A topological framework for the specification and the simulation of discrete dynamical systems

Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto

LaMI, umr 8042 du CNRS, Université d'Évry – GENOPOLE
Tour Evry-2, 523 Place des Terrasses de l'Agora
91000 Évry, France
{aspicher,michel,giavitto}@lami.univ-evry.fr

Abstract. MGS is an experimental programming language for the modeling and the simulation of discrete dynamical systems. The modeling approach is based on the explicit specification of the interaction structure between the system parts. This interaction structure is adequately described by topological notions. The topological approach enables a unified view on several computational mechanisms initially inspired by biological or chemical processes (Gamma and *cellular automata*). The expressivity of the language is illustrated by the modeling of a diffusion limited aggregation process on a wide variety of spatial domain: from cayley graphs to arbitrary quasi-manifolds.

1 Introduction

In this paper, we are interested in the modeling of discrete dynamical systems with some internal structure that arises from their evolution function.

Consider the following discrete process: a particle q moves randomly on a square lattice. So, if q is on node (x, y) at time t, then at time $t + 1$ it occupies one of the node $(x + \epsilon_x, y + \epsilon_y)$ where ϵ_x, ϵ_y are randomly chosen in $\{-1, 0, 1\}$ such that $|\epsilon_x| + |\epsilon_y| \neq 0$. The building of a synchronous cellular automata (CA) that simulates this process is not immediate because the update rules change the state of only one node. It will be much more easy to allow rules that update synchronously a entire subset of the nodes. If this kind of rule is allowed, then it would be trivial to define this process as the simultaneous update of two neighbor cells, one empty and the other containing a particle, by the exchange of their state.

The problem of updating a subset of cells defined by some arbitrary conditions is often solved using an asynchronous dynamics: this avoids that two occupied cells decide at the same step to occupy the same empty neighbor. Another solution is the partitioning of the cells into a coarser graph (this is the approach of lattice gas automata [TM87]).

We see here these two approaches mainly as tricks that do not account that a system can be decomposed into subsystems defined by the requirement that the elements into the subsystems interact together and are truly independent from all other subsystems parallel evolution.

In this view, the decomposition of a system S into subsystems $S_1, S_2, \ldots, S_n$ is *functional*: the state $s_i(t+1)$ of the subsystem S_i depends solely of the previous state $s_i(t)$. However, the decomposition of S into the S_i can depend on the time steps. So we write $S_1^t, S_2^t, \ldots, S_{n_t}^t$ for the decomposition of the system S at time t and we have: $s_i(t+1) = h_i^t(s_i(t))$ where the h_i^t are the "local" evolution functions of the S_i^t. The "global" state $s(t)$ of the system S can be recovered from the "local" states of the subsystems: it exists a function φ^t such that $s(t) = \varphi^t(s_1(t), \ldots, s_{n_t}(t))$ which induces a relation between the "global" evolution function h and the local evolution functions: $s(t+1) = h(s(t)) = \varphi^t(h_1^t(s_1(t)), \ldots, h_{n_t}^t(s_{n_t}(t)))$.

The successive decomposition $S_1^t, S_2^t, \ldots, S_{n_t}^t$ can be used to capture the *elementary parts* and the *interaction structure* between these elementary parts of S. Cf. Figure 1. Two subsystems S' and S'' of S interact if it exists some S_j^t such that $S', S'' \in S_j^t$. Two subsystems S' and S'' are *separable* if it exists some S_j^t such that $S' \in S_j^t$ and $S'' \notin S_j^t$ or vice-versa. This leads to consider the set $\mathcal{S}$, called the *interaction structure* of S, defined by the smaller set closed by intersection that contains the S_j^t.

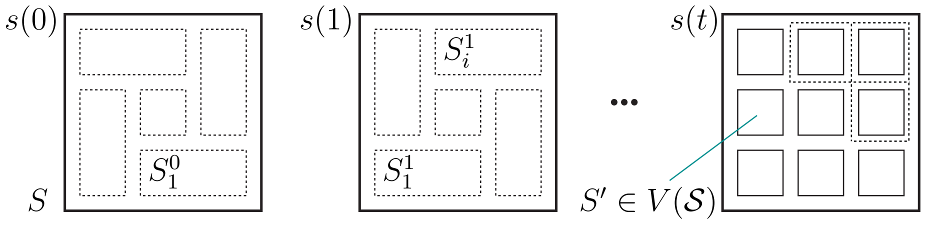

Fig. 1. The interaction structure of a system S resulting from the subsystems of elements in interaction at a given time step.

