Available online at www.sciencedirect.com
ScienceDirect
BioSystems xxx (2006) xxx–xxx
www.elsevier.com/locate/biosystems

Declarative modeling of a neurulation-like process

Antoine Spicher*, Olivier Michel
IBISC, FRE 2873 CNRS, Université d'Évry, Génopole, France

Received 28 February 2005; received in revised form 8 July 2006; accepted 15 July 2006

Abstract

MGS is an experimental programming language dedicated to the modeling and the simulation of a special kind of discrete dynamical systems. *Dynamical systems with a dynamical structure* (or $(DS)^2$) arise when the state space is not fixed *a priori* but is jointly computed with the current state during the simulation. In this case the evolution function is often given through local rules that drive the interaction between some system components. MGS offers a new kind of data structure, *topological collections*, to describe the state of a dynamical system, and a new kind of control structure, *transformations*, to express local and discrete evolution laws. These two notions permit an easy specification of $(DS)^2$.

We propose in this paper a presentation of the MGS language and its main contributions. We show that various topological collections can be unified using concepts developed in combinatorial algebraic topology: *cellular complexes* and *topological chains*. Then we apply the notions brought by MGS to model and simulate the first step towards the simulation of the neurulation process in developmental biology where a sheet of cells evolves to a neural tube. It is a direct description of the modification of the topology of an arbitrary structure expressed in terms of local discrete evolution laws.
© 2006 Elsevier Ireland Ltd. All rights reserved.

Keywords: Computational models for developmental biology; Dynamical systems with a dynamical structure; Combinatorial algebraic topology; Topological collection; Transformation; Rewriting

1. Introduction

Developmental biology investigates highly organized complex systems. One of the main difficulties raised by the modeling of these systems is the handling of their dynamical spatial organization: the spatial organization of an organism that develops depends on its own evolution and its evolution depends on its spatial structure. We call such systems *dynamical systems with a dynamical structure*, or $(DS)^2$ (Giavitto et al., 2002). As an example, let us consider the embryogenesis phenomena: the process starts with a single cell. After several mitoses, the system is composed of several cells that exhibit some spatial organization due to cells growth, cells migration, cells division and cells death. These various processes, implying genetical, mechanical and chemical mechanisms, modify the neighborhood of each cell to develop a particular shape. At the same time, the spatial organization of the embryo influences the propagation of the mechanical stresses, the diffusion of the various chemicals, etc. That is to say, during the embryogenesis process, there is a feed-back between the spatial organization of the embryo and its own evolution which depends on this spatial organization.

From a computer engineering point of view, the phase space of such systems cannot be defined *a priori* and has to be jointly computed with the state of the system (the set of states must be an observable of the system itself, see the work of Giavitto (2003)). So, the modeling and the simulation of these systems have to be supported by dedicated computational models that allow the specification of both structural and functional properties of the system.

* Corresponding author.
E-mail addresses: aspicher@ibisc.univ-evry.fr (A. Spicher), michel@ibisc.univ-evry.fr (O. Michel).

0303-2647/$ – see front matter © 2006 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.biosystems.2006.09.024

Several computational models were inspired by biological processes and are, therefore, good candidates as computational formalisms of the development. As examples, we can cite: Lindenmayer systems (Rozenberg and Salomaa, 1992) (L-systems) inspired by plant growth, the CHAM formalism and Gamma (Banâtre et al., 2001) inspired by chemistry, and Păun systems (Păun, 2001) (P-systems) relying on a biological cell membrane metaphor. Because of their inspirations, these formalisms provide a possible framework to model and simulate the $(DS)^2$ encountered in developmental biology. One of the characteristics of these models is they suit well with a *discrete*, *local* and *declarative* specification of systems:

- Discrete: although the mainstream of the modeling approaches are developed in the framework of continuous models (ODE and PDE), the continuous formalism makes it difficult to express the discrete nature of the biological entities. Moreover the discrete approach leads to an algebraic point of view adapted to the field of formal languages theory used in the above computing models.
- Local: the description of $(DS)^2$ in biology is difficult because of their very complex organization. However, the description of the biological processes consists in the specification of local physico-mechanical interactions that can be specified by local laws. The global evolution is the result of an emergent behavior due to the local application of these laws.
- Declarative: this qualifier is used to contrast these formalisms with procedural approaches. In procedural approaches, the modeler specifies *how* the changes occur rather than *what* happens. A declarative style allows the computational model to be close to a mathematical formalism: it is expressive and brief, theoretically well defined, and close to the subject matter it is intended to describe. As a consequence, formal techniques can be used to validate or to check some properties of the biological model.

