A hypermap framework for computer-aided proofs in space subdivisions - Genus theorem and Euler’s formula

Jean-François Dufourd

Université Louis-Pasteur de Strasbourg
UFR de Mathématique et d’Informatique,
Laboratoire des Sciences de l’Image, de l’Informatique et de la Télédétection (LSIIT, UMR CNRS-ULP 7005),
Pôle API, Boulevard Sébastien Brant, 67400 Illkirch, France

Abstract

This paper presents a new framework to conduct formal proofs concerning the topology of space subdivisions. The subdivisions are modeled by hypermaps specified through the Calculus of Inductive Constructions. Proofs are computer-aided using the Coq system. A significant example is emphasized: the proof of the genus theorem and of the Euler formula for hypermaps.

Key words: subdivisions, hypermaps, formal specification, Coq system, computer-aided proof, genus theorem and Euler’s formula

1 Introduction

This paper presents a new framework to conduct formal proofs in combinatorial topology. In this area, spatial objects are subdivided into cells of different dimensions, i.e. vertices, edges and faces —, equipped with incidence and adjacency relationships. The geometrical forms and localizations of the cells in the plane do not matter. When dealing with geometric problems, an embedding in an Euclidean space must be considered afterwards.

One of the most accurate, homogeneous and flexible models of space subdivisions is the hypermap, a kind of functional multigraph. It is easy to extend to

Email address: dufourd@dpt-info.u-strasbg.fr (Jean-François Dufourd).

Preprint submitted to Geometric Constraints and Reasoning 29 August 2006
any dimension, but also to restrict in order to fit combinatorial varieties of any class. The notion of hypermap and the operations it involves can be axiomatized and constitute a basis to prove properties of subdivisions formally and to build algorithms. Today, such proofs and constructions can be developed interactively and verified by a proof assistant.

In the following, we show how a hypermap framework for surface subdivisions is entirely formalized from scratch in the *Calculus of Inductive Constructions*, and how the proofs are assisted by the Coq system developed at INRIA. Structural induction is the main reasoning mechanism used during the whole process. Then, we illustrate the power of this framework by the proof of the genus theorem and of the Euler formula.

The rest of the paper is organized as follows. In Section 2, we summarize related work in computational topology and geometry, and computer-aided proofs. In Section 3, we recall some basic mathematical materials about subdivisions and hypermaps. In Section 4, we present some preliminary formal specifications and we inductively define a type of free maps, in which all our specifications are rooted. In Section 5, we summarize the central notion of quasi-hypermap, a kind of incomplete hypermap. We also specify hypermaps and their features. In Section 6, we define paths, connectivity and hypermap characteristics. In Section 7, we inductively prove the genus theorem and the Euler relation. Finally, in Section 8, we present some concluding remarks and outline future work.

We progressively recall the main features of the Coq system. We describe and explain the whole process of specification and proof. But the full details of the proofs are out of the scope of this paper.

### 2 Related work

In computer science, models of surface subdivisions, or polyhedra, have been investigated in many ways [21]. In *boundary representation*, they are first represented by their cells at dimensions 0, 1 and 2, *i.e.* vertices, edges, faces —, equipped with incidence and adjacency relationships. The form and exact localization of the cells in space are only considered in a second step. Mathematical concepts have been proposed to model polyhedra in this way, mainly based on graphs. The most advanced are the map models: *combinatorial oriented map, generalized map* and *hypermap*, which is the most general [9, 16, 5]. Map models are used in many libraries or applications in geometry and imagery [11, 3, 6].

We have chosen as a logical specification support the Calculus of Inductive
Constructions, or CiC, a higher order intuitionistic logic based on type theory, \(\lambda\)-calculus and induction. Inductive types are defined in CiC as presentations of algebraic theories with constructors. CiC is implemented in the Coq proof assistant, which offers the specification language Gallina, built-in libraries and proof tactics [4, 2].

Few experiments have been led in geometric proofs aided by a proof assistant. One can mention [14, 18] for classical basic geometry, and [19, 17] for computational geometry. A formalization in Isabelle/Isar of planar graphs is proposed in [1] to model planar triangulations by inductive constructions. The aim is to prove the five colour theorem, the Euler formula being a lemma. However, the chosen model needs to manage an external face specifically. Moreover, it is designed for triangulations only. Using map models instead of graphs would probably help to generalize this work.

