Available online at www.sciencedirect.com

SCIENCE DIRECT®

BioSystems 70 (2003) 149–163

www.elsevier.com/locate/biosystems

Modeling the topological organization of cellular processes

Jean-Louis Giavitto*, Olivier Michel

LaMI u.m.r. 8042 du CNRS, Université d'Évry Val d'Essone—GENOPOLE, Tour Évry-2, 523 Place des Terasses de l'Agora, 91000 Évry, France

Abstract

The cell as a dynamical system presents the characteristics of having a dynamical structure. That is, the exact phase space of the system cannot be fixed before the evolution and integrative cell models must state the evolution of the structure jointly with the evolution of the cell state. This kind of dynamical systems is very challenging to model and simulate. New programming concepts must be developed to ease their modeling and simulation. In this context, the goal of the MGS project is to develop an experimental programming language dedicated to the simulation of this kind of systems. MGS proposes a unified view on several computational mechanisms (CHAM, Lindenmayer systems, Paun systems, cellular automata) enabling the specification of spatially localized computations on heterogeneous entities. The evolution of a dynamical structure is handled through the concept of transformation which relies on the topological organization of the system components. An example based on the modeling of spatially distributed biochemical networks is used to illustrate how these notions can be used to model the spatial and temporal organization of intracellular processes.

Keywords: Computational models for cell simulation; Dynamical systems with a dynamical structure; Spatial organization; Topological collection; Rewriting

1. Introduction

The computer simulation of a biological process implies the definition of a model sufficiently rigorous to lead to a program. Such models are then *formal* but depart from the more traditional mathematical models, e.g. by the high number of heterogeneous components implied in the system description, the complexity and the size of the behaviors specification, the impossibility to "compress" the evolution of the system in an analytic or closed formula, etc. Refer to Hartwell et al. (1999), Wolfram (2002), and Chaitin (2002) for an elaboration on these differences. For example, in the case of the human heart, some computer simulations imply 10^5 cells of about 10 different kinds, each modeled by nonlinear equations capturing the behavior of 50 different ion channels and organized in a realistic geometry (Paniflov and Holden, 1997). Nevertheless, a computer simulation makes possible the systematic exploration of the system's behavior and sometimes to make predictions. This kind of approach is part of the more general idea of *simulated experiments* (also called in silico experiments by biologists and *numerical experiments* by physicists). These experiments are required when in-vivo or in-vitro experiments are out of reach for economical, practical or ethical reasons. Note that the simulation of a computational model (i.e. its run on a computer) is only one of its possible uses: because it is formal, it is possible to reason about it and for example to infer some properties that can be checked against the natural phenomena (see, e.g. Chandy and Misra (1988)

* Corresponding author.
E-mail addresses: giavitto@lami.univ-evry.fr (J.-L. Giavitto), michel@lami.univ-evry.fr, mgs@lami.univ-evry.fr (O. Michel).

for examples of the properties that can be proved on a program).

Besides their simulation, computational models can have a pedagogical, normative, and constructive role in biology. For instance, these models can be used to share knowledge about a given system, as a reference between scientists or to illustrate a set of complex relationships involved in a biological process. Another example is their use as a blueprint in the design of a new biological entity: Biology has reached the point where in addition to the study of already existing natural entities, it has to design new biological artifacts (drug design, artificial metabolic pathways, genetically modified organisms, ...). At last but not least, one may note that a number of notions developed in computer science to investigate the notion of computations have been imported in biology: for instance the notion of programs, memory, information, control (cf. Stengers, 1988; Keller, 1995).

These examples acknowledge the emergence of a new approach in biology, known as *Computational Biology*, where biological entities are considered as information processing systems with the final goal of a better understanding of nature using computer science notions.[1] We make a distinction between this goal and the goals of *bioinformatics* aimed to the development of software tools to support and help the biologists in the analysis and comprehension of biological systems. A good example of the latter is the development of data bases supporting the genome project (Kanehisa, 2000).

The models developed in the framework of Computational Biology are largely centered around the notion of *dynamical systems* (DS) and Tyson et al. (2000) pinpoints the theme:

$$\begin{aligned}\text{gene expression} &\rightarrow \textit{system dynamics}\\ &\rightarrow \text{cell physiology}\end{aligned}$$

It is becoming more and more important as we try to integrate the exponential growth in knowledge of all the cells components in a true understanding of the cell. If this formalization from biology to dynamical system and back to biology is new in molecular biology, it has long been advocated in the domain of the development (Maynard-Smith, 1999; Kaufman, 1995).

In this paper, we advocate that a special class of DS plays a crucial role in the computational modeling of biological processes: the *dynamical systems with a dynamical structure* or $(DS)^2$ in short. The specification of such kind of systems can be very difficult to achieve and new programming concepts must be developed to ease their modeling and simulation. These observations have motivated the development of the `MGS` project.

1.1. Outline

The rest of this paper is organized as follows. In the next section, we present the notion of $(DS)^2$. In Section 3 we sketch the use of term rewriting as a possible paradigm for the computational modeling of $(DS)^2$ through an example borrowed from artificial chemistry. Term rewriting suffers from severe shortcomings for the specification of biological processes. To overcome these drawbacks, we extend the term rewriting framework to handle more general structures using the notions of *topological collection* and *transformation* presented in Section 4. The exposition is restricted to the notions necessary to understand the examples in Section 5. We give four examples of biological processes modeled using the `MGS` experimental programming language: the Eden's model of tumor growth, the action of restriction enzymes, a spatially distributed signaling network and a reaction-diffusion process in an expanding media modeling the growth of a bacteria. We conclude by a summary and a comparison with related approaches.

2. Dynamical systems with a dynamical structure

A dynamical system corresponds to a phenomenon described by a state that evolves in time. The system is characterized by "observables", called the *variables* of the system, which are linked by some relations. The value of the variables evolves in time. A variable can take a scalar value (like a real number) or be of a more complex type like the variation of a simpler value on a spatial domain (for instance, the local concentration of a molecule in each point of a lumen). The set of the

[1] The transfer of concepts and tools between biology and computer science is not a one-way process (Paton, 1994). Often, a computing model inspired initially by a biological phenomena, leads to a formalism used later in simulation of some (other) biological processes.

values of the variables that describe the system constitutes its *state*. The sequence of state changes is called the *trajectory* of the system. Intuitively, a dynamical system is a formal way to describe how a point (the state of the system) moves in the *phase space* (the space of all possible states of the system). It gives a rule, the *evolution function*, telling us where the point should go next from its current location. There exist several formalisms used to describe a DS: ordinary differential equations (ODE), partial differential equations (PDE), iterated equations (finite set of coupled difference equations), cellular automata, etc., following the discrete or continuous nature of the time, the space and the value used in the modeling.

