LaMI

Laboratoire de Méthodes Informatiques

The Topological Structures of Membrane Computing

Jean-Louis Giavitto & Olivier Michel

email(s) : `giavitto` *ou* `michel @lami.univ-evry.fr`

Rapport de Recherche n^o **70-2001**

Novembre 2001

CNRS – Université d'Evry Val d'Essonne
523, Place des Terrasses
F–91000 Evry France

The Topological Structures of Membrane Computing

Jean-Louis Giavitto & Olivier Michel

LaMI u.m.r. 8042 du CNRS
Université d'Evry Val d'Essone
91025 Evry Cedex, France.
[giavitto,michel]@lami.univ-evry.fr

LaMI technical report N 70-2001, November 2001

Abstract

In its initial presentation, the P system formalism describes the topology of the membranes as a set of nested regions. This description is too rough and presents several shortcommings: only the nesting of membranes is taken into account, not their adjacency and there is an artificial distinction between a membrane and its enclosed region.

To answer these problems, we shown that most of the notions used to describe P systems find a natural setting and a smooth extension in the framework provided by topological notions developed in the field of homology theory. Notions like membrane structures, adjacency relationships between membranes, local computations, moves between adjacent membranes, etc., can be specified on top of the notion of chain complex.

Using an appropriate abstract setting, this technical device enables us to reformulate also the computation within a membrane and proposes a unified view on several computational mechanisms initially inspired by biological processes, namely: Gamma and the CHAM, P systems, L systems and cellular automata. These models can be rephrased as the iteration of simple transformations on a topological collection, the difference coming from the topology of the collection.

These theoretical tools are instantiated in `MGS`, an experimental programming language handling various types of membrane structures in a homogeneous and uniform syntax.

Keywords

membrane computing, Gamma, CHAM, P system, L system, cellular automata, group based fields, rewriting, topological collection, declarative programming language, combinatorial algebraic topology, chain complex, chain group.

The authors of this research report can be contacted at:

La.M.I., CNRS UMR 8042
Université d'Évry Val d'Essonne
Tour Évry 2 / 4eme etage
523 Place des terrasses de l'agora
91000 Évry Cedex France

Tel: +33 (0)1 60 87 39 04
Fax: +33 (0)1 60 87 37 89

The `MGS` interpreters are freely available, by sending a demand to `giavitto` *or* `michel @lami.univ-evry.fr` .
The MGS home page is located at url `http://www.lami.univ-evry.fr/~mgs` .

Versions of this report:

- Initial Version: november 2001.

The Topological Structures of Membrane Computing

Jean-Louis Giavitto & Olivier Michel

LaMI u.m.r. 8042 du CNRS
Université d'Evry Val d'Essone
91025 Evry Cedex, France.
[giavitto,michel]@lami.univ-evry.fr

LaMI technical report N 70-2001, November 2001

1 Introduction and Motivations

The original motivation of this work lies in the modeling and the computer simulation of biological *dynamical systems* (DS) with a special focus on DS *with a dynamical structure*. Standard DS exhibit a static structure, that is, the exact phase space of the DS can be known statically before the simulation. This is usually not the case for the DS found in biology [5, 6, 7] like the models conceived for developmental processes (e.g. embryogenesis, plant growing), integrative cell models, protein transport and compartment simulation, etc. In this kind of situation, the dynamic of the 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.

Considering the biological roots of this problem, the dynamical structure and the specification of the dynamics, it is not surprising to consider the formalism of P system, and more generally the approach of membrane computing, as a starting point for developping a dedicated programming language. P systems are new distributed parallel computing models based on the notion of a membrane structure [20, 21]. A membrane structure is a nest of cells represented, e.g., by a Venn diagram without intersection and with a unique superset: the skin. Objects are placed in the regions defined by the membranes and evolve following various transformations subject to some conditions: an object can evolve into another object, can pass trough a membrane or dissolve its enclosing membrane, etc. The computation is finished when no object can further evolve.

The need of more accurate membrane structures. In its initial presentation, the P system formalism describes the topology of the membranes as *nesting*. The nested structures of the membranes can be specified in several ways: as a tree, a Venn diagram, a string of matching parentheses, see figure 1. With respect to the modeling and simulation of concrete biological processes, this description is too rough and presents three main shortcommings.

- Only the nesting of membranes is taken into account, not their adjacency (see figure 2). However, the adjacency relationships of cells are of prime importance in the organization of biological tissues (e.g. for the diffusion of morphogenetic gradient).

- There is an artificial distinction between a membrane and its enclosed region: only the enclosed region is decorated with evolving objects. But in real biological compartments (like cells, vesicles, cargo, organs, etc.) the boundary that defines the compartment is itself the place of active and specific processes (reaction between anchored proteins, hyperstructure [18], ionic chanels, etc.) that need the same computational representation as the region.

- Biological compartmentalization localizes processes at regions of various dimensions (active sites are points and 0-dimensionnal, gene's promoters are localized on one-dimensional molecules, cell membranes are two-dimensional and lumens are three-dimensional regions).

The point we want to emphasize here is that the topological organization of the membrane structure is not fully taken into account in the original formulation of the P systems. We use the term "topological organization" to underline the topological nature of the characteristics we want to consider. Obviously, such topological organization can be supported more or less directly in a genuine P system by *coding*. Figure 3 sketches the coding of the adjacency relationships by specific evolution rules (left diagram), and the coding of the membrane labeling (right diagram).

However, taking explicitly into account topological features in the computational model is interesting *per se* and not only to ease the development of simulations of real biological processes. This has already been acknowledged through the development of some P system generalizations, for example toward graph structured membranes [22]. More generally, if we pinpoint "membrane computing models" as computational devices able to:

1. store and move objects between regions (compartments, loci, positions, ..., specified by the membranes),
2. transform locally the objects stored in a region,
3. create, delete and rearrange locally the organization of the regions,

then it is mandatory to study the organization of the regions, their representations and their handling. In section 2 we introduce the notion of a *chain complex* that can be used for this purpose. A chain complex is a standard construction in the field of algebraic topology that formalizes a faithfull and complete representation of the topological organization of a set of membranes. In addition, the algebraic and combinatorial definition of the involved concepts makes them particularly suited for a computer implementation.

Uniform description of the computational mechanisms. The above presentation shows that two basic computation mechanisms are at work in a membrane computing model: one to process the objects in a region and the second to compute the regions. *This is a two stages model.* From this point of view, P systems exhibit the following two characteristics.

- The type of objects and the evolution mechanism are supposed to be the same for all the regions (e.g.: the evolution rules are based on multiset rewriting, or string rewriting, or splicing systems, but not on both).

- A strict distinction is maintained between the global membrane structure (a tree) and the local computational entities that take places into a region (multisets, strings, etc.).

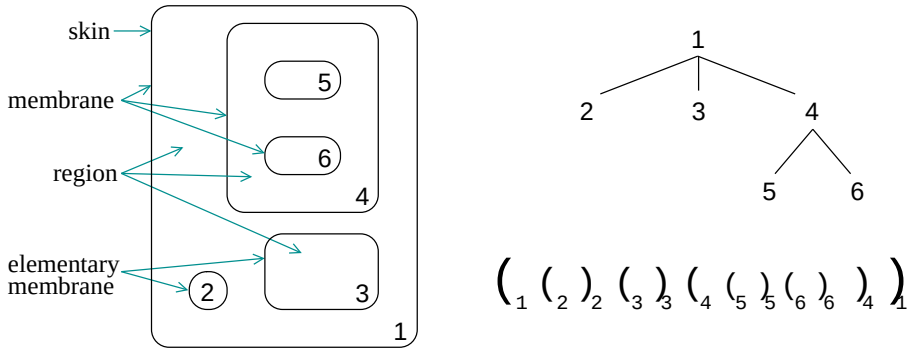


Figure 1: Some representation of the nesting structure of the membranes of a P system: as a Ven diagram, as a tree of regions and as a string of matching parentheses. Regions are numbered from 1 to 6.

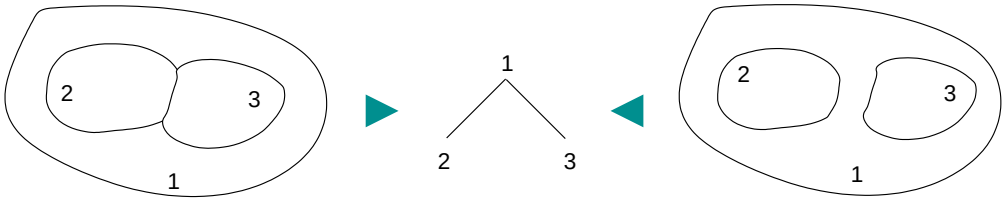


Figure 2: The two different topological situations give the same nesting structure. However, in the diagram to the left, entities in region 2 can pass directly to region 3, which is not the case in the diagram to the right.