Coq specifications and proofs of combinatorial oriented maps and generalized maps are proposed in [20, 7] for geometric modeling. Hypermaps are the combinatorial central structures used in [12] to prove the four colour theorem for a planar subdivision using Coq. The main part of this work is the gigantic proof of the four colour theorem following the pioneer proofs, while using hypermaps and sophisticated proof techniques. This is an impressive result. However, a redundancy appears in the hypermap description and the primitive constructors are neither intuitive nor atomic. Our present hypermap specifications offer more simplicity.

3 Mathematical Aspects

A surface subdivision is a partition of the surface into vertices, edges and faces. Its topology can be described by a hypermap.

Definition 1 (Hypermap)

(i) A hypermap is an algebraic structure \(M = (D, \alpha_0, \alpha_1)\), where \(D\) is a finite set, the elements of which are called darts, and \(\alpha_0, \alpha_1\) are permutations on \(D\).

(ii) If \(y = \alpha_k(x)\), \(y\) is the \(k\)-successor of \(x\), \(x\) is the \(k\)-predecessor of \(y\), and darts \(x\) and \(y\) are said to be \(k\)-linked, or \(k\)-sewn, together.

Example 2 (Hypermap) Let \(D = \{1, \ldots, 15\}\). Table 1 shows functions \(\alpha_0\) and \(\alpha_1\), which are permutations on \(D\). Thus \(M = (D, \alpha_0, \alpha_1)\) is a hypermap. In Fig. 1, \(M\) is drawn on the plane by associating to each dart an oriented arc of curve beginning with a bullet and ending with a small stroke: 0-sewn darts share the same small stroke, while 1-sewn darts share the same bullet.
Table 1
Permutations $\alpha_0$ and $\alpha_1$ of the hypermap in Fig. 1.

<table>
<thead>
<tr>
<th>dart</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_0$</td>
<td>6</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>12</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>11</td>
<td>14</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>$\alpha_1$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>11</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>13</td>
<td>15</td>
</tr>
</tbody>
</table>

Fig. 1. An example of hypermap.

We focus on topology, i.e. dart incidences and adjacencies represented by bullets and strokes. A convention we always adopt in drawings is that $k$-successors turn counterclockwise in the plane around strokes and bullets.

The topological cells of a hypermap, or of the underlying subdivision, can be easily obtained from the permutations using the classical notion of orbit.

**Definition 3** (Orbits and hypermap cells)
(i) Let $D$ be a set and $f_1, \ldots, f_n$ be $n$ functions in $D$. The orbit of $x \in D$ for $f_1, \ldots, f_n$ is the subset of $D$ denoted by $< f_1, \ldots, f_n > (x)$, the elements of which are accessible from $x$ by any composition of the functions $f_1, \ldots, f_n$.

(ii) In hypermap $M = (D, \alpha_0, \alpha_1)$, $< \alpha_0 > (x)$ is the 0-orbit or edge of dart $x$, $< \alpha_1 > (x)$ its 1-orbit or vertex, $< \alpha_1^{-1} \circ \alpha_0^{-1} > (x)$ its face, and $< \alpha_0, \alpha_1 > (x)$ its connected component.

**Example 4** (Hypermap cells) The hypermap example in Fig. 1 contains 7 edges (strokes), 6 vertices (bullets), 6 faces, and 3 connected components. For instance, $< \alpha_0 > (3) = \{3, 5, 4\}$ is the edge of dart 12, $< \alpha_1 > (3) = \{3, 4, 1, 2\}$ its vertex. The (internal) face of 8 is $< \alpha_1^{-1} \circ \alpha_0^{-1} > (8) = \{8, 10\}$ and the (external) face of 13 is $< \alpha_1^{-1} \circ \alpha_0^{-1} > (13) = \{13\}$. The face of 1 is $< \alpha_1^{-1} \circ \alpha_0^{-1} > (1) = \{1, 5, 2, 11, 12, 7, 6, 4, 9\}$. It is both internal and external, because the hypermap is non planar (see below).