Many biological systems are structured, which means that they can be decomposed into parts corresponding to some variables $o_i \in \mathcal{O}_i$ (for convenience we use o_i to denote a part of the whole system and its corresponding state). Then, *sometimes*, the complete state s of the system is simply the product of these variables: $s = (o_1, \dots, o_n) \in \mathcal{O} = \mathcal{O}_1 \times \cdots \times \mathcal{O}_n$. The evolution of the state of the whole system is then viewed as the result of the changes of the state of its parts. In this case, the evolution function h_i of an observable o_i depends only on a limited subset of the state variables of the whole system: $o_i(t + \delta t) = h_i(o_{i_1}, \dots, o_{i_{n_i}})$, where δt denotes an infinitesimal or a discrete increase in time following the continuous or discrete nature of the considered evolution and h_i denotes the evolution function of the ith component. In this case, we say that the DS exhibits a *static structure*:

(1) the state of the system is statically described as a fixed set of variables (this set does not change in time);
(2) the relationships between variables, specified as the functions h_i between o_i and the arguments o_{i_j}, are also fixed and do not change in time.

Moreover, we say that the o_{i_j} are the *logical neighbors* of o_i (because very often, two parts of a system interact when they are physical neighbors). This situation is simple and arises often in elementary physics. For example, a falling stone is statically described by a position and a velocity and this set of variables does not change (even if the value of the position and the value of the velocity change in the course of time).

As pointed out by Giavitto et al. (2002), many biological systems can be viewed as a dynamical system in which not only the values of state variables, but also the *set* of state variables *and/or* the evolution function, change over time. We call these systems *dynamical systems with a dynamic structure* following Giavitto and Michel (2001b), or $(\mathrm{DS})^2$ in short. An obvious example is given by the development of an embryo. Initially, the state of the system is described solely by the chemical state o_0 of the egg (no matter how complex can be this chemical state). After several divisions, the state of the embryo is given not only by the chemical state o_i of the cells, but also by their spatial arrangement.[2] The number of cells, their spatial organization and their interactions evolve constantly in the course of the development and is not handled by one fixed structure $\mathcal{O}$. On the contrary, the phase space $\mathcal{O}(t)$ used to characterize the structure of the state of the system at time t must be computed jointly with the running state of the system. In this kind of situation, the dynamic of the whole system is often specified as several local competing transformations occurring in an organized set of simpler entities. The organization of this set is subject to possible drastic changes in the course of time and is a plain part of the state of the DS.

3. Multiset rewriting and the modeling of biological DS

In view of this last description of a $(\mathrm{DS})^2$, it is tempting to define the evolution of the system as a set of rules specifying the interactions of a part o_i with another part o_j of the system. This schema is reminiscent of the description of a chemical reaction.

Consider, for example, a simple chemical system of two molecule types A and B. The reactions between these two molecule types are given by three rules:[3]

$$A + A \to A + A + B, \qquad A + B \to A + B + B,$$
$$B + B \to B + B + A$$

[2] The neighborhood of each cell is of paramount importance to evolution of the system because of the interplay between the shape of the system and the state of the cells. The shape of the system has an impact on the diffusion of the chemical signals and hence on the cells state. Reciprocally, the state of each cell determines the evolution of the shape of the whole system.

[3] These reaction rules correspond to deterministic second-order catalytic reactions: a collision of two molecules will catalyze the formation of a specific third molecule and the two colliding molecules are regarded as catalysts.

The "$+$" sign that appears in the left- and right-hand sides of the rules means that the linked molecules are present together in the chemical reactor. Thus, the left-hand side (LHS) $A + B$ of second rule can also be equivalently written $B + A$. From a mathematical point of view, it is very convenient to consider $+$ as a formal commutative–associative operator used to construct *multisets*: unlike a set, an element can occur several times in a multiset and a multiset with the six elements A, C, A, D, B, C is simply a formal sum $o = A + C + A + D + B + C$ (in this example, the elements A and C occur twice, and elements B and D occur only one time in the multiset o). The associativity and the commutativity properties of the $+$ operator are simply the expression that the elements of this sum can be rearranged in any order. To simulate the chemical reaction, we simply interpret each rule as a transformation of the multiset. For instance, the first rule specifies that two molecules A taken from the multiset have to be replaced by the three molecules: A, A and B. If this reaction occurs in o at a given time step t_0, then o is transformed in $A + C + A + D + B + C + B$ (one additional B is produced at step t_1). Because several reactions involving different elements occurring in the same time step are possible, the strategy is to apply in parallel as many transformations as possible to the multiset. Such transformations are iterated to model the evolution of the state of the reactor.

In this approach, the chemical reaction rules are interpreted as rules for rewriting the formal sum. Abstractly, we can say that a chemical reaction can be modeled as a *multiset rewriting system*. This computational model focuses on the chemical system at the level of single molecules and is sometimes called *individual-based modeling*: every molecule is explicitly stored and every single collision is explicitly performed. At this level of details, the chemical system is a $(DS)^2$ because the components of the systems are molecules and their number varies in time (there is one variable for each molecule, to record the presence of this molecule in the reactor). Obviously, another formalization is possible: at the coarser level of the chemical concentrations, the chemical system can be described as a DS with a static structure (with one variable for the concentration of each molecule type). This last approach is certainly computationally less expensive, but does not give access to the same level of details as the former.

This modeling paradigm, based on term rewriting, can be extended from this chemical example to other situations and its biological relevance is advocated in several recent papers (Fisher et al., 2000; Manca, 2001; Eker et al., 2002a,b). To paraphrase[4] Fisher et al. (2000): "A biological system is represented as a term of the form $o_1 + o_2 + \cdots + o_n$ where each term o_i represents either an entity of the system or a message addressed to other entities, i.e. signal, command, information, action, etc. The simulation of the physical evolution of the biosystem is achieved through term rewriting, where the LHS of a rule typically matches an entity and a message addressed to it, and where the right-hand side (RHS) specifies the entity's updated state, and possibly other messages addressed to other entities. The operator $+$ that joins entities and messages is associative and commutative, achieving an 'associative commutative soup', where entities swim around looking for messages addressed to them."

A severe shortcoming of this view is the *total lack of (spatial) organization*. For example, the cell cannot be thought as a chemical reactor where the chemicals are homogeneously diluted. On the contrary, the cell exhibit a highly organized spatial structure, with vesicles, cargos, membranes, nucleus, hyperstructures (Amar et al., 2003), etc. And the notion of organization (both spatial organization or more generally the functional organization) is also fundamental at the lower level of pathways and at the higher level of tissues, organs and individuals. The need to represent more structured organizations than multiset of entities and messages is stressed and motivates several extensions of rewriting; see for one example amongst others (Brown and Heyworth, 2000). However, a general drawback with these approaches is that they work with a fixed organization of entities, and it is not obvious at all how to extend this to systems where the organization and number of entities and their relationships are constantly changing.

