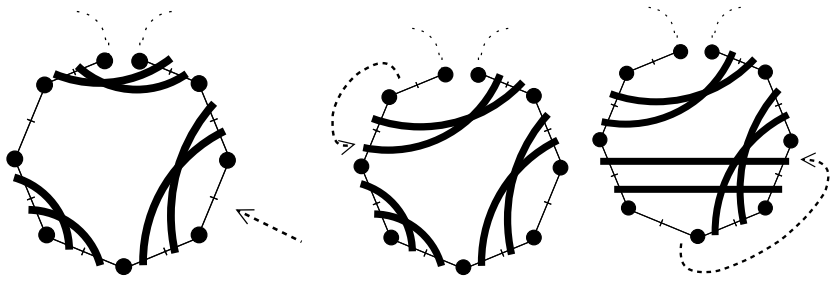
Figure 13 illustrates these operations. Each subfigure shows the result of the operation written in the caption. A dashed line illustrates what the latest operation did, by pointing to the new edge in case of vertex stretching, pointing to the vertex where a removed edge stood before, or by showing the movement of the sewing in case of sliding. The i_k are generated using `idgen`, a function based on `idg` that yields darts that do not belong to any given map. All steps are formally handled in a similar way. Let us use the first step as an exemple.

7.4 Step 1

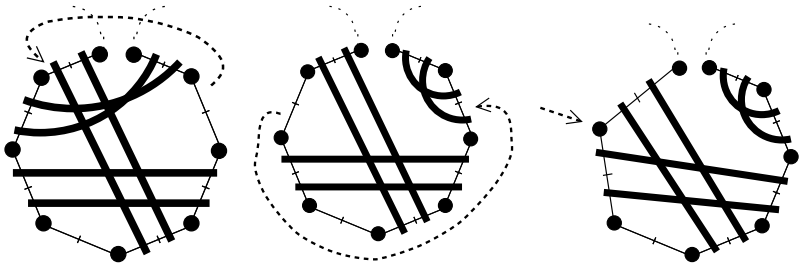
There are three substeps for each step:



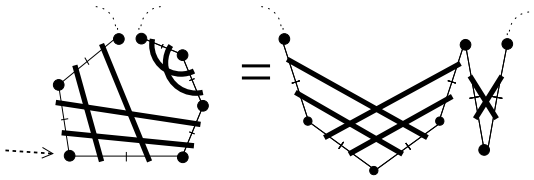
(A) Two views of the starting normal map (B) Step 1: stretch the vertex incident to t_3^2 (C) Step 2: slide the 2-sewing of t_3^0



(D) Step 3: stretch the vertex incident to t_1^2 (E) Step 4: slide the 2-sewing of t_1^0 (F) Step 5: slide the 2-sewing of t_3^1



(G) Step 6: slide the 2-sewing of t_3^2 (H) Step 7: slide the 2-sewing of t_3^0 (I) Step 8: remove the edge incident to t_2^0



(J) Step 9: remove the edge incident to t_2^1

Fig. 13. Steps of the trading theorem

(1) *prove that the map obtained after the previous step satisfies the precondition for the next operation.* Here, there is no previous step, so we take the starting normal map:

LEMMA 3 (PRECONDITION FOR STEP 1 IS SATISFIED) : stretching the vertex incident to t_3^2 with two darts that do not belong to `m` is allowed in any normal map with $r > 2$:

`Lemma pre_step1 : (∀m:nmap)`

```
                (nmr m)>2 -> (pre_stro m x (idgen m 0) (idgen m 1))
```

(2) *perform the operation:*

DEFINITION 14 (RESULT OF STEP 1) : perform the step

```
Definition step1 : λm:gmap λp:((nmr m)>2)
                (gstro (pre_step1 p))
```

(3) *build a library of lemmas describing* `exd`*'s and* `alpha`*'s behaviours after application of* `step1`. It will be used in the following steps.

After step 9, the map has the same structure as $N_{p,q+1,r-2}$, but the darts are named differently, thus we add a renaming step so that the names match. Once this tenth step is over, we prove that the resulting map is observationally equal, and thus topologically equivalent, to $N_{p,q+1,r-2}$. Combining all the intermediate results, we conclude that $N_{p,q,r}$ is equivalent to $N_{p,q+1,r-2}$, thus ending our formal proof of the trading theorem. We would like to stress that the whole proof was entirely checked by the Coq system.

