Interaction Based Simulation of Dynamical System with a Dynamical Structure $(\mathsf{DS})^2$ in MGS

Jean-Louis Giavitto[a], **Olivier Michel**[b], **Antoine Spicher**[b]

[a] **IRCAM - UMR STMS 9912 CNRS, 1 place Igor Stravinsky, 75004 Paris, France**
jean-louis.giavitto@ircam.fr

[b] **LACL, University of Paris-Est Créteil, 61 avenue du Général de Gaulle, 94010 Créteil Cedex, France**
{olivier.michel,antoine.spicher}@u-pec.fr

Keywords: topological collection, topological rewriting, domain specific programming language, cellular complex, topological chain, dynamical systems with a dynamical structure $(\mathsf{DS})^2$, MGS domain specific programming language.

Abstract

We present the domain specific programming language MGS and its approach to the specification of dynamical systems with a dynamical structure or $(\mathsf{DS})^2$. MGS stands for "encore un Modèle Général de Simulation", that is, "yet another general model of simulation". Its declarative approach is based on the notions of chains and cochains well studied in algebraic topology. A careful discussion of the design goals lead us to relax some of the constraints on these mathematical structures to represent in a uniform way various data structures and transformations. In particular, our computational notion of transformation relies on a rewriting mechanism encompassing the usual notions of set, string and term rewriting. These notions are illustrated on two examples involving the implicit computation of a time varying neighborhood: the simulation of the trajectories of flocking birds and the growth of an epithelial tissue. The second example illustrates also the compositionality achieved by the declarative framework. The MGS concepts have been further validated on several large scale simulations of complex biological systems.

1. INTRODUCTION

$(\mathsf{DS})^2$. The use of *dynamical systems* is pervasive in simulation. At any point in time, a dynamical system is characterized by a set of state variables. The evolution of the state over time is specified through a transition function or relation which determines the next state of the system (over some time increment) as a function of its previous state and, possibly, the values of external variables (input to the system). We suppose that the system evolves in discrete time and that the set of variables is finite because models initially formulated in terms of continuous time and/or continuous set of variables (*e.g.*, partial differential equations) are usually discretized for their simulation on a computer.

This description outlines the change of state in time but does not stress that the set of state variables can also change in time. Systems exhibiting a change in the set of state variables have a *dynamical structure*. Note that if the set of state variables evolves in time, so does the transition function. We qualify such systems as $(\mathsf{DS})^2$: *dynamical systems with a dynamical structure* [11].

Computer Science has developed (or appropriated) many languages and tools to help model and simulate dynamical systems. However, the dynamic character of the structure raises a difficult problem: how to define a transition function when its set of arguments (the state variables) is not completely known at the specification time? The answer to this problem lies in the notions of *interaction* and *locality*.

Spatial Organization of the Interactions. Very often, a system can be decomposed into subsystems and the advancement of the state of the whole system results from the advancement of the state of its parts [12]. The change of state of a part can be intrinsic (*e.g.*, because of the passing of time) or extrinsic, that is, caused by some interaction with some other parts of the system.

For physical systems, subsystems are spatially localized and when a locality property[1] holds, only subsystems that are neighbors in space can interact directly. So the interactions between parts are structured by the spatial relationships of the parts.

For abstract systems, in many cases the transition function of each subsystem only depends on the state variables of a small set of parts (and not on the state variables of the whole system). In addition, if a subsystem s interacts with a subset $S = \{s_1, \ldots, s_n\}$ of parts, it also interacts with any subset S' included in S. This closure property induces an abstract spatial structure on the set of parts: the set of parts can be organized as an *abstract simplicial complex* [15].

So, the idea is to describe the global dynamics by summing up the local evolutions triggered by local interactions. And two subsystems s and s' do not interact because they are identified *per se* but because they are neighbors. Such a fea-

[1]The locality property states that matter/energy/information transmissions are done at a finite speed. This property is not always relevant, even for physical systems, for instance because processes may occurs at two separate time scales: changes at the fast time scale may appear instantaneous with respect to the slow one.

ture enables the potential interaction of subsystems that do not yet exist at the beginning of the simulation and that do not know each other at their creation time.

The Chain Approach. It is then tempting to provide a topology (*i.e.*, neighborhood relationships) to the set of subsystems and to identify the state of a dynamical system with a function that assigns a local state to each subsystem. The topology restricts the possible transition functions of a subsystem s: the current state of s only depends on the previous state of s and of its neighbors. Furthermore, the transition function not only specifies the evolution of local states but also the coupled evolution of the topology itself.

Such a structure is well known in algebraic topology: the state can be represented by a *topological chain* that associates some label with each *topological cell* of a *cellular complex*. An (abstract) cellular complex is a formal construction that builds a space in a combinatorial way through more simple objects called topological cells. Each cell represents a simple part of the whole space. The whole structure, corresponding to the partition into topological cells, is then considered through the *incidence relationships*, relating a cell and the cells in its boundary. A topological chain is a function from a cellular complex to a set of labels equipped with some additional algebraic structure [20].