Faces are also defined, through $\alpha_1^{-1} \circ \alpha_0^{-1}$, for a dart traversal in counterclockwise order. All the faces which enclose a bounded (resp. unbounded) region on their left are called internal (resp. external). Others are both internal or internal, when the hypermap is non planar. When it is disconnected, each
planar connected component contains one unbounded face exactly.

Let $M$ be a hypermap, and $d, e, v, f, c$ be its numbers of darts, edges, vertices, faces, connected components, respectively.

**Definition 5** (Euler’s characteristic, genus, planarity)
(i) The Euler characteristic of $M$ is $\chi = v + e + f - d$.
(ii) The genus of $M$ is $g = c - \chi/2$.
(iii) When $g = 0$, the hypermap is said to be planar.

**Example 6** (Countings) In Fig. 1, we have $\chi = 6 + 6 + 7 - 15 = 4$ and $g = 3 - \chi/2 = 1$. Thus, the hypermap is non planar. Keeping only the two connected components on the right, we have $\chi = 2 + 2 + 3 - 3 = 4$, $g = 2 - \chi/2 = 0$ and the planarity.

Note that $\chi$ is always even, but may be negative, and that $g$ is an integer which remains non negative. This makes all the savor of the genus theorem, from which derives Euler’s formula:

**Theorem 7** (of the genus)
(i) $\chi$ is an even integer.
(ii) $g$ is a natural number.

**Corollary 8** (Euler’s formula)
A connected planar hypermap satisfies $\chi = v + e + f - d = 2$.

The representation of a hypermap on a surface is an embedding. Usually, it is a mapping which transforms vertices into points, edges into open Jordan arcs, and faces into regions homeomorphic to open disks. To be proper, the embedding must correspond to a partition of the surface. Moreover, each connected component must be embedded separately. For a connected polyhedron, a genus $g$ is classically defined as the number of holes in the underlying surface. If $g = 0$, the polyhedron has the topology of the sphere. It can be projected on a plane as a subdivision without self-intersection and with an external unbounded face. Otherwise, $g > 0$ and the polyhedron has the topology of the torus with $g$ holes. It cannot be projected onto a plane without self-intersection. The genus is a topological invariant of the surface, which can actually be computed from any minimal (in a certain sense) subdivision, represented for instance by a hypermap. Then, its value coincides with the hypermap’s genus, as defined above [13]. Conversely, in the Euclidean 3D space $\mathbb{R}^3$, each connected hypermap has a proper embedding on a closed oriented surface, such as a sphere or a torus with a sufficient number — the hypermap’s genus $g$ — of holes. When the hypermap is disconnected, the surface is disconnected as well. We thus have a natural minimal subdivision of the surface, *i.e.* a polyhedron. Only planar hypermaps can have a proper embedding on the Euclidean plane $\mathbb{R}^2$, or on the sphere.

5
Example 9 (Genus and planarity) The hypermap in Fig. 1 can never be drawn on the plane without self-intersections.

As a result, each definition or property set for hypermaps is transposable for polyhedra and vice versa. This corresponds to the original note of E. Edmonds in [9]. Actually, in the axiomatic presentation which follows, this fact can be considered as the unique assumption.

4 Basic specifications

In Gallina, the specification language of Coq, we first define an inductive type \texttt{dim} for the two dimensions at stake, which we code \texttt{di0} and \texttt{di1}:

\begin{verbatim}
Inductive dim:Set:= di0: dim | di1: dim.
\end{verbatim}

All objects being typed in Coq, \texttt{dim} is declared with the type \texttt{Set} of all concrete types, and \texttt{di0}, \texttt{di1} with the type \texttt{dim}, as they are its constructors, considered as injective independent functions. At this stage, the decidability of the equality in \texttt{dim} can be proved. Recall that, the logic of Coq being intuitionistic, the excluded middle axiom is not built-in. Therefore, if necessary, the decidability of any predicate must be proved or declared. The equality predicate $=$ is built-in for each inductive type, unlike its decidability. For \texttt{dim}, the latter can be established as a lemma:

\begin{verbatim}
Lemma eq_dim_dec: forall i j : dim, {i=j}+{\neg i=j}.
\end{verbatim}