L-systems are paradigmatic of this approach. They are grammars that declaratively handle sequences. They are especially used in developmental biology, for the simulation of the growth of tree-like structures: see for example the simulation of *Anabaena* growth in Lindenmayer and Jürgensen (1992) and the modeling of plants described in Prusinkiewicz et al. (1990).

The previous computational models (L-systems, P-systems, abstract chemistry) can all be considered as *rewriting systems* on different spatial structures (Giavitto and Michel, 2002a): inclusion diagram for P-systems, tree-like structure for L-systems and an ether (complete graph) for chemistry. However, these spatial structures correspond only to some specific kinds of graphs. In other word, they are one-dimensional and more complex and structured spatial organizations required in developmental biology cannot be handled. For instance, a tissue is a two-dimensional spatial organization and an organ is a three-dimensional one.

Starting from this observation, the MGS project has been developed to study how these models could be unified and generalized to spatial structures of higher dimensions. The contributions of MGS are two-fold. We first propose a unification of the notion of data structure, using the concept of *topological collection*: a collection of elements organized following topological relationships. In MGS, the spatial organization is represented by a *cellular complex* (a notion developed in combinatorial algebraic topology) and the various spatial processes are represented as *topological chains* (a notion very similar to a *field* in physics). To handle topological collections in a declarative manner, we have introduced the notion of *transformations*: functions defined by case using rewriting rules generalized on cellular complexes.

This paper is organized as follows: Section 2 presents the MGS project and its contributions in more details. Section 3 shows through an example how topological collections and transformations can be used to implement a model of a sheet of epithelial cells that locally change their own shape to curve the sheet until the two sides are close enough to glue each other. This example is a premise to the simulation of the neurulation process where the neural plate changes its shape and its topology to form the neural tube. The main contribution of this paper is how MGS notions can be used to model in a declarative way the modifying topology of a developing structure during the course of time.

2. A quick description of the MGS formalism

The MGS project aims at developing a framework dedicated to the modeling and simulation of $(DS)^2$. It is inspired by the computational models described above. MGS is a classical declarative language that has been extended with two structures described below.

2.1. Topological collections

Topological collections are a unified view of the notion of data structure (Giavitto and Michel, 2002a). Here, the data structure is defined as an aggregate of relative elements and the "structure of the space" is used to specify the organization of the data structure. In the MGS project, we advocate that notions developed in the *combinatorial algebraic topology* theory provide a well-suited framework for the description of the structure of the space. This theory has already been used in modeling physical laws in a discrete way. The interested reader can look at the works of Tonti (1974) and Palmer and Shapiro (1993) for an elaboration. Here is a brief but not exhaustive description of the notions required to understand the rest of the paper.

We will call the structure of the space a *cellular complex* (Munkres, 1984). A cellular complex is composed of elements of various dimensions (vertices, edges, faces, etc.) called *topological cells* of dimension n or n-cells. These basic elements are organized following the *incidence relationship* that relies on the notion of boundary: let c_1 and c_2 be respectively an n_1-cell and an n_2-cell with $n_1 < n_2$, c_1 is incident to c_2 if c_1 belongs to the border of c_2. More precisely, if $n_1 = n_2 - 1$, c_1 is called a *face* of c_2, and c_2 is a *coface* of c_1. We can also define the notion of p-neighborhood of two n-cells: two n-cells, c_1 and c_2, are p-neighbors if they are part of the same p-cell (when $p > n$), or if they share a same p-cell in their border (when $p < n$).

A cellular complex C is a set of cells closed for the incidence relationships: if $c \in C$ then the faces of c belongs also to C. Now that we have defined the notion of cellular complex to specify the organization of the data structure, we need to associate data with the topological cells. For example we can associate a concentration with 3-cells, flux with 2-cells, etc. The association of data with cells corresponds to the concept of *topological chain* in algebraic topology (Henle, 1994). We would not describe this notion further in the paper because we will only rely on the fact that values are associated with each cell.