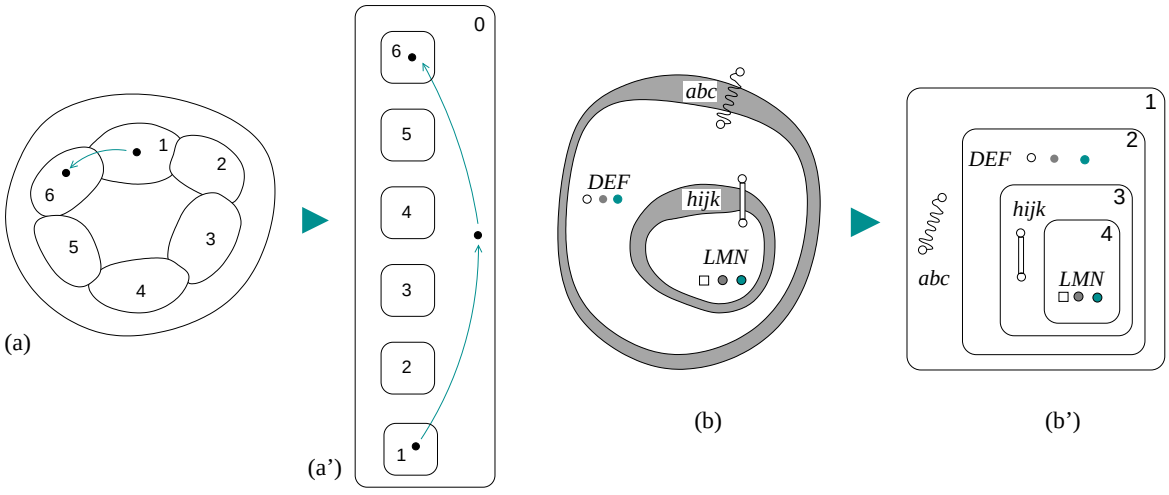


Figure 3: The topological configuration (a) can be coded by the flat membrane structure (a'). Specific transport rules between adjacent compartments are coded by two elementary moves routed between the elementary regions and the top region 0, and then to the final destination. Membranes holding objects (b) (objects are given using italic labels) can be simulated using additional membranes (b').

These characteristics put a burden on the description of the DS, especially when the structure of the system must intrinsically be computed together with its state. A biological motivation to relax these constraints can be illustrated by the simulation of a string of DNA with its coat of activator and inhibitor proteins. The DNA string in the nucleus can be modeled as the object of a splicing system in an enclosing membrane, but it must also be conceived as a region itself endowed with some string rewriting process to take into account the activities and sequential organization of the coat. This example shows that at some level, an entity must be processed as an object in a multiset, while at the same time, at another level, it must be processed as a string. To make this possible, one has to reify the two stages model into a single framework describing with the same device both the computation on objects (of various kind) and the computation on regions.

This unification is not out of reach, because at a sufficiently abstract level, the regions nested in a region R can be conceived as first-citizen objects belonging to R, like the ordinary objects stored in the region. For example, the region 0 in schema (a') of figure 3, can be seen as a multiset of multisets, and then, subject to the same computational mechanism (multiset rewriting) that applies to the atomic objects in an elementary membrane.

It appears that the mathematical device we will introduce to represent adequately arbitrary topological organization of membranes, is also able to support such an uniform specification.

The rest of the paper is organized as follows. Section 2 gives some informations about the notion of chain complexes and defines the notion of topological collection. Based on these notions, the MGS language is described informally in section 3. The topological organization underlying the Gamma programming language and the chemical abstract machine (CHAM), P systems, L systems and cellular automata are formally defined in section 4. The MGS presentation is then completed by some examples covering the previous formalisms in section 5. All examples are processed using the current version of the MGS interpreter. The last section finishes by the review of some directions opened by this research.

2 Cell Complex, Chain Complex and Topological Collections

2.1 Cell Complex

Instead of using a partial order to represent the hierarchichal structure of the membrane's containments, our idea is to use a partial order $<$ to represent the adjacency relationships between the various parts of the membranes. Membranes are supposed to be of any dimension. The mathematical tools we will use are the basic definitions at the start of homology theory. A good introduction is [13] and a standard reference text is [17].

It is convenient to describe the complex shape formed by the membranes together as build from basic blocs called *k-cells*. A k-cell is an homeomorphic image of an open ball in $\mathbb{R}^k$. However, the precise nature of the cell c is not stressed in a purely combinatorial approach until no link is made with point set topology notion. Here, we need only to grad the cells by their dimension and to focus on the connection of cells. A 0-cell is also called a *point* or a *vertex*, a 1-cell is an *edge* and a 2-cell is a *face*. A collection of cells that are fitted together in an appropriate way forms larger structures called *complexes*. Examples of complexes are given in Fig. 4. If an edge e is a side of a face f, we say that e and f are *incident* and we write $e < f$. The incidence relation is a partial order between cells. Let P be the poset of

cells and $x, y \in P$ such that $x < y$ and there is no z such that $x < z$ and $z < y$. Then we write $x \prec y$ and we say that x is a *predecessor* of y or that y is a *successor* of x.

Definition 1 (*Abstract Complex*). An *abstract complex* $\mathcal{K}$ is a poset with a function $\dim : \mathcal{K} \to \mathbb{Z}$ such that $e \prec e'$ implies $\dim e' = 1 + \dim e$. The set $\mathcal{K}_p = \{e \mid e \in \mathcal{K}, \dim e = p\}$ are the p-cells of $\mathcal{K}$. The *dimension* $\dim S$ of a subset $S \subset \mathcal{K}$ is the biggest of the dimensions of the elements of S if it exists.

Given a poset and its partial order $<$, we define the derived $\leq$ and $\preceq$ relationships. We defines now some operations on subsets of complexes. For a subset $S \subseteq P$, the smallest poset containing S is its closure $\overline{S}$. There is two ways for a cell x to be connected with a cell y: because they share a common boundary or because they are both boundaries of a "bigger" cell. Finally, considering an infinite complex may be useful, for instance to represent an unbounded grid. However, each element (vertex or edge) in this grid is connected to only a finite set of other elements. Then, we say that the grid is locally finite.

Definition 2 (*Subcomplex, Star and Shape, Connections and Local Finiteness*). Let $(\mathcal{K}, <)$ be an abstract complex and $S \subseteq \mathcal{K}$ be a subset of $\mathcal{K}$. Then the set $\overline{S} = \{y \mid y \in \mathcal{K}, y \leq x \in S\}$ with the relation $<$ is the subcomplex generated by S. It is called the *closure* of S. The *star* $\operatorname{St} x$ of a cell $x \in \mathcal{K}$ is $\operatorname{St} x = \{y \mid x \leq y \in \mathcal{K}\}$. We define the star of a subset $S \subseteq \mathcal{K}$ to be $\operatorname{St} S = \bigcup_{x \in S} \operatorname{St} x$ and the *closed star* is $\overline{\operatorname{St}} S = \overline{\operatorname{St} S}$. An element x is *above* a set $S \subset \mathcal{K}$ iff $x \in \overline{S}$ or if the elements of the set $\{y \mid y \prec x\}$ are all above S. The *shape* $\operatorname{Shape}(S)$ of a subset $S \subset \mathcal{K}$ is the set of the elements above S. These notions are illustrated in figure 5.

Two cells x and y of an abstract complex $\mathcal{K}$ are *connected*, and we write $x \between y$, iff it exists a cell z such that both x and y belongs to $\overline{\operatorname{St}} z$. In other words, x connected to y requires that $\overline{x} \cap \overline{y} \neq \emptyset$ or that $\operatorname{St} x \cap \operatorname{St} y \neq \emptyset$. Given a set $S \subseteq \mathcal{K}$, we define $(\between \backslash S)$ as the restriction of $\between$ on S: $(\between \backslash S) = \between \cap (S \times S)$. Let $(\between \backslash S)^*$ be the transitive closure of this relation. A subset S of $\mathcal{K}$ is *connected* if $(\between \backslash S)^*$ has only one equivalence class.

A complex $\mathcal{K}$ is *closure-finite* if for all cell $x \in \mathcal{K}$, $\overline{x}$ is a finite set. It is *star-finite* if $\operatorname{St} x$ is a finite set for all x in $\mathcal{K}$. A complex which is both closure-finite and star-finite, is said to be *locally finite*.

2.2 Chain Complex

Figure 4 shows that the poset structure alone is not enough to represent the connections of cells. A cell is not completely described by the simple set of its predecessors. One must represent also some organisation of these predecessors: for example an orientation, or a count if some subcells are identified, etc. This organisation of the set of the predecessors is represented by the notion of *chain*: a chain is a "structured set" of cells. This structure is specified through an abelian group structure and a boundary operator. The abelian group structure is used to describe the gluing of two cells using the group operation (written additively). The boundary operator gives the chain that describes the boundary of a cell, and by extension, the boundary of any chain.