In Gallina, \texttt{i} and \texttt{j} being two dimensions, the decidability of \texttt{i=j}, \textit{i.e.} \texttt{i=j} or \texttt{i<>j}, is conventionally written as the sum \{\texttt{i=j}\}+\{\texttt{~i=j}\}. In accordance with the paradigm proof as program of intuitionistic type theory, when established, the proof is itself a function called \texttt{eq_dim_dec}, with 2 arguments, \texttt{i j : dim}, the result being of the sum type above. An object of this type can be tested using an if\ldots then\ldots else\ldots conditional expression. The lemma is interactively proved with the help of some tactics implementing inference rules. For short, they are not given here. The reasoning is merely a structural induction on both \texttt{i} and \texttt{j}, in fact a simple reasoning by case analysis. Indeed, from each inductice type definition, Coq generates an induction principle, which can be used either to prove propositions or to build total functions on the type.

For the sake of simplicity, we have chosen to identify the type \texttt{dart} with \texttt{nat}, the built-in inductive type of naturals. In addition, the decidability \texttt{eq_dart_dec} of the dart equality is a renaming of \texttt{eq_nat_dec}. Finally, to manage exceptions, a \texttt{nil} dart is a renaming of \texttt{0}:

\begin{verbatim}
Definition dart:= nat.
\end{verbatim}
c. Linking of 2 darts at dimension 0.

d. Linking of 2 darts at dimension 1.

Fig. 2. Action of constructor L.

Definition eq_dart_dec := eq_nat_dec.

Definition nil := 0.

The concept of hypermap is now approached by a more general notion of free map, with the type \texttt{fmap}. Indeed, considering free algebras first is a general trick when dealing with inductive specification and reasoning:

\begin{verbatim}
Inductive fmap : Set :=
  V : fmap |
  I : fmap -> dart -> fmap |
  L : fmap -> dim -> dart -> dart -> fmap.
\end{verbatim}

Once again, it is an inductive type with 3 constructors \texttt{V}, \texttt{I} and \texttt{L}, respectively for the empty (or void) hypermap, the insertion of a dart in a hypermap, and the linking of two darts in a hypermap. The action of \texttt{L} is illustrated in Fig. 2, where sewings are represented by arcs of circle around strokes and bullets. In Coq, \texttt{fmap} is the smaller set of ground terms which can be built from \texttt{V} applying \texttt{I} and \texttt{L}, considered as independent injections. Again, Coq generates an induction principle based on these constructors to prove properties and build functions on free maps. Of course, the direct application of the constructors without restriction allows us to build rather complex objects, even more general than 2-multigraphs. Constraints will come later.

\textit{Observators} of free maps can now be defined. The predicate \texttt{exd} tests whether a dart exists in a hypermap. Its definition is recursive, which is indicated by the keyword \texttt{Fixpoint}, thanks to a pattern matching on \texttt{m} written \texttt{match m with...}. At the end of the parameter list, \texttt{struct m} provides a hint to the proof system to ensure that the recursive calls are performed on smaller terms, thus ensuring termination. The result is \texttt{False} or \texttt{True}, the basic constants of \texttt{Prop}, the built-in type of propositions. Note that terms are in prefix notation with as few parentheses as possible, and that \texttt{ underscore} the anonymous variable, symbolizes a non useful argument. The decidability \texttt{exd dec} of \texttt{exd} directly derives, thanks to a proof by induction on \texttt{m}:
Fixpoint exd(m:fmap)(z:dart){struct m}:Prop:=
match m with
  V => False
  | I m0 x => z=x \ exd m0 z
  | L m0 _ _ _ => exd m0 z
end.

Then, a version of operation $\alpha_k$ of the mathematical definition can be written, but completed with nil for convenience. In the definition of this function denoted $A$, note the use of conditional expressions with decidability functions:

Fixpoint A(m:fmap)(k:dim)(z:dart){struct m}:dart:=
match m with
  V => nil
  | I m0 x => A m0 k z
  | L m0 k0 x y =>
    if (eq_dim_dec k k0)
      then if (eq_dart_dec z x) then y
      else A m0 k z
    else A m0 k z
end.