This is precisely one of the main motivation of the MGS research project. One of our goal is to validate the contribution of a *topological* approach to the specification and simulation of the dynamical organization of biosystems. By superseding the rewriting of terms

[4] We have adapted the terminology.

by the more general notion of transformation of topological collections, we hope to go beyond the limitations of the preceding formalisms.

4. Topological collections and their transformations

The MGS project is aimed at the representation and manipulation of transformations of entities structured by *abstract topologies* (Giavitto and Michel, 2002). A set of entities organized by an abstract topology is called a *topological collection*. Topological means here that each collection type defines a neighborhood relation inducing a notion of *sub-collection*. A sub-collection B of a collection A is a subset of connected elements of A and inheriting its organization from A. The *global transformation* of a topological collection C consists in the parallel application of a set of *local transformations*, see Figs. 1 and 2. A local transformation is specified by a rewriting rule r that specifies the replacement of a sub-collection by another one. The application of a rewriting rule $\beta \Rightarrow f(\beta, \dots)$ to a collection A:

(1) selects a sub-collection B of A whose elements match the *pattern* β,
(2) computes a new collection C as a function f of B and its neighbors,
(3) and specifies the insertion of C in place of B into A.

This framework embeds the rewriting of multisets in the following way. In a multiset, an element is susceptible to interact with any other element, so the abstract topology of a multiset is the topology of a complete connected graph: the neighbors of an element are all the other elements in the multiset. Then, a pattern β can select an arbitrary sub-multiset and a multiset rewriting rule is simply a local transformation in this topology.

The MGS experimental programming language implements the idea of topological collections and their transformations into the framework of a simple dynamically typed functional language. Collections are just new kinds of values and transformations are functions acting on collections and defined by a specific syntax using rules. Functions and transformations are first-class values and can be passed as arguments or returned as the result of an application.

4.1. Collection types

There are several predefined collection types in MGS, and also several means to construct new collection types. The collection types can range in MGS from totally unstructured with sets and multisets to more structured with records, sequences and GBFs (cf. Giavitto and Michel, 2001a, 2002). Other topologies are currently under development and include Delaunay graphs and abstract simplicial complexes for the representation of arbitrary d-dimensional

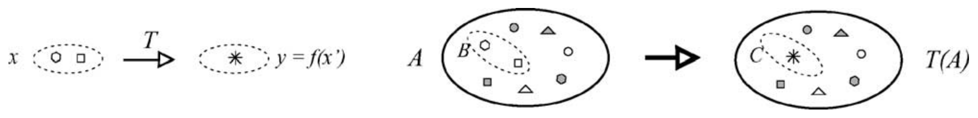

Fig. 1. A local transformation of a topological collection. Collection A is of some kind (set, sequence, array, cyclic grid, tree, term, etc.). A rule T specifies that a sub-collection B of A has to be substituted by a collection C computed from B. The RHS of the rule is computed from the sub-collection matched by the LHS x and its possible neighbors x' in the collection A.

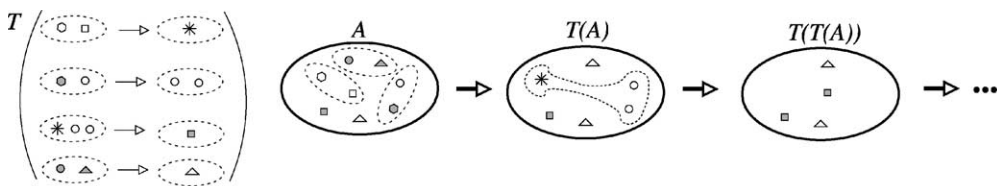

Fig. 2. Transformation and iteration of a transformation. A global transformation T is a set of local transformations applied in parallel and synchronously to make one evolution step. The local transformations do not interact together. A transformation is then iterated to build the successive states of the DS.

neighborhoods. This paper focuses on two families of collection types: *monoidal collection* and *GBF*.

For any collection type `T`, the corresponding empty collection is written `()`:`T`. The name of a collection type is also a predicate used to test if a value has this type: `T`(v) returns true only if v has type `T`. Each collection type can be subtyped. The type declaration `collection U` $=$ `T` introduces a new collection type `U` which is a subtype of `T`. The new type `U` shares the same topology as `T`. However, a value of type `U` can be distinguished from a value of type `T` using the predicate `U` (i.e. the subtyping relation implies that $U(u) \Rightarrow T(u)$, for any value u, but not the reverse). Elements in a collection can be of any type, including collections, thus achieving *complex objects* in the sense of Buneman et al. (1995).

4.2. Monoidal collections

Set, multiset (or bag) and sequences are members of the monoidal collection family. As a matter of fact, a sequence (respectively a multiset) (respectively a set) of values taken in V can be seen as an element of the free monoid V^* (respectively the commutative monoid) (respectively the idempotent and commutative monoid). The join operation in V^* is written by a comma "," and induces the neighborhood of each element: let E be a monoidal collection, then elements x and y in E are neighbors if $E = u, x, y, v$ for some u and v. This definition induces the following topologies. For sets (type `set`), each element in the set is neighbor of any other element (because the commutativity, the term describing a set can be reordered arbitrarily). For multiset (type `bag`), each element is also neighbor of any other (however, the elements are not required to be distinct as in a set). For sequence (type `seq`), the topology is the expected one: an element which is not at the end, has one neighbor on the right.

The comma operator is overloaded in `MGS` and can be used to build any monoidal collection (the type of the arguments disambiguates the collection built). So, the expression `1, 1+1, 2+1, ():set` builds the set with the three elements 1, 2 and 3, while the expression `1, 1+1, 2+1, ():seq` makes a sequence s with the same three elements. The comma operator is overloaded such that if x and y are not monoidal collections, then x, y builds a sequence of two elements. So, the expression `1, 1+1, 2+1` evaluates to the sequence s too.

4.3. GBFs

The acronym GBF stands for "group-based data fields". A GBF is an extension of the notion of array, where the elements are indexed by the elements of a group, called the *shape* of the GBF (see Giavitto and Michel, 2001a). A GBF value associates values to some indices of a shape. This kind of collection can be used to describe uniform and regular topologies like: n-ary trees, n-dimensional grids, circular and screwed grids, archimedian tiling of the plane, etc. For example, the following type declaration:

$$\texttt{gbf}\; Grid2 = <\texttt{north}, \texttt{east}>$$

introduces a new two-dimensional shape called *Grid*2, corresponding to the Von Neuman neighborhood in a classical 2D mesh (a cell above, below, left or right—not diagonal). The two names `north` and `east` refer to the directions that can be followed to reach the neighbors of an element. These directions are the *generators* of the underlying group structure. The list of the generators can be completed by giving equations that constraint the displacement in the shape. For instance:

$$\texttt{gbf}\; Hexagon = <\texttt{east}, \texttt{north}, \texttt{northeast};$$
$$\texttt{east} + \texttt{north} = \texttt{northeast}>$$

defines an hexagonal lattice that tiles the plane, see Fig. 3. Each cell has six neighbors (following the three generators and their inverses). The equation `east` $+$ `north` $=$ `northeast` specifies that a move following `northeast` is the same as a move to `east` followed by a move to `north`.