Over the last forty years, there have been notable efforts to develop comprehensive formulations of physics and geometry based on topological chains [5, 30, 23, 8, 19]. We review some of them in the last section. However, the emphasis has been mainly put on uniform and homogeneous interactions that can be specified by discrete analogues of the differential operators. Such approach cannot be applied to the modeling of agent systems where the interactions between agents are heterogeneous and asynchronous. Moreover they cannot represent all the possible evolutions of the structure and these evolutions are often handled separately, in an imperative settings.

In this paper, we show how the transition function specifying the coupled evolution of state and topology can be defined in a declarative style, using *topological collection rewriting*. *Topological collections* implement various notions of topological chains and *topological rewriting* encompasses both homogeneous and heterogeneous interactions. The declarative style leads to a specification of the systems close to the mathematical formalism: the description is small and expressive, theoretically well founded and close to the concepts used by the modelers. Furthermore, the declarative style is more tractable when one faces the problem of formally deriving some properties of the simulated model [21].

Organization of the paper. The notions presented in this paper are implemented in an experimental domain specific programming language, MGS, designed to investigate the topological approach.

The next section introduces topological collections. This construction unifies all aggregate data structures in MGS. The notion of topological rewriting is then presented in section 3.

These notions are illustrated through two applications exhibiting a dynamic structure: the flock of birds (the neighborhood of each bird evolves in time as a consequence of the flock movement and the trajectory of each bird depends on this neighborhood) and the growing of an epithelial sheet of biological cells (the number of cells and their spatial organization evolve in time). These two examples rely on an unusual kind of topological collection where the neighborhood is implicitly and dynamically computed instead of being explicitly specified.

The paper ends with a conclusion and a discussion about future works.

2. TOPOLOGICAL COLLECTIONS

Topological collections have been introduced in [13] to describe arbitrary complex spatial structures that appear in biological systems [14] and other dynamical systems with a time varying structure [16, 17].

Incidence Structures. An *abstract combinatorial complex* (ACC) $K = (\mathsf{C}, \prec, dim)$ is a set C of abstract elements, called *topological cells*, provided with a partial order $\prec$, called the *boundary relation*, and with a *dimension* function $dim : \mathsf{C} \to \mathbb{N}$ such that for each c and c' in C, $c \prec c' \Rightarrow dim(c) < \dim(c')$. We write $c \in K$ when a cell c is a cell of C.

A cell of dimension 0 corresponds to a point, a 1-dimensional cell corresponds to a line (an edge), a cell of dimension 2 is a surface (*e.g.* a polygon), etc. A cell of dimension p is called a p-cell. For example, a graph is an ACC built only with 0- and 1-cells. An other example is pictured in Fig. 1. Such structures are studied in algebraic topology [20].

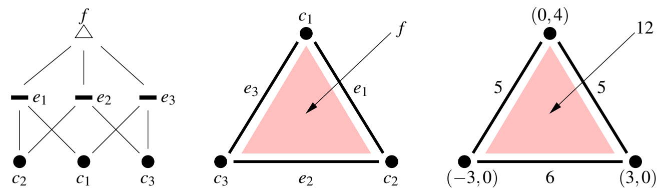

Figure 1. On the left, the Hasse diagram of the boundary relationship of the ACC given in the middle: it is composed of three 0-cells (c_1, c_2, c_3), of three 1-cells (e_1, e_2, e_3) and of a single 2-cells (f). The three edges are the faces of f, and therefore f is a common coface of e_1, e_2 and e_3. On the right, a topological collection associates data with the cells: positions with vertexes, lengths with edges and area with f.

In the context of this paper, we write $\partial_K c$ for the sub-ACC made of the cells of K lower than c for the relation $\prec$: $\partial_K c = (\mathsf{C}', \prec \cap\, \mathsf{C}' \times \mathsf{C}', dim)$ where $\mathsf{C}' = \{c' \,|\, c' \prec c\}$. This ACC is called the *boundary* of c. The *faces* of a p-cell c are the $(p-1)$-cells c' of $\partial_K c$ and we write $c > c'$ or $c' < c$; c' is called a *coface* of c. Two cells c and c' are *q-neighbors* either if they share a common border of dimension q or if they are in the boundary of a q-cell (of higher dimension).

Topological Collections. The next step is to attach a value to each cell of a complex. Algebraic topology takes this value in a commutative group since it gives a natural group structure to the set of chains [20]. We relax this assumption for topological collection: a topological collection C is a function that associates a value from an arbitrary set V with cells in an ACC, see Fig. 1. Thus the notation $C(c)$ refers to the value of C on cell c.

Values associated with cells in the cell complex can be used to capture geometric properties (*i.e.*, $V = \{-1, 0, 1\}$ can be used to select a sub-complex made of oriented cells) or to represent the arbitrary state of a subsystem (a mass, a concentration of chemicals, or a force acting on certain cells).