To summarize, *a topological collection is a cellular complex where values are associated with each cell.* An example of such a cellular complex is given in Fig. 1. Topological collection will be used to represent the states of a dynamical system.

2.2. Transformation

Transformations are functions defined by case used to handle topological collections. Each case is a *rewriting rule* $\alpha \to \beta$ where *pattern* α matches some sub-collection s of a collection S, where expression β computes a new sub-collection s' and substitutes s by s' in S. Transformations are well suited for describing the notion of local evolution law. Indeed, a local law can be considered as a rewriting rule that matches some interacting sub-systems, and that replaces these entities by a combination of the interaction. There are two kinds of transformations available in MGS.

<n,p>-transformations. The *<n,p>-transformations* use the p-neighborhood on cells of dimension n. The pattern α of each rule matches a sequence of n-cells. Two contiguous ın-cells in this sequence have to be p-neighbors. This sequence of neighboring cells is called a *$< n, p >$-path* or simply a *path*. The right-hand side (r.h.s.) of a rule also computes a sequence of values. Finally, the substitution is done element-wise: element i in the matched path is replaced by the ith element in the r.h.s. sequence. This point of view enables a very concise writing of the rules. Nevertheless, it does not allow a topological modification of the cellular complex but only an updating of the associated values.

Patch transformations. A *patch* is a transformation that allows modifications of the topological structure of a collection. The pattern and the r.h.s. specify an arbitrary sub-collection through a set of clauses. The example described in Section 3 shows how transformations can be defined and used in a concrete manner.

2.3. Unification of discrete computational models

The abstract approach of the data structure in MGS and the notions developed on transformations, enable an homogeneous and uniform handling of the computational models quoted above. For example, L-systems can be viewed as rewriting rules (defined by

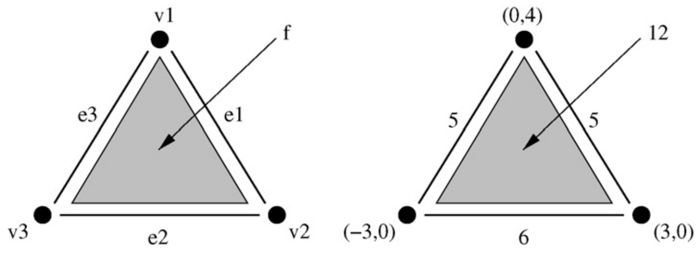

Fig. 1. On the left is an example of a cellular complex: it is composed of three 0-cells (v_1, v_2, v_3), three 1-cells (e_1, e_2, e_3), and a 2-cell f. The boundary of f is formed by its incident cells v_1, v_2, v_3, e_1, e_2 and e_3. Especially, the three edges are the faces of f, and therefore, f is the coface of e_1, e_2 and e_3. On the right, data are associated with the topological cells: positions are associated with vertices, lengths with edges and area with f.

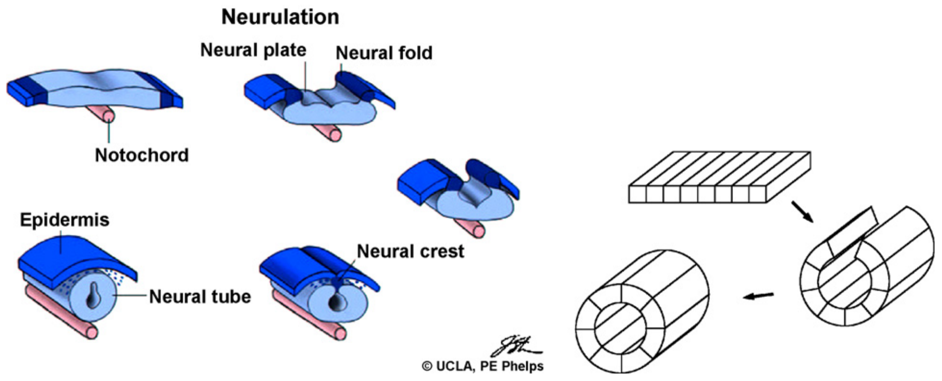

Fig. 2. On the left is a diagrammatic description of the neurulation process (reproduced, with permission from the author, from a drawing by Patricia Phelps at UCLA). Figure on the right shows three steps of our model. At the beginning, the system is a sheet of cells representing the neural plate. Then it is curved by cells deformation. At the final step, the sheet is closed and becomes a true continuous topological cylinder.

the grammar) applied on sequences. As a matter of fact, sequences are one dimensional: elements of a sequence are 0-cells, and two elements are contiguous in the sequence if their associated cells are 1-neighbors. The data stored in the sequence are values associated with each 0-cell in the corresponding topological collection. More generally, by varying the underlying topology, it is possible to emulate various computing models. Reference (Giavitto and Michel, 2002b) shows how Lindenmayer systems, Păun systems, abstract chemistry and cellular automata can be rephrased uniformly as transformations of topological collections.