8 The normalization theorem

8.1 Quasi-normal maps

This theorem is *much* more difficult to prove. Informally, it states:

Theorem 13 *Any open connected 2-gmap is topologically equivalent to one of the normal maps.*

Our proof is in two parts that are joined together using a new subtype of `smap`, the type of *quasi-normal* maps.

Definition 14 *A* `smap` `m` *of dimension 2 is* (p, q, r)-quasi-normal *if:*

- *map* $N_{p,q,r}$ *is* included *in* `m`, *i.e. all its darts and sewings are in* `m`, *with the exception of the 1-sewings of* o_0 *to* $\alpha_1(o_0)$, *which is replaced here by another 1-sewing, thus connecting this* normal section *of* `m` *to the rest of* `m`;
- `m` *has only one face;*
- *all vertices in* `m` *are incident to a boundary;*
- `m` *contains no bump.*

A quasi-normal map is essentially a map that "contains" a normal map and that also has several useful properties. A very important dart in a quasi-normal map is what we call the `limit` dart, denoted *ld*: it is the dart that comes right after the normal part of the `qmap`. Figure 14 shows one such map and its limit dart. Surrounded on the figure is the non-normal part of the quasi-map.

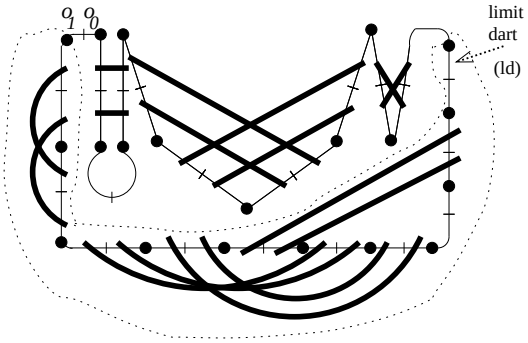


Fig. 14. Example of (1,1,1)-quasi-normal map

The type of quasi-maps is *qmap*, it is defined much like previously as a triplet of a *smap*, a *nmap*, and a proof that the *smap* satisfies the above properties with the *nmap* as the normal section.

The two parts of the normalization theorem are:

- *the cleaning theorem*, that states that any open connected 2-gmap is equivalent to a quasi-normal map the normal section of which is $N_{0,0,0}$;
- *the absorption theorem*, that incrementally "grows" the normal section of the quasi-normal map, until it encompasses the whole map.

In the following subsections, we give a sketch of the proofs.

8.2 Nœtherian induction

Both parts are proved using the proof technique called *Nœtherian induction*, which is an extension of structural induction. Coq uses a variant of this technique based on the builtin notion of *accessibility*.

Let E be any set and R be a binary relation in E: $R \subseteq E \times E$. As usual, for any $x, x' \in E$, we denote $x\ R\ x'$ the fact that there exists an R-arc from x to x'. We say that x is a R-*predecessor* of x' and x' is a R-*successor* of x.

Intuitively, an element $x \in E$ is said to be R-*accessible* if and only if any descending R-chain beginning by x, i.e. $x = x_1, x_2, \ldots, x_k, x_{k+1}, \ldots$ with $x_2\ R\ x_1, \ldots, x_{k+1}\ R\ x_k, \ldots$, is finite. The Calculus of Inductive Constructions expresses that with the notion of *accessibility*.

Definition 15 *An element $x \in E$ is said to be* accessible *by R, or R-*accessible*, if all its R-predecessors are themselves R-accessible. If x has no R-predecessor, x is said to be* immediately R-accessible.

It is clear that, if x is immediately R-accessible, then it is R-accessible. For instance, let $E = \mathbb{N}$ be equipped with the usual natural strict order $<$. Then, 0 is immediately $<$-accessible since it has no predecessor, 1 is $<$-accessible because its only $<$-predecessor is 0, 2 is $<$-accessible because its only $<$-predecessor is 1, etc. Then all the natural numbers are $<$-accessible.

If $E0$ is the set of the elements of E which are immediately R-accessible, the set of all the elements of E which are R-accessible is exactly $(R^*\ E0)$, where R^* as usual denotes the reflexive transitive closure of R.

Definition 16 *The set E equipped with R is said to be* well-founded*, or* Nœtherian*, if every $x \in E$ is R-accessible.*

If the context is clear, it is simply said that R is *well-founded* or *Nœtherian*. In this case, if we want to prove a property P_x for any $x \in E$, the *Nœtherian induction principle* states that we can assume "for free" that P_y is true for any $y \in E$ such that $y\ R\ x$.

