Abstract – A geometric modeller is a piece of software allowing to create, manipulate and modify geometric objects. Modern geometric modellers usually use a mathematical model to represent objects and are developed around a kernel that proposes basic operations to create and manipulate objects called by high level operations. Because of these proceedings, the context is a priori in favour of the use of formal methods software engineering. In this paper we formally specify and verify a geometric modelling core, comparable to the kernel of a geometric modeller where the objects are represented as generalized maps, a classical and powerful representation model. We use for that purpose the B method and its embedded proof system.

Keywords: formal methods, B method, geometric modelling, generalized maps.

1.0 Introduction

A geometric modeller is a piece of software allowing to create, manipulate and modify geometric objects. They are used for instance to design manufactured objects, to model plants in botany, to represent geological layers in geophysics, to produce virtual objects in computer animation. The objects built with these tools become bigger and bigger, the operations for creation and manipulation become more and more complex and thus the development of such modellers become more and more expensive and more and more difficult. In order to decrease that cost and increase the reliability of such software, software engineering methods are to be recommended.

The fundamental design principles behind modern modellers (e.g. Topofil [1]) are twofold: the use of a mathematical model to represent objects, and a design around a kernel that proposes basic operations to create and manipulate objects called by high level operations. Because of these proceedings, the context is a priori in favour of the use of formal methods software engineering. Both share mathematical backgrounds and some kind of genericity and abstraction. Formal methods have already been used in the domain of geometry and geometric modelling.

In this paper we propose to formally specify and verify a geometric modelling core, comparable to the kernel of a geometric modeller that implements a mathematical representation models and the associated basic operations. It is an important step towards more reliability in modellers since we define a rigorous framework in which geometric objects and basic operations are defined and their properties proved on-machine. Our guideline is to stick the more possible to the usual mathematical definitions.

The formal method and methodology we based our work on is the B method [2]. Its favourite applications domains are critical software for transportation systems, and geometric modelling is a new application area.

We have chosen the B method for its ability to handle easily sets and relations also very often used in the mathematical definitions in geometric modelling. Furthermore, this specification language is equipped with many different tools such as animators, provers, code generators, etc. This tooling provides B with a very operational environment.

There are different ways to represent geometric objects, we have chosen a combinatorial model, called the generalized maps or Gmaps. Thus our B specification is composed of an evolutive state that encapsulates a generalized map and operations modifying or observing the Gmap, corresponding to the basic operations of the modelling core. In the following section, we give an overview of the B method. The section 3.0 briefly presents the notion of generalized maps. The state of the B specification is presented in the section 4.0. Then we detail each operation in the section 5.0. The section 6.0 focuses on the proof activity around this specification. Before concluding we discuss related work.

2.0 The B method

The B method allows to specify and design systems: from abstract specifications to executable code. The first specification we write down is called an abstract machine: it is composed of an evolutive state that encapsulates some data and operations modifying and observing the state. Mainly it means to prove that the initialization phase establishes the invariant and that each operation of the machine
preserves the invariant. Thanks to tools such as the Atelier B [3] or B-toolkit [4], proof obligations are automatically generated. Finally B machines can be refined into other more and more concrete components. The ultimate component usually provides an executable model that can be easily translated into a program or a module written in a programming language. Again refinement proof obligations are to be discharged to ensure the correctness of the refinements. Roughly speaking it ensures that the concrete model has the same behaviour than the abstract one. In the work presented here we stay at the specification level and consequently do not use refinement.

### 3.0 What is a $n$-Gmap?

There exist several ways to represent a geometric object. We consider the one that emphasizes the topological structure of the object, that is its subdivision in vertices, edges, faces, etc. Its shape (curves, surfaces) is then obtained by mapping each element of the subdivision to its embedding in space. This mapping is called the embedding. For example we associate a curve to an edge and a surface to a face. Such a model allows to decompose each operation in two counterparts: a topological operation that only modifies the structure of the object, a geometric operation that modifies its embedding. With this approach, the operations have been more rigorously described.