3. Modeling the shape transformation of an epithelial sheet

In this section, we use the concepts of topological collections and transformations for multi-dimensional structures presented above, to specify a model of the shape transformation of an epithelial sheet. This model represents a first step towards the declarative modeling of neurulation.

3.1. Description of the model

The neurulation process consists in a topological modification of the back region of the embryo (see left of Fig. 2)[1]: the neural plate is folding and forms the neural fold. Then, this folding curves the neural plate until the two borders touch each other and make the plate becomes a neural tube. A last step consists in a separation between the neural tube and the epidermis located at the neural crest. The global deformation of the neural plate comes from the local change of the individual cell shapes, cell division and cell migration. The change of the shape geometry is followed by a topological modification that corresponds to the transformation of a plate into a tube. This modification is not trivial to model.

In our example, we have simplified the systems into a sheet of epithelial cells that is folded only by local cell deformation (see right of Fig. 2). After the migration, cells that were on the opposite sides at the beginning become very close to each other. Once they are closed enough, they collapse to make the original sheet of cells becomes a cylinder.

Our mechanical model is inspired from the works of Odell et al. (1981) on the modeling of epithelial cells. In their model, an epithelial cell is described in two dimensions by a masses and springs system. Fig. 3 sketches this representation. A cell is a square composed of four vertices and six edges or *fibers* (four for the sides and two others for the diagonals). The sides of the square correspond to the membrane of the cell, and the diagonals are used to model its inner fibers, and prevent an implosion. Moreover, the cell has a polarity: the fiber at the top is *apical*, the bottom one is *basal*. By contracting the springs of the basal and/or the apical sides, the cell changes its shape.

Each edge corresponds to a spring with a stiffness constant k and a rest length L_0 in parallel with a friction with a damping constant μ. Let L be the length of the spring, the mechanical forces are given by Newton's second law of motion in the equation:

$$\frac{\mathrm{d}^2 L}{\mathrm{d}t} = k(L_0 - L) - \mu \frac{\mathrm{d}L}{\mathrm{d}t} \quad (1)$$

Each vertex is linked to several springs and its acceleration vector can be computed by summing the forces

[1] Image taken from http://www.physci.ucla.edu/research/phelps/index.php and http://biology.kenyon.edu/courses/biol114/Chap14/Chapter_14.html.

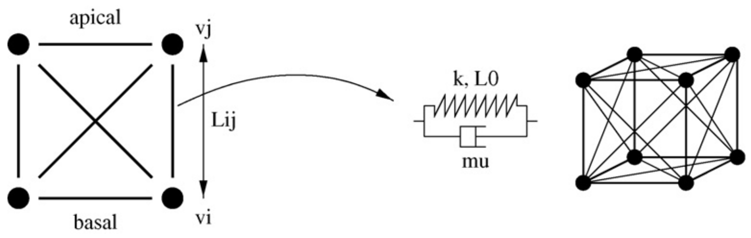

Fig. 3. On the left, Odell's model for the simulation of epithelial cells. On the right, extension of Odell's model to three dimensions.

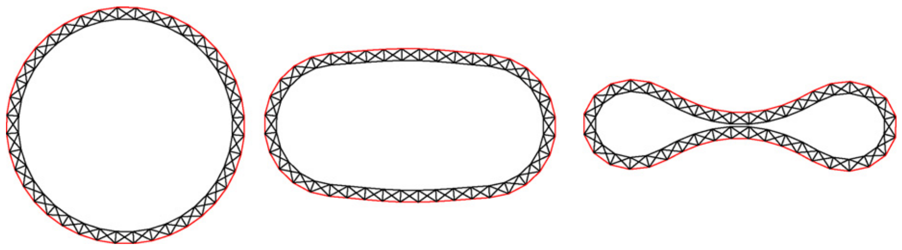

Fig. 4. Application of Odell's model on a ring of cells in MGS. From the left to the right, some screen-shots at different steps of the simulation.