We write $|C|$ for the set of cells for which C is defined. The collection C can be written as a formal sum $\sum_{c \in |C|} v_c \cdot c$ where $v_c \overset{\text{df}}{=} C(c)$. With this notation, the underlying ACC is left implicit but can usually be recovered from the context. By convention, when we write a collection C as a sum

$$C = v_1 \cdot c_1 + \cdots + v_p \cdot c_p$$

we insist that all c_i are distinct. This notation is directly used in MGS to build new topological collections on arbitrary ACC of any dimension. Notice that this addition is associative and commutative: the order of operations used to build a topological collection is irrelevant.

In MGS, topological collections correspond to aggregate data types. These data types differ by the specification of their underlying cellular complex. In the current implementation of the MGS language, usual data structures (records, sets, sequences, trees, arrays, etc.) are represented by special kinds of one-dimensional topological collection, namely vertex-labeled graphs: elements of the data structure are attached to the vertexes and the edges represent the relative accessibility from one element to another in the data structure. MGS also handles more sophisticated spatial structures corresponding to arbitrary ACC of any dimension.

Examples. We detail the topology underlying some usual data structures to show how they can be seamlessly embedded in the topological collection framework.

The topology of sets is the complete graph: each element in the set labels a 0-cell (the cell labeled by a value v is simply v itself) which is in the neighborhood of all the other cells. With this neighborhood structure, topological rewriting (presented in the next section) of sets corresponds to the usual notion of set rewriting.

Records (`C` struct) are another example of collection topology: a dictionary associating values with names. The underlying topology is the fully unconnected graph: the fields of the record form the vertexes of the graph and there is no edge. A record r can be declared using a dedicated syntax

```
{ Area = 10; Speed = [0.0; 1.0; 0.33]; }
```

and the usual dot notation $r.a$ to access the value of the field a can be used as an alternative to the application $r(a)$. The curly bracket notation is a shorthand for the additive expression:

```
10·Area + [0.0; 1.0; 0.33]·Speed
```

where `Area` and `Speed` are the symbols representing the name of the fields. Whereas the addition of collections is restricted in general to arguments that do not share common cells, addition of records sharing some fields is allowed: the expression $r_1 + r_2$ computes a new record r having the fields of both r_1 and r_2: $r(a)$ has the value of $r_2(a)$ if the field a is present in r_2, otherwise it has the value of $r_1(a)$. In this way, the addition of records has the usual semantic of the *asymmetric* merge of records [26] used in object-oriented programming to model inheritance and object updates.

The value of the field `Speed` is a list of three elements. Lists are also topological collections (this illustrates the possibility to nest arbitrary topological collections in MGS). The dedicated syntax, using square bracket, hide the underlying topology of linear oriented graph. Orientation of the cells is taken into account so that in `[1; 2; 3]` the neighbor of the cell labeled by `1` is the cell c labeled by `2` but the only neighbor of c is the cell labeled by `3`. With this topology, topological rewriting of lists is similar to the usual notion of string rewriting.

3. TOPOLOGICAL REWRITING

The next move is to define a suitable notion of topological collection transformation. As mentioned in the introduction, the transformation of a topological collection must be able to express changes in the labels as well as changes in the underlying spatial structure.

Chains and their Shortcomings. It exists in homology theory a notion of chain transformation called *cochains*. A cochain is a group homomorphism from the group of chains to a target group G. Since a cochain d is a group homomorphism, d is completely defined by the values taken on the labeled cell $v_i \cdot c_i$:

$$d(v_1 \cdot c_1 + \cdots + v_n \cdot c_n) = d(v_1 \cdot c_1) + \cdots + d(v_n \cdot c_n)\,.$$

If d is real-valued cochain on real-valued chains, then $d(v_i \cdot c_i)$ can be further rewritten in $v_i.d_i$ where d_i is the value of d on the unit chain $1.c_i$. Therefore d can be defined as a formal sum: $d = \sum g_i.c_i$ where $g_i \overset{\text{df}}{=} d(c_i)$. Thus every real-valued chains can be also viewed as a real-valued cochain. Cochains provide an elegant algebra where discrete analogues of the classical differential operators (boundary, coboundary, Hodge star…) can be defined [7, 19].

We want to lift this approach in MGS. However the notion of cochains presents three drawbacks: (1) topological collections do not have always a group structure; (2) they are not always real-valued; and (3) some natural transformations are not homomorphisms. An example of the last point is given by the merge of two cells into a new one iff they are labeled by the same value in the chain. Let M be this transformation, then $M(v\cdot c_1 + v\cdot c_2) = v\cdot c' \neq M(v\cdot c_1) + M(v\cdot c_2) = v\cdot c_1 + v\cdot c_2$. Nevertheless, a suitable notion of transformation must include the notion of cochain and must respect, in some way, the additive structure of chains.

Topological Rewriting. To meet the previous requirements, we have proposed to define transformation of topological collection by rewriting rules. Topological rewriting can be defined following an approach similar to that taken in [24]: using the additive representation of topological collections, topological rewriting can be simply defined as an adapted version of conditional first-order associative-commutative term rewriting, see [28] for the details.