In our example, $\mathbb{N}$ equipped by $<$ is Nœtherian, since $E0 = \{0\}$, and $(<^* \{0\}) = \mathbb{N}$. Then the Nœtherian induction principle for $(\mathbb{N}, <)$ corresponds to the classical "general induction": to prove P_n for any n, assume P_k for $k < n$ and use that to prove P_n.

Then, the following result can immediately be proved using Nœtherian induction.

Theorem 17 (E, R) *is* Nœtherian *if and only if, for every $x \in E$, there is no infinitely descending R-chain from x.*

In our map case, E is a type of maps, *smap* for the cleaning part, *qmap* for the absorption part. Proving that the processes realizing the two parts by map transformations are finitely terminating needs suitable well-founded relations on these map types.

8.3 The cleaning theorem

This theorem states that any open connected 2-gmap m is equivalent to a quasi-normal map whose normal section is $N_{0,0,0}$. The proof of the cleaning theorem is split in five lemmas, each showing that m is equivalent to a map that satisfies one more property than in the previous step:

(1) step 1: m is equivalent to a map containing $N_{0,0,0}$;
(2) step 2: like step 1, but the map also contains no bump;
(3) step 3: like step 2, but the map also contains no one-edged face;

(4) step 4: like step 3, but the map also contains only one face;
(5) step 5: like step 4, but the map also only has open vertices, thus is quasi-normal.

Step 1 simply renames two darts of an open edge into o_0 and o_1, thus creating the normal section of the map: indeed, normal map $N_{0,0,0}$ only contains these two darts, so in order to be contained by a map, that map only needs to contain the open edge made of o_0 and o_1.

The four other steps ensure one of the other properties of the quasi-maps by repeatingly applying a conservative operation. The convergence is proved by Nœtherian induction. The corresponding relation is denoted $<_{cd}$, which is proved well-founded in `smap`.

Definition 18 *For any* `smap` m *and* m'*, the relation* $m <_{cd} m'$ *is satisfied if the number of closed darts is strictly lower in* m *than in* m'*.*

In step 2, we want to prove that any open connected 2-gmap m has an equivalent containing $N_{0,0,0}$ with no bump. We first use the induction principle to assume that there is such an equivalent for any map $m' <_{cd} m$. Then we test whether there is a bump in m. If not, then m itself is the sought equivalent. If there is one, we remove it with conservative operation `rmbump` and show that the resulting map m'' is smaller than m w.r.t. $<_{cd}$. Applying the induction hypothesis to m'', we find a map with the good properties that is an equivalent of m'', but then also of m by transitivity of equivalence, thus finishing the proof of step 2.

The other three steps, step 3 to step 5, work in the same way, respectively with operations `absorb`, `merge` and `slit`. Figure 15 shows each step of the

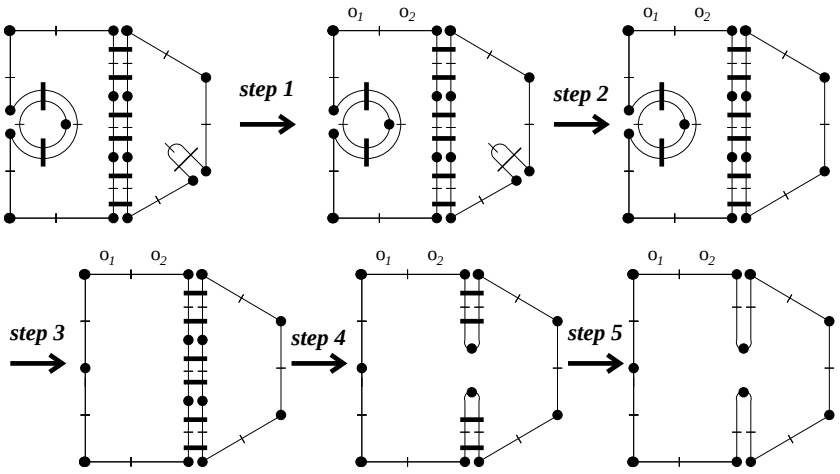


Fig. 15. Example of cleaning

cleaning starting from a sample 2-gmap. It is clear that the number of closed darts in the current map is strictly decreasing from step 2 to step 5.