Formally, a GBF value is a partial function from the shape (a group specified by a finite presentation) to a set of values. Even if the underlying shape is infinite, the domain of a GBF value is finite. The topology of the GBF is the topology of the underlying Cayley graph (Magnus et al., 1976).

4.4. Sub-collection patterns

A pattern β that appears in the LHS of a rule is an expression used to select a sub-collection to be

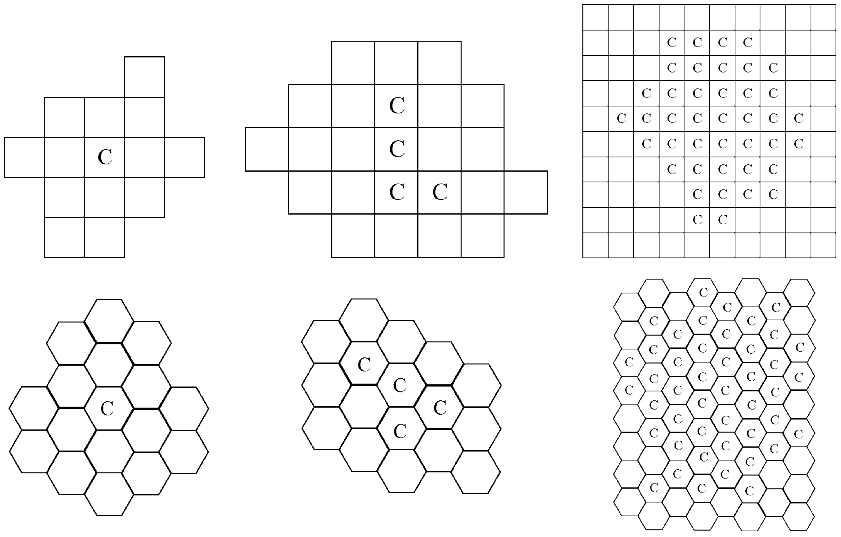

Fig. 3. Eden's model on a grid and on an hexagonal mesh (initial state, and states after the 3 and the 7 time steps). Exactly the same `MGS` transformation is used for both cases. These shapes correspond to a Cayley graph of `Grid2` and `Hexagon` with the following conventions: a vertex is represented as a face and two neighbors in the Cayley graph share an edge in this representation. An empty cell has an undefined value. Only a part of the infinite domain is figured.

replaced. Several operators are available; we will review here only few constructs.

- *literal*: a literal value matches an element with the same value. For example, 123 matches an element in a GBF with value 123.
- *variable*: a pattern variable a matches exactly one element with a well defined value. The variable a can then occur elsewhere in the rest of the rule and denotes the value of the matched element. The identifier of a pattern variable can be used only once in a pattern.
- *record pattern*: the brackets $\{\ldots\}$ are used to match one element whose value is a record (`MGS` record are similar to Pascal's record or `C`'s structure). The content of the brackets can be used to match records with or without a specific field (eventually constrained to a given field type or field value). For instance, $\{a, b : \texttt{string}, c = \texttt{3}, \tilde{}\ d\}$ is a pattern that matches a record with fields a, b and c but no field d. In addition, the type of field b must be `string` and the value of the field c must be the integer 3.
- *empty element*: the symbol `<undef>` matches an element with an undefined value, that is, an element whose position does not belong to the support of the GBF. The use of this basic filter is subject to some restriction: it can occur only as the neighbor of a defined element.
- *neighbor*: the pattern b, p matches a sub-collection composed of an element matched by b neighbor of a sub-collection matched by p.
- *guard*: p/exp matches a sub-collection matched by p if boolean expression exp evaluates to true. For instance, $x, y/y > x$ matches two neighbor elements x and y such that y is greater than x.
- *repetition*: $p+$ matches a sub-collection made of a non-empty repetition of sub-collections matched by p. If p is a pattern variable, then its value refers the sequence of matched elements and not to one of the individual values. For example, `3`$+$ matches a non-empty sub-collection made only of `3`'s.

5. Examples

The purpose of this section is to show the capacity of `MGS` to specify in a concise way several well-known examples corresponding to several biological situations and various computational models. Section 5.1 relies on cellular automata to model a growth process. The two next examples (Sections 5.2 and 5.3) use the

P systems approach to model biochemical reactions. Section 5.3 introduces nested multisets to handle the spatial organization of the compartments within the cell. The last example in Section 5.4 was initially proposed to model the growth of a bacteria, *Anabaena catenula*, based on a reaction–diffusion taking place in an expanding media and using the formalism of L systems. We hope that these examples taken in several fields, will convince the reader of the effectivity of the MGS approach for biological modeling (see also Section 6.2).

5.1. The Eden model

We start with a simple model of growth sometimes called the Eden model (specifically, a type B Eden model; Eden, 1958). The model has been used since the 1960s as a model for such things as tumor growth and growth of cities. In this model, a 2D space is partitioned in empty or occupied cells. We start with only one occupied cell. At each step, occupied cells with an empty neighbor are selected, and the corresponding empty cell is made occupied.

The Eden's aggregation process is simply described as the following transformation *Eden* with only one rule R:

```
trans Eden = {R = x, < undef > ⇒ x, x; }
```

We assume that some arbitrary value is used to represent an occupied cell, other cells are simply left undefined (i.e. without associated value). Then the rule R can be read: an occupied element *x* and an undefined neighbor are transformed into two occupied elements. The transformation Eden defines a function that can then be applied to compute the evolution of some initial state. One of the advantages of the MGS approach, is that this transformation can apply indifferently on grid or hexagonal lattices, or *any* other collection type (see Fig. 3).

5.2. Restriction enzymes

This example shows the ability to nest different topologies to achieve the modeling of a biological structure. We want to represent the action of a set of restriction enzymes on the DNA. The DNA structure is simplified as a sequence of letters A, C, T and G. The DNA strings are collected in a multiset. Thus we have to manipulate a multiset of sequences. The following declarations:

```
collection DNA = seq; ;
collection TUBE = bag; ;
```

introduce a subtype called DNA of seq and a subtype of multisets called TUBE.

