MGS: *a declarative spatial computing programming language*[*]

Jean-Louis GIAVITTO[1], Antoine SPICHER[2]

[1] *UMR 9912 STMS, IRCAM - CNRS - UPMC, 1 place Igor Stravinsky, 75004 Paris, France*
[2] *Université Paris-Est Créteil, LACL, 61 rue du Général de Gaulle, 94010 Créteil, France*
giavitto@ircam.fr;antoine.spicher@u-pec.fr

Abstract

We sketch the rationals of the MGS programming language. MGS is an experimental programming language developed to study the application of several spatial computing concepts to the specification and simulation of dynamical systems with a dynamical structure. MGS extends the notion of rewriting by considering more general structure than terms. The basic computation step in MGS replaces in a *topological collection* A, some subcollection B, by another topological collection C. A topological collection is a set of element structured by a neighborhood relationships describing an underlying space representing a data structure or constraints that must be fulfilled by the computation. This process proposes a unified view on several computational mechanisms initially inspired by biological or chemical processes (Gamma and the CHAM, Lindenmayer systems, Paun systems and cellular automata).

1 Dynamical Systems and their State Structures

A *dynamical system* (or DS in short) is a phenomenon that evolves in time. At any point in time, a dynamical system is characterized by a set of state variables. Intuitively, a DS is a formal way to describe how a point (the state of the system) moves in the *phase space* (the space of all possible states of the system). It gives a rule telling us where the point should go next from its current location. 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).

This description outlines the change of state in time but does not stress that the set of state variables can also change in time and that the *a priori* determination of the phase space cannot always be done. This is a common situation in biology [FB94, FB96, Ros91] where such DS can be found in morphogenetic processes such as in plant growing, developmental biology or evolutionary processes. They also naturally appear in complex adaptive systems which exhibits dynamic networks of interactions and relationships not predefined aggregations of static entities (distributed systems, social networks, etc.).

This kind of systems share the property that the structure of the phase space must be computed jointly with the current state of the system. Note that if the set of state variables evolves in time, so does the transition function. We qualify such systems as $(\text{DS})^1$: *dynamical systems with a dynamical structure* [Gia03].

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?

2 Space for Structuring Computation

The MGS project is an attempt to answer the previous question based on the assumption that the state of a $(\text{DS})^1$ can be dynamically structured in term of *spatial relationships* where only "neighbor" elements may interact.

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 [GGMP02]. 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.

The direct interactions of arbitrary elements in a system are not always allowed nor desirable and only "neighbor" elements may interact. For instance, 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

[*] This presentation relies partially on work presented already in [GM01c, GM02a, GMCS05, GS08b].

[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.

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 a topological organization: the set of parts can be organized as an *abstract simplicial complex* [GMCS05].

The neighborhood relationship, which instantiates the locality property of the computations, can be formalized using concept from algebraic topology, see [GMCS05]. This relation represents *physical* (spatial distribution, localization of the resources) or *logical constraints* (inherent to the problem to be solved).

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

3 Computing in Space, Space in Computation and Spatial Computing

We have described the rational behind the idea of using spatial relationships to structure the specification of a (DS)[1]. However, the use of spatial relationships can invade all domains of computing.

Spatial relationships adequately represent *physical constraints* (spatial distribution, localization of the resources) or *logical constraints* (inherent to the problem to be solved). Physical constraints are given by distributed computations where the computing resources are localized in different places in space.

Logical constraints arise when position and shape are input to computation or a key part of the desired result of the computation, *e.g.* in computational geometry applications, amorphous computing [AAC+00], claytronics [ABC+05], distributed robotics or programmable matter... to cite a few. More generally, logical constraints are given by the accessibility relation involved by a data structure [GM02a]. For instance, in a simply linked list the elements are accessed linearly (the second after the first, the third after the second, etc.). In arrays, a computation often involves only an element and its neighbors: the neighborhood relationships are left implicit and implemented through incrementing or decrementing indices. The concept of logical neighborhood in a data structure is not only an abstraction perceived by the programmer and vanishing at the execution. Indeed, the computation usually complies with the logical neighborhood of the data elements such that some primitive recursion schemes, e.g. catamorphisms and others polytypic functions on inductive types [MFP91] can be automatically defined from the data structure organization.