The set $\mathcal{S}$ has a *topological structure*: $\mathcal{S}$ corresponds to an *abstract simplicial complex*. An abstract simplicial complex [Mun84] is a collection $\mathcal{S}$ of finite nonempty set such that if A is an element of $\mathcal{S}$, so is every nonempty subset of A. The element A of $\mathcal{S}$ is called a *simplex* of $\mathcal{S}$; its *dimension* is one less that the number of its element. The dimension of $\mathcal{S}$ is the largest dimension of one of its simplices. Each nonempty subset of A is called a *face* and the *vertex set* $V(\mathcal{S})$, defined by the union of the one point elements of $\mathcal{S}$, corresponds to the *elementary functional parts* of the system S. The abstract simplicial complex notion generalizes the idea of *graph*: a simplex of dimension 1 is an edge that links two vertices, a simplex d of dimension 2 can be thought of as a surface whose boundaries are the simplices of dimension 1 included in d, etc.

Our proposal is to specify a discrete dynamical system starting from its interaction structure. Our idea is to define directly the set $\mathcal{S}$ with its topological structure and to specify the evolution function h by specifying the set S_i^t and the functions h_i^t. The result is an experimental programming language called `MGS`. In

MGS, the interaction structure $\mathcal{S}$ is defined as a new kind of data structures called *topological collections* and a set of functions h_i^t together with the specification of the S_i^t for a given t are called *transformation.* A transformation is a function defined by the cases S_i^t.

This abstract approach has two main advantages. The first one is that it enables the easy specification of dynamical systems with a dynamical structure (DS)^2. In such systems, the phase space cannot be defined *a priori* and must be computed jointly with the state of the system (in other word, the set of states must be an observable of the system itself, see [Gia03]). The second advantage is that this abstract approach enables an homogeneous and uniform handling of several computational models including CA, lattice gas automata, abstract chemistry, Lindenmayer systems and several other abstract reduction systems.

The rest of the paper is organized as follows. We begin with a short introduction to the MGS experimental programming language, through the notions of topological collection and transformation. Then, section 3 illustrates the application of these notions on a simple CA. The same diffusion-aggregation process is considered on two increasingly more complex topologies. Finally, we review some related works and the perspectives opened by this research.

2 A Brief Presentation of the MGS Language

MGS embeds a complete, impure, dynamically typed, strict, functional language. We focus on the notions required to understand the rest of the paper and we only describe here the major differences between the constructions available in MGS with respect to functional languages like OCAML [Ler96].

2.1 Topological Collections

The distinctive feature of the MGS language is its handling of entities structured by *abstract topologies* using *transformations* [GM02]. A set of entities organized by an abstract topology is called a *topological collection.* Topological means here that each collection type defines a neighborhood relation inducing a notion of *subcollection.* A subcollection S' of a collection S is a subset of connected elements of S and inheriting its organization from S.

Topological Collection and the Representation of a (DS)^2 State. A topological collection is used to represent the complex states s of a dynamical system S at a given time t. The elements of the topological collection are the elements S' of $V(\mathcal{S})$ and each element has a value given by s'. In the context of topological collections, the elements of $V(\mathcal{S})$ are called *positions* and we say that s' is the value associated to the position S'. Often there is no need to distinguish between the positions and their associated value. In this case, we use the term "element of the collection".

Collection Types. Different predefined and user-defined collection types are available in MGS, including sets, bags (or multisets), sequences, several grids and arbitrary topologies. We introduce the collection types along with the examples.