A restriction enzyme is represented as a rule that splits the DNA strings; for instance a rule like:

```
EcoRI = X+, ("G","A","A","T","T","C"),
 Y+   ⇒ (X,"G" ) :: ("A","A",
          "T","T","C",Y) :: () : TUBE;
```

stands for the *Eco*RI restriction enzyme with recognition sequence G^AATTC (the point of cleavage is marked with ^). The X+ pattern filters the part of the DNA string before the recognition sequence. Identically, Y names the part of the string after the recognition sequence. The RHS of the rule constructs a TUBE containing the two resulting DNA subsequences (the :: operator indicates the "consing" of an element at the head of a collection).

We need an additional rule Void for specifying that a DNA string without a recognition sequence must be inserted wrapped in a TUBE. The two rules are collected into one transformation:

```
trans Restriction = {
    EcoRI = ...;
    Void = X+ ⇒ X :: () : TUBE;
}
```

The rule specification order in a transformation is taken into account, and so, the rule Void is used only if rule EcoRI cannot be applied. In this way, the result of applying the transformation *Restriction* on a DNA string is systematically a sequence with only one element which is a TUBE.

The transformation *Restriction* can then be applied to the DNA strings floating in a TUBE using the simple transformation:

```
trans React = {dna ⇒ hd(Restriction(dna))}
```

The operator hd gives the head of the result of the transformation *Restriction*, i.e. a TUBE containing one or two DNA strings. These elements are then merged with the content of the enclosing TUBE. The transformation can be iterated until a fixpoint is reached:

```
React[fixpoint]((
    ("C","C","C","G","A",
      "A","T","T","C","A",
      "A",() : DNA),
    ("T","T","G","A","A",
    "T","T","C","G","G",
      "G",() : DNA),
      () : TUBE));;
```

returns a tube with four DNA strings:

```
("T","T","G" , () : DNA),
("C","C","C","G" , () : DNA),
("A","A","T","T","C" ,
  "A","A" , () : DNA),
("A","A","T","T","C" ,
  "G","G","G" , () : DNA),
() : TUBE
```

5.3. A localized signaling network

We want to sketch the specification in MGS of a spatially distributed biochemical network model proposed by Bugrim (2000). The example focuses on a small signaling network that consists of cAMP and calcium signaling. See Fig. 4 for a more complete description.

The corresponding topological structure mimics the spatial organization of the cell using nested multisets, see Fig. 5. The MGS declarations:

```
collection Volume = bag;
collection Membrane = bag;
collection Environment = Volume;
collection Plasma = Membrane;
collection Cytosol = Volume;
collection EndoRetic = Membrane;
```

are used to introduce some new kinds of multisets (the bag keyword). These kinds are used here mainly to describe the hierarchy of localization and compartments and are used to discriminate between multisets.

The main part of the corresponding MGS program consists in defining the ontology of this application domain: there exist several molecules, each has a name; some exists in two states: active or inactive; some are characterized as receptors; etc. Such ontology is described in MGS using *subtyping*. These subtypes are then used in pattern-matching to select entities with or without some properties. For example, a molecule

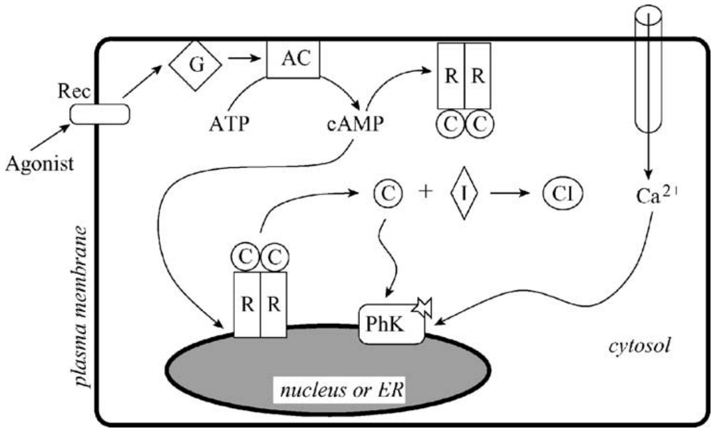

Fig. 4. cAMP and calcium signaling pathways. This schema is reprinted from Bugrim (2000) and the description of the involved pathways is largely inspired by this reference. The different components of the two pathways are localized at various places within the cell. The first steps of the cAMP pathway occur at the plasma membrane, starting with the activation of adrenergic receptors. Then, the cAMP molecules bind to a regulatory sub-unit of the protein kinase A, with the effect of dissociating a catalytic sub-unit C. The localization of PKA depends on a family of anchoring proteins AKAPs that target this kinase to different compartments. In this example, two localizations are considered: the plasma membrane and an internal compartment (e.g. nucleus or endoplasmic reticulum). The calcium pathway starts by the activation of a channel in the plasma membrane. The fraction of PhK associated to the internal compartment is the target of both pathways. A possible inhibitor I of PKA is also considered.

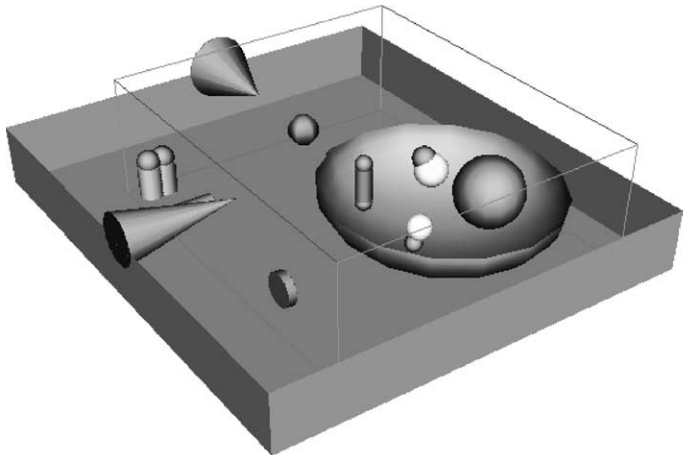

Fig. 5. The spatial organization of the pathway specified as a nest of multisets. The reaction, diffusion and transport processes described in Fig. 4 are modeled as multiset transformations taking place in a nest of multisets. This is reminiscent of the P system paradigm (Paun et al., 2001). This figure is automatically generated by the MGS simulation program. Each box corresponds to a multiset: the external one represents the universe and contains three elements: the agonist molecule pictured as a thin cone, the calcium (the thick cone) and the plasma membrane which is represented as a multiset and figured by a translucid box. The various molecules anchored in the plasma membrane are elements of the corresponding multiset and are figured as various solid volumes. The ellipsoidal container represents the cytosol and the solid sphere in the middle of it, the nucleus. Such figure can be generated at each simulation step to visualize the trajectory of the DS.

is described as a record having or not some fields. Record type may specify the presence or the absence of a field, or the value of a specific field (like in record pattern). For instance:

$\texttt{record}\, \mathit{Molecule} = \{\texttt{name}\};$
$\texttt{record}\, \mathit{Activity} = \{\texttt{activation}\};$
$\texttt{record}\, \mathit{Activated} = \{\texttt{activation} = \texttt{true}\};$
$\texttt{record}\, \mathit{Inactivated} = \{\texttt{activation} = \texttt{false}\};$
$\texttt{record}\, \mathit{ATP} = \texttt{Molecule} + \{\texttt{name} = \texttt{"atp"}\};$

define five record types. The record type declaration is introduced by the keyword `record`. *Molecule* is the type of any record having at least a field named `name`. *Activated* is the type of a record having at least a field named `activation` and with value `true`. This type is a subtype of *Activity* which only requires the presence of the field `activation`. The type ATP corresponds to a molecule named `"atp"`.

