Évry
N° d'ordre : XXXX

Mémoire d'
Habilitation à Diriger des Recherches

présenté à l'
Université d'Évry Val d'Essonne

Spécialité
Informatique

There's Plenty of Room for Unconventional Programming Languages or Declarative Simulations of Dynamical Systems (with a Dynamical Structure)

par :

Olivier Michel

Présenté le 7 Décembre 2007 devant le jury composé de :

MM.	Jean-Pierre Banâtre	*Examinateur*
	Olivier Danvy	*Rapporteur*
	Jean-Louis Giavitto	*Examinateur*
	Gaétan Hains	*Rapporteur*
	Giuseppe Longo	*Examinateur*
	Przemyslaw Prusinkiewicz	*Examinateur*
	Susan Stepney	*Rapporteur*

Acknowledgements – Remerciements

Acknowledgements are too important to be expressed in a language that is not well mastered. For this reason, I choose to mostly write them in French.

On perçoit souvent le chercheur comme une entité isolée, déconnectée du réel et du monde des vivants. Ce n'est pas mon cas. Si j'ai choisi de travailler sur les langages de programmation, c'est qu'à mon sens le verbe est essentiel à la construction et au développement des idées. C'est par des rencontres que j'ai pu forger et développer les notions qui sont exposées ici. Ce sont des échanges (parfois houleux, selon le tempérament des interlocuteurs...) qui permettent de confronter une théorie, un programme, une façon de concevoir et de manipuler les objets qui deviendront les briques d'un langage à construire. C'est quand le langage actuel ne suffit plus pour décrire ce monde que l'on imagine qu'il devient urgent de poser les fondations d'un nouveau moyen d'expression.

J'ai été touché par l'amabilité de tous les membres de mon jury qui me font l'honneur de leur présence. Chacun d'eux a fait preuve d'une grande disponibilité afin de se libérer en une période de l'année notoirement chargée en ce genre d'événement.

C'est lors de l'organisation de la conférence « Unconventionnal Programming Paradigms » en 2004 que j'ai fait la connaissance du Professeur Jean-Pierre Banâtre, un des concepteurs du langage Γ, langage d'une rare élégance qui a été une des inspirations pour développer le langage MGS. Je le remercie vivement d'avoir accepté d'être membre du jury.

C'est durant UPP que j'ai rencontré le Professeur Olivier Danvy. Les discussions que nous avons eues à cette occasion autour des langages, des interpréteurs, des machines virtuelles et de la compilation en général, m'ont éclairé sur les liens forts qui existent entre tous ces domaines. Je suis très honoré qu'il ait accepté d'être rapporteur.

Je croise régulièrement la trajectoire du Professeur Gaétan Hains de par ses travaux et par l'intérêt qu'il veut bien porter aux miens. Je lui suis très reconnaissant pour ses encouragements, ses questions stimulantes et je le remercie d'avoir accepté d'être rapporteur dans mon jury.

L'étendue et la variété des travaux du Professeur Giuseppe Longo représentent pour moi un modèle de parcours scientifique. Son récent ouvrage *Mathématiques et sciences de la nature* résonne en moi et je me reconnais dans ce que j'y lis. C'est un grand honneur de le compter dans mon jury.

L'entrée dans le XXI^{e} siècle a été marquée par ma rencontre déterminante avec le Professeur Przemyslaw Prusinkiewicz. L'émerveillement ressenti devant la lecture de l'ouvrage *The Algorithmic Beauty of Plants* s'est depuis poursuivi dans de nombreuses discussions et travaux qui m'ont beaucoup stimulé et ont inspiré des travaux que l'on retrouve dans ce document. Je tiens à le remercier tout particulièrement de m'avoir fait l'honneur d'accepter d'être membre de mon jury.

I have met Professor Susan Stepney at the University of York during the "Grand Challenge in Non-Classical Computation International Workshop" in 2005. I have been enlightened by her talk and then by her papers. It is a great honor to have her as a referee in my jury. It is a great recomfort and an example to see people working fruitfully in so many different fields like formal methods, non-standard computation, security, etc.

Dans cette histoire, la rencontre avec Jean-Louis Giavitto a été essentielle. Ce sont des discussions en 1992 autour d'un café sur les (déjà) vieillissants langages de programmation qui ont marqué le début d'une aventure dont ce document n'est qu'un élément. Qu'il recoive mes remerciements les plus vifs pour les si nombreuses discussions que nous avons eues et son exigence constante.

Des travaux de recherche se déroulent dans des institutions. Parmi celles-ci, je tiens à exprimer ma gratitude au CNRS, au Genopole®, et à l'université d'Évry. Ces travaux ont débuté au LRI et se sont poursuivis au laboratoire LaMI, devenu depuis le laboratoire IBISC, de l'université d'Évry. Je tiens à remercier le Professeur Pascale Le Gall, responsable de l'équipe SPECIF dans laquelle je me trouvais jusqu'à 2006, ainsi que le Professeur Hanna Klaudel responsable de l'équipe LIS dans laquelle je me trouve actuellement.

Les remerciements ne seraient pas complets s'ils n'incluaient, parmi toutes les personnes avec qui j'ai travaillé, celles qui ont été essentielles pour les développements présentés ici : Julien Cohen et Antoine Spicher ont été de remarquables et redoutables étudiants qui ont marqué de leur présence et de leur gentillesse le projet MGS. Je remercie aussi tous les étudiants de Maîtrise et de DEA qui ont participé au projet : Christof, Damien, Emmanuel, Fabien G., Fabien T., Jean-Vincent, Lionel, Nicolas, Sami, Valérie, Yann.

Enfin, j'aimerais clore ces remerciements en exprimant ma gratitude à tous ceux qui m'ont chaleureusement accueilli et qui font de l'activité de recherche un véritable plaisir : Pierre Barbier de Reuille, Georgia Barlovatz Meimon, Hugues Berry, Daniel Coore, Franck Delaplace, Peter Dittrich, Christophe Godin, Frédéric Gruau, Florent Jacquemard, Jean-Paul Sansonnet, Pietro Speroni di Fenizio, Olivier Temam, et tous ceux que j'ai oublié mais qui savent ce que je leur dois.

Contents

A language that doesn't affect the way you think about programming, is not worth knowing.

Alan J. Perlis
(in *Epigrams In Programming, ACM SIGPLAN, Sept. 1982.*)

We may most aptly say that the Analytical Engine weaves algebraical patterns just as the Jacquard-loom weaves flowers and leaves.

Augusta Ada Byron, Countess of Lovelace
(Comparing Babbage's Analytical Engine with the Jacquard-loom.)

[...]: while it was perhaps natural and inevitable that languages like FORTRAN and its successors should have developed out of the concept of the von Neumann computer as they did, the fact that such languages have dominated our thinking for over twenty years is unfortunate. It is unfortunate because their long-standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer far greater intellectual and computational power.

John Backus
(in *The history of FORTRAN I, II, and III. ACM SIGPLAN Notices, 13(8):165–180, August 1978.*)

[...] un peu moins de systèmes, un peu plus de systématique [...]

Paul Ricœur
(in *La critique et la conviction,* Hachette, 2002.)

Chapter I

Introduction

I.1 Introduction

This dissertation, as every habilitation manuscript, tells a story. And this story, as usual, can be told from different standpoints[1].

I.1.1 Chronologically

The most obvious way is to present the work achieved these last ten years chronologically (see figure II.1 page 12).

My research has first taken place in the 81/2 project and then in the MGS project. The former is devoted to the simulation of dynamical systems through the explicit specification of their trajectory using declarative streams. The latter investigates the simulation of *dynamical systems with a dynamical structure* through sophisticated data structures used to represent dynamic spaces and to manipulate them by rules.

Both projects gave rise to an experimental language, materialized by the development of an experimentation platform (interpreter, fragments of compiler, some static analysis tools, visualizing tools, numerous examples, etc.), to 40 papers[2] with about 10 coauthors and to the supervision of many students: among them, I supervised more than 10 Master's Theses and two Ph.D.'s theses [Coh04b, Spi06b] during the MGS project.

[1] It is not limited to the description of ten years of scientific activities that a story can be told from multiple points of view. I have been deeply moved by A. Kurosawa's movie "Rashōmon", which describes the very same "physical event" as *seen* by four different people corresponding to four different *levels* of reality. I believe that contemplating things from different standpoints is particularly important for someone who pretends to model and simulate.

[2] Including 13 international journals, 4 book chapters and 23 international conferences.

The 81/2 Project

The 81/2 project has grown in the 1990's in a computer architecture team developing new parallel architectures. As a matter of fact, stream representing trajectories are also sequential threads of control and multiple streams mean parallel threads of computation. In addition, the instantaneous state of a dynamical system is often a massive object (*e.g.*, in finite elements methods) that can be computed using data-parallel operations. My contributions in this project arise at a moment where the static core (*i.e.*, the features of the language that can be statically analyzed with respect to the parallel implementation) was well understood and where new dynamic data and control structures had to be designed and developed to face more sophisticated applications.

