Formal specification for a mathematics-based application domain: geometric modeling[1]

Franck Ledoux †, Jean-Marc Mota †, Agnès Arnould ‡, Catherine Dubois†, Pascale Le Gall †, Yves Bertrand ‡

† L.a.M.I., Université d'Évry val d'Essonne,
Cours Monseigneur Roméro, 91025 Evry Cedex, France.
`{jmmota,dubois,fledoux,legall}@lami.univ-evry.fr`

‡ IRCOM-SIC, UMR 6615, Université de Poitiers,
SP2MI, 86962 Futuroscope Cedex, France.
`{arnould,bertrand}@sic.sp2mi.univ-poitiers.fr`

Abstract: In this paper, we discuss the use of formal methods in the domain of geometric modeling. More precisely, the purpose of our work is to provide a formal specification language convenient for geometric modeling with tools. To do it, we are interested in a high-level operation, the chamfering which is already mathematically defined. Then we propose two specifications of it, using the two languages B and CASL, respectively representative of model-oriented approach and property-oriented approach. In order to as well as possible take advantage of the B and CASL characteristics, we explicitly specify the chamfering in B and implicitly in CASL. In both cases, we succeeded in providing a specification easily understandable by the experts of the application domain.

Keywords: Formal specifications, B, B method, algebraic specifications, CASL, geometric modeling, n-dimensional generalized maps, chamfering.

1 Introduction

The aim of this paper is to point out how it may be fruitful to formally specify software whose requirements are naturally based on mathematical notions. For this, we propose a case study concerning a central operation in the modeling of 3-D geometrical objects which mathematically defined: chamfering. This kind of operation is a part of what we call 3-D geometric modeling softwares, or modelers. Generally, the implementation of these operations requires large and sophisticated algorithms. A formal specification provides a both rigorous and abstract definition of these operations. Thus, it is possible to reason about this definition formally on the one hand, and without care of algorithmic details on the other hand.

[1]This work is supported by a french national project (Plan Pluri Formation) between two research teams on geometric modeling (Poitiers, Strasbourg) and a research team on formal methods (Evry).

Modelers are at the center of domains such as Computer-Aided Design (CAD) for manufactured objects, geophysics when the considered objects are geological layers, or botany for modeling the volumic structure of plants. Particularly CAD use operations (union, intersection, ...) which compose simple objects to give more complex ones. These operations handle the objects globally. Moreover, some operations modifies objects only locally.

The operation named chamfering is one of the most important local operations in CAD. It is also one of the most tricky ones, because it modifies both the structure and the form of the objects while preserving some constraints (continuity, for instance). This operation consists in replacing sharp edges of an object by small, flat or rounded patches. It is very frequently used, a lot of manufactured objects are easily constructible with a few elementary volumes, whose most edges are then rounded. It is the case of TV, telephones, among others.

The main idea for the chamfering operation, which makes this operation complex, is not to ask the user to explicitly construct small patches which will replace rounded edges, nor rounded vertices resulting from the edges chamfering. Indeed, the user only picks the edges which are to be rounded. Then, the chamfering operation automatically calculates the resulting object.

Applications based on geometrical modelers are large size softwares (a few millions of lines of code). They are developed by large teams and maintained during several years. These softwares permit to interactively handle objects made up with several hundreds of geometric primitives. Thus, they must be particularly efficient in time and space. The development of geometrical modelers is always accompanied by a large use of informal software engineering methods.

Codes developed by research laboratories are often restricted to geometrical modeling kernels. Consequently, they are of a smaller size than the commercial ones: in general between 10000 and 500000 lines. They are more reused, by adapting them to different application domains which often require specific functions, new object classes, or particular time-space performances. These codes must be very well structured and documented.

These different acknowledgements have led to use formal algebraic specifications in geometric modeling [Duf89, BD94, FBBD96, Duf97]. These experiments have been conclusive. Particularly, a topologic-based modeler kernel, named Topofil, has been designed by this way [BD94]. The formalization phase have been facilitated by the mathematical modeling of geometric objects. Better structured and more efficient, the quality of the developed code have been significantly improved. To achieve these large-size and complex specifications, these experiments have been done by mixing several existing algebraic languages (for example, order sorted-algebras) and introducing some dedicated features like ad hoc iterators. The main advantage of these first experiments was the development of a well structured and rather concise reference document. However, since the semantic counterpart of the specifications was not really defined, it was not possible to reason about these specifications, and also, of course, to use existing verification tools (provers, code generators, test generators).

The ultimate purpose of our work is to provide a formal specification language convenient for geometric modeling with tools. For this, the most natural way is to begin by reusing existing well established specification languages, and if possible already provided with tools. [PD00] presents a complete formalization of a case study also issued from the geometric modeling using a higher logic framework. But, unlike [PD00] which mainly focuses on the ability of handling large proofs, we prefer to express requirements in the simplest and clearest way. Thus, we will use specification languages whose one of the main recognized characteristics is to provide the user with friendly specification primitives ensuring abstract features, easy reading, ... We first experiment with two existing specification languages, respectively the B method [Abr96] and CASL [oLD00, BM00, Mos99]. These two specification languages are representative for the two main approaches to specify data structure, that can be respectively classified as model-oriented or property-oriented. Intuitively and to simplify, the first approach (B, Z, VDM,...) is based on set theory: a state is defined satisfying some invariants properties and some operations, often defined by pre and post conditions, act on it. On the contrary, the second approach (CASL, CLEAR, OBJ,...) is based on algebraic and logic technics: a set of axioms specify the intended properties.

The B method is a complete methodology for formal software development provided with tools (B-toolkit, Atelier B), and already successfully applied to large industrial projects [BBFM99]. The B language [Abr96, Abr91] offers high level objects, for instance sets, relations and functions equipped with powerful operations. So the choice to experiment B in the domain of geometric modeling is based on the intuition that the B method can be used to easily express mathematical models such as those found in this domain. Another more technical reason is that the specification language used to develop Topofil features pre-conditions, a very central notion in B.

The CASL language [BM00, Mos99] has been developed by the algebraic specification community very recently in order to get a common agreement on some basic concepts. It gathers many features found in the different algebraic languages. Consequently this choice allows us to stay in the algebraic flavor and to follow out the first experiment around Topofil. CASL combines the advantages of having many high level primitives and of being equipped with tools, or at least compatible with existing tools [Mos00].

In this paper, the two languages B and CASL will be only slightly presented through the explanations associated to the specifications introduced in relation to the development of our case study. But, of course, the reader can find additional informations in reference documents [Abr96] and [oLD00].

In the second section, we recall our application domain we focus on the chamfering operation. Then we present two specifications of the chamfering operation, using the two languages B and CASL, respectively in the sections 3 and 4. We emphasize that in case of geometric modeling, informal requirements given as mathematic-based description may be rather easily translated into formal specifications. Moreover, we want to provide specifications both in accordance with the style advocated by the underlying specification language and easily understandable by the experts of geometric modeling. Finally, we give some concluding remarks in section 5.

2 A brief overview on geometric modeling

2.1 The topology-based modeling

In geometric modeling, we can distinguish two opposite ways to represent a geometric object: either you are interested in curves, surfaces and numerical approximations, so just the shape of geometric objects, or you take care of the *topology*, i.e. the structure of geometric objects. We are interested in the topological approach, the knowledge of topology being often useful and important. We can quote the following reasons:

- The topological information is significant as a functional information in many applications. For example, in CAD, a 3D object is structured in 2D objects (faces) themselves broken up into 1D objects (edges) and 0D objects (vertices). Such a combinatorial representation of topology is voluntarily redundant with information of embedding (the given of coordinates for each topological element) and represents this structuring efficiently.