applied by the springs. Let p_i be the position vector of vertex i, we have:

$$m\frac{\mathrm{d}^2 p_i}{\mathrm{d}t} = \sum_{j \in \mathrm{neighbors}(i)} \frac{\mathrm{d}^2 L_{ij}}{\mathrm{d}t} \cdot \frac{p_j - p_i}{\|p_j - p_i\|} \quad (2)$$

where L_{ij} is the length of the spring between i and j, and is given by Eq. (1). Fig. 4 presents the application of this model on a ring of cells in MGS.

In Nagpal (2001), this model has been extended in three dimensions. We will use this extension in our model. Instead of representing a cell by a square, we use cubes as shown on the right of Fig. 3: a cell is composed of 8 vertices, 24 edges, 6 faces (the faces of the cube) and 1 volume. Note the inner fibers are not represented, and instead of corresponding to inner fibers in the 2D model, the diagonals represent fibers in the membrane in three dimensions.

3.2. Implementation in MGS

3.2.1. Representation of the system

First of all, we start by defining the type of the values associated with each cell.

```
record Vertex = { px:float, py:float, pz:float,
                  vx:float, vy:float, vz:float,
                  ax:float, ay:float, az:float,
                  m:float } ;;

record Edge = { k:float, L0:float, mu:float, vL:float,
                vi:cell, vj:cell } ;;
```

We define two types of MGS records (a data structure equivalent to a `C struct`) for cells of dimension 0 and 1. The type `Vertex` contains 10 fields used for the Newtonian mechanics: `px`, `py` and `pz` represent the position vector, `vx`, `vy` and `vz` the velocity vector, `ax`, `ay` and `az` the acceleration vector, and `m` the mass. The type `Edge` pulls together the different constants associated with the spring model `k`, `mu` and `L0`. The velocity of the elongation is stored in the field `vL`. Moreover, two fields, `vi` and `vj` refer to boundaries of the edge; they are used to generate an arbitrary orientation of the edges. Note that `r.x` denotes the value associated with the field `x` of a record `r`.

Faces and volumes are not used in this example. However, it is easy to imagine that the volumes contain a kind of "genetic script" and the faces correspond to the place where cells exchange some chemical entities (ion channel, port, etc.). Therefore, these cells would be useful for the simulation of reaction–diffusion processes superimposed with the regulation of a genetic network.

In a second step, we have to initialize the chain representing the filament of cells. MGS can load cellular complexes from external files. So we consider that the structure of filament has been built with a CAD program and imported into MGS. Let `c` be a chain where the position {`px=..., py=..., pz=...`} is associated with each vertex, and where the other cells have no value. The initialization is done by four transformations for each dimension. As an example, we will describe here the initialization of the vertices:

```
trans <0> init_0 =
  { v => v + { m=1.0, vx=0.0, vy=0.0, vz=0.0, ax=0.0,
  ay=0.0, az=0.0 } };;
```

The keyword `trans <0>` means the transformation is dedicated to the elements of dimension 0. For each matched element, the variable `v` contains the position of the vertex. The initialization consists in adding the velocity and the acceleration (null at the beginning) to the record `v`. The addition between two records `r+r'` computes the asymmetric fusion: the result is a record that contains all the fields of `r` and `r'` with a priority for `r'` when a collision occurs. Asymmetric fusion enables to update some fields in a record without knowing all the remaining fields, a feature largely used in MGS programs.

3.2.2. *Mechanical model*

The mechanical model is described by two equations. So we will write two transformations to compute the force applied on each edge, and then the displacement of each vertex.

```
trans <1> update_spring = {
  e:Edge =>
    let vi = self.(e.vi) and vj = self.(e.vj) in
    let L  = dist(vi,vj) in
    let f  = (e.k*(e.L0-L) + e.mu*e.vL) in
    let eij_x = (vj.px - vi.px) / L and eij_y = ... and eij_z = ... in
      e + { vL = vL, fx = f*eij_x, fy = f*eij_y, fz = f*eij_z }
};;
```

This transformation (dedicated to the edges of type `Edge`) is the straightforward translation of Eq. (1). The amplitude of the elastic force is associated with the variable `f`. The orientation of the edges is taken into account: the vector $\vec{e}_{ij} = \frac{p_j - p_i}{\|p_j - p_i\|}$ is computed and the force is distributed along the three directions. The variable `self` denotes the collection that `update_spring` is applied on and `self.(x)` returns the value associated with cell `x` of the collection `self`. The function `dist` computes the distance between its two arguments of type `Vertex`.

