Using Rewriting Techniques in the Simulation of Dynamical Systems: Application to the Modeling of Sperm Crawling

Antoine Spicher and Olivier Michel

LaMI, umr 8042 du CNRS, Université d'Évry – GENOPOLE
Tour Evry-2, 523 Place des Terrasses de l'Agora
91000 Évry, France

{aspicher,michel}@lami.univ-evry.fr

Abstract. Rewriting system (RS) are a formalism widely used in computer science. However, such a formalism can also be used to specify executable models of dynamical systems (DS) by allowing the specification of the evolution laws of the systems in a local manner.
The main drawback of RS is that they are well understood and well known only for terms (a tree-like structure) and that their expressivity is not enough for the representation of complex organizations that can be found in DS.
We propose a framework based on topological notion to extend the notion of RS on more sophisticated structures; the corresponding concepts are validated through the development of an experimental programming language, MGS, dedicated to the simulation of DS. We show how the MGS rewriting system can be used to specify complex dynamical systems and illustrate it with the simulation of the motility of the nematode's sperm cell.

1 Introduction

In this paper, we advocate the use of rewriting techniques for the simulation of complex dynamical systems. The systems we are interested in, are often systems with a dynamical structure [1]. They are difficult to model because their state space is not fixed *a priori* and is jointly computed with the current state during the simulation. In this case the evolution function is often given through local rules that drive the interaction between some system components.

These rules and their application are reminiscent of rewriting rules and their strategy. As a programming language, rewriting systems have the advantage of being close of the mathematical formalism (transparencial referency and declarativeness).

The aim of the `MGS` project is to develop new rewriting techniques on data structures beyond tree-like organization, and to apply these techniques to the modeling and simulation to various dynamical systems with a dynamical structure in biology. The key idea used here to extend rewriting systems to more

general data structures is a topological point of view: a data structure is a set of elements with neighborhood relationship that specifies which elements of the data structure can be accessed from a given one.

This paper is organized as follows. Section 2 recalls the basic notions of rewriting system and sketches its application to the simulation of dynamical systems. Then we present the `MGS` programming language. An example illustrates the introduced notion: the `MGS` simulation approach on a dynamical system with a dynamical structure. The system to be modeled is the motility of a cell, inspired by a previous work [2].

2 Rewriting and Simulations

2.1 A computational Device.

A *rewriting system* [3] (RS) is a device used to replace some part of an entity by another. In computer science, the entities subject to this process are usually expressions represented by formal trees. A RS is defined by a set of rules, and each rule $\alpha \rightarrow \beta$ specifies how a subpart that matches with the pattern α is substituted by a new part computed from the expression β. We call the pattern α the *left hand side* of the rule (*l.h.s*), and β the *right hand side* (*r.h.s*).

We write $e \rightarrow^* e'$ to denote that an expression e is transformed by a series of rewriting in expression e'. It is called a *derivation* of e. The transformation of e into e' can be seen as the result of some computations defined by the rewriting rules and the derivation corresponds to the intermediate results of the computation.

2.2 Rewriting and Simulation.

We will see how rewriting can be used for the simulation of *dynamical systems* (DS), *i.e.*, systems described by a state that changes with the time. Using RS for the simulation of DS means:

- the state of the DS is represented by an expression,
- its evolution is specified by a set of rewriting rules defining local transformation.

Then, given an initial state e, a derivation of e following a RS corresponds to a possible trajectory of the DS.

The role of a rule is to specify an interaction between different parts (atomic or not) of the system, or the answer of the system to an exterior message. So, at a cellular scale, $c + s \rightarrow c'$ means a cell c that receives a signal s, will change its state to c' ; $c \rightarrow c' + c''$ specifies a cell division and $c \rightarrow .$ represents apoptosis. In these examples, operator $+$ denotes the composition of entities into subsystems. The formalism of RS has consequences of the properties of DS taken in considerations, especially on the management of time and space.

Discretized Time. An important point in the modeling of a DS is the handling of time. Clearly the model of time naturally supported by the framework of rewriting is a discrete, event based, model of time: the application of a rule corresponds to some event in the system and this event corresponds to an atomic instantaneous change in the system state.

Locality of Space. The previous operator + that joins entities and messages expresses the spatial and/or the functional organization of the modeled system and is used to denotes interacting parts of the system and the composition of entities into a subsystem. So, on the first hand, the l.h.s and the r.h.s of a rule specify a local part of the system where an interaction occurs. As a consequence, rules represent local evolution laws of the DS. On the other hand, the organization structures specified in the l.h.s and the r.h.s can differ to generate a modification of the structure. This allows the modeling of a special and difficult to represent kind of DS, the *dynamical systems with a dynamical structure* or $(\text{DS})^2$ (see [4]).