- Both computations of topological properties are simpler and algorithms are more efficient. Thanks to a topological structuring, from an algorithmic point of view, it will be easier in a 3D object to reach an adjacent face with a given face, and that without worrying about the shape of these faces. On the contrary, without this topological information, it would be necessary to traverse all the faces of the considered object with the additional risk of errors induced by numerical approximations.

In topology-based modeling, a geometric object is characterized by its structure, i.e its *topology*, and its form, i.e. its *embedding* (see figure 1). So, we have to introduce the two following models:

- a *topological model* describes the topology of the object, i.e. the cells (vertices, faces, edges, volumes...) of the subdivision as well as the various relations between these cells.
- an *embedding model* defines the embedding of the subdivision. This model defines the object shape, or in other words, the way in which the object is embedded in space. To embed an object consists in associating an embedding model with every elementary topological element issued from the topological model. Thus, each cell of the topological model is associated with a geometric object preserving the dimension: a point, a curve, a surface correspond respectively to a vertex, an edge, a face.

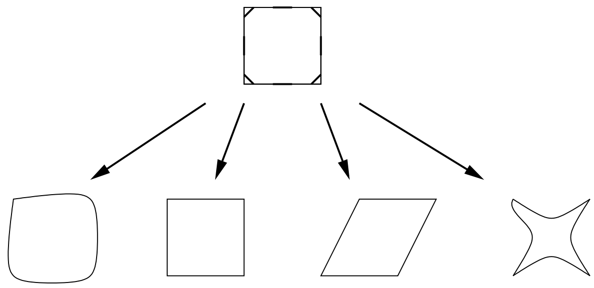

Figure 1: a topological model (n-G-Map) and several associated embeddings.

In this paper, we are only interested in the topological model and more precisely in the model of n-dimensional generalized maps [Lie94] which allows to represent unspecified-sized manifolds, opened or not, and orientable or not.

2.2 What's a n-G-Map ?

The definition of n-G-Map (or n-dimensional generalized map) was introduced by Pascal Lienhardt in [Lie88]. A n-G-Map is a model defining the topology of a subdivision of space and simply defined from a type of basic elements, the *darts*, on which elementary functions, the *involutions*, act. Before giving a mathematical definition, we give some intuitions on what is a n-G-map.

The main underlying idea behind this model is to decompose a complex object in a dart set, each of them carrying the local topological information: for each dimension, each dart is linked to another dart, possibly itself. This local piece of information allows to retrieve the whole topological structure. Let us consider the geometric object of the figure 2(a), which is a rectangle built from two squares. Figure 2(b) shows the same object by highlighting the adjacency relations between the two squares, i.e. two surfaces (dim 2). Figure 2(c) shows the squares in a disjointed way, each square being broken up into four edges (dim 1) bound by their adjacency relations. Lastly, on the figure 2(d), the edges are dissociated in vertices (dim 0), and we continue to reveal their adjacency

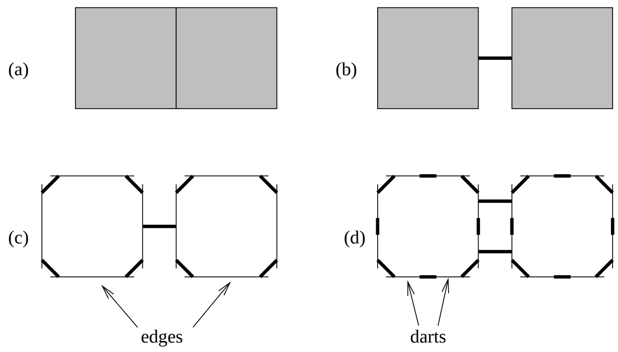

Figure 2: Decomposition of an object.

relations. Thus we have broken up the initial object into objects of dimension 0, which are the *darts*, basic elements of the n-G-Maps represented by thiner lines.

Notice that all the relations which could relate to faces or edges of our basic object were deferred on the darts composing these edges or these faces. Thus some adjacency relations will be modeled by an involution α_i whose index gives the type of an considered adjacency relation. For example, two darts linked by α_0 belong to an edge. A mathematical definition is given by:

Definition 1 *A* **n-dimensional generalized map** *(or* **n-G-map***) is an algebra* $G = (D, \alpha_0, ..., \alpha_n)$ *such as:*
- *D is a finite set of* darts,
- *α_i is an involution*[1] *on D ($0 \leq i \leq n$), and* (c_1)
- *$\alpha_i\alpha_j$ is an involution ($0 \leq i < i+1 < j \leq n$)* (c_2).

Let us consider the figure 3 on which we have reported all the adjacency relations (thicker lines): $\alpha_0, \alpha_1, \alpha_2$. α_0 represents the link between vertices, α_1 the link between edges and α_2 the link between surfaces. Note that a dart can be linked to himself by an α_i. Generally we omit to report these links on the figures.

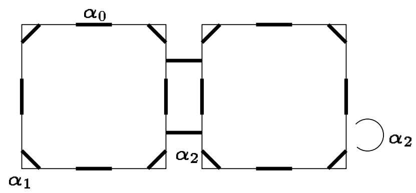

Figure 3: A 2-G-map representing two linked squares.

Vocabulary : Let us consider a n-G-map $G = (D, \alpha_0, ..., \alpha_n)$ and $i \in [0, ..., n]$. A dart is said to be **i-sew** with the dart $d' \in D$ if $d\alpha_i = d'$, **i-free** if $d\alpha_i = d$ and **free** if for any $i \in [1..., n]$, $d\alpha_i = d$.

A n-G-map is a combinatorial structure whose basic element is the dart. It is however natural to retrieve the edges, the faces or volumes which compose the object that we handle. To find this information from a topological point of view, with the n-G-Maps, we introduce the concept of

[1] An application $\alpha_i : D \to D$ is said to be an involution iff $\forall d \in D$ $d\alpha_i\alpha_i = d$ using the notation $d\alpha_i$ for the application of α_i on the dart d.

orbit whose some particular cases, called the *cells*, represent the notions what we wait for. A 0-cell represents a vertex, a 1-cell, an edge, a 2-cell, a face, and a 3-cell, a volume.

Definition 2 *Let us consider a n-G-map $G = (D, \alpha_0, ..., \alpha_n)$, a dart $d \in D$, and P a set of m permutations $\{p_1, \ ... \ , p_m\}$ on D. We call* **orbits** *of d compared to P the unit*

$$\{d' \in D \mid \exists\, p'_1 \in P, \ ... \ , \exists p'_k \in P, \ with \ d' = dp'_1...p'_k\}$$

(we can have $p'_i = p'_j$ with $i \neq j$), and we note this orbit $< p_1, \ ... \ , p_m >$ (b).