Three kinds of transformations are used to define the processes of the Bugrim's model. The first class corresponds to some ancillary transformations. For example

$\texttt{trans}\, \mathit{ActivateReceptor}$

$= \{r : \mathit{Receptor} \to \texttt{r} + \{\texttt{activation} = \texttt{true}\}\}$

is a rule that updates to `true` the field `activation` of an entity *r* of type *Receptor*. This kind of transformations is triggered by a rule of the sole transformation of the second class. This transformation summarizes all the rules corresponding to the description of the biochemistry (there are about 10 reactions in this pathway):

$\texttt{trans}\, \mathit{Biochemistry} = \{$
$\quad \texttt{R1} = a : \mathit{ActiveAgonist},\ p : \mathit{Plasma}$
$\qquad \Rightarrow a + \{\texttt{activation} = \texttt{false}\},$
$\qquad\quad \mathit{ActivateReceptor}(\texttt{p});$
$\quad \ldots$
$\}$

For example, rule `R1` specifies that an active agonist and a plasma membrane interact to inactivate the agonist and to transform the plasma with transformation *ActivateReceptor* (this transformation turns on all the activation fields of the receptors anchored in the plasma membrane).

There is also only one transformation in the last class of transformations. It is used to thread the biochemistry rules amongst the nested multisets:

$\texttt{fun}\, \mathit{Run}(x) = \mathit{Thread}(\mathit{Biochemistry}\,(x));$
$\texttt{trans}\, \mathit{Thread} = \{$
$\quad p : \mathit{Membrane} \Rightarrow \mathit{Run}(p);$
$\quad c : \mathit{Volume} \Rightarrow \mathit{Run}(c);$
$\}$

The transformation *Thread* applies the function *Run* to each entity of type *Membrane* or *Volume* found in the collection argument. The function *Run* consists in running the biochemistry transformation and then iterating the threading.

The complete MGS program is approximatively 150 lines long, including the building of the initial system state. It describes 40 states of molecules and uses 5 auxiliary transformations to define 10 chemical interactions. Several transformations are also used to produce the description of the DS state (the description is generated in a 3D scene description language which is then visualized by an ad-hoc front-end). The complete code can be found from the MGS web page.

5.4. A model of growth for Anabaena catenula

The cyanobacterium *Anabaena* grows in filaments of 100 cells or more. When starved for nitrogen, specialized cells called heterocysts differentiate from the photosynthetic vegetative cells at regular intervals along each filament. Heterocysts are anaerobic factories for nitrogen fixation; in them, the nitrogenase enzyme complex is synthesized and the components of the oxygen-evolving photosystem II are turned off. Plant signals exert both positive and negative regulatory control on heterocyst differentiation. Wilcox et al. (1973) have proposed an activator–inhibitor model of heterocyst differentiation where the (high) concentration of the activator triggers the heterocysts differentiation. The production of the activator is an autocatalytic reaction and also catalyzes the production of the inhibitor. The inhibitor is an antagonist substance that repress the activity of the activator when its concentration is high enough. The diffusion of the inhibitor to the neighboring cells prevents neighbors to become also heterocysts and explains why heterocysts appear in a regular spaced pattern in the filament.

```
record C = { a, h, x, p, type = "C"};;
record D = { a, h, x, p, type = "D"};;
trans T = {
  p1 = e / (C(e) & (e.x >= lm) & (e.p == Right))
  ⇒ { type ="C", a = e.a, h = e.h, x = e.x * longer, p = Left},
     { type ="C", a = e.a, h = e.h, x = e.x * shorter, p = Right};
  p2 = e / (C(e) & (e.x >= lm) & (e.p == Left))
  ⇒ { type ="C", a = e.a, h = e.h, x = e.x * shorter, p = Left},
     { type ="C", a = e.a, h = e.h, x = e.x * longer, p = Right};
  p3 = e / (C(e) & C(left e) & C(right e))
  ⇒ let al = (left e).a and ar = (right e).a
     and hl = (left e).h and hr = (right e).h
     and h = e.h and a = e.a and x = e.x and p = e.p
     in { type = "D", x = x, p = p
          a = a + ... increase in a ...,
          h = h + ... increase in h ... };
  p4 = e / (D(e) & (e.a >= thr) & (e.x < shorter*gr))
  ⇒ { type ="C", a = e.a, h = e.h, x = e.x, p = e.p};
  p5 = e / (D(e) & (e.a < thr) | (e.x >= shorter*gr))
  ⇒ { type ="C", a = e.a/gr, h = e.h/gr, x = e.x*gr, p = e.p};
  p6 = e / C(e)
  ⇒ { type = "D", a = e.a, h = e.h, x = e.x, p = e.p};
};;
```

Fig. 6. The MGS program corresponding to the heterocyst differentiation in *Anabaena*. See Section 5.4 for further explanations.

A computer simulation of this process (Hammel and Prusinkiewicz, 1996) was originally developed in the field of L system and shows the use of *parametric L systems* (Prusinkiewicz and Hanan, 1990; Hanan, 1992) for the modeling of a fundamental mechanism: a morphogenesis driven by a reaction–diffusion process taking place in a growing media. The corresponding parametric L systems is easily translated into a MGS program where each rule corresponds to a production of the L system given in Giavitto and Michel (2002). There is nothing new in this translation and the example is given mainly to show the ability of MGS to express sophisticated L systems. The program is listed in Fig. 6. The output of the program is plotted in Fig. 7.