In this paper, we focus on the topological model and its operations. The geometric aspects are to be incorporated, they are only sketched at the end of the paper. We briefly present a model reflecting this approach, that is the model of the generalized maps, Gmaps for short. Technically it allows to represent unspecified-sized manifolds, opened or not and orientable or not. Our modelling core is based on this representation model. It is also the representation the modeller Topofil [1] is based on.

Let us consider the object small house on the figure 1–A. The figure 1–B illustrates the same object but it highlights that the square and the triangle, 2 faces (of dimension 2), are adjacent. The figure 1–C also breaks up the square into 4 edges (dimension 1) and the triangle in 3 edges. At the end, on the figure 1–D, the edges are dissociated into vertices (dimension 0) that are linked two by two by an adjacency relation. Thus we have represented the small house with the help of 14 atomic objects we call darts. All the relations between faces or edges have been deferred on the darts (cf. figure 1–E). Each relation is formalized by an involution $\alpha_i$ whose index $i$ denotes the dimension of the adjacency relation. For example two darts linked by $\alpha_2$ belong to two objects of dimension 2 that are different but adjacent. A general definition, not bound to a particular dimension, has been introduced by [5]. We follow it in this paper.

**Definition 1:** Let $n \geq 0$, a generalized map of dimension $n$ (or $n$-Gmap) is defined by an $(n+2)$-uplet $G = (D, \alpha_0, \alpha_1, \ldots, \alpha_n)$, such as:
- $D$ is a finite set of darts. 
- $\alpha_i$ is an involution on $G$, $0 \leq i \leq n$ ($C_1$)
- $\alpha_i \alpha_j$ is an involution, $0 \leq i < i + 1 < j \leq n$ ($C_2$)

A dart $d$ is said $i$-free if $\alpha_i(d) = d$ and free if it is $i$-free for all $i$ in $[0, n]$.

### 4.0 The state of the abstract machine BGmap

The specification of the geometric modelling core is described as an abstract machine $BGmap$ parametrized by a natural number $n$ the dimension of the Gmap. Its state is presented in the figure 2 and discussed here. Its operations are described in the next section.

The state is described by two variables: darts the carrier of the Gmap and alpha a function that groups altogether the adjacency relations $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_n$. So the B expression $alpha(i)$ denotes the mathematical object $\alpha_i$. From now, we do not distinguish them anymore and write $\alpha_i$ uniformly in the text (not in the formal specifications). The relation $\alpha_i$ is a $-total$- function that maps each dart of the carrier to another one. The specification introduces the given set $DARTS$ which corresponds to the set of all possible darts, or else the type of the darts, thus $darts \subseteq DARTS$. In addition to the typing
predicates, the invariant clause formalizes the conditions $C_1$ and $C_2$ of the mathematical definition. We specify that $\alpha_i$ is an involution by imposing that \( \alpha(i) = (\alpha(i))^{-1} \).

The mathematical function composition $\alpha_i \alpha_j$ is written in B

\[
(\alpha(i) ; \alpha(j))
\]

A parametrized definition is used for more readability:

\[
is\_invol(g) == ((g = g^{-1});
\]

### 5.0 The basic operations on $n$-Gmaps

The dynamic part of the abstract machine contains the basic operations that allow us to build and manipulate the topology of simple objects, called in this paper the basic operations of the geometric modelling core: add a dart in a Gmap; remove a dart; walk into a Gmap; sew two darts; unsew a dart.

In this part we define and formalize within B each operation.

#### 5.1 Adding a dart

\[
\begin{align*}
\text{res} & \leftarrow \text{Add} = \\
& \text{ANY } d \\
& \text{WHERE } d \in \text{DARTS} - \text{darts} \\
& \text{THEN} \\
& \text{darts} := \text{darts} \cup \{d\} \quad \text{||} \\
& \alpha := \lambda(x). (x \in 0..n \mid \alpha(x) \cup \{d \rightarrow d\}) \quad \text{||} \\
& \text{res} := d \\
& \text{END;}
\end{align*}
\]

Fig. 3. The B specification of the Add operation

This very simple operation adds a new dart, not already present in the carrier of the Gmap, as a free dart. Its specification is written within B in the figure 3.