Then, if we consider a n-G-map $G = (D, \alpha_0, ..., \alpha_n)$, a dart $d \in D$, the **i-cell** $(i \leq n)$ containing d, is the orbit $< \alpha_0, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_n >$ (d). For instance, if we consider the 2-G-map G of the figure 4(a), we can define several cells on G. These various cells are represented in the figure 4(b), 4(c), 4(d) and 4(e); The dart d and its corresponding cells are represented in full bold features. The 0-cell $<\alpha_1, \alpha_2>(d)$ which defines a vertex is represented on the figure 4(b). On the figure 4(c) the 1-cell $<\alpha_0, \alpha_2>(d)$ defines an edge, on the figure 4(d) the 2-cell $<\alpha_0, \alpha_1>(d)$ defines a face, and on the figure 4(e) the orbit $<\alpha_0, \alpha_1, \alpha_2>(d)$ defines the related connected component of b.

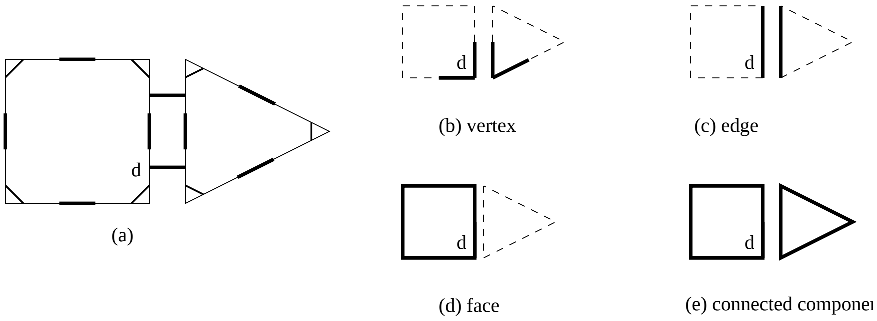

Figure 4: several orbits for a dart d.

2.3 The chamfering operation

Naively, the chamfering operation is often interpreted as being a current operation in joinery: to plane. This interpretation is just an useful and intuitive way to understand the chamfering operation, but not the general result of the chamfering operation in geometric modeling. We would rather speak about *rounding*. In fact, if we consider a solid in dimension 3 and we want to chamfer one of its corners, we are in the intuitive case.

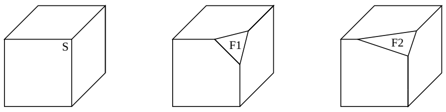

Figure 5: Two ways to chamfer a cube corner.

For example on the figure 5, we chamfer the corner of a cube following two different planes. Thus we replace a vertex S by a face F_1 or F_2. On the other hand, we want now to chamfer an edge; then the result is not geometrically perceptible, but only topologically (see fig. 6). Actually the waited and intuitive result in the edge case is the result of the rounding operation which is a combination between the chamfering of the edge and of all its incident vertices. So the chamfering operation is a geometric operation which modifies both the geometry and the topology of an object. In this paper, we are just interested in the topological chamfering.

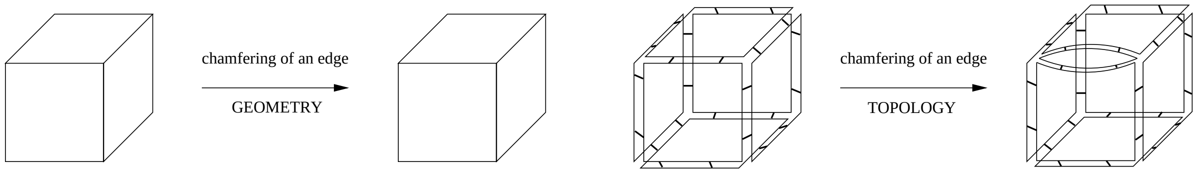

Figure 6: An edge chamfering. The used topological model is one of 2-G-map. We have just reported the adjacency relations between faces i.e. the involution α_2.

The topological approach that we follow is one of Elter [Elt94]: to chamfer a i-cell consists in replacing this cell I by a n-cell N whose the number of edges bordering it is equal to the number of incidental faces with I. For example, in the figure 7, the cases (a) and (b) represent two acceptable topologies, and the case (c) does not. Such an operation is very complex: you have to unsew some darts for a particular dimension, to add new darts and to make several seams (see fig. 8).

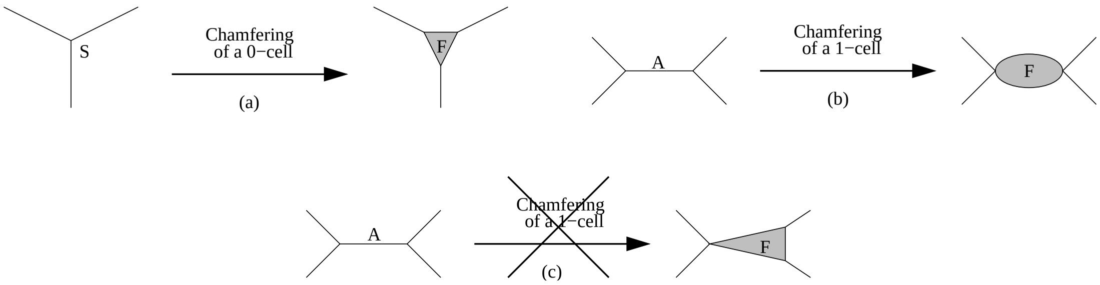

Figure 7: Several topological chamfering examples.

A so complex operation can't be easily defined. Being given a n-G-map G, a first method is to define explicitly and completely the n-G-map resulting from the chamfering of a cell of G. The process is to define each component of the new n-G-map according to the old one. Then we obtain the following definition of the chamfering which is given in [Elt94].

Definition 3 *Let us consider a n-G-map* $G = (D, \alpha_0, ..., \alpha_n)$ *and a i-cell* $C =< \alpha_0, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_n > (d)$ *included in* G. *We denote* D_i *the dart set of* C. *Let us consider a family* $(D_j)_{j\in[i+1,n]}$ *of dart sets such as for each* $j \in [i+1, n]$,

- $D \cap D_j = \emptyset$,
- *for all* $k \in [i+1, n],\ k \neq j,\ D_j \cap D_k = \emptyset$,
- *there exists a bijection* $\varphi^j : D_i \to D_j$. *Moreover, we introduce the bijection* $\varphi^i = id_{D_i}$.

Then, let E *be the n-G-map* $(D^E, \alpha_0^E, ..., \alpha_n^E)$ *defined by:*

1. $D^E = D \cup \bigcup_{j\in[i+1,n]} D_j$

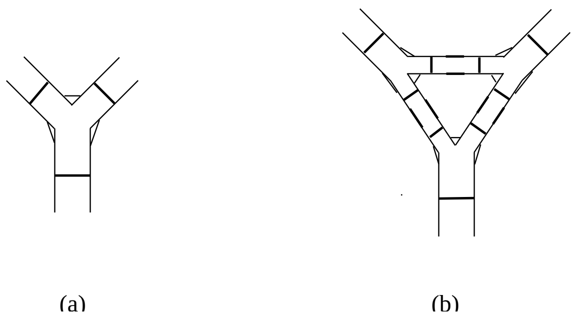

Figure 8: A 0-cell in (a) and its chamfering result in (b).