3 MGS: a Framework for Modeling and Simulating Dynamical Systems using RS

MGS is a project that aims at integrating the formalism of RS in a programming language dedicated to the modeling and the simulation of $(\text{DS})^2$. In this section, we will present this language. MGS embeds a complete, impure, dynamically typed, strict, functional language.

3.1 Topological Collections

One of the distinctive features of the MGS language is its handling of entities structured by *abstract topologies* using *transformations* [5]. The notion of data structures is unified in the notion of *topological collection*, a set of entities organized by an abstract topology. Topological means here that each collection type defines a neighborhood relation inducing a notion of *subcollection*.

Topological Collection and the Representation of a DS State. Topological collections are well-fitted to represent the complex states of DS at a given time. The elements of the topological collection are the atomic elements of DS and each element has a value.

3.2 Transformations

Topological collections represent a possible framework for an extension of RS. Indeed, the neighborhood relationship provides a local view of the structural organization of elements. *Transformations* extends the notion of RS to structures other than trees and they are used to specify evolution functions of modeled DS. A *transformation* of a topological collection S consists in the *parallel application* of a set of *local rewriting rules*. A local rewriting rule r specifies the replacement

of a subcollection by another one. The application of a rewriting rule $\sigma \Rightarrow f(\sigma, ...)$ to a collection S (1) selects a subcollection S_i of S whose elements match the *pattern* σ, (2) computes a new collection S'_i as a function f of S_i and its neighbors, and (3) specifies the insertion of s'_i in place of s_i into s.

Path Pattern. A pattern σ in the l.h.s of a rule specifies a subcollection where an interaction occurs. This subcollection can have an arbitrary shape, making it very difficult to specify. Thus, it is more convenient (and not so restrictive) to enumerate sequentially its elements. Such enumeration will be called a *path*.

Replacement. The right hand side of a rule specifies a collection that replaces the subcollection matched by the pattern in the left hand side.

4 Application to the Simulation of Nematode Sperm Crawling

In this part, we are interesting in implementing a complex biological model proposed by Bottino *et al* [2]. This model simulates the motility of the sperm cell of the nematode *Ascaris suum*. We first describe the model and its discretization in 2D. Then we see how this model can be translated in the `MGS` formalism.

4.1 Description of the model

The sperm of *Ascaris suum* crawls using a lamellipodial protusion, adhesion and retraction cycle. The chemical mechanisms of motility are located in the front of the cell called *lamellipodium*. In this model, the system corresponds to the lamellipodium membrane stuck to the matrix surrounding the cell. First, a fibrous polymerization occurs at the leading edge of the cell creating protusions. These protusions push the cell membrane forward. Then, some elastic energy is stored in the created fibrous gel. During the adhesion step, the protusions stick the matrix with a traction process that makes the cell body traveling. The fibrous gel undergoes a contraction. The final step occurs near the boundary between the lamellipodium and the rest of the cell where the depolymerization of the fibrous gel causes the deadhesion of the membrane. As a consequence, the stored energy is released to pull the cell body forward. This mechanics is moderated by a pH gradient.

The considered continuous equations corresponds to the elastic and tensile stress in the membrane fixed to the extracellular matrix, and to pH distribution to deal with the pH dependence.

Mechanical Forces. The equation given by Bottino *et al.* governing the mechanical forces is:

$$\mu(u)\frac{\partial u}{\partial t} = \nabla.\sigma(u)$$

where u is a position vector. The l.h.s corresponds to the drag force due to the contact between the membrane and the matrix. The r.h.s computes the