The MGS Project

These new sophisticated applications are simulations of *dynamical systems with a dynamical structure*. Paradigmatic examples, in the context of Évry[3] are given by developmental processes in biology. The modelling and the simulation of these processes are one of the goals of the *systemic biology* program. They have motivated the start of the MGS project [GM01b] soon after the involvement of the LaMI into bio-informatics.

MGS is a rule-based language. A rule is similar to a rewriting rule but acts on almost any kind of data structure. This is possible through a unifying point of view where each data structure is equipped with its *neighborhood relationship*. The neighborhood relationship enables the definition of a suitable notion of *part* and a rule specifies the replacement of a part by another part. A set of rules is called a *transformation*. The definition of a transformation is similar to the specification of a function by case and transformations can be smoothly integrated into any functional language like ML.

I.1.2 Concepts and Tools

Chronology can be anecdotic. A scientific domain is often described through its objects of interests and through the tools used to investigate them. So, another way to present my work is to characterize the notions investigated and the tools developed for their study. Looking back on the loom of my work, it clearly appears that the *concept of space* constitutes its main weft and is summarized with the slogan **a data structure is a topological space**.

Data fields

The 81/2 project has focused on the handling of time. It is customary to articulate the concept of time on the three notions of *instant*, *duration* and *succession*. There is no duration in 81/2: an event takes place in an atomic instant and all events are strictly and totally ordered in a temporal succession whose beat is given by a global clock. At a given instant, the events are described by a set of values. Because all these values are logically accessible at the same time instant, they correspond to the notion of *physical field* [Boa83].

[3] I joined the University of Évry after my Ph.D. thesis at the University of Orsay. The University of Évry is strongly linked with the Genopole® institute whose mission is to create and foster a research cluster dedicated to genomics, post-genomics and related sciences. As a result, the computer science research laboratory LaMI from 2000 (and IBISC from 2006) develops strong interactions with biologists for the development of computer methods to support the modelling, the simulation, the analysis and the engineering of complex bio-systems.

Many physical quantities have different values at different points in space. For example, the temperature in a room is different at different points: higher near the heater, lower near the window. The velocity of a flow of water in a stream is larger in narrow channels and smaller where the stream is wide. In these two examples, there is a particular region of space which is of interest for the problem at hand and at every point of this region some physical quantities has a value.

The notion of physical field can be extended to include the idea of data structure. In this point of view, a data structure is a set of places filled by some value. The set of places exhibits some structure: a spatial organization. The neighborhood relationship in a data structure comes from the moves supported in that data structure: which element (place) is accessible from another element (place). For example, in a simply linked list, the elements are accessed linearly (the second after the first, the third after the second...). We call such a data structure a *topological collection*.

The notion of data field is an old one in computer science: it already appears in the development of recurrence equations and goes back at least to [KMW67]. The term "data field" seems to appear for the first time in [CiCL91, YC92] around the `Crystal` project [Che86]. As a matter of fact, the notion of data field is familiar to the systolic programming community [Qui86], especially in high-level approaches as in `Alpha` [GMQS89, QRW95]. B. Lisper has explicitly brought together the notions of data fields and of data parallelism [HL93] (the tutorial [Lis96] is a good introduction to these problems). This approach is also close to the notion of *pvar* or *xapping* [SH86] in the context of the *Connection Machine* [Hil85]. However, in these works, the set of places is simply an integer lattice (places are elements of $\mathbb{Z}^n$).

In the MGS project we have proposed to use topological notions to formalize and unify the presentation of a data structure. The motivation is that we are interested only in the connections between the elements of a data structure, we do not need in a first stage to handle quantitative notions like a distance between elements for instance. Topology is usually presented through the notion of open and closed sets and focus on a point-set point of view, but, in the MGS project we rely on notions developed in *combinatorial topology* [Hen94] to formalize the neighborhood relationships of a topological collection. A topological collection is a field (more precisely a *chain*) on a topological space defined by a *cellular complex*. These notions are developed later in [GM02c] and Section III.3 and rely on algebra (which is especially attractive for computer science because algebraic objects are usually implementable on a computer, which is not obvious when dealing directly with point-set notions).

Group-Based Data Field

In my Ph.D. thesis, I have used a group presentation to define the set of places and their neighborhood relationships. The spatial structure can be depicted by the Cayley graph of the group presentation. This approach goes beyond integer lattice and makes possible the representation of trees (free groups), integer lattices (free Abelian groups), plane tessellation (Archimedian groups) and many kind of twisted and circular grids [Mic96d, Del02, Lar02]. The corresponding data structure is called a *GBF* for *Group-Based data Field*. GBF based on Abelian groups have been implemented in the 8½ project and the corresponding library reused in MGS.

At the same time, Z. Róka considered Cayley graphs to extend the space and the communication links between the cells of a cellular automata [Rók94, Rók95]. However, the use of a group structure to describe a space structure has already been proposed in 1971 in [MP71]. The work is restricted to the study of the *dimension* of a space described by an Abelian group: "[...] We will show that for such groups the dimension equals the number of generators [...] Thus there seems to be a correspondence between Euclidean spaces and free Abelian groups.".

Newtonian and Leibnizian Spaces

At a first look, it may appear that a data field requires the *a priori* definition of its underlying space (its set of places). Indeed, this is the case for the GBF: the set of places are the group elements and has a meaning which is freed from any connection with a particular value attached to a place. This kind of space is called *Newtonian space* and corresponds to a container defined prior any contents.

Traditionally[4] there is another kind of space: if something has a position, that position can be defined only with respect to the other things. For example, the second element of a list can be the second one only because there is a first one. In this point of view, space is nothing apart the things it holds; it is only a consequence of the relationships between things; space is not a stage, which might be either empty or full, onto which things come and go.

Such kind of space is qualified as *Leibnizian*. Lists, but also sets and multi-sets are examples of fields on Leibnizian spaces. Leibnizian spaces are necessary because it is not always possible to foresee the shape of a data structure. For example, if we know for sure that a list will hold only 4 elements, we can use advantageously a quadruple or a vector of size 4 which provides a constant access time.

A Leibnizian approach is also necessary when the structure of a system cannot be statically known.

Declarative programming and Moving Values in a Graph

Not only the data structures of a program have a spatial structure: this also holds for control structures. One may think of the flow chart of a program which exposes the organization of the computations as a graph; but other spatial representations are possible, including higher-dimensional representations like "progress graph" [CES71] used to study the possible deadlocks of a set of concurrent processes (see [Gou00] for the use of homotopy to classify the behaviour of concurrent programs).

Here we focus on declarative programs and we assume that declarative programs are sets of equations. This statement is developed in Section III.4. The 81/2 programming language belongs to the declarative paradigm: the temporal succession of events are defined by a set of fixed-point equations (*i.e.*, recursive definitions). MGS programs are also sets of recursive definitions but between functions (however, imperative features exist in MGS).

These equations can be represented by a graph, the data-flow graph: a node represents an operator in the equations and edges route the data values between the computations. The concept of data-flow is an old notion, which goes back to at least to [Con63]. Since then, many kind of data-flow computation models have been developed; [KM66, TE68, Ada68, Kah74, AG77, Arn81] are amongst the first works. A data-flow graphs comes with its own computation process. We focus here on a network where every node is an autonomous calculator working asynchronously with respect to the other nodes. A node is a black box consuming data on its input links and producing data on its output links. A link is a FIFO with an unbounded capacity. The links are the only interactions between the nodes. The *functional data-flow*, or "pure data-flow" model, is a model where, in addition, the computations made by a node satisfy a functional relationship between the sequence of the input and the sequence of the output.

G. Kahn was the first to study a functional data-flow model and its relationships with the corresponding set of equations. What is now known under the name "Kahn Principle" says that in

[4] We refer here to the debate between I. Newton and G. W. Leibniz on the concept of space [Jam93].

functional data-flow, the sequence of values passing trough an edge is solution of the corresponding set of equations [Kah74]. The formal proof that the execution of the network of processes effectively solves the associated system of equations was done in [Wie80, Fau82].

This is a very important principle since it shows how a computation procedure allows us to solve some systems of equations. The compilation of 81/2 is based on this principle and this includes the computation of GBF specified by recursive definitions [GM01a].

Amalgams

The seminal article [Con63] introducing the concept of data-flow sketches the structuring of a program in computation modules, autonomous and independent, communicating by sending data (discrete items) among half-duplex links. The structure is given by a static graph, that is, a graph which is independent of the values carried through its edges.

This situation is reminiscent of the Newtonian vision and, as one may expect, there is a Leibnizian point of view on data-flow graphs, where the graph is built as the computation proceeds. For example, the computation of the value returned by the application of a recursive function on some argument, is achieved by a data-flow graph that unfolds with each function application (the added part corresponding to a recursive call)[5].

Amalgams have been introduced to enable the construction of data-flow graphs by "gluing" together "open" or "incomplete" graphs. In term of set of equations, amalgams enable the specification of a set of definitions, the merging of two sets of definitions and the evaluation of an expression using a set of definitions. The underlying idea was to model the free references[6] that enable the gluing of graphs on the *border* of some abstract object, because it is through the border that objects can be assembled and pasted together. As the computation proceeds, some parts of the data-flow graph become "complete" (*i.e.*, there is no pending edge left) which triggers the execution of the data-flow graph and the substitution of the completed part by the computed value.