The mechanics of rewriting systems are familiar to anyone who has done arithmetic simplifications: an arithmetic expression can be simplified by repeatedly replacing parts of the expression (*subexpressions*) with other subexpressions. For example, $\frac{7}{3}\cdot\frac{3}{11}\cdot\frac{11}{5} \Rightarrow \frac{7}{11}\cdot\frac{11}{5} \Rightarrow \frac{7}{5}$. The rule that is applied here is: $\frac{x}{y}\cdot\frac{y}{z} \Rightarrow \frac{x}{z}$, where x, y and z are pattern variables representing arbitrary non-null numbers. A transformation generalizes this process to topological collections.

A transformation T is a function specified by a set of rewriting rules $\{p_1 \Rightarrow e_1, \ldots, p_n \Rightarrow e_n\}$ where each p_i is a pattern and each e_i is an expression. An application of such a rule matches a sub-collection with one p_k that is then substituted by the result of expression e_k. In rewriting rules, patterns match sub-expressions, that is, partial sums of the whole sum representing the topological collection which the rule is applied on. It is in this sense that the additive structure of topological collections is preserved (but a transformation is not necessarily an homomorphism).

The formal definition of topological rewriting is less interesting than the syntax of the pattern language used to specify the left hand side (lhs) of a rewriting rule: as a matter of fact, the lhs of a rule must match a sub-collection, that is a subset of C and a sub-relation of the incidence relation $\prec$ of the complex K. This information can be difficult to specify without the help of a dedicated language. We present here a fragment of the MGS *path pattern language*.

Pattern Variables. A pattern variable x matches a cell and its label. The identifier x can be used elsewhere in the rule to refer to the label of the matched cell; the cell itself can be referred through the special identifier $\hat{}x$. This convention avoids the introduction of two identifiers to match a cell and its associated value. Using the additive notation for topological collections, and without the previous convention, this pattern is translated to $x\cdot\hat{}x$ where the variable x ranges over the labels, and where the variable $\hat{}x$ ranges over the cells.

Patterns are *linear*: two distinct pattern variables always refer to two distinct cells.

Conditional rules. A *guard* can be used to specify a condition that must be satisfied by the matching. For instance, expression $x/x > 5$ matches a cell $\hat{}x$ labeled by an integer x greater than 5.

Pattern Composition. The associative operator "," is used to specify a path, *i.e.*, a sequence of elements. A comma implies also some constraints on the incidence relationships linking the two arguments: in the additive notation, the pattern v, w translates to the conditional pattern

$$v\cdot\hat{}v + w\cdot\hat{}w \;/\; \exists\hat{}u :\; (\hat{}v \prec \hat{}u \wedge \hat{}w \prec \hat{}u) \vee (\hat{}u \prec \hat{}v \wedge \hat{}u \prec \hat{}w)$$

In other words, the cells matched by v and w must be neighbors (they share a common cell u in their boundary or they are both elements of the boundary of some cell u).

Notice that the pattern composition operator preserves a notion of locality: only cells that are neighbors are selected.

Strategies. Rule applications are controlled through a *rule application strategy*. Several strategies are available in MGS like the maximal parallel application used in *L-systems* or *P systems*, and the Gillespie stochastic simulation algorithm used in the simulation of chemical reactions [27]. These strategies control the advancement of time in the simulation (synchronous, asynchronous, stochastic, etc.). They are often non-deterministic, *i.e.*, applied on a collection C, only one of the possible outcomes (randomly chosen) is returned by the transformation.

4. APPLICATIONS

Topological collections and topological rewriting are directly supported in the MGS domain specific programing language. We present two examples to illustrate the MGS approach in the simulation of (DS)^2. In the first example, a simulation of *flocking birds*, there is no creation nor destruction of birds, but the neighborhood structure changes in time with the move of the birds. This example introduces the notion of Delaunay collections where the neighborhood structure is not built explicitly by the programmer but is computed implicitly by the run-time using the labels of 0-cells in the collection. The second examples relies also on Delaunay collection for the simulation of the *growth of an epithelial tissue* driven by a diffusion-reaction process and taking into account a simple

```
trans behavior[speed] = {
  separation = ... / ... => ... ;
  cohesion   = ... / ... => ... ;
  alignment  =
    a => let phi = neighborsfold(\x,acc.(acc+x.theta), 0, a)
         and nb  = neighborsfold(\x,acc.(acc+1), 0, a) in
         let dir = phi / nb in
           { x     = a.x + speed*cos(dir) + random(ε),
             y     = a.y + speed*sin(dir) + random(ε),
             theta = dir } · ^a;
}
```

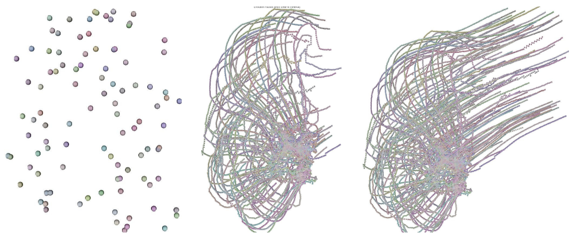

Figure 2. Left: Behavior of a bird expressed as a transformation with three rules. Right: Trajectory of a flock of 50 birds. First plot: the initial state where each bird has a randomly chosen direction. Middle plot: the configuration after 300 iterations. Last plot: after 900 iterations of the transition function `behavior`.

model of the mechanics of the tissue. This second example includes the creation of 0-cells.