Using an abelian group operation to represent the "gluing" c of two cells x in position g and y in position g' means that we can write $c = g + g'$ or $c = g' + g$: the order of the gluing does not matter. The neutral element 0 corresponds to the empty set. And if we add a cell x to a part c, one must be able to "detach" latter the cell x from c. This justifies the use of a group structure for the set of chains. Furthermore, one of the main objectives of the theory

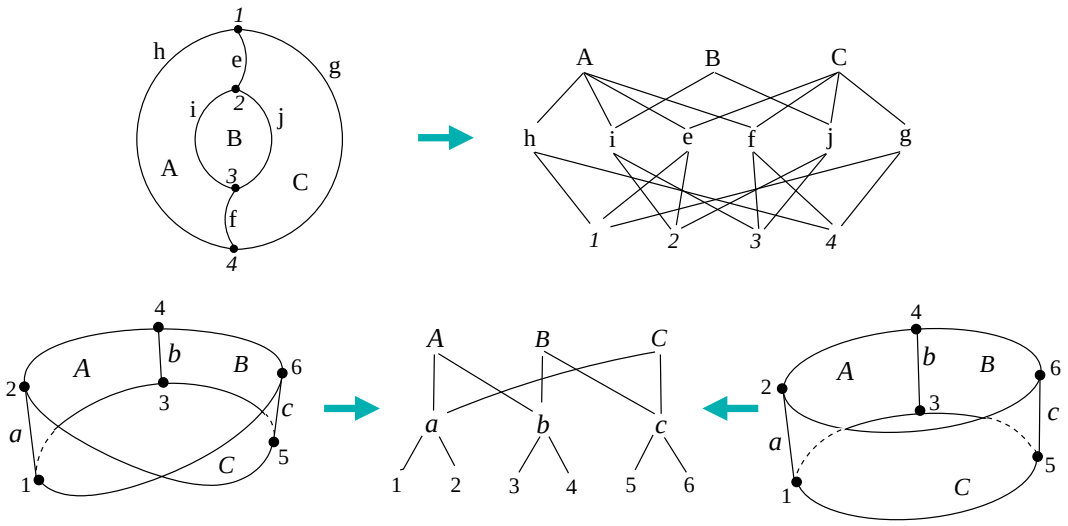


Figure 4: *Top diagrams.*The schema in the right hand side gives the Hasse diagram of the incidence relation of the complex in the left hand side. Faces are denoted by capital letters A, B and C. Edges are denoted by small letters and vertices by numbers. For instance, the face B is bounded by two edges i and j which are themselves bounded by vertices *2* and *3*. This example shows also that an abstract complex is generally not a *lattice*: there is for instance no least upper bound for edges e and f: both faces A and C are incomparable successors of e and f. *Bottom diagrams.* The moebius strip on the left gives the same poset as the cylinder on the right (they are both composed of 3 faces, 3 edges and 6 vertices).

.

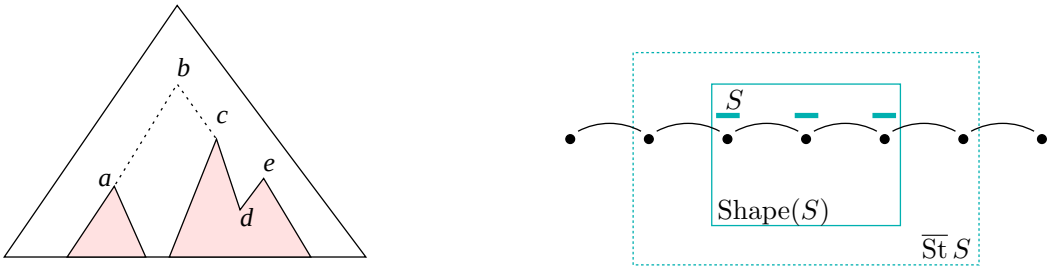


Figure 5: Connection and shape of a set. *Left figure.* We figure symbolically a poset $\mathcal{K}$ by a triangle. The coloured triangle below element a is the subcomplex $\overline{a}$ generated by a. It is also called the *cone* below a. An element x is in the cone below y iff $x \leq y$. The set $\{a, b, c, d, e\}$ is connected because elements are connected two by two. Fo example, a and b are connected because $a \leq b$, idem for c and b. The elements c and e are connected because $d \leq c$ and $d \leq e$. Let $A = \overline{a}$, $C = \overline{c}$ and $E = \overline{e}$ be the closure of $\{a\}, \{c\}$ and $\{e\}$ respectively. Then the set $A \cup C \cup E \cup \{b\}$ is also connected because a closure of a connected set is connected. *Right figure.* The set S consists of three internal vertices of a line graph. We have figured $\overline{\mathrm{St}}(S)$ and $\mathrm{Shape}(S)$.

is to compute the boundary of an arbitrary part of a space, from the boudary defined for an "isolated" cell (to compute the neighbors of an arbitrary membrane). Then, it is natural to require the boundary operator ∂ to be an homomorphism: $\partial(g + g') = \partial(g) + \partial(g')$. These considerations motivate the following definitions.

DEFINITION 3 (*Chain Group with Coefficients and Chain Complex*). Let $\mathcal{K}$ be an abstract complex, and let G denotes an arbitrary abelian group written additively. The neutral element of G is written 0. The set $C_p(\mathcal{K}, G)$ of *p-chain on the complex $\mathcal{K}$ with coefficients in G* is the set of total functions c_p from the set $\mathcal{K}_p$ to G that are zero almost everywhere, that is, $c_p(x) = 0$ for all but a finite number of p-cells of $\mathcal{K}$. The set $C_p(\mathcal{K}, G)$ is an abelian group for the addition of functions. *The* chain group with coefficients in G is defined by: $Chains(\mathcal{K}, G) = C_0(\mathcal{K}, G) \oplus C_1(\mathcal{K}, G) \oplus \dots$ where $\oplus$ is the direct sum of abelian groups.

A *chain complex* $C(\mathcal{K}, G, \partial)$ is a sequence $(C_p(\mathcal{K}, G), \partial_p)_{p\in\mathbb{Z}}$ of the abelian groups C_p and connecting homomorphism $\partial_p : C_p \to C_{p-1}$, called *boundary maps*.

An element c of $C_p(\mathcal{K}, G)$ is called a *p-chain*. $C_p(K, G)$ represents all the way to glue p-cells together. Sometimes we use a subscript p to indicate that a chain c is a p-chain: c_p. In the opposite, for convenience in notation, we shall sometimes delete the dimensional subscript p on the boundary operator ∂_p, and rely on the context to make clear which of these operators is intended. We also abbreviate $C_p(\mathcal{K}, G)$ by C_p, $C(\mathcal{K}, G, \partial)$ by C and use uniformly 0 to denote the neutral element of any abelian group.

An abelian group C_p is *trivial* when its only p-chain is 0 (the element zero of the group of functions). It this case we write $C_p = 0$. A *finite dimensional* chain complex C is such that the C_p are trivial except for at most a finite number of p. If C_p is the trivial group for $p < 0$, we say that C is a *non-negative* chain complex. The *carrier* of c_p is the set of p-cells with a nonzero coefficient in the chain: $|c_p| = \{x \in \mathcal{K}_p \mid c_p(x) \neq 0\}$.

It is customary to use a linear additive notation for a chain c_p: $c_p = \sum_{x\in|c_p|} c_p(x).x$. Indeed, $C_p(\mathcal{K}, G)$ can alternatively be defined as the formal sums with variable $x \in \mathcal{K}_p$ and coefficients in G. Let $c_p = \alpha_1 x_1 + \dots + \alpha_n x_n$ be a chain of $C_p(\mathcal{K}, G)$. Then $\alpha_i \in G$ and we suppose in addition that $\alpha_i \neq 0$ for all i and that $i \neq j$ implies $x_i \neq x_j$.

Example of the $C(\mathcal{K}, \mathbb{Z}/2, \partial)$ Chain Complex. $\mathbb{Z}/2$ denotes the module of relative integers modulo 2. Using $\mathbb{Z}/2$ as the chain coefficients enables the representation of the presence, $c_p(x) = 1$, or the absence, $c_p(x) = 0$, of a p-cell x in a chain c_p. A chain of $C(\mathcal{K}, \mathbb{Z}/2)$ is then simply the characteristic function of a subset of $\mathcal{K}$. An example is given in figure 6. A chain $c = e + f$ corresponds to the function c defined by $c(e) = c(f) = 1$ and $c(x) = 0$ for $x \neq e$ and $x \neq f$. This chain can also be written $c = 1.e + 1.f + 0.g + 0.h + \dots$. It is customary not to write the p-cells with a zero coefficient (in accordance with the additive notation). Thus we have $c = 1.e + 1.f$ or more ambiguously $c = e + f$. Suppose that the chain $c \in C_p(\mathcal{K}, \mathbb{Z}/2)$ is composed of two k-cells s and s'; this is denoted by $c = s + s'$. Suppose than s and s' share only one cell $d \in \mathcal{K}_{p-1}$, see Fig. 6. Then d is not in the border of s because s and s' are glued along d: d is an interior cell. But d is in the boundary of s and in the boundary of s'. Let $\partial_p s = d + \sum x'_j$ and $\partial_p s' = d + \sum x''_k$. Then we must have: $d + \sum x'_j + d + \sum x''_k = \sum x'_j + \sum x''_k$ which is *automatically* achieved because $d + d = 2d = 0$.

2.3 Arbitrary Labeling the Cells of a Complex

Suppose we want to label *some* of the cells of a complex with values taken in an arbitrary set *Val*. Such labeling can be represented by a *partial* function ℓ from $\mathcal{K}$ to *Val*. This partial function can be extended into a total function given the value $\perp$, $\perp \notin Val$, to the cells that have no image by ℓ. Then, the function ℓ *can be seen as a chain* if we give an abelian group structure to $Val \cup \{\perp\}$.

A natural choice is to use Abel(*Val*) the free abelian group generated by the elements of *Val*. We rely on the injection $x \mapsto x$ to represent an element of *Val* by an element of Abel(*Val*) and $\perp$ is represented by 0. This group has a richer structure than *Val* and enables the association of a cell to a "generalized multiset" of *Val* elements. In a generalized multiset, an element can have a negative multiplicity. Alternatively, Abel(*Val*) can be defined as the set of total functions from *Val* to $\mathbb{Z}$.

Remark that if *Val* has already a group structure $+$, the operation in Abel(*Val*) does not coincide with the operation $+_{\text{Abel}}$ in Abel(*Val*). Take for example $Val = \mathbb{Z}$, then $x +_{\text{Abel}} (-x) \neq 0_{\text{Abel}}$. Indeed, both x and $(-x)$ are generators of Abel($\mathbb{Z}$) and they are distinct.

Boundary and Coboundary as Transport Operation. In an arbitrary labeling of a complex, we can interpret the ∂ operations as *transport* operations, see figure 8 and the references [24, 25, 19].