This is by far the most difficult part of the proof, and also clearly the least intuitive. It only deals with quasi-normal maps. It is structured like step 2 of the cleaning theorem, except that it uses another relation, and that instead of simply checking whether the map contains bumps, it computes which of fourteen different classes the map belongs to, applying a different series of conservative operations each time.

The class of the map depends on its configuration near ld. For example, one corresponds to maps where ld is open, another to maps where ld is incident to a twisted ear pattern, another to maps where ld is incident to a bridge pattern, etc., with numerous subcases, that are too long to present in this paper. Then, for each class among the fourteen, an appropriate treatment is applied.

We prove that the process converges to a normal map by Nœtherian induction on a vector computed from the map. As each step in the absorption process lexicographically decreases the vector, and that the smallest vectors are shown to only belong to normal maps, we infer that any `qmap` can be normalized in a finite number of steps. The vector is made of 8 natural numbers $v_0, v_1, \ldots, v_7$ which we use to define well-founded relation $<_q$ on `qmap` :

Definition 19 *For any* `qmap` m *and* m', $m <_q m'$ *is satisfied if the vector of* m *is* lexicographically smaller *than that of* m'.

Each component v_i is a count of darts with a specific feature, or a number reflecting a characteristic of ld in the current map, e.g. a distance to a well chosen other dart. For instance, v_0 is the number of darts in the non-normal section, v_3 is the number of darts between ld and the closest closed dart x, v_1 is 0 if $x \in < \alpha_1 \circ \alpha_0 > (ld)$ and 1 otherwise, etc. We remind that ld is the dart that sits between the normal and non-normal parts of the qmap.

These are values that were empirically devised so that they worked well for our inductive proof. More specifically, the v_i are carefully chosen so that a single step in the absorption process will decrease one of the v_i without altering the v_j for $j < i$. Unfortunately, the complexity of the absorption process makes the definition of v_i themselves quite lengthy and complex. It is necessary to delve into the details of the theorem and its proof in order to understand why these v_i do indeed evolve this way, which makes it hard to describe them succinctly; even harder would be to give an intuitive idea as to why they behave as expected.

The only simple one to grasp is v_0, the number of darts in the non-normal section. We prove that all steps of the absorption theorem either decrease the number of such darts or leave it unchanged and decrease some other v_i. Fur-

thermore, enough v_0-decreasing steps will be applied so that it will eventually reach 0, the v_0 for a normal map. Similarly, we show that no step increases v_1 unless v_0 decreases at the same time, and so forth for the other v_i. In the end, all these results taken together prove that each step lexicographically decreases the vector until $v_0 = 0$, which by definition characterizes normality. In other words, each step brings the map closer to normality.

The strength of these proofs is to certify that these complicated operations are indeed correct and actually lead to the desired result, which cannot be ensured when writing a conventional program. For more detail about the steps and the vector, refer to [11].

9 Conclusion

In this article, we have shown the basic structure of a Coq specification of generalized maps, which topologically model combinatorial surfaces. We then have shown how we used it to adapt and formally prove the first part of the famous theorem of classification of surfaces applying to subdivisions instead, the subdivisions being classified according to the set of surfaces that they subdivide. This shows that the Coq system can actually be used in this field, although the size of our development (over 100,000 lines) [12] suggests that it could benefit from dedicated tactics and commands to relieve the user a little.

We now have a directly usable powerful specification of generalized maps that can be used for further studies. The most obvious use would be to extend our proofs in three directions:
- First, we must prove the second part of our classification theorem which asserts that any two subdivision classes are not confused. Following [26] for the closed surfaces, we must establish the link between the open surface equivalence and the triple (p, q, r) with $r \leq 2$, which can be admitted as the invariant of an open surface. We must prove that our elementary transformations preserve it, what seems easy for p and q, but more difficult for r, for which the orientability notion must be deepened. Then, two distinct normal forms necessarily correspond to inequivalent surfaces. A useful task will be to formally establish the relationship between q and χ, the Euler-Poincaré characteristic (Sect. 4), which is often used to classify surfaces.
- Second, we have to include the closed subdivisions. For instance we could assume, like Griffiths does for surfaces, that puncturing in a closed subdivision, conservatively transforming it, then closing back the puncture, is conservative as a whole.
- Third, we could specify embeddings in order to obtain surface classification from subdivision classification. This work involves to consider classical notions of topology, with continuity, homomorphisms, homology, probably by using an

axiomatic system for the real numbers.