The operation takes no parameter but returns the new dart. The way the new dart is chosen is let unspecified in the abstract machine (see the use of the unbounded choice substitution in the specification). The chosen new dart is inserted in the carrier of the Gmap and every adjacency relation is augmented with the pair \((d \rightarrow d)\): \(\alpha(0) := \alpha(0) \cup \{(d \rightarrow d)\}, \alpha(1) := \alpha(1) \cup \{(d \rightarrow d)\}, \ldots\) It is expressed by using the lambda expression

\[
\lambda(x). \{x \in 0..n \mid \alpha(x) \cup \{d \rightarrow d\}\}.
\]

#### 5.2 Removing a dart

\[
\begin{align*}
\text{Delete}(d) = \\
& \text{PRE } d \in \text{darts} \land \forall (i,i \in 0..n \Rightarrow \alpha(i)(d) = d) \\
& \text{THEN} \\
& \text{darts} := \text{darts} - \{d\} \quad \text{||} \\
& \alpha := \lambda(k). (k \in 0..n \mid \alpha(k) - \{d \rightarrow d\}) \\
& \text{END;}
\end{align*}
\]

Fig. 4. The B specification of the Delete operation

This operation, dual from the previous one, allows us to remove a dart from a Gmap. It only makes sense when this dart is a free one (otherwise we could introduce some ‘mess’ in the Gmap). This condition constitutes the precondition of the B corresponding operation that takes the dart to be removed \(d\) as a parameter (see the figure 4). So the dart \(d\) is removed from the carrier and all the related information is erased from the adjacency relations (but because of the precondition, the involved pair is \((d \rightarrow d)\) for any dimension).

#### 5.3 Traversal in a Gmap: the computation of an orbit

A traversal of a Gmap is determined by a set of permutations \(\{p_1, \ldots, p_m\}\) on the carrier and a dart: its will compute all the darts reachable from the dart \(d\) by successive applications of the permutations. This set of darts is known as the orbit \(\langle p_1, \ldots, p_m \rangle(d)\) of \(d\) wrt the permutations \(p_1, \ldots, p_m\).

This operation is a very general one that will allows us to retrieve for a dart, all the darts belonging to the same vertex, the same face or the same volume. For example, let us consider again the small house of the figure 1–D. We illustrate with it some orbits wrt to the dart \(d\) (in bold font), the orbits are drawn with full lines. \(<\alpha_1, \alpha_2 > (d)\) is represented on the figure 5–A: all the involved darts define a vertex. On the figure 5–B, \(<\alpha_0, \alpha_2 > (d)\) defines an edge. On the figure 5–C, \(<\alpha_0, \alpha_1 > (d)\) defines a face. On the figure 5–D, \(<\alpha_0, \alpha_1, \alpha_2 > (d)\) defines the connected component of \(d\).

We specify the computation of the orbit as the smallest set verifying the axioms given below:

\[
\begin{align*}
& d \in < p_1, \ldots, p_m > (d); \quad (\theta_1) \\
& \forall i \in [1, m], \forall c < p_1, \ldots, p_m > (d) \Rightarrow \\
& p_i(c) \in < p_1, \ldots, p_m > (d). \quad (\theta_2)
\end{align*}
\]

A constant is introduced in the B specification (that is we assume the existence of a function that satisfies the previous axioms): \(\text{orbit(\text{Set, Alphaset, dart})}\) denotes the orbit of the dart \(\text{dart}\) wrt to a set of permutations \(\text{Alphaset}\) in a Gmap whose carrier is \(\text{Set}\). So the previous properties are translated

\[
\lambda(x). (P|E) \text{ denotes the function containing all the pairs } (x \rightarrow E) \text{ such that } x \text{ satisfies the predicate } P.
\]
in B with some others that are typing predicates indicating the nature of the objects Set, AlphaSet etc. (e.g. orbit \( \in \mathbb{P}(DARTS) \ast \mathbb{P}(DARTS \leftrightarrow DARTS) \ast DARTS \to \mathbb{P}(DARTS) \land \)). For all the details see the figure 6. The predicate \((\theta_3)\) is the formal translation of the “smallest set” aspect.