Dynamical Systems with a Dynamical Structure

A data-flow graph is an example of a dynamical system that computes the solution of a set of recursive definitions: the state corresponds to the values on the edges and the evolution is given by the processing of the values by the nodes.

Amalgams are very good examples of dynamical systems with an evolving structure. In amalgams, both the structure of the state (*i.e.*, the set of edges) and the evolution function (the set of nodes) is changing over time (new nodes are added to the data-flow graph when incomplete graphs are completed). We have introduced the term "**dynamical system with a dynamical structure**" to qualify such processes.

This situation contrasts with (the simulation of) a "real" dynamical system "found" in nature where the evolution function usually does not change: this evolution function corresponds to the application of the physical laws and such laws remain the same every time and everywhere. However,

[5] To be complete, one must say that the returned value can also be computed by a static, acyclic but infinite data-flow graph corresponding to the unbounded iteration of the unfolding. It can also be computed by a static finite but cyclic data-flow graph.

[6] A free reference is a name that does not refer to a definition at the time of its use. The concept of name is a central notion in the incremental construction of programs, at a practical level (after all, a *linker* is a tool that resolves free references amongst a set of compiled files) as well as at a theoretical level [BC90, LF93, HO96, LF96, DS96].

natural dynamical system can have a dynamical structure, especially in biology: it is sufficient that the structure of the state evolves.

This is indeed the case, *e.g.*, during embryogenesis: the generation of form is mediated by cellular processes such as the direction and number of cell divisions, changes in cell shapes, cell movement, cell growth, and changes in the composition of the cell membrane and extracellular matrix. So the differential expression of developmental genes is not sufficient to explain the advent of a given shape which is a collective process implying supra-cellular (tissue) events. In other words, there is a *feed-back loop between the form and its evolution via the processes inhabiting this form.* The acknowledgement of this feed-back loop in a dynamical system with a dynamical structure is compatible with any methodological reductionist view: indeed, the situation is similar in the "classical" dynamical systems where the definition of the evolution function implies a feed-back loop between the output and the input of the system.

The idea to simulate this kind of systems is to describe the evolution function through *local interactions*. We will show in Chapter III that these interactions naturally exhibit a topological structure and that the notions used to describe the space of a data-structure can also be used to describe the structure of the interactions. This makes MGS particularly suitable for the modelling and the simulation of such systems.

Modelling and Simulation of Morphogenesis

The project to model and to simulate a complex morphogenesis process goes back to the end of the 8½ project [Seg97] and the failure to achieve it in 8½ was one of the motivations for starting the MGS project. At this time, my focus was on the *C. elegans*, an organism whose complete cell lineage [SSWT83], neural circuitry, and various genes and their functions have been identified. So, a complete synthetic model of *C. elegans* cellular structure and function appears possible [KHL98]. This project has attracted the attention of computer scientists but, so far, only some parts of the development process have been formalized, see for example [KHK+03].

I have not pursued this project in the context of Évry. In 2000, a group of plant biologists have called for a major scientific effort to understand the biological machinery of a plant, *Arabidopsis thaliana*, with enough details to construct a virtual plant that can be used to examine every aspect of a plant's development [CEB+00] (Arabidopsis was the first flowering plant to be completely sequenced at that time). Towards the end of this extremely ambitious 10-years project, the researchers propose to integrate the knowledge gained through studies of single genes into a broader understanding of how these gene networks interact with one another to build cells and tissues. Finally by 2010, plant researchers hope to construct a complete "wiring diagram" of all the biological pathways of Arabidopsis and to build a numerical avatar integrating the cellular level up to the entire organism, making able to see a four-dimensional view of a plant that covers all the details from when the seed germinates to when the next generation seeds fall off the mother plant.

Now, at the end of 2007, we are still very far from reaching this goal, but some specific developmental processes have been modeled at a cellular level. In France[7], a CIRAD-INRA-INRIA project has chosen MGS as the language for the modelling and the simulation of the cell-cell signaling network during the growth of the shoot apical meristem in Arabidopsis [BdR05, BdRBCL+06]. This project is certainly one of the three or four most advanced apex modelling project at the international level (see also in the same PNAS issue the papers by P. Prusinkiewicz [SGM+06] and E. Mjolsness [JHS+06]).

[7] Independently of the above mentioned 2010 Grand Project.

I want to stress that our contribution, as computer scientists, relies on the provided tools (the MGS interpreter) and at a methodological level: people working in the MGS project have no competence in the involved biological processes. However, the notion of dynamical systems with a dynamical structure, the expressive linguistic support offered by MGS and the rapid prototyping abilities of the environment, have made it possible for us to sketch in two days with P. Barbier de Reuille the very skeleton of the model in less than two pages of code. Obviously it has taken then three years of hard work to collect and analyze the relevant data, to fill the details, to formalize the transport and the mechanical models, to display the results of the simulations, to debug it and, finally, to validate the resulting model by new "real" laboratory experiments.

I.1.3 The Next Step: Synthetic Biology

Even if we are very far from the holy Grail of systemic biology, integrated models of biological processes from the molecular level up to the entire organism, a new frontier is envisioned by dreaming biologists and computer scientists.

Since early 2000, researchers in various fields (physics, chemistry, computer science and biology) address biology from an engineering point of view: by creating *standards* (definition of functions, libraries...), *abstractions* (organizing functions by levels) and *decoupling* (separating conception and realization), biology has undergone a similar shift to that of electronics when in 1957 the planar technology for producing transistors was "discovered". *Synthetic biology* [SS78, End05] focuses on the redesign of biological systems for specific purposes. It is not any more the analysis of living organisms for its sole understanding but for the systematic design of artificial biological entities. As a matter of fact, synthetic biology goes way further than genetic engineering by its aims and tools: it proposes to rewire genetic networks from existing organisms to detect[8] or produce[9] chemicals but also to shape new organisms by building them in a bottom-up fashion (from DNA to physiology through metabolic pathways).

But synthetic biology is only at its first steps. Generic tools for developing complex applications from a high-level specification down to the design of genetic networks are still lacking. It is our thesis that unconventional languages have a key role to play to offer new computing models crossing all the levels of systemic biology.

I.2 Organization of the document

The next chapter details my *curriculum vitae*: my research activities since 1992 are given, then my teaching activities and student supervision, my scientific collaborations are reviewed, my administrative tasks are roughly sketched and this chapter is closed by the list of all my publications so far.

Chapter III motivates the work that I have done in the field of declarative languages for unconventional computing. We recall the motivations for designing new programming languages and the key role of spatial relations in existing languages and applications. The generalization of the spatial relationships has led to the unified viewpoint of data structures as topological spaces; a general form of rewriting on those spaces is also presented. The chapter ends by the presentation of papers that are essential to understand the concepts sketched.

[8] For example TNT [Gib04], genetically modified Arabidopsis seeds that change their color while being grown over land mines [Are06], etc.

[9] For example, a precursor of a malaria healing drug, the artemisinic acid [RPO+06], the production of hydrogen with starch and water [ZEM+07], etc.

Designing programming languages is only one third of the work: there is still the necessity to use these languages against real problems to see whether they are well suited. **Chapter IV** shows that the domain of applications of my research, namely dynamical systems with a focus in biology, can be efficiently handled using the concepts of topological spaces and local rewriting by recovering the topology of interactions of the system. Again, the chapter is closed by two sets of essential papers, dealing with biological and non-biological applications.

The last third of work a programming-language designer has to provide is briefly given in **Chapter V**: developing a language generally requires to develop additional tools (for visualisations, for facilitating the writing of redundant code, etc.) that will help the user. Some of the implementations that I have done or participated to are given and the chapter ends by the presentation of papers focusing on implementation details and issues.

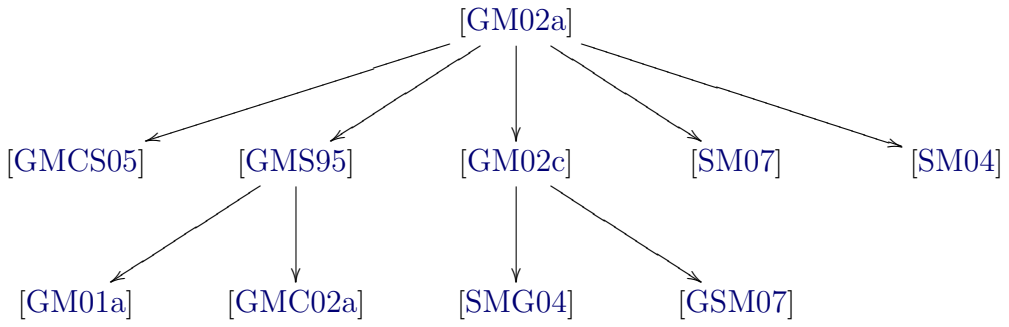
Propositions for further research are made in **Chapter VI**.