The inverse $A^{-1}$ is similar. Auxiliary predicates succ and pred test whether a dart has a k-successor and a k-predecessor (not nil), with the corresponding lemmas, and functions, of decidability, succ_dec and pred_dec.

Finally, destructors are recursively defined. First, $D$ deletes the latest insertion of a dart by I. Second, $B$ and $B^{-1}$ break the latest k-link inserted for a dart by L, forward and backward respectively.

5 Quasi-hypermaps and hypermaps

Free maps having too many degrees of freedom, preconditions written as predicates are introduced for the constructors I and L:

Definition prec_I(m:fmap)(x:dart):= x <> nil \~ exd m x.
Definition prec_L(m:fmap)(k:dim)(x y:dart) :=
exd m x \~ exd m y \~ succ m k x \~ pred m k y.

If I and L are always used under these conditions, the free map built is rather close to a hypermap, which sometimes contains incomplete orbits. It is then called a quasi-hypermap. It satisfies an invariant inv_qhmap defined recursively:

Fixpoint inv_qhmap(m:fmap){struct m}:Prop:=

match \( m \) with
  \( V \Rightarrow \text{True} \)
  | I \( m_0 \) \( x \Rightarrow \text{inv_qhmap} \ m_0 \ \land \ \text{prec_I} \ m_0 \ x \)
  | L \( m_0 \) \( k \) \( x \) \( y \Rightarrow \text{inv_qhmap} \ m_0 \ \land \ \text{prec_L} \ m_0 \ k \ x \ y \)
end.

Although Coq makes it possible to exactly define the type \( \text{qhmap} \) of the quasi-hypermaps, we found it better to carry on with \( \text{fmap} \) and \( \text{inv_qhmap} \).

**Example 10** (Quasi-hypermap) A quasi-hypermap is drawn in Fig. 3 for the example subdivision. Arcs of circles represent partial orbs and circles complete ones.

An important proved property is that, for any quasi-hypermap \( m \) and dimension \( k \), \( (A \ m \ k) \) and \( (A_1 \ m \ k) \) are inverses of each other, which was generally false in free maps. We have also proved that both are injections on their definition domains, i.e. on \( (\text{succ} \ m \ k) \) and \( (\text{pred} \ m \ k) \) respectively.

Then, a real hypermap can be considered as a complete quasi-hypermap, i.e. it is equipped with complete sewings, which is expressed by the invariant:

\[
\text{Definition} \ \text{inv_hmap}(m:\text{fmap}):\text{Prop}:= \text{inv_qhmap} \ m \ \land \ \forall (x:\text{dart})(k:\text{dim}), \text{exd} \ m \ x \rightarrow \text{succ} \ m \ k \ x \ \land \ \text{pred} \ m \ k \ x.
\]

**Example 11** (Hypermap) The hypermap which "closes" the quasi-hypermap in Fig. 3 is drawn in Fig. 4.

Again, it is easier to work with \( \text{fmap} \) and \( \text{inv_hmap} \) than with a type \( \text{hmap} \). As in the mathematical definition, we can prove that for any hypermap \( m \) and dimension \( k \), \( (A \ m \ k) \) and \( (A_1 \ m \ k) \) are permutations on their common domain (\( \text{exd} \ m \)). Then, operations designed to build hypermaps directly, i.e. leading from a hypermap to another hypermap, can be defined using the constructors of free maps.
First, operation $V$ returns a free map, but also a hypermap. However, to be homogeneous with what follows it is hidden by $v$. Second, operation $i$ is the insertion of a new isolated dart which is 0- and 1-sewn to itself, provided that precondition $\text{prec}_i$, in fact $\text{prec}_I$, is respected. Third, operation $l$ $k$-links dart $x$ to dart $y$ once their corresponding links have been broken. Moreover, in order to close their $k$-orbits, the $k$-predecessor of $y$ is linked to the $k$-successor of $y$. This operation is useful, and possible, only if $y$ is not already the $k$-successor of $x$:

Definition $v := V$.

Definition $i(m:\text{fmap})(x:\text{dart}) := L(L(I m x) \text{di0 } x \ x) \text{di1 } x \ x$.

Definition $\text{prec}_l(m:\text{fmap})(k:\text{dim})(x y:\text{dart}) := \text{exd } m x \ \land \ \text{exd } m y \ \land \ y \neq A m k x$.