4.1. The Flocking of Birds

In this example, we simulate the gathering and the displacement of a flock of birds. This simulation is the direct implementation of a model of flocking birds proposed by U. Wilensky and by the development of steering behaviors of boids (generic simulated flocking creatures) invented by C. Reynolds [25].

Whereas there is no leader (each bird obeys to the same set of rules), all the birds seem to follow the same direction. This global motion can be modeled using three local rules for a bird: (1) not to collide with neighbors, (2) to join the group when it is too far and (3) finally to head for the same global direction as its neighbors. They will be detailed later. We first begin by the description of a bird.

Representation of Birds and of the Flock. A bird is represented by two quantities: its position and its direction in space. To simplify the example, we consider a two dimensional model, knowing that the three dimensional representation is a trivial generalization. We use a record composed of three fields: two for the position (the coordinates `x` and `y`), and one for the motion direction (the angle `theta` from the x-axis). We suppose that all birds move with the same speed that is increased if the bird is too far from the group. Let define the record type `Bird`:

```
record Bird = { x, y, theta }
```

A natural neighborhood between birds can be generated using a Delaunay triangulation. This neighborhood induces the definition of a new type of collection used to represent the flock of birds. It is generated from a set of birds (the vertexes) and a function extracting the bird position information. The edges between these vertexes are not explicitly given by the programmer but computed and maintained transparently by the run-time using a Delaunay triangulation [1] of the positions.

Thus, the new type of collection `Flock` based on a Delaunay graph whose elements are values of type `Bird` is specified as follows:

```
delaunay Flock(b:Bird) = [b.x; b.y]
```

Birds Behavior. The basic flocking model consists of three simple steering local behaviors which describe how an individual bird maneuvers based on the positions and velocities of its nearby flockmates:

1. ***Separation***: when a bird is too close to one of its neighbors, it changes its direction.
2. ***Cohesion***: when a bird is too far from one of its neighbors, it tries to get closer by increasing its speed.
3. ***Alignment***: otherwise, it chooses a direction which is the average of the direction of its neighbors.

The transformation given at the left of Fig. 2 implements the behavior of the birds. A rule specifies the evolution of one bird. Only the rule `alignment` is presented here. The others have a similar form with an additional guards holding for the separation condition or for the cohesion condition. In the rule `alignment`, the average direction `dir` is computed by summing first the directions of the neighbor birds and their numbers.

These sums are specified using the higher-order function `neighborsfold`. The expression `neighborsfold`$(f, init, c)$ iterates a binary reduction function f over the labels of the neighbors of c to build up a return value. The argument *init* is used to initialize the accumulator. The notation `\`$\mathtt{x}_1, \ldots, \mathtt{x}_k$`.`*expr* is the MGS notation for an anonymous function (lambda abstraction) with arguments $\mathtt{x}_1, \ldots, \mathtt{x}_k$ and body *expr*. Then, the record `a` is updated: the bird position and direction are computed according to the average direction `dir`. Note that some noise is introduced to make the simulation less deterministic. Finally, this new value is associated with the cell matched by the left hand side, achieving an update of the (local) state of the bird.

The transformation applies the rules with a maximal parallel strategy and rule applications are prioritized following their declaration order. In other words, for each bird in parallel, MGS tries to apply the rule `separation` and if the guard does not hold, the rule `cohesion` is applied and if its guard does not hold, the rule `alignment` is applied. Right of Fig. 2 illustrates three iteration steps of this process.

4.2. The Growth of an Epithelial Tissue

In this example, we abstract individual biological cells[2] in a tissue by disks localized in a 2D Euclidean space. Cells push away each other and consequently change their positions in space and their immediate neighborhood. Thus, this neighborhood is required to be dynamically computed according to the position of the disks in the plane.

The state of a biological cell is encoded by a record which includes the required informations. The following type declarations

```
record MechaCell = { px, py, vx, vy, ax, ay }
record BioCell   = { a, b, da, db }
record Cell      = MechaCell + BioCell
```

specify three record types: `MechaCell` representing the 2D position, velocity and acceleration of a cell; `BioCell` representing the biological state of a cell with two diffusing chemicals (`a` and `b`) and their first respective derivatives; `Cell` contains the fields of both `MechaCell` and `BioCell`.

The spatial organization of the whole epithelial tissue is represented by a 2D Delaunay topological collection:

```
delaunay Tissue(c:Cell) = [c.px; c.py]
```

Such an use of the Delaunay neighborhood has already been successfully done in systems biology for the modeling of cells population [18, 2].