Chapters III–V end with a section presenting a selection of papers related to them. In the electronic form of this document, clicking on the provided link enables to retrieve the corresponding paper. A booklet gathering these works is also available and a page reference to this booklet is provided for each reference.

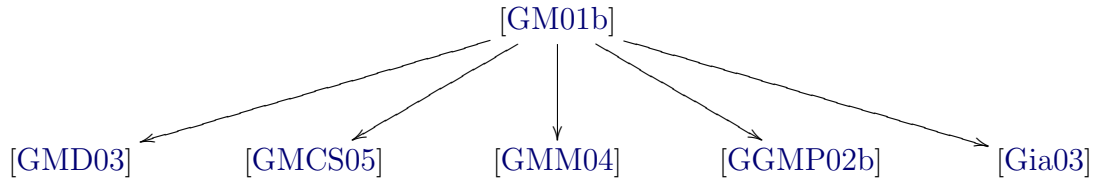
I.3 Multiple Reading Pathways

In this section we give some reading dependencies between the articles, on a thematic basis, not chronogically. These are various pathways to traverse the work evocated here. On the electronic version of this document, the references are clickable and forward to the corresponding electronic version of the denoted work.

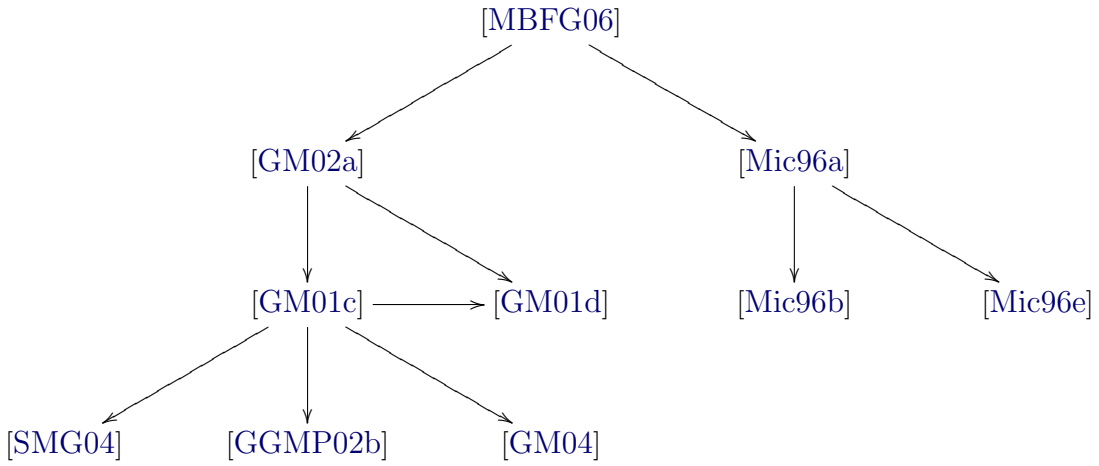
I.3.1 Topological Notions in a Programming Language



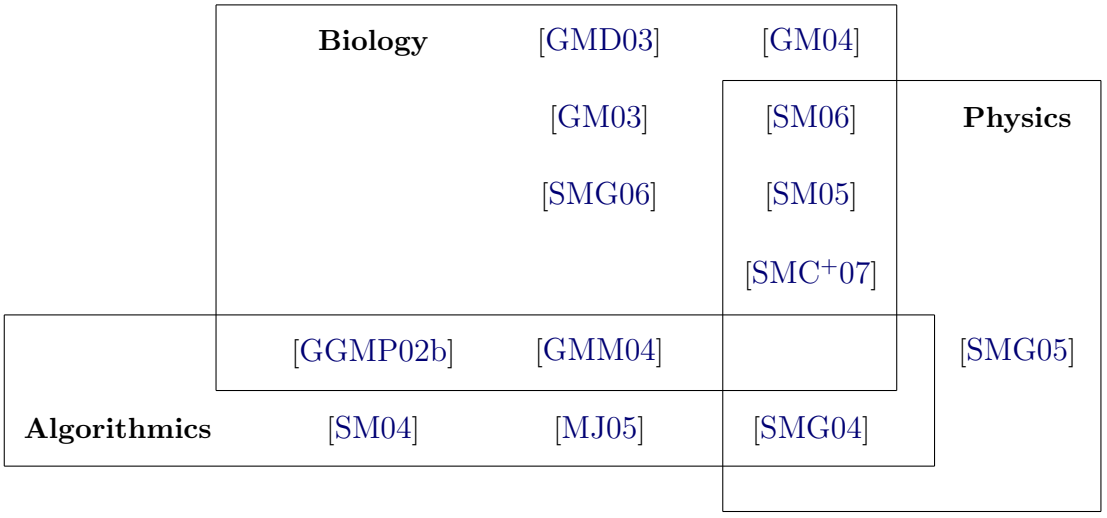
I.3.2 Dynamical Systems with a Dynamical Structure



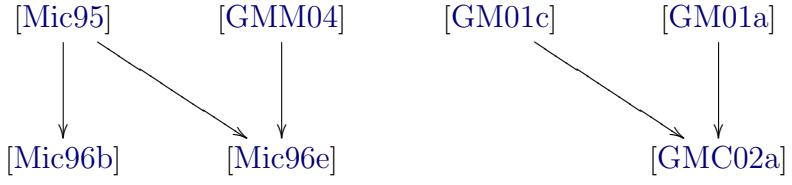
I.3.3 Unconventional Languages



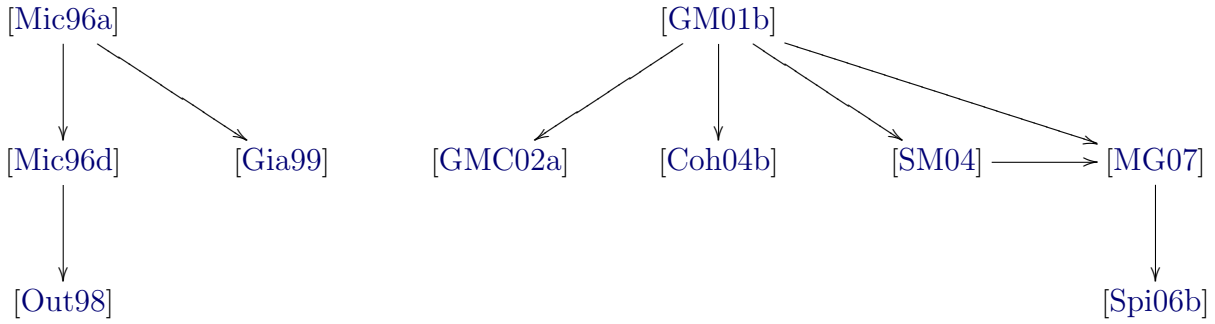
I.3.4 Simulations And Applications



I.3.5 Declarative Languages



I.3.6 Implementation Techniques



Chapter II

Curriculum Vitae

Olivier Michel,

37 years old, French citizenship.

IBISC Lab. - FRE 2873 CNRS - LIS project
Tour Évry 2 - 523, Place des terrasses de l'Agora
91000 Évry Cedex
phone: +33 (0)1 60 87 39 10
email: `michel@ibisc.univ-evry.fr`

Education

1996 Ph.D. at University Paris XI, Orsay.
Title: *Dynamical Representations of Space in a Declarative Simulation Language.*
Research Lab.: L.R.I. (U.M.R. 8623 du C.N.R.S.)
Chairman: L. Puel (LRI - U. of Orsay - Paris XI), Referees: G. Bernot (LaMI - U. of Évry) & P. Sallé (ENSEEIHT/INPT), Examiners: E. Ashcroft (Arizona State University) & J.-L. Giavitto (LRI - CNRS), Supervisor: J.-P. Sansonnet (LRI - CNRS).

1992 DEA[1] **M.I.S.I.** from the University Paris VI, Versailles Saint-Quentin.
Research Masters thesis : "A software framework for the experimentation of parallel simulation algorithms: applications to Time-Warp", supervisor V. Vèque (LRI).

Positions

September 2000 *Associate Professor* at University of Évry.

September 1997 *Assistant Professor* at University of Évry.

[1] The French DEA is an A+5 research oriented degree from University and a pre-requisite for Ph.D studies. The training period for the DEA degree is concluded by a research Masters thesis.