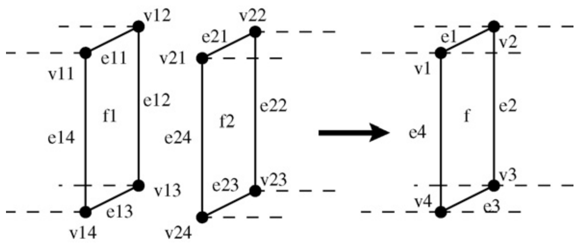

Fig. 5. Scheme of a rewriting rule for the topological transformation from the sheet of cells into a cylinder. To simplify the figure, diagonals are not drawn. The dotted edges represent the rest of the cellular complex.

```
  trans <0> integration [delta_t = 0.01] = {
    vi:Vertex =>
  let forces = cofacesfold (
     (fun e acc ->
        let ve = self.(e) in
        if (vi == ve.vi) then
          { fx = acc.fx+ve.fx, fy = ..., fz = ...}
        else
          { fx = acc.fx-ve.fx, fy = ..., fz = ...}
        fi),
     { fx=0.0, fy=0.0, fz=0.0 },
     vi
  ) in
     vi + { ax = forces.fx / vi.m, ay = ..., az = ...,
            vx = vi.vx + delta_t * vi.ax, vy = ..., vz = ...,
            px = vi.px + delta_t * vi.vx, py = ..., pz = ...}
} ;;
```

The transformation `integration` computes the new positions of vertices by using Eq. (2) (to obtain the updates acceleration) and a classical Euler integration approximation given by

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \Delta t\vec{a}(t) \quad \text{and} \quad \vec{p}(t + \Delta t) = \vec{p}(t) + \Delta t\vec{v}(t) \tag{3}$$

Value Δt corresponds to a discretization of time, and is given as an optional argument, here `delta_t`, whose default value is 0.01. In the right-hand side of the rule, the function `cofacesfold` corresponds to a basic fold on the sequence of the cofaces of `vi`. These cofaces are the edges whose `vi` is one of their boundaries. The function given in argument sums up the force applied by each edge on `vi`. Remark that a conditional test is used to take account of the orientation of the edge; as a consequence, the force is added or subtracted.

3.2.3. *Topological surgery*

The previous transformations allow the sheet to be folded. Nevertheless, no intersection between cells is checked. The following *patch transformation* merges two very close faces where one face is opposite to another. This transformation is *ad hoc* and the initial rest lengths have been appropriately chosen to make the curvature realistic.

Fig. 5 shows a diagrammatic view of the rewriting rule we want to specify in MGS. Cells are gathered two by two and are merged. Moreover, the rest of the cellular complex (represented by the dotted edges) has to be considered for the reconstruction. As an example, the unmatched cofaces of the vertices v_{11} and v_{21} must be cofaces of v_1. The patch transformation is the following:

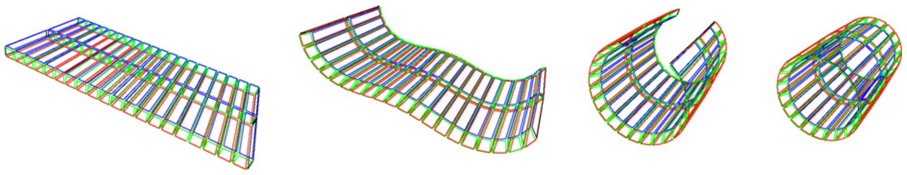

Fig. 6. Simulation of a neurulation-like process in MGS: from the left to the right, a sheet of epithelial cells is curving until the hems sew together to form a tube.

```
patch surgery = {
  f1:[dim=2, (e11,e12,e13,e14) in faces]
  v11 < e11 > v12 < e12 > v13 < e13 > v14 < e14 > v11

  f2:[dim=2, (e21,e22,e23,e24) in faces]
  v21 < e21 > v22 < e22 > v23 < e23 > v24 < e24 > v21

  [P(v11,v12,v3,v14,v21,v22,v23,v24)]
  =>
    ‘v1:[dim=0,faces=(),
         cofaces=(‘e1,‘e4)@unmatched_cofaces(v11,v12),
         average_0(v11,v21)]
    ...
    ‘e1:[dim=1,faces=(‘v1,‘v2),
         cofaces=(‘f)@unmatched_cofaces(f1,f2),
         average_1(e11,e21)]
    ...
    ‘f:[dim=2,faces=(‘e1,‘e2,‘e3,‘e4),
        cofaces=cofaces(f1,f2),
        average_2(f1,f2)]
} ;;
```