For any collection type T, the corresponding empty collection is written ():T. The join of two collections C_1 and C_2 (written by a comma: C_1,C_2) is the main operation on collections. The comma operator is overloaded in MGS and can be used to build any collection by joining its two arguments. To spare the notations, the empty sequence can be omitted in the definition of a sequence: 1, 2, 3 is equivalent to 1, 2, 3, ():seq.

2.2 Transformations

Transformation are used to specify the evolution function of a $(\mathrm{DS})^2$. The *global transformation* of a topological collection s consists in the *parallel application* of a set of *local transformations.* A local transformation is specified by a rule r that 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,
3. and specifies the insertion of s'_i in place of s_i into s.

One should pay attention to the fact that, due to the parallel application strategy of rules, *all distinct instances s_i of the subcollections matched by the σ pattern are "simultaneously replaced" by the $f(s_i)$.*

The MGS experimental programming language implements the idea of transformations of topological collections into the framework of a functional language: collections are just new kind of values and transformations are functions acting on collections and defined by a specific syntax using rules. Transformations (like functions) are first-class values and can be passed as arguments or returned as the result of an application.

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

A path pattern *Pat* is a sequence or a repetition *Rep* of *basic filters.* A basic filter *BF* matches one element. The following (fragment of the) grammar of path patterns reflects this decomposition:

$$Pat ::= Rep \mid Rep \texttt{ , } Pat \qquad Rep ::= BF \mid BF\texttt{/}exp \qquad BF ::= \texttt{cte} \mid \mathrm{id} \mid \texttt{<undef>}$$

where cte is a literal value, id ranges over the pattern variables and *exp* is a boolean expression. The following explanations give a systematic interpretation for these patterns:

literal: a literal value `cte` matches an element with the same value. For example, 123 matches an element with value 123.

empty element the symbol `<undef>` matches an element with an undefined value, that is, an element whose position does not have an associated value.

variable: a pattern variable a matches exactly one element with a well defined value. The variable a can then occur elsewhere in the rest of the rule and denotes the value of the matched element.

neighbor: b`,` p is a pattern that matches a path which begins by an element matched by b and continues by a path matched by p, the first element of p being a neighbor of b.

guard: p/exp matches a path matched by p if boolean expression exp evaluates to `true`.

Elements matched by basic filters in a rule are distinct. So a matched path is without self-intersection.

Right Hand Side of a Rule. The right hand side of a rule specifies a collection that replaces the subcollection matched by the pattern in the left hand side. There is an alternative point of view: because the pattern defines a sequence of elements, the right hand side may be an expression that evaluates to a sequence of elements. Then, the substitution is done element-wise: element i in the matched path is replaced by the ith element in the r.h.s. This point of view enables a very concise writing of the rules.

2.3 Short Example

We give an example that imply the transformation of a sequence of elements.

Bubble Sort in `MGS`. The bubble sort consists in (1) comparing two neighbors elements in a sequence and swapping them if they are not in order; (2) repeating the first step until a fixed point is reached. This specification is immediate in `MGS` and can be written:

```
trans bubble_sort = {      x, y / (x > y) => y, x      }
```

The keyword `trans` introduces the definition of a transformation by a set of rules. Here there is only one rule. This transformation can be applied to a sequence `s := 4, 2, 3, 1` until a fixed point is reached: `bubble_sort['iter='fixpoint](s)`. The value of the predefined optional parameter `'iter` indicates that the application of the function `bubble_sort` must be iterated until a fixed point is reached. The results is `1, 2, 3, 4` as expected.

3 (DS)^2 on Complex Topologies

In this section we present the use of `MGS` to model and simulate more complex systems. Our running example will be a diffusion limited aggregation process on different spatial domains. Diffusion Limited Aggregation, or DLA, is a fractal growth model studied by two physicists, T.A. Witten and L.M. Sander, in the 80's. The principle of the model is simple: a set of particles diffuse randomly on a given spatial domain. Initially one particle, the seed, is fixed. When a mobile particle collides a fixed one, they stick together and stay fixed. For the sake of simplicity, we suppose that they stick together forever and that there is no aggregate formation between two mobile particles.