2. $\forall d \in D,\ \forall k,\ (0 \leq k \leq n), d\alpha_k^E = \begin{cases} d\varphi^{i+1} & \textit{if } k = i+1 \textit{ and } d \in D_i \quad (2.1) \\ d\alpha_k & \textit{else} \quad (2.2) \end{cases}$

3. $\forall j,\ (i < j \leq n),\ \forall d \in D_i,\ \forall k,\ (0 \leq k \leq n), d\varphi^j\alpha_k^E = \begin{cases} d\alpha_k\varphi^j & \textit{if } 0 \leq k < i \quad (3.1) \\ d\alpha_{k+1}\varphi^j & \textit{if } i \leq k < j \quad (3.2) \\ d\varphi^{j-1} & \textit{if } k = j \quad (3.3) \\ d\varphi^{j+1} & \textit{if } k = j+1 \quad (3.4) \\ d\alpha_k\varphi^j & \textit{if } j+1 < k \leq n \quad (3.5) \end{cases}$

E is called the n-G-map resulting from the chamfering of C in G.
We denote $E = Che(C, G, (D_j)_{j\in[i+1,n]})$.

Remarks: This definition calls for some comments:

- The functions $\varphi^j,\ j \in [i+1, n]$ enable us to handle the new added darts.
- The item (2.2) indicates that the functions of the n-G-map E are equal to those of the n-G-map G on all the old darts.
- The item (2.1) says how the old darts are connected to the new added darts.
- The condition 3 indicates how the new darts are connected between them. Let us notice that the items (3.1),(3.2) and (3.3) give the links between new darts according to links between darts of G.

In the preceding definition, we totally define the new n-G-map from the old one: we explicitly give its dart set and each of its involution is explicitly described. We finally have a very operational definition. As the B language is based on a set theory and offers sets, relations and functions equipped with powerful operations, we have the necessary tools to directly specify the definition 3. On the contrary, specifying using the axiomatic approach led us to abstract this definition. N-G-maps check some invariants and we do not make the most of this information in the preceding definition. Then, the new approach consists in using these invariants: we explicitly give some modifications (seam, unseam, adding of dart,...) and some other changes are implicitly deduced. Let us consider a basic operation like the seam: when you 2-sew two darts d and d', a lot of others are 2-sewed too in order to preserve the n-G-map invariants. In an implicit definition, we just want to say d and d' are 2-sewed. But it is not sufficient, we have to ensure it is the only changes, i.e. excepted that d and d' are 2-sewed and the modifications implied by the invariants, there is no other change. To solve this problem, we introduce an inclusion concept on the n-G-maps, which enables to compare two n-G-maps. Let us give its definition.

Definition 4 *Let us consider a n-G-map $G = (D, \alpha_0, ..., \alpha_n)$ and a n-G-map $G' = (D', \alpha'_0, ..., \alpha'_n)$. Then we say G* **is included in** *G' iff:*

- *$D \subseteq D'$,*
- *$\forall i \in \{0, .., n\}$, $\forall d, d' \in D$, if $\alpha_i(d) = d'$ with $d' \neq d$ in G then $\alpha'_i(d) = d'$ in G',*

And we denote: $G \subseteq G'$.

Using the inclusion notion, we can implicitly define the basic operation of sewing: when you 2-sew two darts d and d' in a n-G-map G, the result is the smallest n-G-map containing G and such as d and d' are sewed. In the same way, we are now able to define the chamfering operation implicitly. The n-G-map resulting from the chamfering operation is defined as the smallest n-G-map, within the meaning of inclusion previously defined, checking some properties. We have the following definition.

Definition 5 *Let us consider a n-Gmap $G = (D, \alpha_0, ..., \alpha_n)$, a dart d in D and $i \in [0, n[$.*
We notice D_i the dart set of the cell $C =< \alpha_0, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_n > (d)$ and $L = (D^L, \alpha_0^L, ..., \alpha_n^L)$ the largest n-G-map included in G such as: $\forall d \in D_i,\ d\alpha_{i+1}^L = d$.
Let us consider a family $(D_j)_{j \in [i+1,n]}$ of dart sets such as for each $j \in [i+1, n]$,

- *$D \cap D_j = \emptyset$,*
- *for all $k \in [i+1, n]$, $k \neq j$, $D_j \cap D_k = \emptyset$,*
- *there exists a bijection $\psi^j : D_i \to D_j$.*

Then $I = (D^I, \alpha_0^I, ..., \alpha_n^I)$ is the smallest n-G-map which contains L and such as:

1. *$D^I = D \cup \bigcup_{j \in [i+1,n]} D_j$*
2. *for all $d \in D_i$, $d\alpha_{i+1}^I = d\psi^{i+1}$*
3. *for all $d \in D_i$, for all $k \in [i+1, n-1]$, $d\psi^k \alpha_{k+1}^I = d\psi^{k+1}$*
4. *for all $d_1, d_2 \in D_i$,*
   - *for all $k \in [0, i-1]$, $d_1\alpha_k = d_2 \Rightarrow d_1\psi^n\alpha_k^I = d_2\psi^n$*
   - *for all $k \in [i+1, n]$, $d_1\alpha_k = d_2 \Rightarrow d_1\psi^n\alpha_{k-1}^I = d_2\psi^n$*

We denote $I = Chi(C, G, (D_j)_{j \in [i+1,n]})$.

Remarks: This definition calls for some comments:

- As before, the functions ψ^j, $j \in [i+1, n]$ enable to handle the new added darts.
- The items (2) and (3) indicate that for each dart of D^i, we add $n-i$ darts sewed in a particular way.
- The item (4) indicates how are sewed certain added darts (those reached by ψ^n).

- There are only the seams to be defined explicitly. As we obtain a n-G-map, the other seams follow from the invariants. Which are the seams to explicitly give? For that, we use a characteristic of chamfering: a i-cell is replaced by a n-cell which can be completely determined from the starting cell using a dual concept. For example, if we consider the figure 9(a), i.e. a 0-cell in a 2-G-map, we have in the figure 9(b), the corresponding 2-cell after chamfering. The principle is simple: if we consider the function g defined from the dart set of the figure 9(a) to the dart set of the figure 9(b), and associating the darts as marked on the figure, and two darts d and d' included in the figure 9(a):

  - if d and d' are linked by α_j $(j < i)$, then $g(d)\alpha_j = g(d')$,
  - if d and d' are linked by α_j $(j > i)$, then $g(d)\alpha_{j-1} = g(d')$.

  In our case, the function g is simply the function ψ^n of the definition 5.

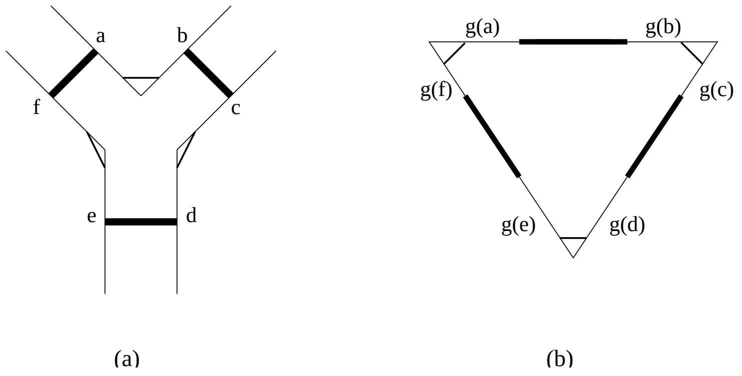

Figure 9: a 0-cell and a 1-cell and their "duals" in a 2-G-map.

Finally, we have two distinct definitions, one providing an explicit characterization of the chamfering operation, the other giving an implicit characterization. Of course, these two definitions provides the same n-G-map as result providing that they share the added dart set.