This specification of gmaps could probably also be used as a basis for work in the discrete geometry field, where each voxel could be represented by a cubic map, which would then be combined to form discrete surfaces [2]. We could also use the *program extraction* facility of Coq to automatically generate from our proof a program that computes the (p, q, r) of any gmap. Moreover, a reasonable but still nontrivial modification of our proof could yield a certified program listing all the conservative operations used during normalization.

Finally, our long-term project is to revisit the foundations of the computational geometry, using combinatorial map concepts for topology and Coq for the specification and extraction of certified fonctional algorithms, rather than to produce them from scratch, like in [22], without proof of correctness, or like [21], with proof of total correctness. However, it will be necessary to study the insertion of numerical computations and round-off errors in such a formal proof framework, what always remains a difficult challenge in theorem provers.

Acknowledgements

We are very grateful towards the reviewers and P. Lienhardt whose criticisms, requests and suggestions have allowed to improve this article substantially.

References

[1] Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Construction. Springer-Verlag (2004).

[2] Bertrand, G., Malgouyres, R.: Some topological properties of surfaces in Z^3. Journal of Mathematical Imaging and Vision **11** (1999) 207–221

[3] Bertrand, Y., Dufourd, J.-F.: Algebraic specification of a 3D-modeler based on hypermaps. Graphical Models and Image Processing **56**:1 (1994) 29–60

[4] Brisson, E., Representing Geometric Structures in d Dimensions: Topology and Order. *In* ACM Symposium on Computational Geometry (1989) 218-227

[5] Brun, C., Dufourd, J.-F., Magaud, N.: Designing and proving correct a convex hull algorithm with hypermaps in Coq. Computational Geometry - Theory and Applications **45**:8 (2012) 436-457

[6] Bryant, R.P., Singerman, D.: Foundations of the Theory of Maps on Surfaces with Boundaries. Quat. J. Math. Oxford **36** (1985) 17-41

[7] Chou, S.C.: Mechanical geometry theorem proving. D. Reidel Publishing Company (1988)

[8] Coq Development Team: The Coq proof assistant reference manual. http://coq.inria.fr/doc/main.html

[9] Coquand, T., Huet, G.: Constructions: a higher order proof system for mechanizing mathematics. *In* EUROCAL (1985) Springer-Verlag LNCS **203**

[10] Cori, R.: Un Code pour les Graphes Planaires et ses Applications. Société Math. de France, Astérisque **27** (1970)

[11] Dehlinger, C.: Spécifications et preuves en Coq pour les surfaces combinatoires et leur classification. PhD Thesis, Strasbourg University (2003), http://dpt-info.u-strasbg.fr/~ jfd/DD-CGTA13/theseCD.ps.

[12] Dehlinger, C.: Coq Development on Combinatorial Surfaces. Strasbourg University (2003), http://dpt-info.u-strasbg.fr/~ jfd/DD-CGTA13/ngold.tar.gz.

[13] Dehlinger, C., Dufourd, J.-F.: Formalizing generalized maps in Coq. Theoretical Computer Science **323** (2004) 351–397

[14] Dehlinger, C., Dufourd, J.-F.: Formalizing the trading theorem in Coq. Theoretical Computer Science **323** (2004) 399–442

[15] Dehlinger, C., Dufourd, J.-F., Schreck, P.: Higher-Order Intuitionistic Formalization and Proofs in Hilbert's Elementary Geometry. *In* Automated Deduction in Geometry (2000). Springer-Verlag LNAI **2061** 306–323

[16] Dufourd, J.-F.: Formal specification of topological subdivisions using hypermaps. Computer-Aided Design **23**:2 (1991) 99–115

[17] Dufourd, J.-F.: An OBJ3 functional specification for the boundary representation. *In* ACM Siggraph Symp. on Solid Modelling (1991) 61–72

[18] Dufourd, J.-F.: Algebras and formal specifications in geometric modelling. The Visual Computer **13** (1997) 61–72

[19] Dufourd, J.-F.: Polyhedra genus theorem and Euler formula: A hypermap-formalized intuitionistic proof. Theor. Comput. Sci. **403**:2-3 (2008) 133-159