The orbit constant function is used to define the corresponding B operation Orbit. Its arguments are respectively a dart \(d\) from the carrier of the Gmap, a subset of integers indexSet, that denotes the adjacency relations to take into account. It computes the orbit of \(d\) wrt the permutations \(\alpha(i)\) for \(i \in IndexSet\). with the help of the constant orbit (see the figure 7).

### 5.4 Sewing 2 darts

Intuitively, this operation allows us to link together objects of the same dimension, along an object of the predecessor dimension. For example, let us have a look at the cubes of the figure 8–A. Gluing them requires to detect the faces that will be involved (in grey in the figure), we call them the sewing orbits. More precisely we need to know the 2 darts that determine the faces, \(d\) and \(d'\) in the figure 8–B. The sewing operation will consist in linking \(d\) and \(d'\) together (for the dimension 3). This link will generate other darts to be linked in order to build a new Gmap that preserves the properties related to the composed involutions \((C_2)\). The induced modifications in the adjacency relations appear in the figure 9.

Indeed, once \(d\) and \(d'\) are linked by \(\alpha_3\), the Gmap model imposes to link also:

\(- \alpha_0(d)\) and \(\alpha_0(d')\) by \(\alpha_3\) (in order to establish again the
the last example, 8 darts for the cube face, and 6 darts for the sewing are isomorphic. Indeed it is a fundamental precondition of the operation. We cannot sew a cube and a tetrahedron, we should have to sew a “square” and a “triangle”: the set of darts that determine these two faces ($f_3$ and $f_4$ in the figure 10) are not isomorphic. For an object of dimension $n$ less or equal to 3, the isomorphism test can be reduced to the comparison of the cardinality of both sets of darts. In the last example, 8 darts for the cube face, and 6 darts for the tetrahedron face. However we specify the more general definition of this test: the set of isomorphisms between $O_1$ and $O_2$ is not empty where $O_1$ ($O_2$) is the set of darts denoting the sewing orbit (a face in the example, it could be for instance an edge or a vertex) of the first (second) object. Both sewing orbits are denoted by a dart ($d$ and $d'$ in the example). Formally the sewing-orbit for a dart $d$ and a dimension $i$ is defined as the reachable darts from $d$ by using the involutions $\alpha_{i-1}$, $\alpha_i$ and $\alpha_{i+1}$ or else $\text{orbit}(darts, ran(\{i-1, i, i+1\} \cup alpha), d)$

Then the $\text{Sew}$ operation consists in choosing in the set of isomorphisms between $O(d_1)$ and $O(d_2)$ an isomorphism that links $d_1$ et $d_2$, called $iso$. Because of the precondition, such a function exists (there may have several) and updating the adjacency relations with $iso$. The links relative to darts (in $\alpha_i$) involved in the sewing are modified according to $iso$, the others are unchanged. This is easily specified in B by using the overriding operator $\triangleleft$.

The mathematical definition of the sewing operation [6] contains the precondition $d_1$ and $d_2$ are $i$-free. We have replaced in our B formalization that condition by a stronger one: the darts of the sewing-orbit of $d_1$ are $i$-free (written in B as $O(d_1) \triangleleft alpha(i) = \text{id}(O(d_1))$), idem for the sewing-orbit of $d_2$. This adaptation of the mathematical definition has been required when trying to prove the proof obligations generated by the $\text{Sew}$ operation. The weaker precondition requires a large implicit knowledge regarding results about the Gmaps. However, we can establish as an assertion that the weaker precondition implies ours.