This process leads to a simple CA with an asynchronous update function or a lattice gas automata with a slightly more elaborate rule set. The purpose of this section is twofold. Firstly, we want to show that the MGS approach enables the specification of a simple generic transformation that can act on arbitrary complex topologies. Secondly, we show how to specify MGS topological collections that correspond to standard CAs.

3.1 The DLA Evolution Function in MGS

The transformation describing the DLA behavior is really simple. We use two symbolic values `‘mobile` and `‘fixed` to represent respectively a mobile and a fixed particle. There are two rules in the transformation: (1) if a diffusing particle is the neighbor of a fixed seed, then it becomes fixed; (2) if a mobile particle is neighbor of an empty place, then it may leave its current position to occupy the empty neighbor. Note that the order of the rules is important because the first has priority over the second one. Thus, we have :

```
trans dla = {
    ‘mobile, ‘fixed   =>  ‘fixed, ‘fixed
    ‘mobile, <undef>  =>  <undef>, ‘mobile
}
```

3.2 DLA on Uniform Topologies : GBF

Group-based data fields (GBF in short) are topological collections used to define topologies with a *uniform* neighborhood: a position cannot be distinguished only by looking at its neighbors. This implies for example that each position has the same number of neighbors.

A GBF is an extension of the notion of array where the elements are indexed by the elements of a group called the *shape* of the GBF [GMS95,GM01]. For example:

```
gbf Grid2 = < north, east >
```

defines a GBF collection type called *Grid2*, corresponding to the Von Neumann neighborhood in a classical array (a cell above, below, left or right – not diagonal). The two names `north` and `east` refer to the directions that can be followed to reach the neighbors of an element. These directions are the *generators* of the underlying group structure. The r.h.s. of the GBF definition gives a *finite presentation* of the group structure [Sha90]. The list of the generators can be completed by giving equations that constraint the displacements in the shape:

```
gbf Torus2 = < north, east; 12*east = 0, 40*north = 0 >
```

defines a 40×12 torus: if one starts from a position and follows 12 times the `east` direction, one finds oneself at the same position. Another example is the definition of an hexagonal lattice that tiles the plane:

```
gbf Hexa2 = < east, north, northeast; east + north = northeast >
```

Each cell has six neighbors (following the three generators and their inverses). The equation `east + north = northeast` specifies that a `northeast` move is equivalent to a move following the `east` direction followed by a move following the `north` direction.

A GBF value of type `T` is a partial function that associates a value to some group elements (the group elements are the positions of the collection and the the empty GBF is the everywhere undefined function). The topology of `T` is easily visualized as the Cayley graph of the presentation of `T`: each vertex in the Cayley graph is an element of the group and vertices x and y are linked by an edge labeled `g` if there is a generator `g` in the presentation such that $x + \mathtt{g} = y$.

Figure 2 represents the final state of a DLA process, where each initially mobile particle has been fixed. The plot at the left is the application, until a fixed point has been reached, of the transformation `dla` to a `Torus2`. The plot at the right is the fixed point reached by the same transformation `dla` to a `Hexa2`. In this last simulation, particles are initially packed on the right and the initial static particle was on the left part of the lattice. Particles are constrained to move on a rhombus. This explains the asymmetries of the figure.

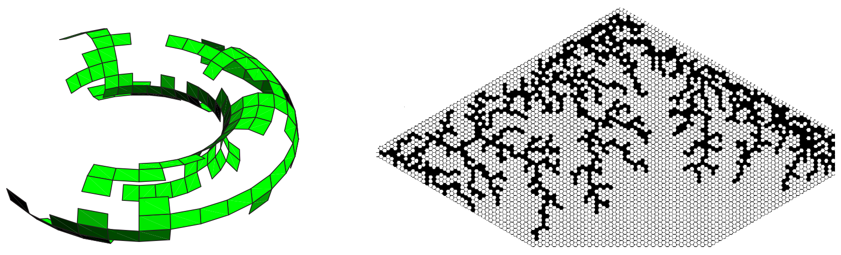

Fig. 2. Example of DLA on two different topologies: a torus (left) and an hexagonal meshes (right)

3.3 Arbitrary topology : DLA on Cellular Complexes

Beyond Graphs. The interaction structure of the previous examples can be adequately described by a graph: two positions are connected by an edge if they are neighbors. Sequences correspond to linear graphs and GBFs provide Cayley's graphs with regular connectivity. This last family of topologies is convenient to represent regular spaces where all elementary parts have the same spatial properties, but they cannot be used for instance to model a sphere (there is no regular graph on a sphere that corresponds to the Cayley graph of some group).