The pattern of a patch rewriting rule is composed of clauses. As an example, clause `f:[dim=2, (e1,e2,e3) in faces]` specifies a cell `f` of dimension 2, that has at least three faces called `e1`, `e2` and `e3`. The `:[...]` part of the clause is optional. Moreover, we can use the operators < and > between two clauses: `a<b` *means that the cell matched by the clause* `a` *is a face of the cell matched by* `b`. To simplify the program, the diagonals are not specified in the `surgery` patch. A predicate `P` is also used to check some additional geometrical properties of the vertices (both squares have to be very close).

The right-hand side specifies, within a syntax close to the patch pattern one, the elements that are built to replace the matched one. As an example, we create a new cell `'v1` of dimension 0, that has two new elements (`'e1` and `'e4`), and the old cofaces of `v11` and `v12` that have not been matched (they are given by the function `unmatched_cofaces`) as cofaces. The value associated with `'v1` is computed by the function `average_0(v11,v21)`. The `@` operator denotes the sequence concatenation. Fig. 6 shows four steps of the animation generated by MGS on a sheet of 20×2 cells.

3.2.4. *A moving token in the ring of cells*

To show that the cylinder is really closed after the topological surgery step, we put a token in the volumes. We want it to move along the sheet from one side to another. After the cylinder formation, this token should move along the ring without being blocked.[2] The following transformation can also be viewed as a substance transport between biological cells.

To simulate the token movement we associate values with the volumes (3-cells). At the beginning, each cube holds the value `'cell`: this is a symbol, i.e. a constant built from a name (here “*cell*”). However, we associate the value `'token` with a cell, and `'back`, with one of its neighbors. This second symbol is used to force the token to go along one direction.

```
trans <3,2> token = {
  ‘back, ‘token, ‘cell => ‘cell, ‘back, ‘token ;
  ‘back, ‘token        => ‘token, ‘back
} ;;
```

This transformation is applied on the element of dimension 3, and we consider the two-neighborhood relationship between them. Therefore, the pattern `'back, 'token, 'cell` means that we want to find a path of three cubes whose values correspond to `'back`, `'token` and `'cell` in this order. We rewrite `'cell`, `'back`, `'token` that makes the token move forward. The second rule corresponds to the case when the token is at an extremity. It cannot go forward and goes in the opposite direction.

4. Conclusion

In this paper we have presented a new approach for the declarative modeling of $(DS)^2$. Such kind of dynamical systems are very common in developmental biology. Our approach relies on the representation of the state of the $(DS)^2$ as a topological collection (a data structure based on notions developed in algebraic topology). The evolution laws of the $(DS)^2$ are specified

[2] The movie of this simulation is available at http://www.ibisc.univ-evry.fr/~mgs/ImageGallery/EXEMPLES/Neurulation/neurulation.avi.

using transformations, which are rewriting rules acting on topological collections. This unifying point of view enables a clear and concise description of models of $(DS)^2$. We have shown that the notions brought by MGS allow to take into account systems that have a multi-dimensional structure by giving a straight description in MGS terms of a model of simulation of the neurulation process.

The MGS programming style corresponds to the rule-based programming paradigm. Rule-based programming is currently experiencing a renewed period of growth with the emergence of new concepts and systems that allow a better understanding and a better usability. However, the vast majority of rule-based languages (like expert-systems) are funded on a logical approach (computation is a logical deduction and a deduction step is specified by a rule) which is not adequate to describe various spatiotemporal processes. We hope that the previous section has demonstrated the ability of MGS to express easily and concisely the building of sophisticated spatial structures, like the ones needed to model developmental processes.