**Sew** ($i, d_1, d_2$) =

PRE $d_1 \in darts$ ∧ $d_2 \in darts$ ∧ $i \in 0..n$ ∧
isos($O(d_1), O(d_2), i$) $\neq \emptyset$ ∧
$O(d_1) \triangleleft alpha(i) = \text{id}(O(d_1))$ ∧
$O(d_2) \triangleleft alpha(i) = \text{id}(O(d_2))$

THEN

ANY iso, Iso
WHERE iso $\in$ isos($O(d_1), O(d_2), i$) ∧ isos($d_1) = (d_2)$ ∧
Iso = iso $\cup$ iso$^{-1}$

THEN

alpha($i$) := alpha($i$) $\triangleleft$ Iso

END

END;
5.5 Unsewing a dart

Unsewing a dart \(d\) according to a dimension \(i\) only makes sense if \(d\) belongs to the carrier of the Gmap, \(i\) is less or equal to the dimension of the Gmap and \(d\) is linked to another dart different from itself. Let us notice that this dart is unique and can be denoted by \(\alpha_i(d)\). If all these conditions are met, we can suppress the \(i\)-link between \(d\) and its image and also between the darts of the sewing-orbits \(O(d)\) and \(O(\alpha_i(d))\). The suppression means here to make them \(i\)-free: \(\alpha(i) := \alpha(i) \Leftrightarrow (\text{id}(O(d) \cup O(\alpha(i)(d))).\)

As previously, we have slightly modified one of the preconditions of the operation. Instead of writing \(d\) must have been sewed at the \(i\)-dimension, we write that the image of the \(i\)-sewing orbit of \(d\) is the \(i\)-sewing orbit of \(\alpha_i(d)\). It is expressed in B by \(\alpha(i)[O(d)] = O(\alpha(i)(d))\). The complete specification of the operation is detailed in the figure 13.

\[
\text{unSew} \quad \text{(i, d)} = \begin{align*}
\text{PRE} & \ d \in \text{DARTS} \land i \in \text{darts} \land \nabla \ i \in 0..n \land \alpha(i)[O(d)] = O(\alpha(i)(d)) \\
\text{THEN} & \quad \alpha(i) := \alpha(i) \Leftrightarrow (\text{id}(O(d) \cup O(\alpha(i)(d)))
\end{align*}
\]

Fig. 13. Specification of the UnSew operation

6.0 Proofs

In this section, we discuss the correctness proof of the previous abstract machine. It means proving that the initialization establishes the invariant and that the operations preserve the invariant. The initialization which assigns to the variables \text{darts} and \text{alpha(i)} the empty set trivially establishes the invariant. The proof obligations (PO) regarding the preservation of the invariant by the operations are automatically generated by the Atelier B tool [7] that offers 2 demonstrators, an automatic one and an interactive one. The distribution of the required effort to prove the generated POs can be seen in the table 2.

The table mentions that only 45\% of the proof obligations have been automatically proved. However this percentage increases if we count the 6 POs that have been proved by a very blind process we can sum up like this: when one prover cannot anymore simplify the goal, we use the other one and so on until the proof is done.

On the other hand, when this empirical method fails -for 6 other POs-, the proof is done thanks to the embedded interactive prover. These cases have required to exhibit properties about involutions and to introduce intermediate lemmas. For some of them, we have carried on our effort in order to turn intermediate lemmas into a proof rule. Then it becomes a rule usable in any proof, not only by the interactive prover but also by the automatic one.

In conclusion the BGmap machine has been entirely proved correct, with a moderate effort. The next step would be to capitalize this proof experience into a dedicated prover. Main pieces such as proof rules, theories are present, it still remains to develop specialized tactics to obtain a better automation.