Suppose that we want to valuate the cells of the chains by an element of *Val*. We use the previous encoding based on Abel(*Val*) for the chain coefficients. We define the boundary of a cell x by:

$$\partial x = \sum_{y \prec x} y \qquad \text{and extend } \partial \text{ linearly:} \qquad \partial(\sum \alpha_x x) = \sum \alpha_x \partial x$$

Consider a cell x that has several successors in the chain. Then the effect of ∂ as a transport operation is to send to x the coefficients of theses successors. The result is conveniently gathered as a formal sum in Abel(*Val*) and no coefficients are lost. We can then further interpret "the collision at cell x of the transported values" using an homomorphism to resolve the "collisions" and to compute the final value of x.

If operators ∂_p transport values from a cell to its predecessor, it exists a family of dual operator that moves values from a cell to its successor. Such operators are the dual (in a precise sense, see [17]) of the boundary maps ∂_p.

To be more concrete, suppose that the cells in figure 8 (left) are valuated by reals, that is, we consider chains in $C(\mathcal{K}, \text{Abel}(\mathbb{R}))$. For instance, take $\omega = 1.6$ and $\omega' = 3.1$ in chain ℓ_2. Then

$$\partial(1.6s + 3.1s') = 1.6a + 1.6b + 1.6c + (1.6 +_{\text{Abel}} 3.1)d + 3.1f + 3.1e$$

We say that the value 1.6 coming from s and the value 3.1 coming from s', collide at cell d. We want to combine colliding values into a real to get again a real valued chain. Suppose that the combination function is the sum of reals. Then we would use the homomorphism h from Abel($\mathbb{R}$) to $(\mathbb{R}, +)$ that interprets the $+_{\text{Abel}}$ as the usual $+_{\mathbb{R}}$. The homomorphism h between the groups of values, is easily extended into an homomorphism on chains, by defining $h(\alpha x) = h(\alpha)x$ for all cell x and then using linearity. Instead of using a function h to combine the colliding values, we can work directly with chains in $C(\mathcal{K}, (\mathbb{R}, +))$. In this way, the combining function is directly the group operation of the chain coefficients. However, using Abel($\mathbb{R}$) and then an *a posteriori* homomorphism h is more general. For instance,

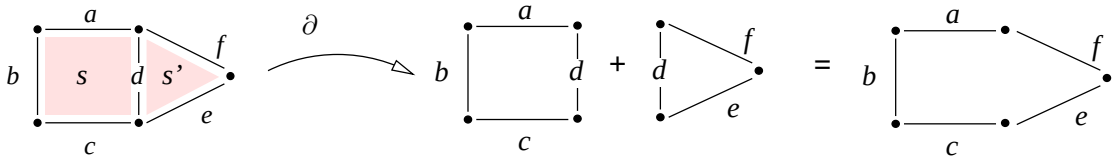
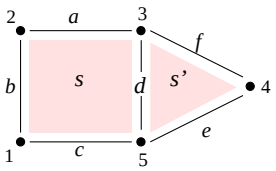


Figure 6: Example of the application of the boundary operator on a $C(\mathcal{K}, \mathbb{Z}/2)$ chain. $\partial(s + s') = \partial s + \partial s' = (a + b + c + d) + (d + e + f) = a + b + c + e + f$ because $d + d = 0$.

$\mathcal{K}_0 = \{1, 2, 3, 4, 5\}$
$\mathcal{K}_1 = \{a, b, c, d, e, f\}$
$\mathcal{K}_2 = \{s, s'\}$



$\ell(1) = \delta, \quad \ell(2) = \alpha$
$\ell(3) = \beta, \quad \ell(4) = \gamma$
$\ell(a) = \rho, \quad \ell(b) = \kappa$
$\ell(c) = \sigma, \quad \ell(d) = \tau$
$\ell(s) = \omega$
and $\ell(x)$ undefined for the others x.

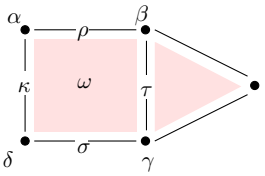


Figure 7: The labeling of the cells of an abstract complex. The figure in the left gives the abstract complex $\mathcal{K}$ and its p-cells $\mathcal{K}_p$ (for $p = 0, 1, 2$). The labeling ℓ is defined on the right. In this diagram, we indicate the images of the function ℓ by writing next to each cell the value of the function on that cell. This function has for codomain the set $Val = \{\alpha, \beta, \gamma, \delta, \rho, \tau, \sigma, \kappa, \omega\}$ which *a priori* do not have an abelian group structure. The function ℓ can be written as a chain of $C(\mathcal{K}, \mathrm{Abel}(Val))$: $\ell = \delta.1 + \alpha.2 + \beta.3 + \gamma.4 + \rho.a + \kappa.b + \sigma.c + \tau.d + \omega.s$. However, note that in $C(\mathcal{K}, \mathrm{Abel}(Val))$ there are also chains like $(\alpha +_{\mathrm{Abel}(Val)} \beta).1$ which would represents a function f such that $f(1) = \{\alpha, \beta\}$ and undefined elsewhere.

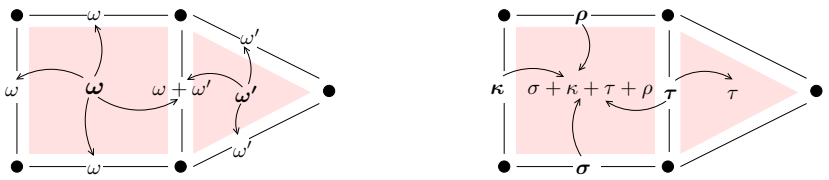


Figure 8: Depiction of the boundary and coboundary operation on chains. We consider the abstract complex already used in figure 7. The effect of taking the boundary operator ∂ on $\ell_2 = \omega.s + \omega'.s'$ is pictured by the diagram in the left. The figure in the right gives the effect of taking the coboundary δ of the 1-chain $\ell_1 = \rho.a + \kappa.b + \sigma.c + \tau.d$. The coboundary operators δ^p are the dual homomorphisms of the operators ∂_p (see [17]). In these two figures, the curved arrow indicate values (in bold) being transferred from a p-cell to the preceding $(p-1)$-cells (for ∂) and from a $(p-1)$-cell to the succeeding p-cells (for δ).

suppose that we work with coefficients in $(\mathbb{R}, +)$ but we want to combine the colliding values by multiplication. This is not easily expressed. But using Abel($\mathbb{R}$) at the first place, we have just to change the function h. The combination function must not depend on the order of the combinations and then the chain $(\alpha + \beta)x$ must be equal to the chain $(\beta + \alpha)x$. Intuitively, one can see the interest of using an abelian group for the coefficients.

2.4 Topological Collection

A "snapshot" of a P system will be described by a topological collection. A topological collection associates a value to some cells of a complex. In addition, we must be able to speak of the carrier of the collection (the cells that have a value), of the neighbors of an element, of subcollections and of the boundary of a subcollection. All these notions can be developed on top of the notion of chain complex presented above.

Definition 4 (*Simple Topological Collection*). A *simple topological collection type* is a quadruple $\mathcal{T} = (\mathcal{K}, B, \partial, \mathit{Val})$ such that $\mathcal{K}$ is a finite-dimensional, non-negative, locally-finite abstract complex and $C(\mathcal{K}, B, \partial)$ is a chain complex. A *simple topological collection* is a pair $(\mathcal{T}, c)$ where $\mathcal{T}$ is a topological collection type $(\mathcal{K}, B, \partial, \mathit{Val})$ and c is a chain: $c \in \mathit{Chains}(\mathcal{K}, B \odot \mathit{Val})$. The product $B \odot \mathit{Val}$ denotes the cartesian product $B \times \mathrm{Abel}(\mathit{Val})$.

Often we omit to mention the type $\mathcal{T}$ of the topological collection when it is clear from the context; we says directly that a chain c is a simple topological collection (or more simply is a collection) and we write $c \in \mathcal{T}$ if $\mathcal{T}$ is the type of c. The chain complex $C(\mathcal{K}, B, \partial)$ is called the *form* of the type.

If c is a collection, and $x \in \mathcal{K}_p$, then $c(x) = (g, u)$ with $g \in B$ and $u \in \mathrm{Abel}(\mathit{Val})$ and we say that *the value of c at x is u*. The functions c_b and c_v are the first and second projection of c. That is, $c_b(x) = g$ and $c_v(x) = u$ for $c(x) = (g, u)$. The functions c_b and c_v associate an element of a group to a cell and then are chains: $c_b \in \mathit{Chains}(\mathcal{K}, B)$ and $c_v \in \mathit{Chains}(\mathcal{K}, \mathrm{Abel}(\mathit{Val}))$. For all collection c we have $|c_v| \subset |c|$ and $|c_b| \subset |c|$. The set $\mathrm{Residu}(c) = \{x \in \mathcal{K} \mid c_b(x) = 0_B \text{ and } c_v(x) \neq 0_{\mathrm{Abel}(\mathit{Val})}\}$ is called the *residue* of the collection. A collection c is *residue-free* if $\mathrm{Residu}(c) = \emptyset$. A topological collection c is *flat* if $c_v(x) = 0$ or $c_v(x) \in \mathit{Val}$ for all $x \in \mathcal{K}$.

2.5 Simple Transformation of a Topological Collection

Now, we want to state precisely the notion of *local computation*. A local computation would be done by some kind of rewriting mechanism that substitutes a subcollection c' in c by another one. If only c'_v is changed, then there is no change in the structure of the P system. Deleting or creating new membranes corresponds to a change in c'_b (and accordingly in c'_v).