This shortcoming motivates the development of more powerful (arbitrary) topologies. First we would like to represent irregular meshes. However, we also want to represent heterogeneous spaces. As an example, let consider the electrostatic laws. They depend on the geometry of the system and some values must be associated to a dimension: the distribution of electric charges corresponds to a *volumic* density while the electric flux through a surface is associated to a *surface*. Note that *balance equations* often link these values, such as the Gauss theorem for our electrostatic example. See also the work of E. Tonti [Ton74] for an elaboration.

As a consequence, the interaction structure that describes a system S may contains simplices with dimension greater than one. In general, any component of any dimension and their interactions should appear in the description of the system, and we should be allowed to associate some values to them.

Arbitrary Topological Collection as Cellular Complex. The right framework to develop a topological collection type that allows the representation of arbitrary topologies is the *combinatorial algebraic topology* theory.

In this framework, a topological collection is a *cellular complex*: a collection of objects of various dimension called *k-cell*, where k is the dimension. To be more practical, 0-cells are vertices, 1-cells are edges, 2-cells are faces, 3-cells are volumes, etc. To build some arbitrary complex domain, the domain is divided into a cellular partition. Each cell represents a simple part of the domain and the cells are glued together: a k-cell c_1 is incident to a $(k-1)$-cell c_2 if $c_2 \subset \partial c_1$, where ∂c_1 denotes the border of c_1. This boundary relation ∂ can be used to specify the neighborhood relationships in a topological collection: *two k-cells are neighbors if they share an incident $(k-1)$-cell or if they are incident to a same $(k+1)$-cell.* This definition of a topological collection is consistent with the previous one.

G-map in `MGS`. There are several specializations of the notion of cellular complex. For instance abstract simplical complex evoked in the introduction are special cases of cellular complexes. In `MGS` one can use *generalized map* or (Gmap) [Lie91] to build arbitrary topologies. The topological objects that can be described by Gmaps are *quasi-manifolds*.

There are several ways to specify and build Gmaps in `MGS`. A Gmap can be build as the result of construction operations like various products, extrusion, suspension, pasting, gluing, etc. A perhaps simpler way is to edit manually the Gmap in an interactive CAD tool like MOKA[1] and to import the result in `MGS`.

The figure 3 shows applications of the DLA transformation on different kinds of objects built with Gmaps. As a matter of fact, the change of topological collection doesn't affect the transformation and we still apply the same `dla` transformation. In these examples, the topological collections have dimension 2 and the values are associated only to 2-cells. The 2-cells are neighbors if they have a common edge on their boundary. In the top of the figure, only the 1-cells are figured, and in the bottom, only the 2-cells that hold a value are represented.

[1] `http://www.sic.sp2mi.univ-poitiers.fr/moka/`

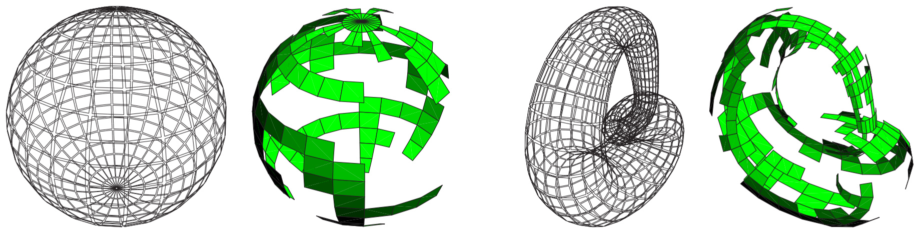

Fig. 3. DLA on complex objects (topology and final state). On the left: a sphere with 18 parallels and 24 meridians. On the right: a Klein's bottle.