Definition $l(m:\text{fmap})(k:\text{dim})(x y:\text{dart}) :=$

let $x_k := (A m k x)$ in
let $y_k := (A_{-1} m k y)$ in
let $m_1 := (B_{-1} (B m k x) k y)$ in
$L(L m_1 k x y) k y_k x_k$.

In fact, $l$ can merge two distinct $k$-orbits or split a $k$-orbit, depending on the fact that $x$ and $y$ are in different $k$-orbits of $m$ or not.

Example 12 (Linking two darts in a hypermap: merging vs splitting) Fig. 5 illustrates both cases for $l m \text{ di0 } x y$. A sewing break is represented by a small zigzag. The figure would be similar for $\text{di1}$.

We have proved that $i$ and $l$, when respecting their preconditions, preserve $\text{inv}_{\text{hmap}}$. This is what is conventionally called a proof obligation. Finally, the role of the operation $\text{clos}$ is to complete a quasi-hypermap into a hypermap, which was proved. It is easily defined on free maps by a structural induction:

Fixpoint $\text{clos}(m:\text{fmap}) :=$
match $m$ with
Example 13 (Closure of a quasi-hypermap) If \( m \) is the quasi-hypermap in Fig. 3, then \( \text{clos} \ m \) is exactly the hypermap in Fig. 4.

6 Paths, connectivity and characteristics

The existence \( \expf \) of a path from a dart to another within a hypermap face is inductively defined on free maps and its decidability \( \expf\ _\text{dec} \) proved:

Fixpoint \( \expf(m:\text{fmap})(z\ t:\text{dart})\{\text{struct} \ m\}:\text{Prop}\ :=\ 
\text{match} \ m \ \text{with} \ 
\text{V} => \text{False} \\
\text{I} m0 \ x => \expf m0 z \ t \lor z=x \land t=x \\
\text{L} m0 \ di0 \ x \ y => \expf m0 z \ t \\
\text{L} m0 \ di1 \ x \ y => \expf m0 z \ t \\
\text{end.} \)
Some explanations for pattern $L \text{ m0 di0 x y}$ of $\text{expf}$ definition lie in Fig. 6. Its \textit{reflexivity} and \textit{transitivity} are easily obtained by induction for quasi-hypermaps. However, its \textit{symmetry} is only satisfied on real hypermaps. We have established it for closures only. This property helps to understand the properties of the faces better, but it is never used in the subsequent proofs.

Finally, the \textit{connectivity} of a quasi-hypermap, or of a free map, which is actually the same, is specified by a \text{Fixpoint} as a binary relation $\text{eqc}$ stating that two darts belong to the same hypermap connected component. Using induction, we have its decidability $\text{eqc}_{\text{dec}}$, and quickly its \textit{reflexivity}, \textit{symmetry} and \textit{transitivity} for any free map.

To define hypermap characteristics, we must work with a Coq library module which contains all the features for $\mathbb{Z}$, the integer ring. The number $\text{nd}$ of darts in a quasi-map can be immediately defined by induction. It increases by 1 for pattern $I \text{ m0 x}$ only:

\begin{verbatim}
Fixpoint nd(m:fmap):Z:=
  match m with
    V => 0
  | I m0 x => nd m0 + 1
  | L m0 _ _ _ => nd m0
  end.
\end{verbatim}

The definition of the number $\text{ne}$ of edges is more difficult to read. Let us focus on the most interesting pattern $L \text{ m0 di0 x y}$. If, in the closure ($\text{clos m0}$), $y$ is already the 0-successor of $x$, then $L \text{ m0 di0 x y}$ only helps to close the 0-orbit common to $x$ and $y$. Thus, $\text{ne}$ does not change. Otherwise, $x$ and $y$ are in different 0-orbits which are merged, and $\text{ne}$ decreases by 1. The case of the number $\text{nv}$ of vertices is similar, replacing $\text{di0}$ by $\text{di1}$:

\begin{verbatim}
Fixpoint ne(m:fmap):Z:=
\end{verbatim}
match m with
  | V => 0
  | I m0 x => ne m0 + 1
  | L m0 di0 x y => ne m0 -
    if eq_dart_dec (A (clos m0) di0 x) y then 0 else 1
  | L m0 di1 x y => ne m0
end.