The *restriction* $c\backslash S$ of a topological collection c by a set S is the chain $c\backslash S$ defined by $(c\backslash S)(x) = c(x)$ if $x \in S$ and by $(c\backslash S)(x) = 0$ elsewhere. A restriction is too general to represent a subcollection: a subcollection is a connected part of a collection. It must be represented by a chain too.

Definition 5 (*Split, Patch and Subcollection*). Let c be a chain and c' and c'' be two chains such that $|c'| \cap |c''| = \emptyset$ and $c = c' + c''$. Then we say that c' and c'' are a *split* of the chain c and we write $c \trianglerighteq c'$, $c \trianglerighteq c''$ and $c'' = \complement_c c'$ or $c' = \complement_c c''$. A chain c' is a *patch* of the chain $c \in \mathit{Chains}(\mathcal{K}, G)$, if $c \trianglerighteq c'$ and if $\mathrm{Shape}\,|c'|$ is a connected set of $\mathcal{K}$. Let c be a collection; a collection c' is a subcollection of c if $c' = c\backslash|c'|$ and if c'_b is a patch of c_b.

Now, we can define the basic transformation step which is used in the MGS language. The basic intuition hidden behind this definition is sketched in figure 9. Note that we do not describe a device to select a subcollection into a collection, neither we give conditions on the gluing of the substituted subcollection. We just specify that untouched parts of the collection must remain unchanged, both from the value point of view (condition 1) and the shape point of view (condition 2).

DEFINITION 6 (*Simple Transformation*). Let c and d be collections with respective subcollections c' and d'. Then d is a *simple transformation* of c' by d' if the two following conditions hold:

1. $\mathsf{C}_c c' = \mathsf{C}_d d'$
2. $\text{Shape}\,|\mathsf{C}_c c'| = \text{Shape}\,|\mathsf{C}_d d'|$

If a function f such that $d' = f(c\backslash|\,\overline{\text{St}}\,c'|)$ exists, then the substitution is said *computed by* f.

Note that there is several possible variations on the notion of "computed by f" to accommodate the possible variation on the neighborhood notion.

3 MGS: a Programming Language based on Topological Collections and their Transformations

The experimental programming language MGS[1] instantiates 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. MGS is an applicative programming language: operators acting on values combine values to give new values, they do not act by side-effect. In our context, dynamically typed means that there is no static type checking and that type errors are detected at run-time during evaluation. Although dynamically typed, the set of values has a rich type structure used in the definition of pattern-matching, rules and transformations.

The approach of MGS, focusing on the notion of topological collection, emphasizes the spatial aspect of a data structure: a collection is seen as a set of *places* or *positions* organized by a *topology* defining the *neighborhood* of each element in the collection. This approach is part of a long term research effort [12] developed for instance in [8] where the focus is on the substructure and in [9] where a general tool for uniform neighborhood definition is developed.

We will see in section 4 that several usual data structures have a natural topology. In the rest of this section, we sketch some of the language constructs without relying on a particular collection type. Thus, by collection we understand a topological collection, as described formally in the previous section. In section 5, some examples illustrate the expressive power of the approach and give a more concrete flavor of the language.

3.1 Computing with Topological Collections

The computation of a new collection is done by a structural combination of the results of more elementary local computations involving only a small and static subset of the initial

[1] MGS is the acronym of *"(encore) un **M**odèle **G**éneral de **S**imulation (de système dynamique)"* (yet another General Model for the Simulation of dynamical systems).

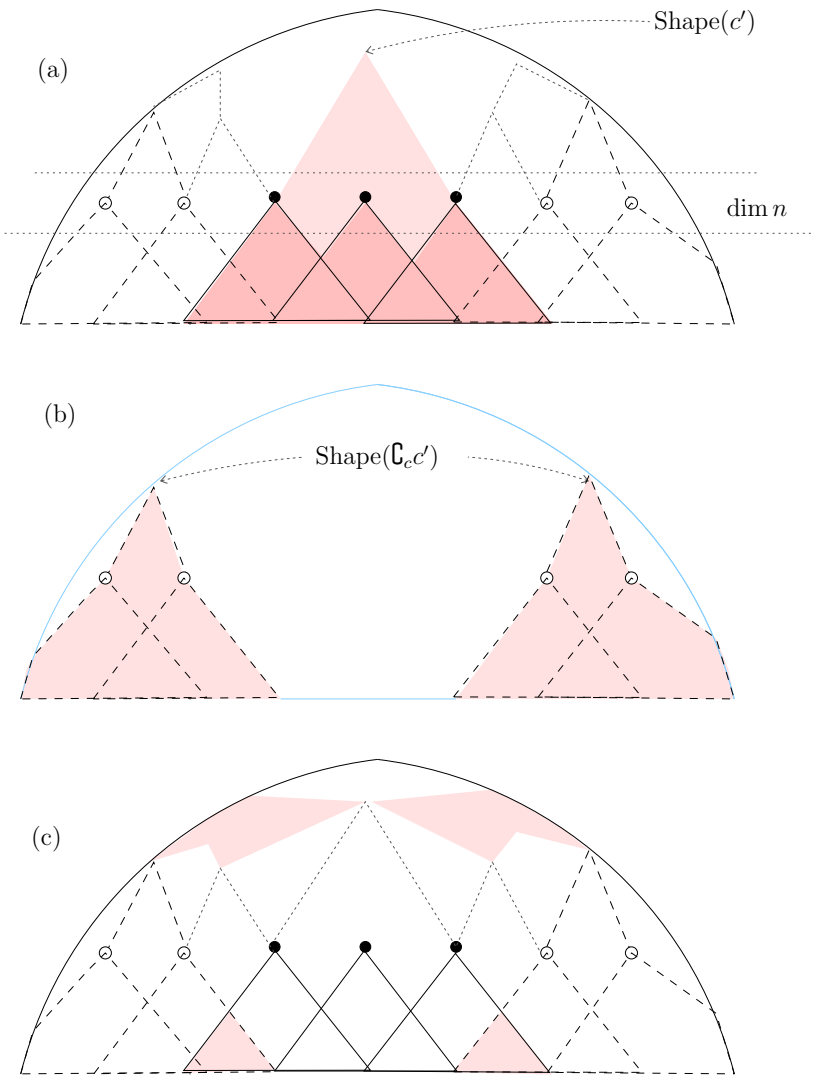


Figure 9: Parts of a complex involved in a substitution. We have pictured symbolically the abstract complex $\mathcal{K}$ as a Hasse diagram (cf. Fig. 5). The carrier of the chain c consists in all the n-cells pictured as circle (diagram (a)). The three black circles in the middle specify the carrier of the subcollection c'. Consequently, the four empty circles are the carrier of $c'' = \complement_c c'$.
The shape Shape(c') of c' is sketched as the gray region in diagram (a): the subcomplex $\overline{|c'|}$ spanned by c' is in dark gray while the p-cells above this subcomplex are in light gray. The shape Shape(c'') is sketched in gray in diagram (b). This part of the complex must remain unchanged across a simple transformation.
The diagram (c) has two gray regions, one near the top and one near the bottom (each is composed of several parts). The region near the bottom, corresponds to the intersection Shape(c') $\cap$ Shape(c''). Cells in this region have a dimension less than n. The definition of a simple transformation says that this region must remain unchanged in the final result (because it belongs to the shape of c'' and then must not be touched by the transformation). The region near the top corresponds to the p-cells x, $p > n$, such that $\overline{x}$ has an intersection both in $\overline{|c'|}$ and $\overline{|c''|}$. The definition of a simple transformation does not say anything about such cells.

collection. "*Small and static subset*" makes explicit that only a fixed subset of the initial elements are used to compute a new element value. "*Structural combination*", means that the elementary results are combined into a new collection, irrespectively of their precise value. The global organization of the new collection results of the combination of these local changes. These characteristics lead to the following abstract computational mechanism:

1. a subcollection A is selected in a collection C;
2. a new subcollection B is computed from A and a local neighborhood;
3. the collection B is substituted for A in C.

This process is pictured in Fig. 10 and is formalized by the notion of *simple transformation* developed in the previous section.

A transformation, without the "simple" qualifier, consists in several non interacting simple transformations applied in parallel to a collection. Back to our application area (Cf. section 1) a transformation corresponds to one evolution step of a spatially distributed DS. Then, the iteration of transformations builds the entire DS trajectory, Cf. Fig. 11.

In addition to the specification of the underlying organization, the definition of a simple transformation requires the specification of the subcollection A and the replacement B. This specification defines a ***rule*** and must adapt several constraints and variations.

3.2 Patterns, Rules and Transformations

A transformation T is a set of rules:

$$\texttt{trans}\ T = \{\ \ldots \quad \mathit{rule}; \quad \ldots\ \}$$

When there is only one rule in the transformation, the enclosing braces can be dropped. A rule is a basic transformation taking the following form:

$$\mathit{pattern}\ \Rightarrow\ \mathit{expression}$$

where *pattern* in the left hand side (lhs) of the rule matches a subcollection A of the collection C on which the transformation is applied. The subcollection A is substituted in C by the collection B computed by the *expression* in the right hand side (rhs) of the rule. Each collection kind comes with its own specific behavior for the pasting of B into $\complement_C A$.