Proposition 1 *Let us consider a n-Gmap $G = (D, \alpha_0, ..., \alpha_n)$, a dart d included in D and $i \in [0, n[$. We denote C the i-cell $< \alpha_0, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_n > (d)$. Moreover, we introduce a family $(D_j)_{j\in[i+1,n]}$ of dart sets such as for each $j \in [i+1, n]$,*

- $D \cap D_j = \emptyset$,
- *for all* $k \in [i+1, n]$, $k \neq j$, $D_j \cap D_k = \emptyset$,
- *there exists a bijection* $\psi^j : D_i \to D_j$.

Then, we have:

$$Che(C, G, (D_j)_{j\in[i+1,n]}) = Chi(C, G, (D_j)_{j\in[i+1,n]})$$

The complete proof of the proposition 1 is given in [LMA+00]. The proof carries out in a traditional mathematical way and is just technical. The idea of the proof is the following: we consider a n-G-map which checks the definition 3 and we prove it checks the different assertions of the definition 5 and is the smallest checking them.

Now we are interested in the formalization of the chamfering operation with B and CASL. These two languages have some different characteristics which imply different styles of specification. B allows to easily formalize the chamfering explicit definition 3. In the following section, we present this approach. Afterward, in an other section, we will see how CASL allows to naturally specify the chamfering implicit definition 5.

3 The B approach

We provide an abstract model of a 3-G-map within the B language under the form of an abstract state machine in two parts. We comment this model below.

The scope of our formal model is to give a comprehensive representation of the chamfering operation. Consequently our abstract machine is composed of a state that hides the 3-G-map that may evolute and an operation allowing to chamfer a cell of the encapsulated 3-G-map.

MACHINE
GChanf

SETS
DARTS

ABSTRACT_VARIABLES
darts, *alpha*

INVARIANT
$darts \subseteq DARTS \wedge$
$alpha \in 0 \,..\, 3 \rightarrow (darts \rightarrow darts) \wedge$

$\forall\ (i).(i \in 0 \,..\, 3 \Rightarrow is\_an\_involution(alpha(i))) \wedge$

$\forall\ (i, j).(i \in 0 \,..\, 1 \wedge j \in i{+}2 \,..\, 3 \Rightarrow is\_an\_involution\ (alpha(i);alpha(j))\ )$

DEFINITIONS
$Cell(dart,dim) == \mathbf{closure}(\ \bigcup\ (Dim).(Dim \in 0 \,..\, 3 \wedge Dim \neq dim \mid alpha(Dim)))\ [\{dart\}]$;

$is\_an\_involution(g) == ((g;g)=\mathbf{id}(darts))$;

$is\_a\_partition(f,nw\_darts,dim) == \forall\ (i,j).(i \in\ dim{+}1 \,..\, 3 \wedge j \in\ dim{+}1 \,..\, 3 \wedge i \neq j \Rightarrow$
$\mathbf{ran}(f(i)) \cap \mathbf{ran}(f(j)) = \emptyset \wedge$
$nw\_darts = \bigcup\ (l).(l \in\ dim{+}1 \,..\, 3 \mid \mathbf{ran}(f(l))))$

Specification B: The abstract machine state.

3.1 Presentation of the abstract machine state

The clauses ABSTRACT_VARIABLES and INVARIANT are nothing less than a translation of the mathematical definition 3 of a n-G-map (adapted to the dimension 3). Let us call this mathematical definition an informal description. Thus we introduce 2 variables *darts* and *alpha*. The set *darts* (D in the informal definition) is defined as a subset of $DARTS$ that is the universe of all the possible darts. The function *alpha* gathers all the involutions α_0, α_1, α_2 and α_3: the B object $alpha(i)$ formalizes the mathematical object α_i. Its description is two-party: first we express that it is a total function from *darts* to *darts*, and then that it is an involution by using the composition operator (;) provided by B. The last part of the invariant predicate expresses the condition (c_2) of the informal definition.

3.2 The chamfering operation

The chamfering operation of a dart d (of the 3-G-map) for the dimension 0, 1 or 2 in our case is seen here as an operation that modifies the 3-G-map on which it acts: it adds some new darts, sews the new darts and modifies some old seams. The old darts (by opposition to the new supplementary ones brought by the operation) affected by the operation are those of the i-cell of the dart d, denoted in our B model by $Cell(d, i)$ inlined as $\textbf{closure}(\bigcup (Dim).(Dim \in 0 \,..\, 3 \wedge Dim \neq i \mid alpha(Dim)))[\{d\}]$. This expression specifies the i-cell of a dart d as the darts that may be reached from d by applying one involution (different from $alpha(i)$), the darts that may in turn be reached from the previous ones and so on. It is written essentially as the range of the reflexive and transitive closure of all the involutions except $alpha(i)$. In fact we are only concerned with the images of the dart d.

To specify the chamfering operation, we started from the operational chamfering definition 3 and adapted it to a comprehensive B presentation. The set *new_darts* of new darts added in the 3-G-map is specified as any subset of $DARTS$ that disjoined with *darts*, the set of darts are currently involved in the 3-G-map. For each dimension j in $[i+1, 3]$, any dart of the i-cell concerned by the chamfering is mapped to a different new dart of *new_darts*, phi(j)(d). We can liken $phi(j)$ to a renaming substitution of $Cell(d, i)$, consequently an injective total function from $Cell(d, i)$ to *new_darts*. Furthemore, the different renaming substitutions do not overlap. These requirements are formalized in the B language by the following predicates: $phi \in i + 1..3 \to (Cell(d, i) \rightarrowtail new\_darts) \wedge is\_a\_partition(phi, new\_darts, i)$ where $\rightarrowtail$ indicates injective function and *is_a_partition*, defined as a B definition, specifies that the ranges of $phi(i+1)$ ... $phi(3)$ form a partition of *new_darts*. Notice that in the definition 3 $Cell(d, i)$ and $ran(phi(j))$ are called respectively D_i and D_j. In the B specification, let us remark that the variable $\varphi(j)$ is only defined for $j \geq i+1$ since $\varphi(i)$ simply represents the identity on the dart set D_i.

The global structure of the generalized substitution that describes the chamfering operation is as follows:

$darts := darts \cup new\_darts ||$

$alpha := \lambda\ (k).(k \in 0 \,..\, 3 \mid alpha\ (k) \lhd\!\!-\ \{\ d1,\ b2 \mid E_1 \vee\ E_2\ \vee\ E_3\ \vee\ E_4\ \vee\ E_5\ \})$

Informally, it can be read: *update the set of darts by adding the new darts and update the different involutions by overloading each of them by a set of new seams* $d_1 \mapsto d_2$ *that satisfy the constraints* E_1, E_2, E_3, E_4 *or* E_5.

We can classify the different conditions in two groups: the last case allows to sew a new dart with a dart of the i-cell, E_1, E_2, E_3 and E_4 all concern the sewing of new darts. More precisely, E_1 and E_2 concern two new darts introduced by a same $phi(j)$. Our five cases relate to the seven cases of definition 3:

- the constraint E_1 is the translation of the property 3.1 of the mathematical definition 3. This case only concerns the involutions $alpha(k)$ where k is included in 0 to $dim - 1$. Informally, with E_1, if two darts of $Cell(d, i)$ were k-sewed, then we k-sew the new darts they are mapped to by $phi(j)$.