The specification of the system dynamics is based on three coupled models: a biomechanical model, a biochemical model and a cellular model.

A Biomechanical Model. The global behavior of the tissue is modeled by a spring-mass system based on elastic and viscous forces. The movement of a cell c is given by the Newton's equation of dynamics:

$$m.\mathrm{a}_c = F_{\text{elastic}} + F_{\text{viscous}} = \sum_{c',c} k(L_{cc'} - L_0)\frac{\mathrm{p}_{c'} - \mathrm{p}_c}{L_{cc'}} - \mu.\mathrm{v}_c \quad (1)$$

where p_c, v_c and a_c are respectively the position, the velocity and the acceleration of c, and c' represents a neighbor cell of c at a distance of $L_{cc'}$. The spring term depends on the constant k and the rest length L_0, and μ is a friction coefficient.

Eq. (1) is integrated for each cell by a Euler scheme implemented in the following transformation:

[2]The reader must pay attention not to confuse biological and topological cells.

```
trans Mechanics = {
  c => let Fvisc = { fx = -mu*c.vx, ... } in
       let F = neighborsfold(Felastic(c),Fvisc,c)
       in (c+{ ax = F.fx / m,
               vx = c.vx + dt*c.ax,
               px = c.px + dt*c.vx, ... }) · ^c
}
```

The `neighborsfold` expression computes the summation of Eq. (1); function `Felastic(`c, c', f`)` computes the elastic force between c and c' and sums it to an accumulator f. As a curryfied function, the first argument is specified while the others are provided by the `neighborsfold` operator.

The Biochemical Model. The biochemical model describes the diffusion-reaction of two morphogens modeled as a simplified version [35] of the Turing's diffusion-reaction model [34]. The evolution of the two chemicals is also implemented with only one transformation `DiffusionReaction`, using also a `neighborsfold` to compute the Laplacian of the concentrations.

The Cellular Model. An additional transformation defines the division of a cell when the concentration of `b` increases above a given threshold `split`:

```
trans Division = {
  c / (c.b > split) =>
    (c + { a = 1/3*c.a, b = 2/3*c.b,
           px = c.px + random(ε),
           py = c.py + random(ε) }) · newcell(0) +
    (c + { a = 2/3*c.a, b = 1/3*c.b,
           px = c.px + random(ε),
           py = c.py + random(ε) }) · newcell(0)
}
```

The coefficients used to compute the asymmetric repartition of chemical concentrations in the daughter cells are arbitrary. The two daughter cells occupy almost the same position (disturbed a little by the term `random(ε)`); the elastic forces will quickly push them away from each other. The two new cells states are associated with new topological 0-cells created by the primitive `newcell` (the additive notation is used). No further specification of the neighborhood (*e.g.*, some additional 1-cell between the vertexes) is needed since it will be computed automatically by the Delaunay triangulation.

Integration of the Three Models. In MGS, transformation are ordinary functions and can be arbitrarily composed. This is the key to the coupling of the 3 models. The iteration of a function can be specified by the MGS option `iter`. It allows to deal with different time scales: assuming for example that the mechanical process is faster than the chemical process, the whole model is captured by the following evolution function:

```
fun evolve(tissue) =
   Division(
      DiffusionReaction[dt=Δ1](
         Mechanics[dt=Δ2, iter=N](tissue)))
```

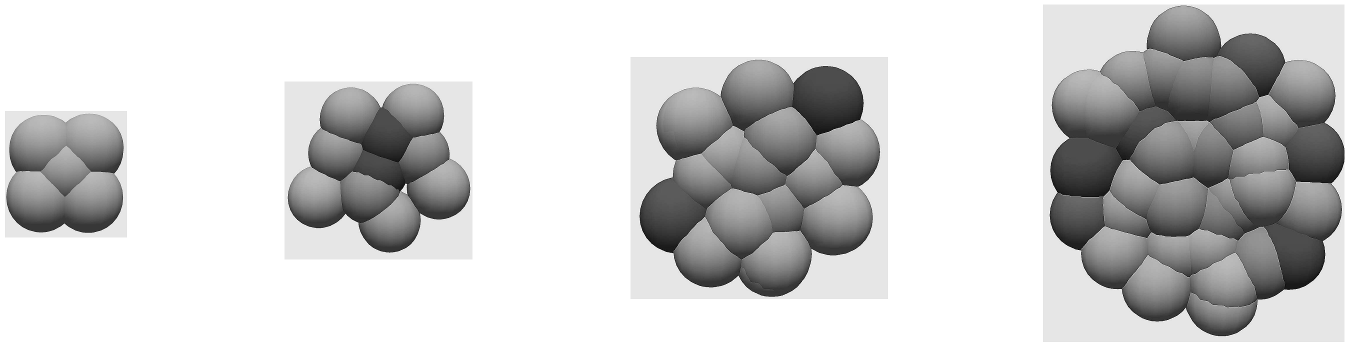

Figure 3. Four steps in the growth of a tissue of cells. The color of a cell is correlated with the concentration of the morphogen that triggers the cell division (black cell will divide). Here the simulation is done in 3D.

where the argument `dt` corresponds to the time step parameters used in transformations `Mechanics` and `DiffusionReaction`. Here transformation `Mechanics` is iterated N times (such that $\Delta_1 = N\Delta_2$) before the application of one diffusion-reaction step. Finally, transformation `Division` checks for some possible cellular divisions.