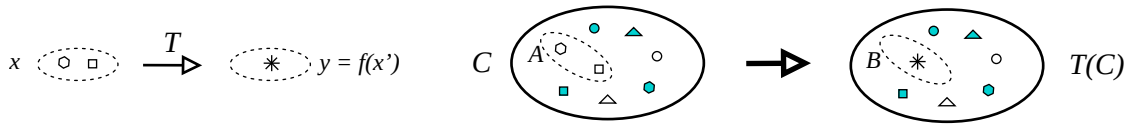


Figure 10: A simple transformation of a collection. Collection C is of some kind (set, sequence, array, cyclic grid, tree, term, etc). A rule T specifies that a subcollection A of C has to be substituted by a collection B computed from A. The right hand side of the rule is computed from the subcollection matched by the left hand side x and its possible neighbors x' in the collection C.

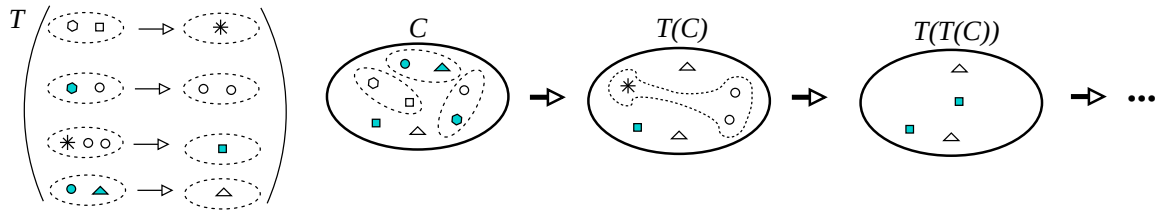


Figure 11: Transformation and iteration of a transformation. A transformation T is a set of simple transformations applied "in parallel" to make one evolution step. The simple transformations do not interact together. A transformation is then iterated to build the successive states of the system.

We present the pattern expressions that have a generic meaning, that is, they can be interpreted against any collection kind. The grammar of such pattern expressions is the following

$$Pat ::= x \mid \{...\} \mid p, p' \mid p+ \mid p* \mid p : P \mid p/exp \mid p \texttt{ as } x \mid (p)$$

where p, p' are patterns, x ranges over the pattern variables, P is a predicate and exp is an expression with a boolean value. The explanations below give an informal semantics for these patterns.

variable: a pattern variable x matches exactly one element in the collection (i.e. a k-cell). The name x can then occurs elsewhere in the rule.

state pattern: $\{...\}$ are used to match one element (a k-cell) whose value is a record. 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,

$$\{a,\ b : \texttt{string},\ c = 3,\ \tilde{}d\}$$

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.

neighbor: p, p' is a pattern that matches two connected collections p and p'. For example, x, y matches two connected elements. The connection relationship is introduced in section 2 and depends of the collection kind.

repetition: pattern $p+$ (resp. $p*$) matches a non empty aggregate of connected elements (resp. a possibly empty aggregate).

binding: a binding p `as` x gives the name x to the collection matched by p. This name can be used in the rest of the rule. For example, $p+$ `as` x identifies under the name x the subcollection matched by $p+$.

guard: p/exp matches the collections matched by p verifying the condition exp. For instance, $y \,/\, y > 3$ matches a cell valued by an integer greater than 3. Pattern $p : P$ abbreviates $(p \texttt{ as } x)/P(x)$ where x is a fresh variable.

Here is a contrived example. Pattern

$$(x : int/x < 3) + \texttt{ as } S \quad / \quad 10 < \texttt{Fold}((\backslash a, b.\, a + b),\ 0,\ S)$$

selects a connected collection S of integers less than 3, such that the sum of the elements in S is greater than 10. (The generic operator `Fold` reduces a collection using a binary function, which is supposed to be associative and commutative, and an initial value. The notation $\backslash a, b.\, exp$ denotes the lambda abstraction of the variable a and b over the expression exp.) If this pattern is used against a linear sequence, S denotes a subsequence. If this pattern is used against a set, then S denotes a subset. Etc. See section 4.

3.3 Managing the Applications of a Transformation

A transformation is a set of rules. When a transformation is applied to a collection, the strategy is to apply as many rules as possible in parallel. A rule can be applied if its pattern matches a subcollection. Several features are used to have a control over the choice of the rules applied within a transformation. For instance, a priority can be associated to each rule to specify a precedence order within each class (the priority of inclusive rules may be used to specify the relative order of their applications).

A transformation T can be used like a unary function. For instance, a transformation can be passed as an argument to another function. It makes able to sequence and compose transformations very easily.

The expression $T(c)$ denotes the application of one transformation step to the collection c. As said above, a transformation step consists in the parallel application of the rules (modulo the rule application's features). A transformation step can be easily iterated:

$T[n]\,(c)$	denotes the application of n transformation steps to c
$T[\texttt{fixpoint}]\,(c)$	application of the transformation T until a fixpoint is reached
$T[\texttt{fixrule}]\,(c)$	idem but the fixpoint is detected when no rule applies

In addition to the standard transformation step strategy, two other *application modes* exist. In the *stochastic mode*, the choice of the exclusive rule to apply is made randomly. The priorities of the exclusive rules are then considered as the relative probability of their effective application (when they can apply). In *asynchronous mode*, only one exclusive rule is applied in one transformation step.

4 The Topology of Sets, Multisets, Sequences and Arrays

In this section, we show that several classical data structures can be seen from a topological point of view. The notion of transformation introduced in the previous section on such collection, allows us to recover some well-known computational models. More precisely:

- using transformation on multisets, we recover Gamma [1] and P system like models;
- using transformation on sequences, we recover the L system formalism [23];
- using transformation on arrays, we retrieve cellular automata [26].

We sketch how these well known models can be roughly rephrased and mimicked in the framework of topological collections. The representations given are only approximations of the exact computation mechanisms, because we do not fully consider the very basic details (they are very relevant for the study of the formal expressive power of each formalism but are not considered here, as a programming language always embeds a lot of small extensions required to facilitate the programmer's life). Section 5 gives examples of `MGS` programs that have been initially proposed as paradigmatic examples of these formalisms.

4.1 Monoidal Collections

Consider a monoid M over an alphabet A with an operation written "$,$". Let m be an element of M. If M is free, then m is a representation of a sequence of elements in A. Moreover, if M is not free because operation $,$ is commutative, then m represents a multiset of elements in A. And if $,$ is also idempotent (i.e. $x , x = x$), then m represents a set. See [14].

It is not a coincidence that the neighborhood relationship in definition 2 and the join operation here are denoted by the same comma. We say that x and y belonging to A are neighbors in m iff $m = u , x , y , v$ or $m = u , y , x , v$ with u and v elements of M. This implies that:

- In a set, an element x is neighbor of any other element y;
- The neighborhood relationship in a multiset is the same as the neighborhood relationship in a set: two arbitrary elements are always neighbors. The difference is that the same element may appear more than one time in the multiset.
- The neighborhood relationship in a sequence is the expected one: if the sequence has at least two elements, then all elements except the first and the last have two neighbors (called the *left* and the *right* neighbor). The first and the last element have only one neighbor (respectively a right and a left neighbor). If the sequence is reduced to a singleton, then this singleton as no neighbor.

These topologies can be described as abstract complexes in the following manner.

The topology of sets. A set V is represented by a topological 0-collection on a one dimensional form with vertices V and only one edge $\top$. The function ∂_1 is defined by $\partial_1 \top = \sum V$. With this definition, an element of V is connected with any other element. The chain group describing a set is then particularly simple: $C_p = 0$ for $p \neq 0$, $K_0 = V$ and $C_0 = C_0(\mathcal{K}, \mathbb{Z}/2 \odot V)$. A set V corresponds to the chain $\sum_{x \in V} x.x$.

Let c' be the subcollection to be replaced by d' into the collection c to give a new collection d. The fixed strategy used to build d from d' and $c'' = \complement_c c'$, is simply to set $\top_d = |c''| \cup |d'|$.

This description is only combinatorial and does not admit a geometric realization. Indeed, a geometric 1-cell is homeomorphic to the interval $[0, 1]$ and then admits only two 0-cells in its boundary. If one insists to have a geometric realization of topological sets, then shifting the dimension of the cells by one is enough: the elements of V are the many edges of a unique polygonal face.

The topology of multisets. A multiset M of elements $e \in E$ can be represented by a set $\hat{M} \subseteq \mathbb{N} \times E$. If $e \in M$ with multiplicity n, then the n elements $(p_1, e), (p_2, e), ..., (p_n, e)$ where

the p_i are n arbitrary distinct integers, belong to $\hat{M}$. The multiset M is represented as the 1-collection associated to the set $\hat{M}$.

With this encoding, two arbitrary multiset elements are connected, in accordance with the fact that any submultiset can be matched and replaced in a Gamma rule. Furthermore, the application of one Gamma rule on a multiset M is the parallel application of simple transformation and therefore, an MGS transformation.

The topology of sequences. A sequence $\ell = <\ell_1, \ell_2, \ldots, \ell_n>$ is a 0-collection whose form is a chain complex of dimension 1. Let i_k be n rationals in increasing order; the underlying complex $\mathcal{K}$ is defined by

$$\begin{aligned} \mathcal{K}_0 &= \{i_1, \ldots, i_n\} \qquad \text{such that } i_j < i_{j+1} \\ \mathcal{K}_1 &= \big\{(i_1, i_2), (i_2, i_3), \ldots, (i_{n-1}, i_n)\big\} \\ \partial(i,j) &= i + j \end{aligned}$$

(the last sum is a formal sum, the operator + is not the addition of rationals). The form of the sequences is $C(\mathcal{K}, \mathbb{Z}/2, \delta)$. Hence, ℓ is represented by the chain $\sum_{1 \le j \le n} \ell_j.i_j$.