- the constraint E_2 is an adaptation of the property 3.2 of the mathematical definition 3. In E_2 it is the same situation except that we expect then the old darts to be $(k+1)$-sewed. In the same way, is the translation of the property 3.5.
- E_3 catches the points 3.3 and 3.4. It is based on the property 3.4 but some cases stand by, more precisely those that concern the use of $\varphi(i)$. These corresponding cases will be found in the condition E_5. The property 3.3 sews the darts in a direction and the property 3.4 in the opposite one. So, instead of writing another case, we did it in the same one. Informally, this case applies to new darts concerned by $phi(k)$ and $phi(k-1)$.
- our last condition E_5 corresponds to the property 2.1 and to the cases not treated in the third case. The property 2.1 sews the darts in a direction and the untreated cases in the opposite one.

Notice that the property 2.2 of the definition 3 is not present in our previous cases. Indeed this property caracterises the unchamfered part of the object. In B, with the overriding, we can modify a subset of a variable exactly as the chamfering operation which modifies only a part of the object. For this reason we do not have to express this property formally since it is implicitly included in overriding semantics.

OPERATIONS

Chamfering $(d,\ i) =$
PRE $d \in darts\ \wedge\ i \in 0\ ..\ 2$
THEN

ANY *phi*, *new_darts*
WHERE
$new\_darts \subseteq DARTS$ - $darts\ \wedge$
$phi \in i{+}1\ ..\ 3 \rightarrow (Cell(d,i) \rightarrowtail new\_darts)\ \wedge$
$is\_a\_partition(phi, new\_darts, i)$

THEN

$darts := darts \cup new\_darts\ ||$

$alpha := \lambda\ (k).(k \in 0\ ..\ 3\ |$
$alpha\ (k) \lhd\!\!-\ \{\ d1,\ d2\ |\ d1 \in Cell(d,i) \cup new\_darts \wedge d2 \in Cell(d,i) \cup new\_darts \wedge ($
$(k \in 0\ ..\ i{-}1 \wedge \exists\ (j,d3).(j \in i{+}1\ ..\ 3 \wedge d3 \in Cell(d,i)$
$\wedge\ d1 = phi(j)(d3) \wedge d2 = phi(j)(alpha(k)(d3))))$

$\vee\ (\ \exists\ (j,d3).(j \in i{+}1\ ..\ 3 \wedge k \in i\ ..\ j{-}1 \wedge d3 \in Cell(d,i)$
$\wedge\ d1 = phi(j)(d3) \wedge d2 = phi(j)\ (alpha(k{+}1)(d3))))$

$\vee\ (k \in i{+}2\ ..\ 3 \wedge \exists\ (d3).(d3 \in Cell(d,i) \wedge ((d1 = phi(k)(d3) \wedge d2 = phi(k{-}1)(d3))$
$\vee\ (d2 = phi(k)(d3) \wedge d1 = phi(k{-}1)(d3)))))$

$\vee\ (\ \exists\ (j,d3).(j \in i{+}1\ ..\ 3 \wedge k \in j{+}2\ ..\ 3 \wedge d3 \in Cell(d,i)$
$\wedge\ d1 = phi(j)(d3) \wedge d2 = phi(j)\ (alpha(k)(d3))))$

$\vee$ ($k = i$+1 $\wedge$ (($d1 \in$ *Cell(d,i)* $\wedge$ $d2 = phi(i+1)(d1)$)
$\vee$ ($d2 \in$ *Cell(d,i)* $\wedge$ $d1 = phi(i+1)(b2)$)))))})
END
END
END

Specification B: The chamfering operation.

3.3 Conclusion

In this section we have proposed an abstract model of a 3-G-map equipped with the chamfering operation. This description generates some proof obligations that we have not yet verified mechanically. Indeed the model is correct if the chamfering operation preserves the invariant. Verifying that *the new seams preserve the involutions* is a specially difficult task and will require to prove numerous properties about involutions. Then a next step will be to write different refinements of the initial model, more and more concrete and operational in the hope of producing executable code or at least pseudo-code.

Let us conclude this first approach by some remarks about the appropriateness of using B for the geometric modeling and more precisely the chamfering operation.

- The B language is based on the set theory and offers many relations and functions with powerful operations. Moreover, the concept of the abstract state machine allows to have encapsulate basic data types and to make its operations directly correspond to the modeler primitives: this represents an appropriateness between the abstract interface and the concrete interface.
- The B formalism allows us to obtain quite easily a formal specification from a mathematical definition. This is a very strong argument because this step *informal/formal* is often delicate in other application domains or with other formal methods.
- The successive refinements may lead us to reasoned (because every choice generates proof obligations) and reasonable implementation choices.

4 CASL approach

CASL is a language of algebraic specification. So, in a traditional way, we have to define in the first time the various necessary data types. To do it, we use some CASL libraries [RMS00] which particularly provides the data types for sets and functions. Indeed, contrary to the language B, sets and functions are not inherent in CASL which is based on the first order logic. In the second time, we define the operations and predicates, as it happens the chamfering.

4.1 The abstract data types

4.1.1 The basic data types

On looking the n-G-map definition, we have to introduce some basic data types in order to define a type named *3Gmap* representing the 3-G-map. Then we define first the following data types:

- a type *Dart*,
- a type *Dim*, which provides the integers 0, 1, 2 and 3 that are only dimensions used in a 3-G-map,
- a type *Function[S]*, which provides applications on elements of type S.

Since a dart has not a particular characteristic, we simply state darts by a sort:

sort *Dart*

In a 3-G-map, we have to manipulate the functions $\alpha_0, \alpha_1, \alpha_2, \alpha_3$. It is useful to have a generic operation, named α, enabling us to write α_i, for any i included in 0, 1, 2 and 3. Such an operation allows to quantify on the value of i and so to be more concise. So we define a subtype of $\mathbb{N}$, named *Dim*, which would contain the natural integers less than or equal to 3. A specification of natural integers (Nat) is given in the library NUMBERS [RMS00].

sort $Dim = \{n : \ Nat.\ n \leq 3\}$

So *Dim* is defined as a subsort of the sort Nat: a value n in Nat is the embedding of some value from *Dim* iff n is less than or equal to 3. The last basic data type necessary to define the data type *3Gmap* is *Function[S]*. This type must provide functions whose field is included in a finite set, the darts of the 3-G-map. The first-order logic is sufficient for this kind of function. Indeed, it is possible to represent a function with finite field like a *finite set of pair*, which checks some properties. To define the data type FUNCTION, we use the library STRUCTUREDDATATYPES which includes the data types *FiniteMap* [RMS00]. Intuitively, a finite map is a function which takes its arguments in a finite set and return values in a finite set. To specify 3-G-map, we need functions taking darts in arguments and returning darts, so a particular case of finite map. The head of the new specification FINITEFIELDFUNCTION looks like:

---

spec FINITEFIELDFUNCTION[**sort** S] =
  FINITEMAP[**sort** S][**sort** S **fit sort** $T \to S$]
    with $FiniteMap[S, S] \to Function[S]$
then
  ...
end

Specification CASL: The head of the *FiniteFieldFunction* specification.

The specification FINITEFIELDFUNCTION is parametrized by a specification which just consists in a *sort S*. FINITEFIELDFUNCTION is an extension of an instantiation of the named specification FINITEMAP[**sort** S][**sort** T]. We indicate how it "fits" by giving a map from the parameter signature to the argument signature (keyword **fit**). If it is not ambiguous, the fitting may be left implicit. Moreover, we apply a translation (keyword **with**) on this specification. We declare explicitly the translation of the symbol $FiniteMap[S, S]$ to the new symbol $Function[S]$. Afterward and by instantiation, we will obtain functions taking darts in arguments and returning darts too.