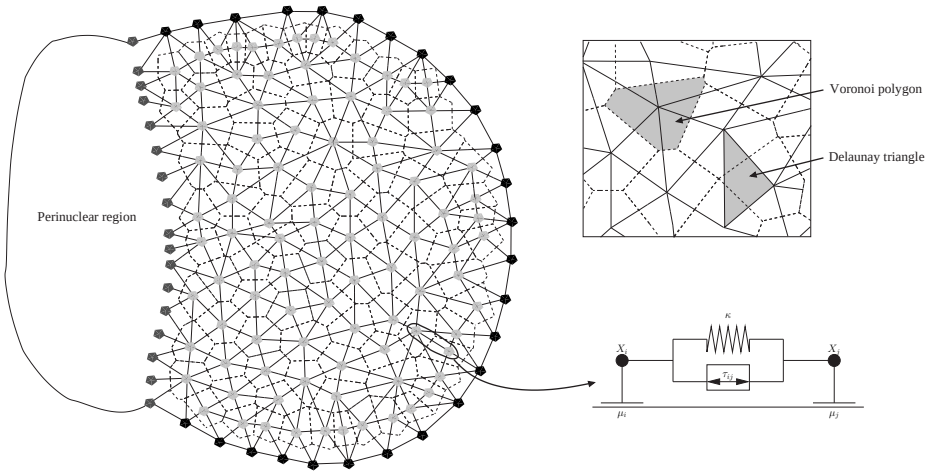

Fig. 1. The nematode sperm cell. At the left, a schematic diagram showing the cell organization: on the left, the nuclear region is found, on right is the lamellipodium. Its discretization is done by the nodes. The plain edges correspond to the Delaunay neighborhood. The dashed edges are the boundaries of the Voronoi polygons. At top right, a figure of a zoomed part of the mesh is given. At bottom right, a Delaunay edge links two nodes with a spring of modulus κ in parallel with a tensile element of stress τ. A friction of coefficient μ appears when a node is in contact with the exterior tissue (these diagrams are inspired by figures from [2]).

mechanical forces from the stress given by σ = Elastic Stress − Tensile Stress. All the coefficients depend on distribution of the pH.

pH Distribution. The pH distribution follows a diffusion equation with a leak. But, this distribution is done in a shorter time scale. Therefore, considering a quasi-static approximation, we obtain:

$$\mathrm{D}\nabla^2[\mathrm{H}^+] = \mathrm{P}([\mathrm{H}^+] - [\mathrm{H}^+]_{\mathrm{ext}})$$

where $[\mathrm{H}^+]$ is the proton concentration at a given position, $[\mathrm{H}^+]_{\mathrm{ext}}$ is the external proton concentration, D and P are properties of the cell. The l.h.s represents the diffusion and the r.h.s is the leakage.

4.2 The Finite Element Model

This 2D surface is divided into finite elements in order to approximate the previous continuous differential equations. Each element corresponds to a node of a mesh (see figure 1). A node represents a Voronoi tessellated cell are represented by dashed edges on figure 1. The neighborhood of each Voronoi polygon is given by a Delaunay triangulation and is figured as plain edges.

There are three kinds of element: (1) the lamellipodial boundary (*Bnodes*) where the protusion occurs (in black), (2) the interface (*NRnodes*) between the lamellipodium and the cell body (in dark grey) where the retraction is done, and (3) the interior (*Inodes*) of the lamellipodium (in light grey).

Polymerization and Depolymerization. The polymerization and the depolymerization of the gel respectively correspond to the creation and the deletion of *Inodes*. Two thresholds give the upper and the lower lengths of a Delaunay edge. Let X_i and X_j be two nodes and l_{ij} the length of the Delaunay edge linking X_i and X_j. If $l_{ij} > l_{\max}$, a node is created is the middle of the edge with a pH being the average between the pH at X_i and X_j. In practice, nodes are created near the boundary. On the opposite, if $l_{ij} < l_{\min}$ and X_i is a *NRnode*, X_j is deleted.

Discretization of the Continuous Equations. The previous equations are translated; for a node X_i:

$$\frac{\partial u_i}{\partial t} = \frac{1}{\mu_i} \sum_j (\kappa |X_i - X_j| - \tau_{ij}) C_2^{ij} \frac{X_j - X_i}{|X_j - X_i|} \quad \text{(mechanical forces)}$$

$$\sum_j C_1^{ij} ([\mathrm{H}^+]_i - [\mathrm{H}^+]_j) = \frac{\mathrm{P}}{\mathrm{D}} ([\mathrm{H}^+]_i - [\mathrm{H}^+]_{\mathrm{ext}}) \quad \text{(pH distribution)}$$

In these two equations, the ∇ operators of the continuous one are replaced by a finite iteration over the neighbors X_j of the node X_i. The computation is local and well-suited to a rewriting framework. The coefficients C_k^{ij} depend on geometrical properties of the Voronoi/Delaunay triangulation (such as the area of Voronoi polygons). In the first equation, the term $\kappa|X_i - X_j|$ corresponds to the elastic force between X_i and X_j (on bottom right of figure 1). In fact, the Delaunay edges are considered as an elastic element (emulated by a spring of modulus κ) in parallel with a tensile element (with the stress τ_{ij}). The coefficient μ_i represents the drag effect.

4.3 `MGS` Implementation

The translation of the model in terms of transformations and topological collections is straightforward.

Data Structures. First, we have to represent a node of the Delaunay graph. We use an `MGS` *record* (a data structure equivalent to a `C struct`) composed by 8 fields:

```
record Node = { px:float, py:float, vx:float, vy:float
                H:float, pH:float, Bflag:bool, NRflag:bool };
```

Fields `px` and `py` represent the position of the node, `vx` and `vy` the speed vector, and `H` and `pH` the proton concentration and the pH. We also define three predicates `Inode`, `Bnode` and `NRnode`, to determine the type of the node. They use the booleans `Bflag` and `NRflag` of a node.