An MGS rule $c' => d'$ applied to a topological sequence c corresponds to a substitution with result d. The strategy used to glue the new subcollection d' and $c'' = \complement_c c'$ into the result d is the following:

- if $d' = 0$ (that is, the MGS rule cancel c') then $\text{Shape}(d) = \text{Shape}(c'')$;
- if $d' \neq 0$, then $\delta c' = \delta d'$ (operator δ is the coboundary operator defined by: $\delta i_k = (i_{k-1}, i_k) + (i_k, i_k + 1)$ if i_{k-1} and i_{k+1} exist; the δ in the left hand side must be taken in the form of c while the δ in the right hand side must be taken in d). This condition, together with $d = d' + c''$, is sufficient to specify completely $\text{Shape}(d)$: $\text{Shape}(d) = \text{Shape}(d') \cup \text{Shape}(c'') \cup |\delta c'|$.

These rules are just the formal expression of inserting d' in place of c' and corresponds to the behavior of L system rules on a word.

4.2 Arrays and their Extensions

We have showed in [12, 9] that usual arrays are a special case of labelled Cayley graphs. These structures are called "group based fields" (GBF) and subsume arrays, trees, circular buffer, etc. There is no room to develop this approach here, but it is sufficient to consider the case of free abelian groups to handle standard grids of cellular automata in any dimension.

Let G^n be the free abelian group generated by $d_1, \ldots, d_n$. We associate to this group the abstract complex $(\mathcal{G}^n, <)$ defined by:

$$\begin{gathered} \mathcal{G}_0^n = G^n \\ \mathcal{G}_1^n = \big\{(x,y) \mid x \in \mathcal{G}_0^n, y \in \{d_1, \ldots, d_n\}\big\} \\ \partial_1(x,y) = x +_{\mathcal{G}_0} (x +_{G^n} y) \end{gathered}$$

The abstract complex $\mathcal{G}^n$, which is simply the Cayley graph of G^n, is not finite but locally-finite. The strategy used in MGS to paste the result of a simple transformation into the collection c is very simple: only the values of the chains are allowed to change, there is no change in c_b.

5 Examples

The following examples are freely inspired by examples given for Gamma, P systems and L systems and term rewriting.

Erastothene's Sieve on a Set. The idea is to generate a set with integers from 2 to N (with rules *Generate* and *Succeed*) and to replace an x and an y such that x divides y by x (rule *Eliminate*). The result is the set of the prime integers less than N.

$$\begin{array}{rcl} \texttt{trans}\ \mathit{Generate} & = & \{x, \mathit{true}\}\ =>\ x, \{x+1, \mathit{true}\}; \\ \texttt{trans}\ \mathit{Succed} & = & \{x, \mathit{true}\}\ =>\ x; \\ \texttt{trans}\ \mathit{Eliminate} & = & (x, y\ /\ y\,\texttt{mod}\,x = 0)\ =>\ x; \end{array}$$

With these definitions, the expression

$$\mathit{Eliminate}[\texttt{fixrule}]\Big(\mathit{Succed}\big(\mathit{Generate}[N](\{2, \mathit{true}\}, \texttt{set} : ())\big)\Big)$$

computes the primes up to N. The expression $(a, \texttt{set} : ())$ build a set by joining the element a to the empty set $\texttt{set} : ()$. So the expression $\mathit{Generate}[N](\{2, \mathit{true}\}, \texttt{set} : ())$ applies N times the transformation *Generate* to a singleton. The transformation *Succed* is applied only one times and then transformation *Eliminate* is applied until a fixpoint is reached.

Sorting a Sequence. A kind of bubble-sort is immediate:

$$\texttt{trans}\ \mathit{Sort}\ =\ (x, y\ /\ y < x)\ =>\ y, x;$$

(This is not really a bubble-sort because swapping of elements can take at arbitrary places; hence an out-of-order element does not necessarily bubble to the top in the characteristic way.)

Eratosthene's Sieve on a Sequence. The idea is to refine the previous algorithm using a sequence. Each element i in the sequence corresponds to the previously computed ith prime P_i and is represented by a record $\{\mathit{prime} = P_i\}$. This element can receive a candidate number n, which is represented by a record $\{\mathit{prime} = P_i, \mathit{candidate} = n\}$. If *candidate* is divisible by the stored number *prime*, (rule *Test1*), then the candidate number is deleted. If the candidate number passes the test (rule *Test2*), then the element transforms itself into a record $r = \{\mathit{prime} = P_i, \mathit{ok} = n\}$. If the right neighbor of r matches $\{\mathit{prime} = P_{i+1}\}$ without a field *candidate* nor *ok*, then the candidate n skips from r to the right neighbor. When there is no right neighbor to r, then n is prime and a new element is added at the end of the sequence. The first element of the sequence is distinguished (it is just an integer, not a record) and generates the candidates.

$$\begin{array}{l} \texttt{trans}\ \mathit{Eratos} = \{ \\ \qquad \begin{array}{rcl} \mathit{Genere1} & = & n : \mathit{integer}\ /\ \tilde{}\texttt{right}\ n \\ & & \quad =>\ n,\ \{\mathit{prime} = n\}; \\ \mathit{Genere2} & = & n : \mathit{integer}, \{\mathit{prime}\ \texttt{as}\ x, \tilde{}\mathit{candidate}, \tilde{}\mathit{ok}\} \\ & & \quad =>\ n+1,\ \{\mathit{prime} = x, \mathit{candidate} = n\}; \end{array} \end{array}$$

$$
\begin{array}{rcl}
\mathit{Test1} & = & \{\mathit{prime}\ \texttt{as}\ x, \mathit{candidate}\ \texttt{as}\ y, \tilde{}\mathit{ok}\}\ /\ y\ \texttt{mod}\ x = 0 \\
 & & =>\ \{\mathit{prime} = x\}; \\
\mathit{Test2} & = & \{\mathit{prime}\ \texttt{as}\ x, \mathit{candidate}\ \texttt{as}\ y, \tilde{}\mathit{ok}\}\ /\ y\ \texttt{mod}\ x <> 0 \\
 & & =>\ \{\mathit{prime} = x, \mathit{ok} = y\}; \\
\mathit{Next} & = & \{\mathit{prime}\ \texttt{as}\ x1, \mathit{ok}\ \texttt{as}\ y\}, \{\mathit{prime}\ \texttt{as}\ x2, \tilde{}\mathit{ok}, \tilde{}\mathit{candidate}\} \\
 & & =>\ \{\mathit{prime} = x1\}, \{\mathit{prime} = x2, \mathit{candidate} = y\}; \\
\mathit{NextCreate} & = & \{\mathit{prime}\ \texttt{as}\ x, \mathit{ok}\ \texttt{as}\ y\}\ \texttt{as}\ s\ /\ \tilde{}\texttt{right}\ s \\
 & & =>\ \{\mathit{prime} = x\}, \{\mathit{prime} = y\}; \\
\} & &
\end{array}
$$

Each rule has a name, and some rule applications are illustrated in figure 12. The function `left` (resp. `right`) gives the left (resp. right) neighbor of its argument, if it exists, or else the undefined value. Thus, this transformation can be applied only to topological collection which have a defined left and right neighborhood relation. The expression

$$\mathit{Erasto}[N]((2, \texttt{seq} : ()))$$

executes N steps of the Erastothene's sieve. For instance $\mathit{Erasto}[100]((2, \texttt{seq} : ()))$ computes the sequence: 42, $\{\mathit{candidate} = 42, \mathit{prime} = 2\}$, $\{\mathit{ok} = 41, \mathit{prime} = 3\}$, $\{\mathit{prime} = 5\}$, $\{\mathit{prime} = 7\}$, $\{\mathit{prime} = 11\}$, $\{\mathit{prime} = 13\}$, $\{\mathit{ok} = 37, \mathit{prime} = 17\}$, $\{\mathit{prime} = 19\}$, $\{\mathit{prime} = 23\}$, $\{\mathit{prime} = 29\}$, $\{\mathit{prime} = 31\}$, $\texttt{seq} : ()$.

The game of life. The game of life is a special kind of cellular automata. A cell of the cellular automaton (a vertex of the corresponding topological collection) takes one of the two values 0 and 1. The evolution of this value depends on the values of the neighbors (if the sum of the neighbor's value is between two given level, the current state is set to 1 and else it is set to 0). The corresponding `MGS` program is the following. It begins by the declaration of a new topological collection type:

$$\texttt{gbf}\ \mathit{Grid2}\ \ =\ \ < X, Y >$$

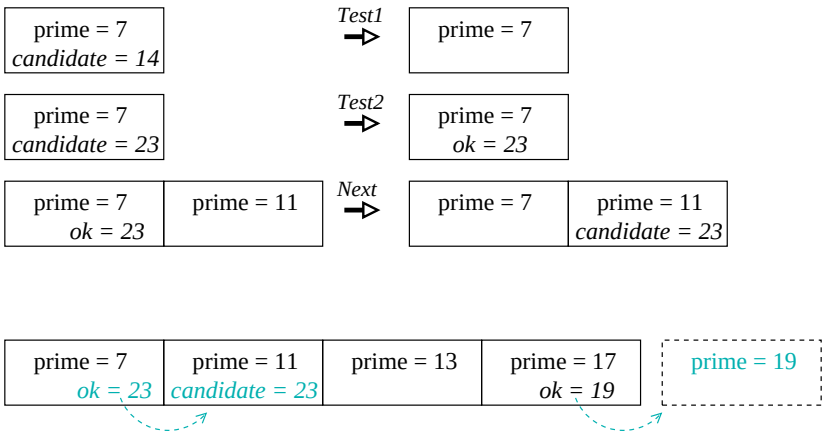


Figure 12: The *Eratos* program. Some rule instantiations and a fragment of the sequence built by the transformation *Eratos*.

this statement declares a new collection type, based on the group based field topology described in section 4.2, with an X and an Y neighborhood relation. In this case, this declaration simply specify the topology of an infinite grid with two dimensions named X and Y. The evolution function of the cellular automata is given by the transformation:

$$\texttt{trans}\ \textit{evolve} = \quad x \quad => \quad \texttt{let}\ s = \texttt{FoldNeighbors}((\backslash a, b.\, a + b),\ 0,\ x)$$
$$\texttt{in}\ \ \texttt{if}\ (s < 3)\ \texttt{or}\ (s > 4)\ \ \texttt{then}\ 0\ \ \texttt{else}\ 1\ \ \texttt{fi}$$

the function $\texttt{FoldNeighbors}(f, e, x)$ makes a fold between the values of the neighbors of x with the binary function f and the initial value e (f is supposed to be an associative-commutative function with neutral element e). The operator `FoldNeighbors` is applicable in all topology (in a set it gives all the elements in the set, in a sequence it gives the considered element together with its left and right neighbors, etc.).