4.1.2 The 3-G-map data type

We define a 3-G-map by using a free specification which imposes some semantic constraints: we just consider the model restricted to (the isomorphism class of) its inital models. Then, a 3-G-map is declared with a type constructor and some additional constraints on this type. We obtain the following specification.

---

spec 3GMAP =
  ...
then
  free
  { **type** *3Gmap* ::= $gmap(D : FinSet[Dart]; \alpha : FiniteMap[Dim, Function[Dart]])$;
    var G : *3Gmap*
    - $\forall i : Dim.\ G.\alpha_i :: G.D \to G.D$
    - $\forall i : Dim.\ involution(G.\alpha_i, G.D)$
    - $\forall i, j : Dim.\ j > i + 1 \Rightarrow involution(G.\alpha_i \circ G.\alpha_j, G.D)$

  }
end

Specification CASL: The 3-G-map data type.

We declare a 3-G-map by using a type: each element of type *3Gmap* is composed of a dart set D (the type *FinSet[Dart]* is a "fitting" of the type *FinSet[S]* coming from the library STRUCTURED-DATATYPES) and a generic function α which provides the functions $\alpha_0, \alpha_1, \alpha_2, \alpha_3$. Besides such a combinatorial structure has to check some properties to be a 3-G-map, these properties are given by three axioms:

- the first ensures that the application α_i have their domains equal to D and their ranges included in D,
- the second and third ensure the involution properties. Let f be a variable of type *Function[S]*, D be a variable of type *FinSet[Dart]*, *involution(f, D)* is a predicate which holds if f is an involution on the set D.

By using a free specification, each element of the carrier set corresponding to *3Gmap* is exactly what is attempted.

4.2 The chamfering operation

To specify the chamfering in CASL, we start from the definition 5. So we have to specify the n-G-map inclusion, denoted by $\subseteq$:

---

  pred $\_\_ \subseteq \_\_\ :\ 3Gmap \times 3Gmap$;
vars $G,\ G'\ :\ 3Gmap$
• $G \subseteq G' \Leftrightarrow$
$\forall d : Dart.\ d \in G.D \Rightarrow (d \in G'.D \wedge \forall i : Dim.\ \neg(iFreeDart(G, d, i)) \Rightarrow G.\alpha_i(d) = G'.\alpha_i(d))$

Specification CASL: The 3-G-map inclusion operation.

We can remind of that by using the inclusion, we let some seams implicit. In fact, we let the invariants "work". The chamfering is an operation taking a 3-G-map G, a dart d and a dimension i in arguments, and giving a 3-G-map in result. d and i indicate in a single way the cell that we want to chamfer. We name it C. The chamfered 3-G-map is then specified as being the smallest 3-G-map checking a local predicate. We obtain the following specification:

---

spec CHAMFERED3GMAP =
  3GMAP
then
  op $\_\_.\psi{-}{-}$: $\mathit{3Gmap} \times \mathit{Dim} \to \mathit{Function}[\mathit{Dart}]$;
  vars G : $\mathit{3Gmap}$; d, d' : Dart; i, j : Dim
  - %[ψ_range] $\neg(G.\psi^i(d) \in G.D)$
  - %[ψ_def] $G.\psi^i(d) = G.\psi^j(d') \Rightarrow (d = d' \wedge i = j)$

  pred *isChamfered* : $\mathit{3Gmap} \times \mathit{3Gmap} \times \mathit{Dart} \times \mathit{Dim}$;
  vars G, Gafter : $\mathit{3Gmap}$; d : Dart; i : Dim
  - %[*isChamf_def*] $\mathit{isChamfered}(\mathit{Gafter}, G, d, i) \Leftrightarrow$
    $(G.\mathit{unsewSet}(G.\mathit{cell}(G, d, i), i+1) \subseteq \mathit{Gafter}$ %%A_1
    $\wedge$
    $\forall d' : \mathit{Dart}.d' \in \mathit{cell}(G, d, i) \Rightarrow (G.\alpha_{i+1}(d') = G.\psi^{i+1}(d')$ %%A_2
    $\wedge$
    $\forall k : \mathit{Dim}.\ ((i+1 \leq k \wedge k \leq 2) \Rightarrow \mathit{Gafter}.\alpha_{k+1}(G.\psi^k(d')) = G.\psi^{k+1}(d'))$ %%A_3
    $\wedge$
    $\forall d1, d2 : \mathit{Dart}.\ \forall k : \mathit{Dim}.\ k \leq 2 \Rightarrow$
    $((k < i \wedge G.\alpha_k(d1) = d2) \Rightarrow \mathit{Gafter}.\alpha_k(G.\psi^n(d1)) = G.\psi^n(d2)))$
    $((k > i \wedge G.\alpha_k(d1) = d2) \Rightarrow \mathit{Gafter}.\alpha_{k-1}(G.\psi^n(d1)) = G.\psi^n(d2)))$ %%A_4

  op $\_\_.\mathit{chamfering}$: $\mathit{3Gmap} \times \mathit{Dart} \times \mathit{Dim} \to ?\ \mathit{3Gmap}$;
  vars G : $\mathit{3Gmap}$; i, j : Dim; d : dart
  - %[*cham_dom*] $\mathit{def}\ G.\mathit{chamfering}(d, i) \Leftrightarrow (d \in G.D \wedge i < 3)$
  - %[*cham_def1*] $\mathit{isChamfered}(G.\mathit{chamfering}(d, i), G, d, i)$
  - %[*cham_def2*] $\forall G' : \mathit{Gmap}.\ \mathit{isChamfered}(G', G, d, i) \Rightarrow G.\mathit{chamfering}(d, i) \subseteq G'$

end

Specification CASL: The chamfering operation.

To specify the chamfering operation, we stay very close to the definition 5. In this definition, we begin by introducing the family $(D_j)_{j\in[i+1,n]}$ of dart sets added during the chamfering and the family $(\psi_j)_{j\in[i+1,n]}$ of bijections between the previous added darts and the darts of the chamfering cell. In our specification, we explicitly define the family $(\psi_j)_{j\in[i+1,n]}$ of functions which are necessary to specify the items 1, 2, 3 and 4 of the definition 5. The family $(D_j)_{j\in[i+1,n]}$ is then implicitly defined as the ranges of the bijection ψ_j. These functions are specified by the introduction of the operation $\_\_.\psi{-}{-}$, which takes the current n-G-map as argument. Then we obtain a deterministic addition of new darts for a given n-G-map. Thanks to the axiom ψ*_range*, we know that D and D^j are disjointed and the axiom ψ*_def* ensures D_i and D_j $(i \neq j)$ are disjointed and ψ^j is an injection (and even a bijection as ψ is a total function). Now we have to specify the properties checked a

3-G-map I resulting from the chamfering of a i-cell defined by a dart d and a dimension i in a 3-G-map G. This properties are given in the predicate *isChamfered*: *isChamfered*(I, G, d, i) holds iff the conditions A_1, A_2, A_3 and A_4 hold.