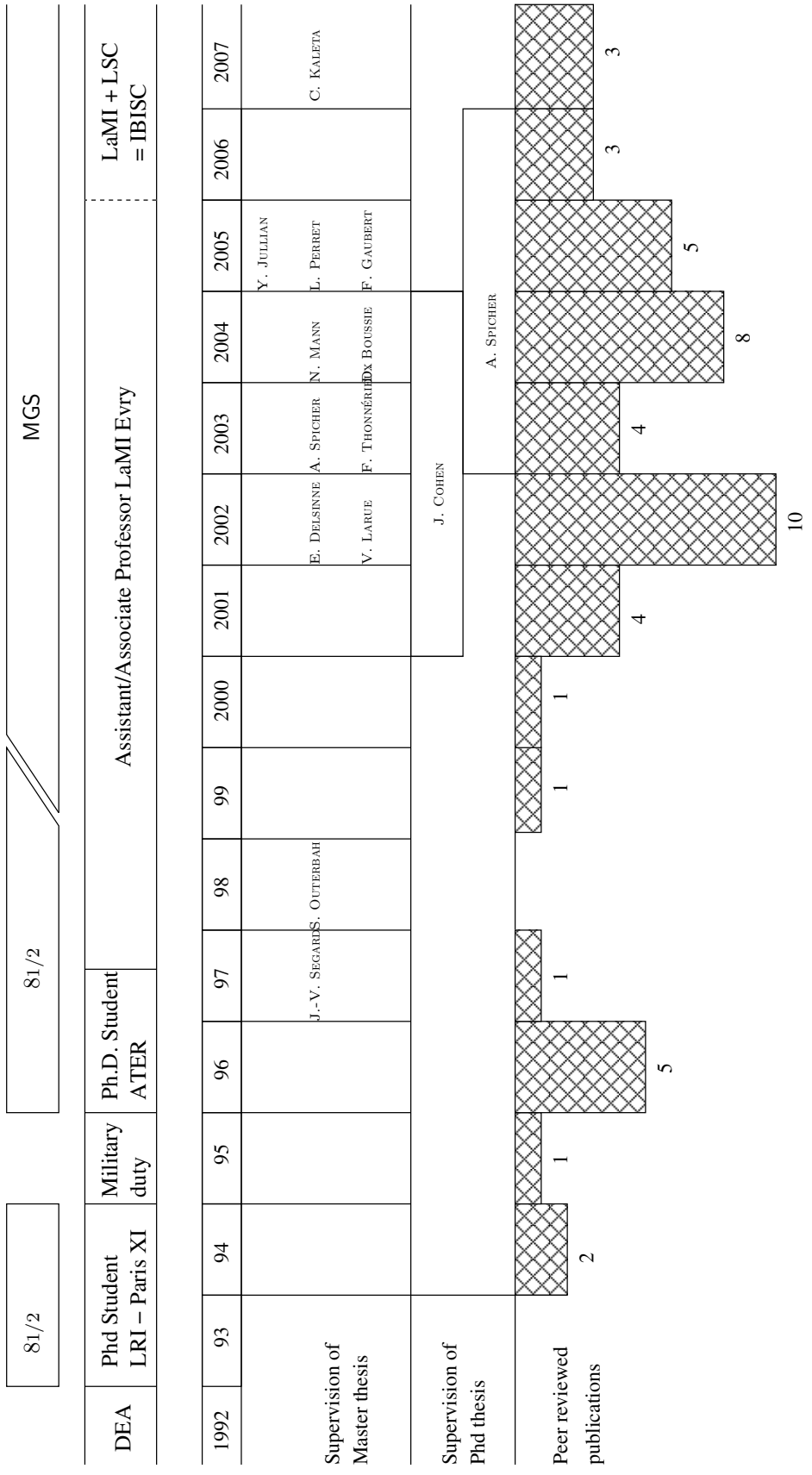


Figure II.1: Short presentation of my research and supervising activities.

II.1 Research Activities

My research activities are focused on *declarative languages* and are centered on the following two directions:

1. dynamicity and time-representation (1992-2000),
2. the development of topological data-structures for the simulation of dynamical systems with a dynamic structure (2000-).

II.1.1 1992-2000: Introducing Dynamicity in the $8_{1/2}$ Programming Language

The principal aim of the $8_{1/2}$ project[2] was the definition of a high-level parallel declarative computation model for the simulation of large dynamical systems. This computation model has been materialized by the development of a programming language also called $8_{1/2}$, and has been validated by a set of experimentation platforms (interpreter, compiler, visualizing tool, workbench for the data distribution strategies, etc.) My work has focused on the definition, the study and the development of dynamical representations of space within a declarative framework.

I have introduced in $8_{1/2}$ two new data structures, *GBF* and *amalgams*, by proposing a formalization and studying their implementation. GBF (group-based data-fields) allow the definition of regular and homogeneous spaces, while the amalgams allow the construction, through some computations, of heterogeneous and *ad-hoc* spaces. These two new notions have direct applications in the domain of simulation of highly dynamical systems (as for example growing processes in biology). They also find a direct application in computer science by defining a new framework for (1) the definition, analysis and implementation of recursive data (the GBF define for example a unified framework encompassing the notion of array and tree); (2) the definition and formalization of new incremental programming mechanisms that were arising then in languages like `Java` (amalgams allow the formalization of an instantiation mechanism by implicit name capture, and program extension) and (3) allow a declarative construction of data-flow graphs. I first studied GBF and amalgams on their own and they have been afterwards introduced and integrated into the declarative framework of the language $8_{1/2}$ to define the language $8_{1/2_{\mathcal{D}}}$.

In that work, I have shown, through numerous and significant examples, the pertinence of the choices made. They put into evidence the gain of expressiveness brought by the improvement of the notion of space, and by the primitives allowing the definition of objects onto those spaces. The notions of GBF and Amalgam allow the definition, in a very concise manner, of regular and irregular data structures, within a declarative framework, and open some new perspectives for the parameterization and the incremental construction of declarative programs.

II.1.2 2000- : The MGS Project

The Context of the Project

My current research activities take place in the MGS project[3]. This project evolves following two complementary directions:

1. to study and develop the integration of topological notions and tools in programming languages;

[2] The web site of the project is: `http://www.ibisc.univ-evry.fr/pub/Otto/`

[3] The home page of the project, where the language interpreters are available is `http://mgs.ibisc.univ-evry.fr`

2. to apply those notions and tools to the conception and development of new data- and control-structures that are expressive and efficient for the modeling and simulation of dynamical systems with a dynamic structure.

This research is concretized by the development of an eponymous experimental programming language and its application to the modeling and simulation of dynamical systems. We have a particular focus in the field of biology and morphogenesis.

High-level Domain Specific Programming Languages

My research work in that domain focuses on the definition of new data- and control- structures allowing the representation of spatial and temporal data for the modeling and simulation of dynamical systems in a rigorous framework, close to the end-user and the mathematical tools used in that domain (dynamical systems in biology, chemistry and physics).

The two main applications domains that have motivated the development of these new mechanisms are the analysis and representation of complex relations (for example spatial ones) and the simulation of dynamical systems where the structure has to be computed jointly with the evolution of the system (as it is the case in developmental biology, at any level from the cells to the individuals).

A privileged application domain was the modeling and simulation in systemic biology (more specifically in post-genomic and in integrative simulation of developmental processes). We are now (since 2007) considering the modeling, simulation and conception of new computational resources brought by nanosciences and molecular biology.

This research is based on the development of a new computation paradigm (by generalizing the approaches of the chemical computing, membrane computing, Lindenmayer systems and cellular automatons) and new tools for the interpretation, typing and efficient compilation of declarative languages. This research meets the fields of the "self-*" (self-sustaining, self-healing, self-organizing) systems that are emerging in the software community and whose long-term goal is to identify and develop in today's computers the fundamental requirements to ensure properties of flexibility, adaptivity, robustness, self-healing that are common to the complex systems of living matter.

II.2 Teaching Activities and Student Supervision

II.2.1 Teaching Activities

Since 1997, I have taught each year between 192h and 240h (for a total for the last 8 years of over 1600 hours), but for 2006 and 2007, since I was on sabbatical[4]. My teaching covers the following domains and levels:

- 2nd year at University ("DEUG SDM"): the `C` language,
- 2nd year at University ("DEUG MIAS"): Unix, operating system, network, development tools,
- 3rd year at University ("IUP-Miage"): logics, human-computer interaction,
- 3rd year at University ("Licence de Mathématiques"): data structure and algorithms,

[4]The sabbatical took the form of a "delegation CNRS" for 2005-2006 and 2006-2007.

- 3rd year at University ("Licence Informatique"): logics, functional programming, compilation,
- 4th year at University ("Maitrise Informatique"): λ-calculus, combinatory logics,
- 5th year at University ("DEA Informatique"): basic mathematics for the computer scientist, advanced programming, data-parallelism, unconventional languages, dynamical systems,
- 5th year at Engineer School ENSIIE ("Mastere Bio-Informatique"): modeling and simulation.

I also took part in 2003 in a mission of teaching at Bobo-Dioulasso (Burkina Faso, Africa) where I gave two lectures: one for 21 hours of mathematics and functional programming; one for 10.5 hours of logics.