The complete MGS code, including the building of a global initial state and the output of the state at each time steps, is less than 150 lines. Four snapshots of the system evolution are given in Fig. 3.

5. CONTRIBUTIONS, RELATED WORKS AND PERSPECTIVES

In the introduction we have exposed the rational behind the design of the MGS experimental programming language. MGS is used as a vehicle to study the implementation of basic notions in algebraic topology and their variations suitable for the modeling and the simulation of $(\mathsf{DS})^2$. Morphogenesis was a driving application domain and MGS has been used in the modeling of several developmental processes in systems biology [2, 29]. MGS constructions are *formal* because they are based on the tools of algebraic topology embedded in a declarative framework, *practical* because we have shown that they generalize well known data and control structures, and *ubiquitous*: MGS transformations are able to express discrete analogues of differential operators [17] but they have been also successfully used in the programing of various algorithmic tasks.

The two application examples in this paper rely on Delaunay topological collections. The 1-cells in this kind of collection are computed rather that explicitly enumerated. Delaunay collections are smoothly embedded in MGS showing that the notion of topological collection accepts a wide range of variations.

The use of chains and cochains to structure the modeling and the simulation of a physical systems can be traced back at least to Branin [5] who applied these notions to network analysis and circuit design. Later, Tonti [30, 31] and co-authors developed comprehensive discrete formulations of physical laws from first principles [22, 32]. Several studies have subsequently developed this approach in the field of physical modeling and CAD, notably by Shapiro using the Chain programming language [23] and various follow-up [6, 10, 8, 9]. (Co-)chains have also been used in numerical computation as a tools to structure and generalize the notion of mesh [3].

One major goal of these studies is to unravel a proper set of definitions and differential operators that make it possible to operate the machinery of multivariate calculus on a finite discrete space. The motivation is to find an equivalent calculus that operates intrinsically in discrete space, without the reference to the discretization of an underlying continuous process. This line of research is particularly developed in the field of geometric modeling, with several recent achievements [7, 19].

These works do not focus on the modeling of dynamical structures in the way it is developed in this paper. The corresponding technical apparatus focuses on uniform computations and also on the metric structure while MGS relies on the combinatorial structure. For example, we emphasize a use of ACC which does not require n-cells to be homeomorphic to n balls [33]. The combinatorial approach is less constrained and then potentially more amenable to algorithmic computations. Algorithmic computations can be done in the above mentioned approaches but without a dedicated support from the corresponding tools. For instance, one can specify imperative mesh subdivision algorithms (using nest of iterators) while MGS enables a declarative specification through a very concise set of rules [28]. In addition, specifying transformations through rewriting rules is strictly more expressive than cochains.

The framework presented in this paper, may be enriched and extended in several directions. First, the declarative approach enjoys a very concise and expressive programming style. However, the generation of efficient code from the declarative specification is an open question. Currently, the MGS framework is only available as an interpreter (see the web page at `http://mgs.spatial-computing.org` to access to the sources and various applications). Second, we want to develop further applications in synthetic and integrative spatial systems biology, as well as in some more unconventional fields, like musical analysis [4]. Another direction of future research consists of the automatic inference of system properties by adapting invariant extraction and model checking techniques that have been developed for other kind of rewriting systems.

REFERENCES

[1] F. Aurenhammer. Voronoi diagrams–a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

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

[3] G. Berti. Generic programming for mesh algorithms: Towards universally usable geometric components. In H. A. Mang, F. G. Rammerstorfer, and J. Eberhardsteiner, editors, *Proceedings of the Fifth World Congress on Computational Mechanics (WCCMV*. IACM, 2002.

[4] L. Bigo, O. Michel, and A. Spicher. Spatial programming for music representation and analysis. In *Spatial Computing Workshop 2010, a satellite workshop of SASO 2010*, Budapest, Sept. 2010. ACM press.

[5] F. Branin. The algebraic-topological basis for network analogies and the vector calculus. In *Symposium on generalized networks*, pages 453–491, 1966.

[6] J. A. Chard and V. Shapiro. A multivector data structure for differential forms and equations. *Math. Comput. Simul.*, 54(1-3):33–64, 2000.

[7] M. Desbrun, E. Kanso, and Y. Tong. Discrete differential forms for computational modeling. In *Discrete differential geometry: an applied introduction*, pages 39–54. Schröder, P, 2006. SIGGRAPH'06 course notes.

[8] A. DiCarlo, F. Milicchio, A. Paoluzzi, and V. Shapiro. Solid and physical modeling with chain complexes. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 73–84. ACM, 2007.