### Table 2: Proofs of the BGmap machine

<table>
<thead>
<tr>
<th>Operation</th>
<th>inter Proof</th>
<th>auto Proof</th>
<th>%Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALISATION</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Add</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Orbit</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sew</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>unSew</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Delete</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>BGmap</td>
<td>22</td>
<td>12</td>
<td>10</td>
</tr>
</tbody>
</table>

This dedicated prover would be a very effective help to formalize other representation models.

7.0 Related work

Different experimentations with formal methods in the fields of geometry and geometric modelling have been undertaken. We can cite works about constructive geometry in the computer-proof-aided tool Coq [8] like [9], [10], [11]. Regarding geometric modelling and more precisely combinatorial surfaces modelling, a large amount of work has been done using algebraic approaches. The modeller Topofil has been developed from algebraic specifications [1]. The language Casl [12] is the support of works for specifying and designing high level operations like rounding [13]. No proof is done with this last approach. The emphasis is put on code and test generation. Type theory and calculus of inductive constructions have been also studied for combinatorial geometry with the proof of Jordan theorem [14], [15].

The closest work to ours is the Coq formalization of generalized maps done by DEHLINGER and DUFORD in [16]. In fact they have developed a hierarchy of types, so they have first specified unconstrained free maps\(^2\) and then have introduced Gmaps themselves as well-formed free maps. Lastly they defined sewn maps or smaps, that are Gmaps built only with the help on a high-level operation. They showed that both definitions (of Gmaps and smaps) are equivalent. This work is the basis of the Coq specification and proof of a fundamental theorem of geometry, the theorem of classification of surfaces according to numerical characteristics.

The specification style is functional and inspired from the algebraic specifications presented in [1] while our style is more imperative and closer to mathematical definitions. However the proof aspects are very similar. Indeed DEHLINGER and DUFORD in [16] usually formulate an operation at the level of free maps and then show that when applied to Gmaps this operation preserves the constraints imposed by Gmaps.

8.0 Conclusion

The B method is well adapted to validate a modelling core. Indeed, the structure of an abstract machine allows us to organize clearly the specification. The state describes the topological model. It can be as in this paper an existing

\(^2\)The \(\alpha_i\) and their compositions are not constrained to be involutive
model or a new one, or a variation about an existing one. The dynamic part contains the topological operations. The structure of a B operation is well adapted to specify the basic modelling operations of a topology-based model: a precondition that announces the usage constraints of the operation and a description that formalizes the behaviour of the operation. The high level relational operators of the B language, as the overriding or the restriction allow us to translate in a very concise way the mathematical definitions while staying very close to them. The proofs to validate the specifications are not fully automatic. However by developing dedicated theories and tactics, this effort could decrease.

The B method has allowed to get the specification as close as possible to the requirements, i.e. the mathematical definitions. These specifications are, according to us, both readable and understandable by modelling experts, thanks to the expressive and powerful B syntax, in a style familiar to them. We won’t claim a same slogan about proofs, an approach press the button is not yet ready but not so far for verification of other models.

We have focused in this paper on the topogical aspects. Nevertheless we will have to study the geometric aspects. For this purpose we have to consider an embedding model. The simplest one, said linear, associates a point \( P \) (characterized by its co-ordinates in a reference for example) to each vertex of the Gmap. Consequently all the darts which compose the vertex have the same embedding, the point \( P \): here is a new invariant of the specification. The embedding of an edge is then deduced from the embedding of its ends which are two points \( P_1 \) and \( P_2 \), it is a segment \( [P_1 P_2] \). We would like to find again this separation in the formalisation. Roughly speaking, we could define a new abstract machine including the previous one that would be refined with the embedding information. This work is in progress, validation proofs are on the way.

We have achieved our goal that was to formally specify a modelling core. We could now go further and try to synthesise executable code from our formal specifications. The B refinement process is the key for that. Two main steps remain: first, remove the non-determinism in the operations Add and Sew; second, implement the computation of the constant orbit as an operation on graphs.

### 9.0 References