4 Conclusion and Perspectives

This paper only focuses on a part of the features available in MGS that can be used to develop computer models of complex discrete dynamical systems that are concise and mathematically well-founded. The approach has been applied successfully to several biological processes (the growth of a tumor, the flocking of birds, colonies of ants foraging for food, the heterocysts differentiation during *Anabaena* growth, etc.) as well as more algorithmic problems (flow on graphs, various sorting algorithms, Hamiltonian path, prime number generation, etc.).

The modeling of $(DS)^2$ through their interaction structure is part of a long term research effort [Gia03]. The topological approach presented here provides a unified description of several computational models. Obviously, Lindenmayer systems [Lin68] correspond to transformations on sequences. Chemical computation, as in Gamma [BFM01], can be emulated using transformations on bags.

There exists strong links between GBF and cellular automata, especially considering the work of Z. Róka [Rók94] which has studied CA on Cayley graphs. However, our own works focus on the construction of Cayley graphs as the shape of a data structure and we develop an operator algebra and rewriting notions on this new data type. This is not in the line of Z. Róka who focuses on synchronization problems and establishes complexity results in the framework of CA.

The perspectives opened by this work are numerous. From the applications point of view, we are targeted by the simulation of developmental processes in biology [GM03,GMM04]. At the language level, the study of the topological collections concepts must continue with a finer study of transformation kinds. Several kinds of restrictions 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. The efficient compilation of a MGS program is a long-term research. We have considered in this paper only one-dimensional paths, but a general n-dimensional notion of path exists and can be used to generalize the substitution mechanisms of MGS.

The sources of the current MGS implementation, as well as several examples and technical reports, are freely available at `http://mgs.lami.univ-evry.fr`.

Acknowledgments. The authors would like to thank the MOKA team at the Univ. of Poitier, J. Cohen at LaMI, F. Jacquemard at 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

[BFM01] Jean-Pierre Banâtre, Pascal Fradet, and Daniel Le Métayer. Gamma and the chemical reaction model: Fifteen years after. *Lecture Notes in Computer Science*, 2235:17–44, 2001.

[Gia03] J.-L. Giavitto. 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*, pages 208 – 233, Valencia, June 2003. Springer.

[GM01] J.-L. Giavitto and O. Michel. Declarative definition of group indexed data structures and approximation of their domains. In *Proceedings of the 3nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-01)*. ACM Press, September 2001.

[GM02] Jean-Louis Giavitto and Olivier Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49:107–129, 2002.

[GM03] J.-L. Giavitto and O. Michel. Modeling the topological organization of cellular processes. *BioSystems*, 70(2):149–163, 2003.

[GMM04] J.-L. Giavitto, G. Malcolm, and O. Michel. Rewriting systems and the modelling of biological systems. *Comparative and Functional Genomics*, 5:95–99, February 2004.

[GMS95] J.-L. Giavitto, O. Michel, and J.-P. Sansonnet. Group based fields. In I. Takayasu, R. H. Jr. Halstead, and C. Queinnec, editors, *Parallel Symbolic Languages and Systems (International Workshop PSLS'95)*, volume 1068 of *LNCS*, pages 209–215, Beaune (France), 2–4 October 1995. Springer-Verlag.

[Ler96] X. Leroy. The Objective Caml system, 1996. Software and documentation available on the web at http://pauillac.inria.fr/ocaml/.

[Lie91] P. Lienhardt. Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1):59–82, 1991.

[Lin68] A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[Mun84] James Munkres. *Elements of Algebraic Topology*. Addison-Wesley, 1984.

[Rók94] Zsuzsanna Róka. One-way cellular automata on Cayley graphs. *Theoretical Computer Science*, 132(1–2):259–290, 26 September 1994.

[Sha90] Igor' Shafarevich. *Basic Notions of Algebra*. Springer, 1990.

[TM87] T. Toffoli and N. Margolus. *Cellular automata machines: a new environment for modeling*. MIT Press, Cambridge, 1987.

[Ton74] Enzo Tonti. The algebraic-topological structure of physical theories. In P. G. Glockner and M. C. sing, editors, *Symmetry, similarity and group theoretic methods in mechanics*, pages 441–467, Calgary, Canada, August 1974.