- The condition A_1 corresponds to the fact that the 3-G-map I has to contain the 3-G-map L. L is the result of the operation $G.unsewSet(G.cell(G, d, i), i + 1)$ which provides the largest 3-G-map included in G and such as all the darts belonging in[2] $G.cell(d, i)$ are i-unsewed[3];
- The condition A_2 and A_3 respectively catch the items 2 and 3. The formalization is obvious, we have just to precise we consider the darts belonging in the cell which we we want to chamfer.
- The condition A_4 is the exact formalization of the item 4.

Finally, by definition, the 3-G-map resulting from the chamfering of the cell defined by d and i in G is the smallest 3-G-map such as *isChamfered*(I, G, d, i) holds. The chamfering operation is easily specified by this way (axioms *cham_def1* and *cham_def2*). Besides, this operation is partial because just defined for a 0,1, or 2-cell included in the 3-G-map (axiom *cham_dom*).

4.3 conclusion

In this section we have specified the 3-G-map model by creating the type *3Gmap*, and the chamfering operation. Notice that this specification is a piece of a larger specification containing other classical topological operations [LMA+00] and which has been successfully parsed with the CASL Tool Set (CATS)[Mos00] which allows the parsing of full CASL syntax (including libraries). To conclude, we can make some remarks about the CASL using for the topology-based modeling.

- Naturally, CASL offers a great specification modularity which allows to put in place a dependence hierarchy between sorts. Thanks to it, we are able to easily break up a problem into subproblem. This modularity is essential for understanding a problem.
- CASL has a pretty-printing syntax which allows to stay close to mathematical definitions. This point is very important because it restricts formalization mistakes and so improve the confidence in the appropriateness of the specification with respect to the requirements.
- Like the B method, CASL provides a demonstration support which will be useful to refine the specification for example.

5 Conclusion

This paper presents two formal specifications of the chamfering operation, a complex operation used in geometric modeling. Each of them sticks to one mathematical definition: the B specification is close to the operational definition, while the CASL specification starts from the implicit definition which uses an ordering about n-G-maps. The features of both B and CASL allow to specify the initial mathematical definitions quite easily. The useful B features are basically the abstraction provided by the notion of abstract machines and relational operations, such as overriding and closure. The observational style of CASL allows to reach the most abstract definition - the implicit one. Moreover,

[2] *cell* is an operation which provide the dart set included in a particular cell. Thus, *G.cell*(d, i) provides the dart set included in the cell $< \alpha_0, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_n >$ (d) itself included in G.

[3] *unsewSet* is an operation which unsew a dart set for a particular dimension. $G.unsewSet(C, i)$ unsews all the darts of C for the involution α_i.

the CASL pretty-printing allows us to obtain a specification very close to the initial mathematical definition.

Because of the two languages different nature, we used them in two different abstraction levels. In both cases, we succeeded in providing a specification easily understandable by the experts of the application domain. From our point of view, the main reason is that we especially undertake to develop specifications preserving as much as possible the style advocated by the designers of the specification languages. For this, we were induced both to write several times our specification and to reformulate the informal requirements (here the mathematical definitions). Writing a formal specification for the chamfering operation brings us a significant feedback on the understanding, and this on the adequate definition of it. We have two equivalent mathematical definitions at our disposal, each of them being very close of either the B specification or the CASL specification. We have a high level of confidence of our specification to the requirements. The mathematical "foundations" of geometric modeling allow us to reason on the requirement side (using mathematical reasoning) in order to provide elegant specifications. So, with respect to the use of formal specifications, informal requirements expressed using mathematics specifically present the advantage of both being able to adjust requirement expression to the advocated specification style and providing specifications easily understandable by the experts of the domain. Nevertheless, in our case, the equivalence of the two specifications is informally established by the fact that the two mathematical definitions (the implicit one and the operational one) are mathematically proved. It is not fully satisfactory. A further work will consist in proving that the B specification or another CASL specification that follows the B approach can be seen as a refinement of the CASL specification. This step basically consists in a formal proof of the paper-and-pencil proof of the equivalence of both definitions reported in [LMA+00].

More generally speaking, B and CASL have some important and complementary syntactical features well adapted to the topologic-based geometry. In order to determine the most adapted language to the geometric modeling domain, and if one or the two languages B and CASL may play this role, we have to pursue this study by the one of the tools.

References

[Abr91] J. R. Abrial. *The B reference manual.* Edinburgh Portable Compilers, 17 Alva Street, Edinburgh EH2 4PH, UK, 1991.

[Abr96] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.

[BBFM99] Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. METEOR: A successful application of B in a large project. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *FM'99 – Formal Methods*, Lecture Notes in Computer Science, pages 369–387. Springer Verlag, September 1999.

[BD94] Y. Bertrand and J.-F. Dufourd. Algebraic Specification of a 3D-modeller Based on Hypermaps. *Computer vision, graphical model, and image processing*, 56(1):29–60, janvier 1994.

[BM00] Michel Bidoit and Peter D. Mosses. A gentle introduction to CASL. Tutorial, CoFI Workshop at the 3rd European Joint Conferences on Theory and Practice of Software (ETAPS'2000), Berlin, Germany, April 2000.

[Duf89] J.-F. Dufourd. Algebraic Map-Based Topological Kernel for Polyhedron Modelers. In *Conf. Eurographics*, pages 301–312, Hamburg, Germany, 1989. Elsevier.

[Duf97] J.-F. Dufourd. Algebras and formal specifications in geometric modeling. *The Visual Computer*, 13:131–154, 1997. Springer-Verlag.

[Elt94] H. Elter. *Etude de structures combinatoires pour la représentation de complexes cellulaires.* PhD thesis, Université de Strasbourg, 1994.

[FBBD96] L. Fuchs, D. Bechmann, Y. Bertrand, and J.-F. Dufourd. Formal specification for free-form curves and surfaces. In *Spring Conference on Computer Graphics*, Bratislava, juin 1996.

[Lie88] P. Lienhardt. Extension of the notion of map and subdivisions of a three-dimensional space. In M. Wirsing R. Cori, editor, *Proceedings of the 5th Annual Symposium on Theoretical Aspects of Computer Science (STACS '88)*, volume 294 of *LNCS*, pages 301–311, Bordeaux, France, February 1988. Springer.

[Lie94] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 1994.

[LMA+00] F. Ledoux, J.-M. Mota, A. Arnould, C. Dubois, P. Le Gall, and Y. Bertrand. Formal specification for a mathematics-based application domain: geometric modeling. Technical report, Département d'informatique, Université d'Evry Val d'Essonne, Evry, France, 2000.

[Mos99] P. D. Mosses. CASL: A guided tour of its design. *Lecture Notes in Computer Science*, 1589:216–240, 1999.

[Mos00] Till Mossakowski. CASL: From Semantics to Tools. In Susanne Graf and Michael Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 93–108. Springer Verlag, April 2000.

[oLD00] CoFI (Common Framework Initiative) Task Group on Language Design. Casl the common algebraic specification language summary, 2000. ftp://ftp.brics.dk/Projects/CoFI.

[PD00] Puitg and Dufourd. Formalizing mathematics in higher-order logic: A case study in geometric modelling. *TCS: Theoretical Computer Science*, 234, 2000.

[RMS00] M. Roggenbach, T. Mossakowski, and L. Schroder. Basic datatypes in CASL. CoFI Note L-12 - Version 0.4.1, http://www.brics.dk/Projects/CoFI/Notes/L-12/index.html, May 2000.