II.2.2 Supervision of MsC Theses

I supervised the work of many Masters (DEA) trainees or 3rd year of engineer school:

Masters Theses with a Specific Computer Science Focus

1. S. Outerbah, 1998, "Introduction d'une notion de référence distante dans un formalisme de capture de noms et implémentation d'une plate-forme d'expérimentation", D.E.A. Informatique, 100% supervision.
2. E. Delsinne, 2002, "Structures de données indexées par un groupe, isomorphismes de GBF abéliens et extensions aux structures automatiques", E.N.S. Cachan and University of Rennes-I, 50% supervision.
3. V. Larue, 2002, "Structures de données indexées par un groupe : représentation graphique et extension au cas non abélien", D.E.A. INFO, 100% supervision.
4. A. Spicher, 2003, "Typage et compilation de filtrage de chemins dans des collections topologiques", D.E.A. AMIB, 100% supervision.
5. F. Thonnérieux, 2003, "Réalisation d'une interface graphique pour le traitement des sorties du programme MGS", IIE, 100% supervision.
6. L. Perret, 2005, "Intégration des types de données algébriques dans MGS", École Centrale Paris, 100% supervision.
7. Y. Jullian, 2005, "Conception et développement d'un éditeur graphique de filtre pour MGS", IIE, 100% supervision.

Masters Theses with a Specific Bio-Computing Focus

1. J.-V. Segard, 1997, "Modèles de morphogénèse biologique dans un langage déclaratif de simulation", D.E.A. of Cognitive Science of L.I.M.S.I, U.P.R. 3251 du C.N.R.S, 50% supervision.
2. N. Mann, 2004, "Hyperstructures et modélisation de chimie artificielle dans le langage `MGS`", D.E.A. AMIB, 100% supervision.
3. D. Boussié, 2004, "Simulation en `MGS` du déplacement du spermatozoïde du nématode *Ascaris Suum*", D.E.A. AMIB, 100% supervision.

4. F. Gaubert, 2005, "Simulation stochastique et modélisation de chimie artificielle dans le langage MGS", D.E.A. AMIB, 100% supervision.

5. C. Kaleta, 2007, "Outils de visualisation pour la simulation de systèmes dynamiques à structure dynamique", Erasmus M1 student in Computer Science, Universität Jena & Université d'Évry.

II.2.3 Supervision of Ph.D.

I supervised for 100% the scientific work of 2 Ph.D. students[5]. These students are:

- J. Cohen, 2004, "Intégration des collections topologiques et des transformations dans un langage fonctionnel". J. Cohen is now assistant professor at the University of Nantes.

  J. Cohen has participated in the development of an interpreter for the MGS language using a higher-order abstract syntax scheme and in the definition of a generic pattern matching algorithm for the topological collections. He also developed a typing strategy for a sub-part of the language.

- A. Spicher, 2006, "Transformation de collections topologiques de dimension arbitraires. Application à la modélisation de systèmes dynamiques". A. Spicher is now a post-doc at INRIA Lorraine.

  A. Spicher has developed a large number of applications of MGS in the biological domain. He has also done a great deal of theoretical work in the definition of a general formalization of the notion of topological collection based on abstract combinatorial topology and a probabilistic semantics for MGS.

II.2.4 Ph.D. Committee

I was one of the examiners of the Ph.D. thesis jury of A. Merlin with H. Thuillier (chairman), R. Di Cosmo & E. Violard (referees), Q. Miller (examiner), G. Hains (supervisor). The defense took place in 2004, december the 7$^{\text{th}}$ at the LIFO, University of Orléans.

II.3 Scientific Collaborations

The French scientific community in computer science is structured around national working groups in various research fields. The exact administrative structure varies in time and has been called AS, GDR, PRC, etc.

II.3.1 Past Collaborations and Scientific Involvement

My past scientific activities included:

- The members of the 81/2 project participated to the "PRC GDR C3" and then to "PRS" and more precisely to the working group "ParaDe" directed by L. Bougé.

- The 81/2 project took part in the new "GDR de Programmation", parallelism working group.

[5] under the administrative responsibility of Jean-Louis Giavitto, Research Director at CNRS.

- In 1997-2000, we set up with F. Delaplace from the LaMI, and with J.-L. Giavitto, F. Cappello and C. Germain from the LRI, a working group on "*Meta-Computing and Distributed High-Performance Computation*".
- I am a former member of the "GDR ALP" with the proposition of the working group "LODEC" (Languages and Tools for Deduction under Constraints) in the action "parallel and concurrent programming using logical and functional languages and debugging tools".
- I was a member of the editorial board of the French computer-science journal TSI, from 1998 to 2002.
- I served as PC in the following conferences: CC'99, CC'00, JFLA'04, RULE'04.
- I was a member of the CNRS AS "Topologie et Calcul" lead by E. Goubault (CEA),
- I was a member of the CNRS AS "Nouveaux Modèles et algorithmes de graphes pour la biologie" lead by M. Habib (LIRMM).
- I was a member of the CELLIA working group at IBISC; I take part to the working group "Simulation en génomique : vers l'épigénèse" and "Synthetic Biology" hosted by `genopole`®.
- I was the head of the PC for the sixteenth edition of the JFLA (in 2005), the only French conference on applicative languages.

II.3.2 Involvement in ACI and ANR Programs

I am currently involved in the following ACI and ANR programs:

- I am the member of the ACI IMPBIO project "VICANNE" headed by J.-P. Mazat.
- I co-organized with J.-P. Banâtre, P. Fradet and J.-L. Giavitto, the conference UPP'04 (Unconventionnal Programming Paradigms). This conference received some support by the EEC (IST program) and the NSF. Its goal was to gather for three days researchers in the field of unconventional programming (*bio-inspired computing*, *chemical computing*, *amorphous computing*, *generative programming* and *autonomic computing*).
- I co-organized in July, 18$^{\text{th}}$, with F. Gruau (LRI) and H. Berry (INRIA FUTURS) a one day workshop on *Amorphous Computing*, with D. Coore, one of the founder of that computation model. The web page of the workshop is is `http://amorphous.ibisc.univ-evry.fr/`
- I am the head of an ACI "*Jeune Chercheur*" since 2004, "NANOPROG : une approche langage pour le nanocalcul et la simulation des nanosystèmes biologiques"; other participants include J. Cohen, and A. Spicher of the IBISC Lab, and F. Gruau of the LRI Lab.
- I am part of the "ANR blanche AUTOCHEM" headed by T. Priol (IRISA) with J.-P. Banâtre (IRISA), P. Fradet (INRIA Grenoble), J.-L. Giavitto (IBISC - U. of Évry), H. Klaudel (IBISC - U. of Évry), A. Spicher (LORIA), T. Collette (CEA LIST), C. Gamrat (CEA) and V. David (CEA).
- I am an *advisor* (with A. Spicher) in the French (Paris) team for the iGEM'07 international competition on *Synthetic Biology*. The website of iGEM'07 is `http://parts.mit.edu/r/parts/igem/index.cgi`

- I am maintaining with S. Bottani (MSC - University Paris 7) two web sites on Synthetic Biology for the French community: http://sb.ibisc.fr/ and http://www.ibisc.univ-evry.fr/pub/pmwiki/pmwiki.php

II.3.3 "Delegation CNRS"

In 2005–2006 and 2006–2007 I was the recipient of a "Delegation CNRS". During those two years, I have visited many of my collaborators using my grant "ACI NANOPROG".

During the first year of the delegation, I was still in charge of the administrative responsibility of the first Masters year ("M1") at the University of Évry.

II.3.4 International Collaborations

I have international collaborations with:

- P. Prusinkiewicz, University of Calgary, Canada, Dpt of Computer Science. Since 2000, we have met a couple of times and are working on the problem of spatial representations in programming language and the specification of dynamical systems with a dynamical structure.
- G. Malcolm, University of Liverpool, U.K, Dpt of Computer Science. We are collaborating on the study of the rewriting systems for the modeling of biological systems.
- M. Gheorghe, University of Sheffield, U.K, Dpt of Computer Science. We are working on formal aspects of languages for bio-computing (P systems, molecular X machines, ...).
- P. Dittrich, Universität Friedrich-Schiller, Jena, Germany, Bio Systems Analysis Group. We have started a collaboration in the use of MGS for the representation in artificial chemistry of his notion of *organizations*. I have visited the U. of Jena for 2 weeks.
- D. Coore, University of the West Indies, Jamaica, Dept of Computer Science. We have initiated in 2007 a collaboration on the definition of a new programming languages based on the notions developed in *amorphous computing* and aimed at defining the behaviour of large population of asynchronous, unreliable, local communicating *living matter*. D. Coore has come for one month as a visiting Professor in Évry in July 2007.

II.3.5 National Collaborations

I have national collaborations with:

- C. Godin, AMAP team, Modeling Plants unit, CIRAD-INRA-INRIA on problems of computer multi-scale representations in languages. MGS has been used in the Ph.D. thesis of C. Godin's student P. Barbier and lead to a publication in the PNAS.
- I took part in the "Plan Pluri Formation (PPF)" between the University of Évry and the University of Poitiers on "Méthode et outils formels pour l'animation de modèles topologiques et géométriques. Application à la simulation en post-génomique", 2002-2005. I participated to the study of the use of G-maps for simulation in biology.
- H. Berry of the ALCHEMY team, INRIA/PCRI of L.R.I, University Paris XI on the computational properties of a programmable material in the field of synthetic biology.

- P.-E. Moreau, INRIA Lorraine. We have been working together during J. Cohen's Ph.D.'s work on associative-commutative matching developed in the ELAN project.
- F. Jacquemard of L.S.V./INRIA on the exploration of large state space in the field of cryptographic protocols and the use of topological tools for the representation and composition of musical scores.