[9] A. DiCarlo, F. Milicchio, A. Paoluzzi, and V. Shapiro. Discrete physics using metrized chains. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 135–145. ACM, 2009.

[10] R. Egli and N. F. Stewart. Chain models in computer simulation. *Math. Comput. Simul.*, 66(6):449–468, 2004.

[11] J.-L. Giavitto. Topological collections, transformations and their application to the modeling and the simulation of dynamical systems. In *14th International Conference on Rewriting Technics and Applications (RTA'03)*, volume 2706 of *LNCS*, pages 208–233, Valencia, June 2003. Springer.

[12] J.-L. Giavitto, C. Godin, O. Michel, and P. Prusinkiewicz. *Modelling and Simulation of biological processes in the context of genomics*, chapter "Computational Models for Integrative and Developmental Biology". Hermes, July 2002.

[13] J.-L. Giavitto and O. Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49:107–129, 2002.

[14] J.-L. Giavitto and O. Michel. Modeling the topological organization of cellular processes. *BioSystems*, 70(2):149–163, 2003.

[15] J.-L. Giavitto, O. Michel, J. Cohen, and A. Spicher. Computation in space and space in computation. In *Unconventional Programming Paradigms (UPP'04)*, volume 3566 of *LNCS*, pages 137–152, Le Mont Saint-Michel, Sept. 2005. Spinger.

[16] J.-L. Giavitto, O. Michel, and F. Delaplace. Declarative simulation of dynamicals systems : the $81/2$ programming language and its application to the simulation of genetic networks. *BioSystems*, 68(2–3):155–170, feb/march 2003.

[17] J.-L. Giavitto and A. Spicher. Topological rewriting and the geometrization of programming. *Physica D*, 237(9):1302–1314, jully 2008.

[18] M. C. Gibson, A. B. Patel, R. Nagpal, and N. Perrimon. The emergence of geometric order in proliferating metazoan epithelia. *Nature*, 442:1038–1041, Aug. 2006.

[19] L. Grady and J. Polimeni. *Discrete Calculus: Applied Analysis on Graphs for Computational Science*. Springer, 2010.

[20] J. G. Hocking and G. Young. *Topology*. Dover publications, New-York, 1988.

[21] J. W. Lloyd. Practical advantages of declarative programming. In M. Alpuente, R. Barbuti, and I. Ramos, editors, *GULP-PRODE 1994 Joint Conference on Declarative Programming*, vol. 1, pages 18–30, 1994.

[22] C. Mattiussi. The finite volume, finite element, and finite difference methods as numerical methods for physical field problems. *Advances in Imaging and Electron Physics*, 113:1–146, 2000. *NOTES:* Very highly recommended. Tonti diagrams. Fits many mimetic approaches into a general framework.

[23] R. S. Palmer and V. Shapiro. Chain models of physical behavior for engineering analysis and design. *Research in Engineering Design*, 5:161–184, 1993. Springer International.

[24] J.-C. Raoult and F. Voisin. Set-theoretic graph rewriting. In *Proceedings of the International Workshop on Graph Transformations in Computer Science*, pages 312–325, London, UK, 1994. Springer-Verlag.

[25] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 25–34, July 1987.

[26] D. Rémy. Syntactic theories and the algebra of record terms. Technical Report 1869, INRIA-Rocquencourt, BP 105, F-78 153 Le Chesnay Cedex, 1992.

[27] A. Spicher, O. Michel, M. Cieslak, J.-L. Giavitto, and P. Prusinkiewicz. Stochastic p systems and the simulation of biochemical processes with dynamic compartments. *BioSystems*, 91(3):458–472, March 2008.

[28] A. Spicher, O. Michel, and J.-L. Giavitto. Declarative mesh subdivision using topological rewriting in mgs. In *Int. Conf. on Graph Transformations (ICGT) 2010*, volume 6372 of *LNCS*, pages 298–313, Sept. 2010.

[29] A. Spicher, O. Michel, and J.-L. Giavitto. *Understanding the Dynamics of Biological Systems: Lessons Learned from Integrative Systems Biology*, chapter Interaction-Based Simulations for Integrative Spatial Systems Biology. Springer Verlag, Feb. 2011.

[30] E. Tonti. On the mathematical structure of a large class of physicial theories. *Rendidiconti della Academia Nazionale dei Lincei*, 52(fasc. 1):48–56, Jan. 1972. Scienze fisiche, matematiche et naturali, Serie VIII.

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

[32] E. Tonti. A direct discrete formulation of field laws: The cell method. *Computer Modeling in Engineering & Sciences*, 2(2):237–258, 2001.

[33] A. Tucker. An abstract approach to manifolds. *Annals of Mathematics*, 34(2):191–243, 1933.

[34] A. M. Turing. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc. of London*, Series B: Biological Sciences(237):37–72, 1952.

[35] G. Turk. Generating textures for arbitrary surfaces using reaction-diffusion. In T. W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 289–298, July 1991.