[20] Dufourd, J.-F.: An Intuitionistic Proof of a Discrete Form of the Jordan Curve Theorem Formalized in Coq with Combinatorial Hypermaps. J. Autom. Reasoning **43**:1 (2009) 19-51

[21] Dufourd, J.-F., Bertot, Y.: Formal Study of Plane Delaunay Triangulation. *In* Interactive Theorem Proving Conference ITP'10. Springer-Verlag LNCS **6172** (2010) 211–226

[22] Dufourd, J.-F., Puitg, F.: Fonctional specification and prototyping with oriented combinatorial maps. Computational Geometry - Theory and Applications **16** (2000) 129–156

[23] Firby, P. A., Gardiner, C. F.: Surface Topology. Ellis Horwood Ltd. (1982)

[24] Fomenko, A. T.: Differential Geometry and Topology. Consultant Associates (1987)

[25] Fomenko, A. T.: Visual Geometry and Topology. Springer-Verlag (1994)

[26] Gallier, J., Xu, D.: A Guide to the Classification Theorem for Compact Surfaces. Geometry and Computiong **9**, Springer-Verlag (2013)

[27] Gelernter, H.: Realization of a geometry theorem proving machine. *In* Computers and Thought (1963) MacGraw-Hill 134–152

[28] Gonthier, G.: Formal Proof - The Four Color Theorem. Notices of the AMS **55**:11 (2008) 1382–1393

[29] Griffiths, H.: Surfaces. Cambridge University Press (1981)

[30] Jacques, A.: Constellations et graphes topologiques. *In* Combinatorial Theory And Applications Symp. (1970) 657–673

[31] Kahn, G.: Elements of Constructive Geometry, Group Theory, and Domain Theory. Coq contribution, http://coq.inria.fr/contribs-eng.html.

[32] Knuth, D.E.: Axioms and Hulls. Springer-Verlag (1992) LNCS **606**

[33] Lienhardt, P.: Subdivisions of N-Dimensional Spaces and N-Dimensional Generalized Maps. *In* Computational Geometry ACM Symp. (1989) 228–236

[34] Lienhardt, P.: Topological models for boundary representation - A survey. Computer Aided Design **23** (1991) 59–81

[35] Lienhardt, P.: N-Dimensional Generalized Combinatorial Maps and Cellular Quasi-Manifolds. *In* Int. J. Comput. Geometry Appl.**4**:3 (1994) 275-324

[36] Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)

[37] Massey, W.S.: Algebraic Topology, An Introduction. Springer (1977)

[38] Meikle, L.I., Fleuriot, J.: Mechanical Theorem Proving in Computational Geometry. *In* Automated Deduction in Geometry ADG'04. Springer-Verlag LNCS **3763** (2004) 1–18

[39] Parent, C.: Synthesizing proofs from programs in the calculus of inductive constructions. *In* Mathematics of Program Construction (1995). Springer-Verlag LNCS **947** 351–379

[40] Paulin-Mohring, C.: Inductive Definition in the System Coq - Rules and Properties. *In* Typed Lambda-calculi and Applications (1993). Springer-Verlag LNCS **664**

[41] Pichardie, D., Bertot, Y.: Formalizing Convex Hulls Algorithms. *In* Theorem Proving in HOL Conf. (2001). Springer-Verlag LNCS **2152** 346–361.

[42] Puitg, F., Dufourd, J.-F.: Formal specifications and theorem proving breakthroughs in geometric modelling. *In* Theorem Proving in HOL Conf. TPHOL'98. Springer-Verlag LNCS **1479** (1998) 401–427

[43] Puitg, F.: Preuves en modélisation géométrique par le calcul des constructions inductives. PhD Thesis, Strasbourg University (1999).

[44] Puitg, F., Dufourd, J.-F.: Formalizing mathematics in higher-order logic: A case study in geometric modelling. Theoretical Computer Science **234** (2000) 234 1–57

[45] Requicha, A.A.G.: Representations for Rigid Solids. ACM Computing Surveys **12**:4 (1980) 437–464

[46] Seifert, H., Threlfall, W.: A Textbook of Topology. Pure and Applied Mathematic , Academic Press, Inc. (1980).

[47] Tutte, W.E.: Graph Theory. *In* Encyclopedia of Mathematics and its Applications. Addison Wesley, Reading, MA (1984)

[48] von Plato, J.: The axioms of constructive geometry. Annals of pure and applied logic **76** (1995) 169–200