As an example, consider parallel computing. Parallel computing deals with both logical and physical constraints: computations are distributed on physical distinct computing resources but the distribution of the computation is a parameter of the execution, a choice done at a logical level to minimize the computation time, and does not depend on some imposed physical localizations induced solely by the problem to be solved.

[DHGG06] introduce the term *spatial computing* to recognize that space is not an issue to abstract away but that computation is performed distributed across space and that space, either physical or logical, serves as a mean, a resource, an input and an output of a computation.

4 Unifying Several Biologically Inspired Computational Models

One of our additional motivations is the ability to describe generically the basic features of four models of computation: Gamma (chemical computing) [BFL01], P systems (membrane computing) [Pău01], L systems [RS92] and cellular automata (CA) and related models of crystalline computing [Mar98, Tof04]. They have been developed with various goals in mind, e.g. parallel programming for Gamma, calculability and complexity issues for P systems, formal language theory and biological modeling for L systems, parallel distributed model of computation for CA (this list is not exhaustive). However, all these computational models rely on a biological or biochemical metaphor. They have been used extensively to model DS. So, it is then very tempting to ask if a uniform framework may encompass these formalisms, at least with respect to simulation purposes.

We assume that the reader is familiar with the main features of these formalisms. A Gamma program, a P or a L system and a CA can be themselves viewed abstractly as a discrete dynamical system: a running program can be characterized by a state and the evolution of this state is specified through *evolution rules*. From this point of view, they share the following characteristics:

Discrete space and time. The structure of the state (the multiset in Gamma, the membranes hierarchy in a P system, the string in a L system and the array in a CA) consists of a discrete *collection* of values. This discrete collection of values evolves in a sequence of discrete time steps.

Temporally local transformation. The computation of a new value in the new state depends only on values for a fixed number of preceding steps (and as a matter of fact, only one step).

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

"Structural combination", means that the elementary results are combined into a new collection, irrespectively of their precise value. "Small and static subset" makes explicit that only a fixed subset of the initial elements are used to compute a new element value (this is measured for instance by the diameter of the evolution rule of a P systems, the local neighborhood of a CA, the number of variables in the right hand side of a Gamma reaction or the context of a rule in a L system).

Considering these shared characteristics, the main difference between the four formalisms appears to be the organization of the collection. The abstract computational mechanism is always the same:

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

We call these three basic steps a *transformation*. In addition to transformation specification, there is a need to account for the various constraints in the selection of the subcollection A and the replacement B. This abstract view makes possible the unification in the same framework of the above mentioned computational devices. The trick is just to change the organization of the underlying collection.

Constraining the Neighborhood. There is *a priori* no constraint in the case of Gamma: one element or many elements are replaced by zero, one or many elements. The means that the topology underlying a multiset makes all elements connected.

In the case of P systems, the evolution of a membrane may affect only the immediate enclosing membrane (by expelling some tokens or by dissolution): there is a *localization* of the changes. The topology of P systems are nested multiset: in a membrane all element are connected but membranes are organized in a strict hierarchy.

L systems are based on string (or sequence of elements). This also exhibit locality: the new collection B depends only on the context of A (the immediate neighbors in the string) and it is inserted at the place of A and not spread out over C.

For CA, the changes are not only localized, but also A and B are constrained to have the same shape: usually A is restricted to be just one cell in the array and B is also one cell to maintain the array structure.

5 MGS

MGS generalizes the previous formalisms by considering collection of elements with arbitrary neighborhood relationships and by developping a suitable notion of rewriting of such collections.

We see a collection as a set of *places* or *positions* organized by a *topology* defining the *neighborhood* of each element in the collection and also the possible subcollections. To stress the importance of the topological organization of the collection's elements, we call them "topological collection".

The topology needed to describe the neighborhood in a set or a sequence, or more generally the topology of the usual data structures, are fairly poor [GM02a]. So, one may ask if the machinery needed is worthwhile. Actually, more complex topologies are needed for some biological modeling applications or for self-assembly processes [GS08a, SMG11]. And more importantly, the topological framework unify various situations. notions, see [GM01b].