II.4 Administrative Tasks

In our growing University, administrative tasks are required at two levels: for the research lab and for the teaching department. I take part in both as described below.

II.4.1 Research Lab

At the level of the research lab, I am very involved in all regular tasks. Among them, I take part in:

- I am a former member of the "CARI" (*Center for Administration of Computer Resources*) of the University of Évry where I represented the laboratory from 1997 to 2002.
- I am a member of the "Laboratory Council" since 2000 where I represent the assistant/associate professors.
- Since 1997 I am a member of the *hardware and software* commission of the lab where we define and manage the computer resources of the lab.

II.4.2 Teaching Department

Since the beginning of our teaching department, we are lacking administrative help. For that reason, we have to manage the department by ourselves. Among the various activities in which I took part, I can cite:

- I am a member, since 2000, of the "commission de spécialistes" (this commission reviews and hires the faculty members in computer science) where I represent the assistant/associate professors.

  This year, I am he "vice-president" of the commission for the assistant/assiociate professors ("collège B").
- During 3 years, I was in charge of the 4^{th}-year courses in computer science at the University ("Maîtrise Informatique").
- I am this year in charge of the 3^{rd}-year courses in computer science at the University ("L3 Informatique")

Each time, it involves various tasks like finding teaching assistants for the courses, making the course schedules, interact with the administrative staff for setting-up the jurys, reviewing the student applicants files, updating the "règlements du contrôle des connaissances", ...

I also took part to the LMD-committee which produced the new teaching structure based on the LMD structure: "Master Sciences et Ingénierie, mention Informatique et Systèmes". This team

effort required a great deal of work since we had to re-think the whole structure of the teaching activities, from scratch.

II.4.3 Administration of Computer and Network resources: 1998-2002

From the end of 1998 to the beginning of 2002, I have managed **all alone** all the computer resources and network of the research lab (computers, X terminals, printers, active network resources, ...) and the common services (home account savings, e-mail, DNS, ftp, web server, ...). In 2000, the lab has moved from it previous location to its current one: I had to define all the new system architecture from network organization to the distribution/update of computer operating systems and software (Linux and Windows). After 2002, I have supervised the activity of E. Faure, our system engineer.

II.5 Software Developments, Publications and Communications

My research has always been organized in two parts: theoretical work embedded into concrete development of software tools, models and languages.

II.5.1 Software Developments

I took part in the following software developments, which are all open-sources projects:

- the 81/2 programming language, available at `http://www.ibisc.univ-evry.fr/pub/Otto/` and consists in over 36k lines of `ML` and `C` source code,
- a distributed version of the amalgam formalism, available at `http://www.ibisc.fr/~michel/amalgame.tar.gz` and consists in about 3k lines of `ML` and `C` source code,
- the MGS programming language, available by request at `http://mgs.ibisc.univ-evry.fr` and consists in over 50k lines of `ML`, `C++` and `C` source code; it includes many external libraries (`qhull` for the computation of Delaunay and Voronoï tessellations, `GNU gsl` for various random numbers generators, `nauty` for the computation of graph isomorphisms, ...)

II.5.2 International Journals

In each section, publications are sorted by alphabetic order using the first author as the key, then in chronological order. All publications are available at the url `http://www.ibisc.univ-evry.fr/~michel/WWW/bib.html`

[J1] **Olivier Michel**. Design and implementation of 81/2, a declarative data-parallel language. *Computer Languages*, 22(2/3):165–179, 1996. special issue on Parallel Logic Programming.

[J2] Jean-Louis Giavitto and **Olivier Michel**. The topological structures of membrane computing. *Fundamenta Informaticæ*, 49:107–129, 2002.

[J3] Patrick Amar, Pascal Ballet, Georgia Barlovatz-Meimon, Arndt Benecke, Gilles Bernot, Yves Bouligand, Paul Bourguine, Franck Delaplace, Jean-Marc Delosme, Maurice Demarty, Itzhak Fishov, Jean Fourmentin-Guilbert, Joe Fralick, Jean-Louis Giavitto, Bernard

Gleyse, Christophe Godin, Roberto Incitti, François Képès, Catherine Lange, Lois Le Sceller, Corinne Loutellier, **Olivier Michel**, Franck Molina, Chantal Monnier, René Natowicz, Vic Norris, Nicole Orange, Helene Pollard, Derek Raine, Camille Ripoll, Josette Rouviere-Yaniv, Milton Saier, Paul Soler, Pierre Tambourin, Michel Thellier, Philippe Tracqui, Dave Ussery, Jean-Claude Vincent, Jean-Pierre Vannier, Philippa Wiggins, and Abdallah Zemirline. Hyperstructures, genome analysis and I-cells. *Acta Biotheoretica*, 50, 2002.

[J4] Jean-Louis Giavitto, **Olivier Michel**, and Julien Cohen. Pattern-matching and rewriting rules for group indexed data structures. *ACM SIGPLAN Notices*, 37(12):76–87, December 2002.

[J5] Jean-Louis Giavitto, **Olivier Michel**, and Franck Delaplace. Declarative simulation of dynamicals systems: the $8_{1/2}$ programming language and its application to the simulation of genetic networks. *BioSystems*, 68(2–3):155–170, feb/march 2003.

[J6] Jean-Louis Giavitto and **Olivier Michel**. Modeling the topological organization of cellular processes. *BioSystems*, 70(2):149–163, 2003.

[J7] Jean-Louis Giavitto and **Olivier Michel**. Modeling the topological organization of cellular processes. *Physics of Life*, August(3), 2003. See `http://www.physicsoflife.com/index.html`. ("Physics of Life" is an Elsevier electronic Journal selecting articles that have been published in 22 contributing journals from Elsevier Science, covering Physics, Biology, Chemistry and Medicine with a focus on biological physics research).

[J8] Jean-Louis Giavitto, Grant Malcolm, and **Olivier Michel**. Rewriting systems and the modelling of biological systems. *Comparative and Functional Genomics*, 5:95–99, February 2004.

[J9] Antoine Spicher and **Olivier Michel**. Declarative modeling of a neurulation-like process. *BioSystems*, 87:281–288, February 2006.

[J10] **Olivier Michel**, Jean-Pierre Banâtre, Pascal Fradet, and Jean-Louis Giavitto. Challenging questions for the rationales of non-classical programming languages. *International Journal of Unconventional Computing*, 2006.

[J11] Antoine Spicher, **Olivier Michel**, Mikolaj Cieslak, Jean-Louis Giavitto, and Przemyslaw Prusinkiewicz. Stochastic P systems and the simulation of biochemical processes with dynamic compartments. *BioSystems*, In press, 2007.

The two following publications are in a French journal, *Technique et Science Informatique* (Computer Science and Technique).

[J12] Jean-Louis Giavitto, **Olivier Michel**, Jean-Pierre Banâtre, and Pascal Fradet. Modèles de programmation non-conventionnels. *Technique et Science Informatique*, 23:177–186, 2004. Compte-rendu de l'atelier international UPP'04. (not reviewed)

[J13] Antoine Spicher and **Olivier Michel**. Représentation et manipulation de structures topologiques dans un langage fonctionnel. *Technique et Science Informatique*, 2007.

II.5.3 Edition

I have been the (co-)editor of two conference proceedings and one special issue in a journal:

[E1] Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel, editors. *Unconventional Programming Paradigms (UPP'04)*, volume 3566 of *LNCS*, Le Mont Saint-Michel, France, September 2005. ERCIM – NFS, Springer Verlag. Revised, selected and invited papers. 367 p. ISBN: 3-540-27884-2. http://www.springeronline.com/3-540-27884-2.

[E2] Olivier Michel and Pierre Weis, editors. *Seizièmes Journées Francophones des Langages Applicatifs (JFLA'05)*, number 16. INRIA, 2005. http://jfla.inria.fr/2005/actes/actes-jfla-2005.tgz.

[E3] Langages applicatifs, théorie et applications. *Technique et Science Informatique*, Hermes Science, In press. 2007.

II.5.4 Book Chapters

[B1] A. Zemirline, P. Ballet, L. Marcé, P. Amar, P. Ballet, G. Bernot, F. Delaplace, Jean-Louis Giavitto, **Olivier Michel**, J.-M. Delosme, R. Incitti, P. Bourgine, C. Godin, F. Képès, P. Tracqui, V. Noris, J. Guespin, M. Demarty, and C. Ripoll. *Modelling and Simulation of biological processes in the context of genomics*, chapter "Cellular-automata, Reaction-Diffusion and Multiagents Systems for Artificial Cell Modelling". Hermes, July 2002. Also published as a tutorial chapter of the proceedings of the workshop "Modélisation et simulation de processus biologiques dans le contexte de la génomique", 17-21 mars 2002, Autran, France.

[B2] Jean-Louis Giavitto, Christophe Godin, **Olivier Michel**, and Przemyslaw 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.

[B3] Jean-Louis Giavitto and **Olivier Michel**. *Molecular Computational Models: Unconventional Approaches*, chapter Modeling Developmental Processes in MGS, pages 1–46. Idea Group, 2004.