6 Summary and Final Remarks

We have shown in section 2 that most of the notions used to describe P systems (membrane structures, local computations, moves between adjacent membranes) find a natural setting and a smooth extension in the framework provided by topological notions developed in the field of homology theory.

We have defined a topological collection c to be a chain on a given chain complex that describes the topology of the collection and a labeling of the cells. A simple transformation replaces a subchain c' by another subchain, preserving the topological structure of the complement of c' in c.

This abstract view enables the unification in a same programming language of several biologically or biochemically inspired computational models, namely: Gamma and the CHAM, P systems, L systems and cellular automata. These models can be rephrased as the iteration of simple transformations on a topological collection; the difference coming from the topology of the collection (section 4). However, we do not claim that we have achieved a useful theoretical framework encompassing the four cited formalisms. We advocate that few notions and a single syntax can be consistently used to allow the merging of these formalisms *for programming* purposes.

It leads to the development of an experimental programming language called `MGS`. `MGS` is a vehicle used to investigate general notions of collections and transformations and to study their adequacy to the simulation of various biological processes. Simple examples of `MGS` programs are given in section 5. All examples are processed using the current version of the `MGS` interpreter.

Currently, two versions of an `MGS` interpreter exist: one written in `OCAML` (a dialect of `ML`) and one written in `C++`. There are some slight differences between the two versions. For instance, the `OCAML` version is more complete with respect to the functional part of the language. These interpreters are freely available[2]. In these current `MGS` implementations, sets, multisets, sequences and group based fields (which generalize functional arrays) of elements are supported. The elements in a collection can be any kind of values: basic types, records or arbitrary nesting of collections. The values of the record's fields are also of any kind, thus achieving complex objects in the sense of [3].

[2] see `http://www.lami.univ-evry.fr/mgs`

The interested reader will find in [10] a more complete presentation of the language. The technical report [11] gives more details on the topological formalization of collections and transformations. As a matter of fact, we have simplified the presentation given here. For instance, for the sake of the simplicity, we have restricted ourself to avoid the dual notions of cochains and coboundaries. However, this is the right general formal setting to fully develop the notion of topological collection.

The report [11] also develops several examples of `MGS` programs (the tokenization of a sequence of letters, the computation of the convex hull of a set of points in $\mathbb{R}^3$, the computation of the maximal segment sum, a Turing diffusion-reaction process, a grow model of cellular tissues, the computation of a disjonctive normal form of a set of clauses represented as nested sets, etc.).

At the language level, the study of the topological collections concepts must continue with a finer study of transformations. Several kinds of restriction can be put on the transformations, leading to various kind of pattern languages and rules. The complexity of matching such patterns has to be investigated. We also want to develop a type system that can handle nested collections, along the lines developed in [2]. At last but not least, we want to know if the topological spaces built by transformations can be characterized through a non standard type system. We also begin the study of a generic implementation of topological chain complex, based on the G-map data structure [15] to represent arbitrary join/neighborhood relationships. The efficient compilation of a `MGS` program is a long-term research effort.

The applications opened by this preliminary work are numerous. From the applications point of view, we are challenged by the simulation of the topological changes at the early development of the embryo. This is an actual example of tissues formation and fusion requiring complex topology beyond what is accessible using simple data-structures. Another motivating application is the case of a spatially distributed biochemical interaction networks, for which some extension of rewriting have been advocated, see [4, 16].

Acknowledgments

The authors would like to thanks the members of the "Simulation and Epigenesis" group at Genopole for fruitful discussions and biological motivations. They are also grateful to F. Delaplace and J. Cohen for numerous challenging questions and useful comments. The friendly atmosphere of WMC'01 has raised many stimulating questions that have greatly improved an earlier version of this paper and suggested many future developments. This research is supported in part by the CNRS, the GDR ALP, IMPG and Genopole/Evry.

References

[1] Banatre, J. P., Metayer, D. L.: *A new computational model and its discipline of programming*, Technical Report RR-0566, Inria, 1986.

[2] Blelloch, G.: *NESL: A nested data-parallel language (version 2.6)*, Technical Report CMU-CS-93-129, School of Computer Science, Carnegie Mellon University, April 1993.

[3] Buneman, P., Naqvi, S., Tannen, V., Wong, L.: Principles of programming with complex objects and collection types, *Theoretical Computer Science*, **149**(1), 18 September 1995, 3–48.

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

[5] Fontana, W.: Algorithmic Chemistry, *Proceedings of the Workshop on Artificial Life (ALIFE '90)* (C. G. Langton, C. Taylor, J. D. Farmer, S. Rasmussen, Eds.), 5, Addison-Wesley, Redwood City, CA, USA, February 1992, ISBN 0-201-52570-4.

[6] Fontana, W., Buss, L.: "The Arrival of the Fittest": Toward a Theory of Biological Organization, *Bulletin of Mathematical Biology*, 1994.

[7] Fontana, W., Buss, L.: *Boundaries and Barriers, Casti, J. and Karlqvist, A. edts,*, chapter The barrier of objects: from dynamical systems to bounded organizations, Addison-Wesley, 1996, 56–116.

[8] Giavitto, J.-L.: A framework for the recursive definition of data structures., *Proceedings of the 2nd Imternational ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-00)*, ACM Press, September 20–23 2000.

[9] Giavitto, J.-L., Michel, O.: Declarative definition of group indexed data structures and approximation of their domains., *Proceedings of the 3nd Imternational ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-01)*, ACM Press, September 2001.

[10] Giavitto, J.-L., Michel, O.: MGS: a Rule-Based Programming Language for Complex Objects and Collections, *Electronic Notes in Theoretical Computer Science* (M. van den Brand, R. Verma, Eds.), 59, Elsevier Science Publishers, 2001.

[11] Giavitto, J.-L., Michel, O.: *`MGS:` a Programming Language for the Transformations of Topological Collections*, Technical Report 61-2001, LaMI – Université d'Évry Val d'Essonne, May 2001.

[12] Giavitto, J.-L., Michel, O., Sansonnet, J.: Group-Based Fields, *Parallel Symbolic Languages and Systems (Int. Workshop PSLS'95)*, LNCS 1068, Springer, 1996.

[13] Henle, M.: *A combinatorial introduction to topology*, Dover publications, New-York, 1994.

[14] Hoogendijk, P. F., Backhouse, R. C.: Relational Programming Laws in the Tree, List, Bag, Set Hierarchy, *Science of Computer Programming*, **22**(1–2), April 1994, 67–105.

[15] Lienhardt, P.: Topological models for boundary representation : a comparison with n-dimensional generalized maps, *Computer-Aided Design*, **23**(1), 1991, 59–82.

[16] Manca, V.: Logical string rewriting, *Theoretical Computer Science*, **264**, 2001, 25–51.

[17] Munkres, J.: *Elements of Algebraic Topology*, Addison-Wesley, 1984.

[18] Norris, V., Fralick, J., Danchin, A.: A *SeqA* hyperstructure and its interactions direct the replication and sequestration of DNA, *Molecular Microbiology*, **37**, 2000, 696–702.

[19] Palmer, R. S., Shapiro, V.: Chain Models of Physical Behavior for Engineering Analysis and Design, *Research in Engineering Design*, **5**, 1993, 161–184, Springer International.

[20] Paun, G.: Computing with Membranes: An Introduction, *Bulletin of the European Association for Theoretical Computer Science*, **67**, February 1999, 139–152.

[21] Paun, G.: From Cells to Computers: Computing with Membranes (P systems), *Biosystems*, **59**(3), March 2001, 139–158.

[22] Paun, G., Sakakibara, Y., Yokomori, T.: P Systems on Graphs of Restricted Forms, *Publ. Math. Debrecen*, 2001, (to appear).

[23] Rozenberg, G., Salomaa, A.: *Lindenmayer Systems*, Springer, Berlin, 1992.

[24] Tonti, E.: The algebraic-topological structure of physical theories, *Symmetry, similarity and group theoretic methods in mechanics* (P. G. Glockner, M. C. Sing, Eds.), Calgary, Canada, August 1974.

[25] Tonti, E.: The reason for analogies between physical theories, *Appl. Math. Modelling*, **1**, June 1976, 37–50.

[26] Von Neumann, J.: *Theory of Self-Reproducing Automata*, Univ. of Illinois Press, 1966.