Now, we come back to our initial goal of specifying the dynamical structure of a DS. A collection is used to represent the state of a DS. The elements in the collection represent either entities (a subsystem or an atomic part of the DS) or messages (signal, command, information, action, etc.) addressed to an entity. A subcollection represents a subset of interacting entities and messages in the system. The evolution of the system is achieved through transformations, where the left hand side of a rule typically matches an entity and a message addressed to it, and where the right and side specifies the entity's updated state, and possibly other messages addressed to other entities. If one uses a multiset organization for the collection, the entities interact in a rather unstructured way, in the sense that an interaction between two objects is enabled simply by virtue of their both being present in the multiset (the topology underlying a multiset makes all elements connected). More organized topological collections are used for more sophisticated spatial organization.

We do not claim that topological collection are a useful theoretical framework encompassing all the previous formalisms. We advocate that few notions and a single syntax can be consistently used to allow the merging of these formalisms *for programming* purposes.

To go Further. The interested reader may refer to [GM02a, GM01a, GMC02] for the spatial approach of data structure. The mathematical concepts behind the MGS approach are presented in [GM01b, Spi06, GS08b, SMG10]. The modeling and the simulation of (DS)[1] are exposed for example in [GGMP02, Gia03, SMG04, GS08b]. Applications in systems biology are presented in [GM03, SM07, MSG09, SMG11]. Algorithmic applications are sketched in [GM01c, MJ05, SMG10]. Relation with novel approaches of computing, like P systems or autonomic computing, are showed in [GM02c, DHGG06, GS08a, GS08b, GMS08]. Further examples and documentation can be found at the MGS home page:

`mgs.spatial-computing.org`

Acknowledgments. The authors would like to thanks Olivier Michel for stimulating discussions and implementation works. They are also grateful to H. Klaudel, F. Pommereau, F. Delaplace and J. Cohen for many questions, encouragements and sweet cookies. This research is supported in part by the CNRS, the ANR projects AutoChem and SynBioTIC.

References

[AAC+00] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous computing. *CACM: Communications of the ACM*, 43, 2000.

[ABC+05] Burak Aksak, Preethi Srinivas Bhat, Jason Campbell, Michael DeRosa, Stanislav Funiak, Phillip B. Gibbons, Seth Copen Goldstein, Carlos Guestrin, Ashish Gupta, Casey Helfrich, James F. Hoburg, Brian Kirby, James Kuffner, Peter Lee, Todd C. Mowry, Padmanabhan Pillai, Ram Ravichandran, Benjamin D. Rister, Srinivasan Seshan, Metin Sitti, and Haifeng Yu. Claytronics: highly scalable communications, sensing, and actuation networks. In Jason Redi, Hari Balakrishnan, and Feng Zhao, editors, *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, SenSys 2005, San Diego, California, USA, November 2-4, 2005*, page 299. ACM, 2005.

[BFL01] 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.

[DHGG06] André De Hon, Jean-Louis Giavitto, and Frédéric Gruau, editors. *Computing Media and Languages for Space-Oriented Computation*, number 06361 in Dagsthul Seminar Proceedings. Dagsthul, `http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=2006361`, 3-8 september 2006.

[FB94] Walter Fontana and Leo W. Buss. "the arrival of the fittest": Toward a theory of biological organization. *Bulletin of Mathematical Biology*, 1994.

[FB96] W. Fontana and L. Buss. *Boundaries and Barriers, Casti, J. and Karlqvist, A. edts,*, chapter The barrier of objects: from dynamical systems to bounded organizations, pages 56–116. Addison-Wesley, 1996.

[GGMP02] 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. Also republished as an high-level course in the proceedings of the Dieppe spring school on "Modelling and simulation of biological processes in the context of genomics", 12-17 may 2003, Dieppes, France.

[Gia03] 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.

[GM01a] 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.

[GM01b] J.-L. Giavitto and O. Michel. `MGS`: a programming language for the transformations of topological collections. Technical Report 61-2001, LaMI – Université d'Évry Val d'Essonne, May 2001.

[GM01c] Jean-Louis Giavitto and Olivier Michel. Mgs: a rule-based programming language for complex objects and collections. In Mark van den Brand and Rakesh Verma, editors, *Electronic Notes in Theoretical Computer Science*, volume 59. Elsevier Science Publishers, 2001.