The definition of the number \( nf \) of faces is more complicated, except for the patterns \( V \) and \( I m0 x0 \), where it is trivial. Hence, for the pattern \( L m0 di0 x y \), with \( mc, x0 \) and \( x_1 \) defined as in the following specification, there are three cases illustrated in Fig. 7. The pattern \( L m0 di1 x y \) is similar, with conditions \( y = x1 \) and \( \text{expf} mc x y0 \):

\[
\text{Fixpoint } nf(m: \text{fmap}): \mathbb{Z} := \\
\text{match } m \text{ with} \\
  \text{V} \Rightarrow 0 \\
  | I m0 x \Rightarrow nf m0 + 1 \\
  | L m0 di0 x y => 
    let mc := clos m0 in 
    let x0 := A mc di0 x in 
    let x_1 := A_1 mc di1 x in 
    nf m0 + 
    if eq_dart_dec y x0 then 0 else 
    if \text{expf\_dec} mc x_1 y then 1 else -1 \\
  | L m0 di1 x y => 
    let mc := clos m0 in 
    let x1 := A mc di1 x in 
    let y0 := A mc di0 y in 
    nf m0 + 
    if eq_dart_dec y x1 then 0 else 
    if \text{expf\_dec} mc x y0 then 1 else -1 \\
end.
\]

Finally, the definition of the number \( nc \) of connected components does not pose any problem with the condition \( \text{eqc\_dec} (clos m0) x y \). The definition in \( \mathbb{Z} \) of the Euler characteristic \( ec \) immediately follows:

\[
\text{Fixpoint } nc(m: \text{fmap}): \mathbb{Z} := \\
\text{match } m \text{ with} \\
  \text{V} \Rightarrow 0 \\
  | I m0 x \Rightarrow nc m0 + 1 \\
  | L m0 _ x y \Rightarrow nc m0 - \text{if eqc\_dec (clos m0) x y then 0 else 1} \\
end.
\]

\[
\text{Definition } ec(m: \text{fmap}): \mathbb{Z} := \text{nv m + ne m + nf m - nd m}.
\]
Case 1 (y = x₀): the same faces remain.

Case 2 (y <> x₀ \land \expf x₁ y): a new face is created.

Case 3 (y <> x₀ \land \neg \expf x₁ y): one face is removed.

Fig. 7. Variation of the number of faces.

7 Genus theorem and Euler’s formula

Everything is set to prove the first part of the genus theorem, which is the parity of \((\text{ec } m)\) for a quasi-hypermap \(m\), as well as for a free map \(m\). Indeed, it is rather surprising that the proof does not require the invariant \((\text{inv}_\text{qhmap} \ m)\). During the proof, numerous simple linear equation, disequation and inequation systems in \(\mathbb{Z}\) are solved automatically calling the Coq tactic \textsf{omega}, which works in Presburger arithmetics. Useful properties of the parity predicate \texttt{Zeven} in \(\mathbb{Z}\) are already present [4]. The second part of the genus theorem immediately follows, this time necessarily under the \((\text{inv}_\text{qhmap} \ m)\) condition.

Theorem genus\_1: \forall m: \text{fmap}, \texttt{Zeven} (\text{ec} \ m).

Theorem genus\_2: \forall m: \text{fmap},
\hspace{1em} \text{inv}_\text{qhmap} m \rightarrow 2 \ast (\text{nc} \ m) \geq (\text{ec} \ m).
The structures of the proofs for both subtheorems are the same. They are based on an elementary structural induction on $m$ and on the linking dimension. The proofs can be outlined together as follows with four cases:

- **Case 1**: $m = V$. Since $nv V = ne V = nf V = nd V = nc m = 0$ by definition, $ec m$ is null, hence even and such that $2 * (nc m) >= (ec m)$.

- **Case 2**: $m = I m0 x$, with the *induction hypothesis*: even $(ec m0)$ and $2 * (nc m0) >= (ec m0)$. Then, from the previous specifications, $nv m = (nv m0) + 1$. Similar equalities apply to $ne, nf, nd$ and $nc$. Hence, $ec m = (ec m0) + 2$, which remains even, and we always have $2 * (nc m) >= (ec m)$.