The perspectives opened by this work are numerous. The notions developed here must be further validated through the development of large scale examples. The MGS language is currently used to model several biological processes (see (Barbier de Reuille et al., 2006) for a sophisticated use of MGS in plant development).

In this paper, we have not discussed the problem raised by the rule application strategy in a transformation. By changing the rule application strategy, that is the way to select the sub-collections to be replaced, we can change the model of time used in the simulation. In this paper, we use a maximal parallel strategy, which corresponds to the rule application strategy used in L- and P-systems. This strategy suits well a macroscopic approach. However, the right handling of chemical reactions at a microscopic scale requires a sequential rewriting strategy. The mixing of several strategies is a domain of future work.

At last but not least, we have to mention that the current MGS prototype relies on naïve data structures and algorithms to implement cells of any dimension and their transformations. The efficient evaluation of patch patterns is a subject of future work, as well as the compilation of MGS programs.

Acknowledgments

The authors would like to thank J.-L. Giavitto at IBISC, the MOKA team at the University of Poitier and the members of the "Simulation and Epigenesis" group at Genopole for technical support, stimulating discussions and biological motivations. This research is supported in part by the CNRS, GDR ALP, IMPG, University of Évry and Genopole/Évry.

References

Banâtre, J.-P., Fradet, P., Le Métayer, D., 2001. Gamma and the chemical reaction model: 15 years after. Lecture Notes Comput. Sci. 2235, 17–44.

Barbier de Reuille, P., Bohn-Courseau, I., Ljung, K., Morin, H., Carraro, N., Godin, C., Traas, J., 2006. Computer simulations reveal properties of the cell–cell signaling network at the shoot apex in Arabidopsis. PNAS 103 (5), 1627–1632.

Giavitto, J.-L., 2003. Invited talk: topological collections, transformations and their application to the modeling and the simulation of dynamical systems. In: Rewriting Technics and Applications (RTA'03), vol. LNCS 2706 of LNCS, Valencia. Springer, pp. 208–233.

Giavitto, J.-L., Godin, C., Michel, O., Prusinkiewicz, P., 2002. Modelling and simulation of biological processes in the context of genomics, "Computational Models for Integrative and Developmental Biology". Hermes. Also republished as an high-level course in the proceedings of the Dieppe spring school on "Modelling and simumation of biological processes in the context of genomics", May 12–17, 2003, Dieppes, France.

Giavitto, J.-L., Michel, O., 2002a. Data structure as topological spaces. In: Proceedings of the third International Conference on Unconventional Models of Computation UMC02, vol. 2509, Himeji, Japan. Lecture Notes in Computer Science, pp. 137–150.

Giavitto, J.-L., Michel, O., 2002. The topological structures of membrane computing. Fundamenta Informaticae 49, 107–129.

Henle, M., 1994. A Combinatorial Introduction to Topology. Dover Publications, New York.

Lindenmayer, A., Jürgensen, H., 1992. Grammars of development: discrete-state models for growth, differentiation, and gene expression in modular organisms. In: Ronzenberg, G., Salomaa, A. (Eds.), Lindenmayer Systems, Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology. Springer Verlag, pp. 3–21.

Munkres, J., 1984. Elements of Algebraic Topology. Addison-Wesley.

Nagpal, R., 2001. Programmable self-assembly: constructing global shape using biologically-inspired local interactions and origami mathematics. PhD Thesis, Massachusetts Institute of Technology.

Odell, G.-M., Oster, G., Alberch, P., Burnside, B., 1981. The mechanical basis of morphogenesis. I. Epithelial folding and invagination. Dev. Biol. 85 (2), 446–462.

Palmer, R.S., Shapiro, V., 1993. Chain models of physical behavior for engineering analysis and design. Res. Eng. Des. 5, 161–184 (Springer International).

Păun, G., 2001. From cells to computers: computing with membranes (P systems). Biosystems 59 (3), 139–158.

Prusinkiewicz, P., Lindenmayer, A., Hanan, J.S., et al. 1990. The Algorithmic Beauty of Plants. Springer-Verlag.

Rozenberg, G., Salomaa, A., 1992. Lindenmayer Systems. Springer, Berlin.

Tonti, E., 1974. The algebraic-topological structure of physical theories. In: Glockner, P.G., Sing, M.C. (Eds.), Symmetry, Similarity and Group Theoretic Methods in Mechanics. Calgary, Canada, pp. 441–467.