A Delaunay graph is a predefined type of a topological collection available in `MGS`. This type of collection is parameterized by a function that returns the position of an element in space to automatically compute the neighborhood. So we define this function for our example:

```
delaunay(2) D2 = fun elt ->
  if Node(elt) then (elt.px, elt.py)
               else error("bad element type") fi ;;
```

Evolution Laws. Now that we have a representation for the data, we have to specify the evolution laws from the discrete equations. We start with checking the structure to create or delete nodes. The following `MGS` rules compute both polymerization and depolymerization:

```
polymerization:   Xi, Xj / (length(Xi,Xj) > lmax) =>
                    Xi, {pH=(Xi.pH+Xj.pH)/2,...}, Xj
depolymerization: Xi:Inode, Xj / (length(Xi,Xj) < lmin) => Xj
```

where function `length` returns the length between two nodes, and `Xi:Inode` specifies that the node `Xi` must be a `Inode`. As soon as these rules are applied, the Delaunay neighborhood is automatically updated

After that, the pH distribution has to be updated to take account of the new or the deleted nodes. The equation provides the value of the proton concentration of a node as a function of the proton concentration of its neighbors:

```
trans update_pH = {
  Xi:Bnode => ...; Xi:NRnode => ...;
  Xi => let num = neighborsfold(
                    (fun Xj acc -> C1(Xi,Xj) * Xj.H + acc),
                    0, Xi) + (P/D) * H_ext
        and den = neighborsfold(
                    (fun Xj acc -> C1(Xi,Xj) + acc),
                    0, Xi) + (P/D)
        in Xi + {H = num/den, pH = -log10(num/den) }
}
```

The transformation `update_pH` is composed by 3 rules. The two first deal with the boundary conditions of the equation, and the last one applies the equation. The function `neighborfold` is used to evaluate the sum of proton concentration of the neighbors of `Xi` balanced by the coefficient C_1^{ij}. `neighborfold` corresponds to a basic fold on the sequence of the neighbors of `Xi`. Finally, `Xi` is replaced by `Xi+{H = num/den, pH = -log10(num/den)}` that denotes the new value of `Xi` where the fields `H` and `pH` are updated. To deal with the quasi-static approximation, this

transformation is iterated until a fixpoint is reached. This iteration corresponds to the resolution of inverting a matrix as Bottino *et al.* do.

To end one step of the simulation, the force equation has to be computed and the velocities and positions of the nodes updated. The implementation of this transformation is quite similar to `update_pH`.

5 Discussion and Conclusion

The simulation developed here mimics in `MGS` the initial model developed by Bottino *et al.* and implemented in Matlab [6]. One of the main motivations for the development of this example, was to compare the conciseness and the expressivity of the `MGS` programming style compared to a more traditional programming language. Our opinion (which is subjective) is that the developed code is more concise and more readable, for instance because the management of the Voronoi tessellation and the Delaunay triangulation is completely transparent to the programmer. From the point of view of the performance, our approach is comparable (with respect to the few indications available into the articles of Bottino *et al.*) despite that the current `MGS` interpreter is a prototype version.

Acknowledgments.

The authors would like to thank J.-L. Giavitto and J. Cohen at LaMI, D. Boussié, F. Jacquemard at INRIA/LSV-Cachan and the members of the "Simulation and Epigenesis" group at Genopole for technical support, stimulating discussions and biological motivations. This research is supported in part by the CNRS, GDR ALP, IMPG, University of Évry and Genopole/Évry.

References

1. Giavitto, J.L.: Invited talk: Topological collections, transformations and their application to the modeling and the simulation of dynamical systems. In: Rewriting Technics and Applications (RTA'03). Volume LNCS 2706 of LNCS., Valencia, Springer (2003) 208 – 233
2. Bottino, D., Mogilner, A., Roberts, T., Stewart, M., Oster, G.: How nematode sperm crawl. Journal of Cell Science **115** (2002) 367–384
3. Dershowitz, N., Jouannaud, J.P.: Rewrite systems. In: Handbook of Theoretical Computer Science. Volume B. Elsevier Science (1990) 244–320
4. Giavitto, J.L., Godin, C., Michel, O., Prusinkiewicz, P.: "Computational Models for Integrative and Developmental Biology". In: Modelling and Simulation of biological processes in the context of genomics. Hermes (2002).
5. Giavitto, J.L., Michel, O.: The topological structures of membrane computing. Fundamenta Informaticae **49** (2002) 107–129
6. Bottino, D.: *Ascaris suum* sperm model documentation (2000)