- **Case 3**: $m = L m0 di0 x y$, with the *induction hypothesis*: even $(ec m0)$ and $2 * (nc m0) >= (ec m0)$. Let $mc := clos m0$ be the closure of $m0$. Two subcases arise (see Fig. 7 again):
  - **Case 3.1**: $(A mc di0 x) = y$, *i.e.* $y$ is the 0-successor of $x$ in $mc$. Then, from the definitions, each characteristic keeps its value and the result follows.
  - **Case 3.2**: $(A mc di0 x) = y$, *i.e.* $y$ is not the 0-successor of $x$ in $mc$. Then, $ne m = (ne m0) - 1$ by definition, and, setting $x1 := (A1 mc di1 x)$, there are two more subcases:
    - **Case 3.2.1**: $expf mc x1 y$, *i.e.* $x1$ and $y$ belong to the same face of $mc$. Then, from the definitions, the only variations of the characteristics are $ne m = (ne m0) - 1$ and $nf m = (nf m0) + 1$. Hence $ec m = ec m0$ and the result follows.
    - **Case 3.2.2**: $~expf mc x1 y$, *i.e.* $x1$ and $y$ do not belong to the same face of $mc$. Then, $nf m = (nf m0) - 1$ by definition. Hence, since $ne m = (ne m0) - 1$, we have $ec m = (ec m0) - 2$, which may be negative but remains even. At last, two subcases arise for $nc$:
      - **Case 3.2.2.1**: $eqc mc x y$, *i.e.* $x$ and $y$ belong to the same connected component of $mc$. Then, $nc m = nc m0$, and the result follows.
      - **Case 3.2.2.2**: $~eqc mc x y$, *i.e.* $x$ and $y$ are not connected in $mc$. Then, $nc m = (nc m0) - 1, ec m = (ec m0) - 2$, hence $2 * (nc m) >= (ec m)$.

- **Case 4**: $m = L m0 di1 x y$. The reasoning is exactly the same as in *Case 3*, replacing $di0, x0, x1$ and $y$ by $di1, x1 := (A mc di1 x), x$ and $y0 := (A mc di0 y)$, respectively.

Finally, the **genus** function and the planarity predicate **planar** are defined and the proof of the Euler formula is directly obtained, from a genus corollary:

**Definition genus** $(m : fmap) := (nc m) - (ec m)/2$.

**Definition planar** $(m : fmap) := genus m = 0$

**Theorem genus_corollary** : forall $m : fmap$,

$$inv_qhmap m \rightarrow genus m \geq 0.$$
Theorem Euler_formula: forall m:fmap,
   inv_qhmap m -> planar m -> ec m / 2 = nc m.

8 Conclusion

We have proposed a new formal framework in Coq to build constructive proofs in computational topology using a map hierarchy leading to hypermaps. The computer-aided inductive proofs we have presented as benchmarks for the genus theorem and Euler’s formula are completely combinatorial (see the website [10]). Moreover, our types and atomic operations allow to build easily any derivative map type or polyhedron operation of the literature.

The Coq system turned out to be a precious auxiliary to guide and check all the proofs. The development we have presented represents about 10,000 lines of Gallina, including several hundreds of definitions, lemmas and theorems. The lack of facilities to manage term congruences, particularly permutations of map constructions by $I$ and $L$, was the main drawback of the Coq system.

Furthermore, we have begun to revisit the foundations of computational geometry using formal specifications, proofs, maps and hypermaps in the way of [8]. In this field, many notions are sheerly topological and derive from the methods we have presented. The others, which concern geometric embedding, real numbers and round-off errors, are more difficult to tackle. We think that progress will arise from appropriate axiomatics of the numbers, or axiomatics allowing to bypass them, like the one of D. Knuth in [15] for orientation in the plane and convex hull.

Finally, we also intend to use a promising feature of provers based on the Curry-Howard isomorphism, namely the extraction of programs from proofs. Such programs are automatically correct, or certified, with respect to formal specifications [2]. This way, we hope that we will be able to extract correct programs of computational topology or geometry from constructive proofs.

References