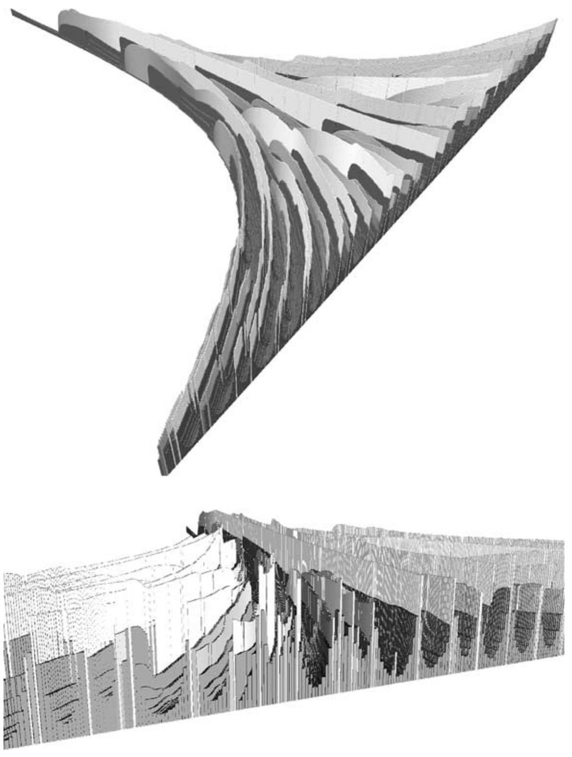
Fig. 7. Heterocysts differentiation in *Anabaena* filament. In the upper graphic, the time goes from upper-left to lower-right corner. Each slice (lower graphic) corresponds to the state of a growing filament and represent a sequence of cells. The height of a cell represent the activator concentration. Cells are pictured in red when the activator is greater than a given level triggering differentiation. Gray cells are vegetative ones. This type of visualization, called “space-time extrusion” has been developped in Hammel and Prusinkiewicz (1996).

In the previous code, the state of a cell is implemented as a record with field `a` for the concentration of the activator, `h` is the concentration of the inhibitor, `p` is the cell polarity, `x` is the length of the cells and `type` indicates if the cell is an heterocystis (`C`) or a vegetative (`D`) cell. The guard in rule `p1` selects right-polarized cells with a length greater than some level `lm`. Note in rule `p3` the way the neighboring elements are accessed using the `left` and `right` displacement operators. Rule `p1` and `p2` specify a cell division (two cells are substituted to one). For a more detailed explanation of the biological processes involved, please refer to Hammel and Prusinkiewicz (1996).

6. Summary and related work

6.1. Summary

In this paper we advocate the development of new programming languages dedicated to the modeling and simulation of dynamical systems with a dynamic structure, a class of systems at the core of the computational biology applications.

One of the main difficulties raised by this kind of systems, is the specification of the dynamic organization and interaction of the system components. To face this problem, we propose an approach founded on the notion of rewriting. However, to handle the complexity of the spatial and functional organization of biological systems, we extend this approach from the usual multiset rewriting formalism (widely used in artificial chemistry, see Dittrich, 2000) to the more general notion of transformation of topological collections.

The proposed approach is exemplified with four examples of biological processes, at three different levels: biomolecules (with the example of restriction enzymes), biological pathways (with a spatially distributed biochemical network) and tissues (with an Eden’s model and the growth of *Anabaena catenula*). All examples run on an experimental platform that can be downloaded from the MGS home page at URL: http://mgs.lami.univ-evry.fr.

6.2. Comparison with existing formalisms

It is interesting to compare transformations on topological collections with some existing formalisms:

GAMMA and the CHAM, P systems, L systems and cellular automata.

Considering multisets, topological transformations of multisets mimic multiset rewriting introduced by the GAMMA parallel programming language (Banatre and Metayer, 1986) and later formalized by the CHAM formalism (Berry and Boudol, 1990). As mentioned above, a multiset is a too weak structure to cope with the complex organization of biological systems.

P systems, introduced by Paun (2001), stress the notion of membrane structure and are a possible answer to the previous drawback. Some entities are placed in the regions defined by the membranes and evolve following various transformations: an entity can evolve into another entity, can pass trough a membrane or dissolve its enclosing membrane. P systems, in their basic definition, are able to represent the containment relationships of biological entities; however, see Paun et al. (2001) for an extension handling more sophisticated relationships. In contrast with the P system approach, the transformation in `MGS` are not implicitly linked to a multiset but must be threaded from the top-level structure (see the transformation *Thread* in Section 5.3). We are working on incorporating such feature in `MGS`, leading to a more agent-based programming style.

Transformation of sequences corresponds to the L system formalism. This formalism was introduced by Lindenmayer (1968) for simulating the development of multicellular organisms. Related to abstract automata and formal languages, this formalism has been widely used for the modeling of plants. An L system can be roughly described as a grammar where the productions are applied in parallel, in a nondeterministic manner. It can be also viewed from a string rewriting perspective and then topological transformations on sequences correspond to the case of parametric context-sensitive L systems (Giavitto et al., 2002).

At last, transformations on GBFs have to be compared with the cellular automata formalism. There are several differences. The notion of GBF extends the usual square grid of CA to more general Cayley graphs. The value of a cell can be arbitrarily complex (even another GBF) and is not restricted to take a value in a finite set. Moreover, the pattern in a rule may match an arbitrary domain and not only one cell as it is usually the case for CA. For example, the Eden model of Section 5.1 cannot be coded by only one rule in a cellular automata if one wants to avoid that two distinct occupied cells preempt the same unoccupied cell.

To summarize, `MGS` proposes actually a unified view on these computational mechanisms initially inspired by biological processes (CHAM, P systems, L systems and cellular automata). However, we do not claim that we have achieved a useful theoretical framework encompassing these formalisms. We advocate that few notions and a single syntax can be consistently used to allow the merging of these formalisms for simulation purposes. The key notions involved are:

- a unified view on data structures using an abstract neighborhood relationship: the *topological collections*;
- a general device to compute new topological collections from a given topological collection, based on an abstract rewriting mechanism: the *transformation* of a topological collection;
- the representation of the state of a biosystem by a topological collection and the specification of the evolution function as a transformation.

The use of a rewriting mechanism as a foundation for biosystems modeling has already been defended in Fisher et al. (2000). In `MGS`, the use of a general abstract neighborhood operator (the commas that appear in the LHS of the rules of a transformation), makes the specification of a transformation largely independent of the precise neighborhood relationship involved by the collection. This feature allows for instance *exactly the same handling* for multisets, sequences and grids, when the evolution rules are *isotropic* (i.e. when there is no need to distinguish between neighbors solely by their spatial position, see the examples in Section 5.1). Relying on a general abstract neighborhood operator also implies that the evolution rules of the biosystem are *local*, which is often the case considering the nature of the physical laws involved (cf. Tonti, 1974 for the algebraic-topological structure underlying physical theories). In addition, the neighborhood operator avoid the need for a global coordinate system: a point which has been stressed as essential for the easy modeling of developmental processes in the works of P. Prusinkiewicz (see, e.g. Prusinkiewicz, 1999; Fisher et al., 2000).