[GM02a] J.-L. Giavitto and O. Michel. Data structure as topological spaces. In *Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02*, volume 2509, pages 137–150, Himeji, Japan, October 2002. Lecture Notes in Computer Science.

[GM02b] J.-L. Giavitto and O. Michel. Pattern-matching and rewriting rules for group indexed data structures. In *ACM Sigplan Workshop RULE'02*, pages 55–66, Pittsburgh, October 2002. ACM.

[GM02c] 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.

[GMC02] J.-L. Giavitto, O. Michel, and J. Cohen. Pattern-matching and rewriting rules for group indexed data structures. *ACM SIGPLAN Notices*, 37(12):76–87, December 2002. Selected papers from the PPDP satellite workshops. Revised version from [GM02b].

[GMCS05] 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, September 2005. Spinger.

[GMS08] Jean-Louis Giavitto, Olivier Michel, and Antoine Spicher. *Software-Intensive Systems and New Computing Paradigms*, volume 5380 of *LNCS*, chapter Spatial Organization of the Chemical Paradigm and the Specification of Autonomic Systems, pages 235–254. Springer, november 2008.

[GS08a] Jean-Louis Giavitto and Antoine Spicher. *Systems Self-Assembly: multidisciplinary snapshots*, chapter Simulation of self-assembly processes using abstract reduction systems, pages 199–223. Elsevier, 2008. doi:10.1016/S1571-0831(07)00009-3.

[GS08b] Jean-Louis Giavitto and Antoine Spicher. Topological rewriting and the geometrization of programming. *Physica D*, 237(9):1302–1314, jully 2008.

[Mar98] Norman Margolus. Crystalline computing. In Kathleen P. Hirons, Manuel Vigil, and Ralph Carlson, editors, *Proc. Conf. on High Speed Computing LANL * LLNL*, pages 249–255, Gleneden Beach, OR, April 1998. LANL. LA-13474-C Conference, UC-705.

[MFP91] E. Meijer, M. Fokkinga, and R. Paterson. Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire. In *5th ACM Conference on Functional Programming Languages and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 124–144, Cambridge, MA, August 26–30, 1991. Springer, Berlin.

[MJ05] O. Michel and F. Jacquemard. *An Analysis of a Public-Key Protocol with Membranes*, pages 281–300. Natural Computing Series. Springer Verlag, 2005.

[MSG09] Olivier Michel, Antoine Spicher, and Jean-Louis Giavitto. Rule-based programming for integrative biological modeling – application to the modeling of the λ phage genetic switch. *Natural Computing*, 8(4):865–889, December 2009.

[Pău01] Gheorghe Păun. From cells to computers: computing with membranes (P systems). *Biosystems*, 59(3):139–158, March 2001.

[Ros91] Robert Rosen. *Life itself: a comprehensive inquiry into the nature, origin, and fabrication of life*. Columbia Univ Pr, 1991.

[RS92] G. Ronzenberg and A. Salomaa, editors. *L systems: from formalism to programming languages*. Springer Verlag, February 1992.

[SM07] Antoine Spicher and Olivier Michel. Declarative modeling of a neurulation-like process. *BioSystems*, 87(2-3):281–288, February 2007.

[SMG04] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. A topological framework for the specification and the simulation of discrete dynamical systems. In *Sixth International conference on Cellular Automata for Research and Industry (ACRI'04)*, volume 3305 of *LNCS*, Amsterdam, October 2004. Springer.

[SMG10] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. Declarative mesh subdivision using topological rewriting in mgs. In *Int. Conf. on Graph Transformations (ICGT) 2010*, volume 6372 of *LNCS*, pages 298–313, September 2010.

[SMG11] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. *Understanding the Dynamics of Biological Systems: Lessons Learned from Integrative Systems Biology*, chapter Interaction-Based Simulations for Integrative Spatial Systems Biology. Springer Verlag, February 2011.

[Spi06] Antoine Spicher. *Transformation de collections topologiques de dimension arbitraire – Application à la modélisation de systèmes dynamiques*. PhD thesis, Université d'Evry, December 2006.

[Tof04] Tommaso Toffoli. A pedestrian's introduction to spacetime crystallography. *IBM Journal of Research and Development*, 48(1):13–30, 2004.