Acknowledgements

The authors would like to thank P. Prusinkiewicz at University of Calgary where some insights have found their right formulation. They are also grateful to F. Delaplace, J. Cohen and the members of the "Simulation and Epigenesis" group at GENOPOLE-Evry for fruitful discussions, biological motivations and challenging questions. C. Boin and N. Thibault have developed parts of the Bugrim model in MGS. The friendly atmosphere of the workshops IPCAT'01 and WMC'01 has also raised many stimulating questions that have suggested many developments and rethinking. Finally, the comments of the referees have greatly improved the presentation of the paper. This research is supported in part by the CNRS, the GDR ALP, IMPG, GENOPOLE and the University of Evry.

References

Amar, P., Ballet, P., Barlovatz-Meimon, G., Benecke, A., Bernot, G., Bouligand, Y., Bourguine, P., Delaplace, F., Delosme, J.-M., Demarty, M., Fishov, I., Fourmentin-Guilbert, J., Fralick, J., Giavitto, J.-L., Gleyse, B., Godin, C., Incitti, R., Képès, F., Lange, C., Sceller, L.L., Loutellier, C., Michel, O., Molina, F., Monnier, C., Natowicz, R., Norris, V., Orange, N., Pollard, H., Raine, D., Ripoll, C., Rouviere-Yaniv, J., Saier, M., Soler, P., Tambourin, P., Thellier, M., Tracqui, P., Ussery, D., Vincent, J.-C., Vannier, J.-P., Wiggins, P., Zemirline, A., 2003. Hyperstructures, genome analysis and I-cell. Acta Biotheoretica (in press).

Banatre, J.P., Metayer, D.L., 1986. A new computational model and its discipline of programming. Technical Report RR-0566, INRIA.

Berry, G., Boudol, G., 1990. The chemical abstract machine. In: Conference Record 17th ACM Symposium on Principles of Programmming Languages, POPL'90, San Francisco, CA, USA, 17–19 January, 1990. ACM Press, New York, pp. 81–94.

Brown, R., Heyworth, A., 2000. Using rewriting systems to compute left Kan extensions and induced actions of categories. J. Symbolic Comput. 29 (1), 5–31.

Bugrim, A.E., 2000. A logic-based approach for computational analysis of spatially distributed biochemical networks. In: ISMB, San Diego, CA, 2000.

Buneman, P., Naqvi, S., Tannen, V., Wong, L., 1995. Principles of programming with complex objects and collection types. Theor. Comput. Sci. 149 (1), 3–48.

Chaitin, G.J., 2002. Meta-mathematics and the foundations of mathematics. Bull. Eur. Assoc. Theor. Comput. Sci. 77, 167–179.

Chandy, K.M., Misra, J., 1988. Parallel Program Design: A Foundation. Addison-Wesley, Reading, MA.

Dittrich, P., Ziegle, P., Banzhaf, W., 2001. Artificial chemistry—a review. Artificial Life 7, 225–275.

Eden, M., 1958. In: Yockey, H.P. (Ed.), Symposium on Information Theory in Biology. Pergamon Press, New York, p. 359.

Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Talcott, C., 2002a. Pathway logic: executable models of biological networks. In: Proceedings of the Fourth International Workshop on Rewriting Logic and Its Applications (WRLA'2002). Vol. 71 of Electronic Notes in Theoretical Computer Science. Elsevier, Amsterdam.

Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Meseguer, J., Sonmez, J., January 2002b. Pathway logic: symbolic analysis of biological signaling. In: Proceedings of the Pacific Symposium on Biocomputing, pp. 400–412.

Fisher, M., Malcolm, G., Paton, R., 2000. Spatio-logical processes in intracellular signalling. BioSystems 55, 83–92.

Giavitto, J.-L., Michel, O., 2001a. Declarative definition of group indexed data structures and approximation of their domains. In: Proceedings of the 3rd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-01). ACM Press, New York.

Giavitto, J.-L., Michel, O., 2001b. MGS: a rule-based programming language for complex objects and collections. In: van den Brand, M., Verma, R. (Eds.), Electronic Notes in Theoretical Computer Science, vol. 59. Elsevier, Amsterdam.

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

Giavitto, J.-L., Godin, C., Michel, O., Prusinkiewicz, P., 2002. Modelling and simulation of biological processes in the context of genomics. Genopole Evry, Ch. "Computational Models for Integrative and Developmental Biology" (final proceedings and tutorials).

Hammel, M., Prusinkiewicz, P., 1996. Visualization of developmental processes by extrusion in space-time. In: Proceedings of Graphics Interface '96, pp. 246–258.

Hanan, J.S., 1992. Parametric L-systems and their application to the modelling and visualization of plants. Ph.D. thesis, University of Regina.

Hartwell, L.H., Hopfield, J.J., Leibler, S., Murray, A.W., 1999. From molecular to molecular cell biology. Nature 402, 47–52.

Kanehisa, M., 2000. Post-Genome Informatics. Oxford University Press, Oxford.

Kaufman, S., 1995. The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press, Oxford.

Keller, E.F., 1995. Refiguring Life: Metaphors of Twentieth-Century Biology. Columbia University Press, New York.

Lindenmayer, A., 1968. Mathematical models for cellular interaction in development, Parts I and II. J. Theor. Biol. 18, 280–315.

Magnus, W., Karrass, A., Solitar, D., 1976. Combinatorial Group Theory: Presentations in Terms of Generators and Relations. Dover, New York.

Manca, V., 2001. Logical string rewriting. Theor. Comput. Sci. 264, 25–51.

Maynard-Smith, J., 1999. Shaping Life: Genes, Embryos and Evolution. Yale University Press, New Haven, CT.

Paniflov, A.V., Holden, A.V. (Eds.), 1997. Computational Biology of the Heart. Wiley, Chichester.

Paton, R. (Ed.), 1994. Computing with Biological Metaphors. Chapman & Hall, London.

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

Paun, G., Sakakibara, Y., Yokomori, T., 2001. P systems on graphs of restricted forms. Publ. Math. Debrecen.

Prusinkiewicz, P., 1999. Modeling of spatial structure and development of plants: a review. Sci. Horti. 74, 113–149.

Prusinkiewicz, P., Hanan, J., 1990. Visualization of botanical structures and processes using parametric L-systems. In: Thalmann, D. (Ed.), Scientific Visualization and Graphics Simulation. Wiley, Chichester, pp. 183–201.

Stengers, I., 1988. D'une Science à L'autre. Les Concepts Nomades. Le Seuil, Paris, France.

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.

Tyson, J., Borisuk, M., Chen, K., Novak, B., 2000. Computational Modeling of Genetic and Biochemical Networks. Analysis of Complex Dynamics in Cell Cycle Regulation. MIT Press, Cambridge, MA, pp. 287–306.

Wilcox, M., Mitchison, G.J., Smith, R.J., 1973. Pattern formation in the blue-green alga, *Anabaena*. I. Basic mechanisms. J. Cell Sci. 12, 707–723.

Wolfram, S., 2002. A new kind of science. Wolfram Media.