[B4] **Olivier Michel** and Florent Jacquemard. *An Analysis of a Public-Key Protocol with Membranes*, pages 281–300. Natural Computing Series. Springer Verlag, 2005.

II.5.5 Publications in International Conferences (with review)

[IC1] **Olivier Michel** and Jean-Louis Giavitto. Design and implementation of a declarative data-parallel language. In *post-ICLP'94 workshop W6 on Parallel and Data Parallel Execution of Logic Programs*, S. Margherita Liguria, Italy, 17June 1994. Uppsala University, Computing Science Department.

[IC2] **Olivier Michel**, Jean-Louis Giavitto, and Jean-Paul Sansonnet. A data-parallel declarative language for the simulation of large dynamical systems and its compilation. In Institute for System Programming of the Russian Ac. of Sci., editor, *SMS-TPE'94: Software for Multiprocessors and Supercomputers*, Moscow, 21–23 September 1994. Office of Naval Research USA & Russian Basic Research Foundation.

[IC3] Jean-Louis Giavitto, **Olivier Michel**, and Jean-Paul 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 *Lecture Notes in Computer Sciences*, pages 209–215, Beaune (France), 2–4 October 1995. Springer-Verlag.

[IC4] **Olivier Michel**, Dominique De Vito, and Jean-Paul Sansonnet. 81/2: data-parallelism and data-flow. In E. Ashcroft, editor, *Intensional Programming II: Proc. of the 9th Int. Symp. on Lucid and Intensional Programming*. World Scientific, May 1996.

[IC5] **Olivier Michel**. Introducing dynamicity in the data-parallel language 81/2. In Luc Bougé, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, *EuroPar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Sciences*, pages 678–686. Springer-Verlag, August 1996.

[IC6] **Olivier Michel**. A straightforward translation of D0L Systems in the declarative data-parallel language 81/2. In Luc Bougé, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, *EuroPar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Sciences*, pages 714–718. Springer-Verlag, August 1996.

[IC7] Dominique De Vito and **Olivier Michel**. Effective SIMD code generation for the high-level declarative data-parallel language 81/2. In *Euro Micro '96*, pages 114–119. IEEE Computer Society, 2–5 September 1996.

[IC8] Jean-Louis Giavitto, Dominique De Vito, and **Olivier Michel**. Semantics and compilation of recursive sequential streams in 81/2. In H. Glaser and H. Kuchen, editors, *Ninth International Symposium on Programming Languages, Implementations, Logics, and Programs (PLILP'97)*, volume 1292, pages 207–223, Southampton, 3-5 September 1997.

[IC9] Jean-Louis Giavitto, **Olivier Michel**, and Franck Delaplace. Declarative simulation of dynamical systems : the 81/2 programming language and its application to the simulation of genetic networks. In *Proceedings of IPCAT 2001 (Workshop on Information Processing in Cells and Tissues)*, August 2001.

[IC10] Jean-Louis Giavitto and **Olivier Michel**. MGS: Implementing a unified view on four biologically inspired computational models. In *Pre-proceedings of WMC-CdeA 2001 (Workshop on Membrane Computing, Curtea de Arges)*. Research Report 17/01 of the Universitat Rivira I Virgili, Tarragona, Spain, August 2001.

[IC11] 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.

[IC12] Jean-Louis Giavitto and **Olivier 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.

[IC13] Jean-Louis Giavitto and **Olivier Michel**. Accretive rules in Cayley P systems. In *Preproceedings of WMC-CdeA 2002 (Workshop on Membrane Computing, Curtea de Arges)*. MolCoNet european network 2002-1, August 2002.

[IC14] Jean-Louis Giavitto and **Olivier Michel**. Pattern-matching and Rewriting Rules for Group Indexed Data Structures In *ACM - Rule'02, Pittsburgh* (USA), October 2002.

[IC15] Jean-Louis Giavitto and **Olivier Michel**. Data Structure as Topological Spaces. In *Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02*. October 2002, Himeji, Japan. LNCS 2509.

[IC16] Jean-Louis Giavitto, **Olivier Michel**, and Julien Cohen. Accretive rules in cayley P systems. In Gh. Paun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing 2002*, pages 319–338. Springer, 2003. LNCS 2597.

[IC17] **Olivier Michel** and Florent Jacquemard. An analysis of the needham-schroeder public-key protocol with MGS. In G. Mauri, G. Paun, and C Zandron, editors, *Preproceedings of the Fifth workshop on Membrane Computing (WMC5)*, pages 295–315. EC MolConNet - Universita di Milano-Bicocca, June 2004.

[IC18] 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 *Lecture Notes in Computer Sciences*, pages 238–247, LNCS, Amsterdam, October 2004.

[IC19] Antoine Spicher and **Olivier Michel**. Declarative modeling of a neurulation-like process. In *Sixth International Workshop on Information Processing in Cells and Tissues*, 2005.

[IC20] Antoine Spicher and **Olivier Michel**. Using rewriting techniques in the simulation of dynamical systems: Application to the modeling of sperm crawling. In *Fifth International Conference on Computational Science (ICCS'05)*, volume I, pages 820–827, 2005.

[IC21] Jean-Louis Giavitto, **Olivier Michel**, and Antoine Spicher. Computation in space and space in computation. In J.-P Banâtre, P. Fradet, Jean-Louis Giavitto, and **Olivier Michel**, editors, *Unconventional Programming Paradigms (UPP'04)*, number LNCS 3566, pages 137–152. ERCIM– NSF, Springer Verlag, 2005.

[IC22] Antoine Spicher, **Olivier Michel**, and Jean-Louis Giavitto. Algorithmic self-assembly by accretion and by carving in MGS. In *7th International Conference on Artificial Evolution*, 2005.

II.5.6 Publications in National Conferences (with review)

[C1] Jean-Louis Giavitto, Jean-Paul Sansonnet, and **Olivier Michel**. Inférer rapidement la géometrie des collections. In *Workshop on Static Analysis, Bordeaux*, 1992.

[C2] Jean-Louis Giavitto and **Olivier Michel**. Calcul distribué de champs de données. In P. Weis, editor, *Journées Francophones des Langages Applicatifs (JFLA99)*, Avoriaz, February 1999. INRIA.

[C3] Jean-Louis Giavitto and **Olivier Michel**. Un cadre pour la définition récursive de données. In C. Dubois, editor, *Journées Francophones des Langages Applicatifs (JFLA00)*, Mont Saint-Michel, February 2000. INRIA.

[C4] **Olivier Michel**, Jean-Louis Giavitto, and Julien Cohen. MGS : transformer des collections complexes pour la simulation en biologie. In L. Rideau, editor, *Journées Francophones des Langages Applicatifs (JFLA02)*, Anglet (France), January 2002. INRIA.

[C5] Antoine Spicher and **Olivier Michel**. Stratégie d'application stochastique de règles de réécritures dans le langage MGS. In *Journées Francophones des Langages Applicatifs*. INRIA, 2006.

[C6] Antoine Spicher and **Olivier Michel**. Manipulations de structures topologiques dans un langage déclaratif pour la simulation. In *11ème Journées du GT "Animation et Simulation" (GTAS'2004)*, Reims, juin 2004. AFIG et LERI, Université de Reims.

II.5.7 Research Reports and Contracts Reports

[R1] **Olivier Michel**. Design and implementation of 81/2, a declarative data-parallel language. Technical Report 1012, Laboratoire de Recherche en Informatique, December 1995.

[R2] **Olivier Michel** and Jean-Louis Giavitto. Amalgams: Names and name capture in a declarative framework. Technical Report 32, LaMI – Université d'Évry Val d'Essonne, January 1998. also avalaible as LRI Research-Report RR-1159.

[R3] Jean-Louis Giavitto and **Olivier Michel**. `MGS`: a programming language for the transformations of topological collections. Technical Report 61-2001, LaMI – Université d'Évry Val d'Essonne, May 2001.

[R4] Jean-Louis Giavitto and **Olivier Michel**. The topological structures of membrane computing. Technical Report 70-2001, LaMI – Université d'Évry Val d'Essonne, November 2001.

[R5] Jean-Louis Giavitto, Christophe Godin, **Olivier Michel**, and Przemyslaw Prusinkiewicz. Computational models for integrative and developmental biology. Technical Report 72-2002, LaMI – Université d'Évry Val d'Essonne, March 2002. draft version of [GGMP02b].

[R6] Jean-Louis Giavitto, **Olivier Michel**, and Julien Cohen. Pattern-matching and rewriting rules for group indexed data structures. Technical Report 76-2002, LaMI – Université d'Évry Val d'Essonne, June 2002.

[R7] **Olivier Michel**, Florent Jacquemard, and Jean-Louis Giavitto. Three variations on the analysis of the needham-schroeder public-key protocol with MGS. Technical Report LaMI-98-2004, LaMI – Université d'Évry - CNRS, May 2004. 25 p.

[R8] Antoine Spicher, **Olivier Michel**, and Jean-Louis Giavitto. A topological framework for the specification and the simulation of discrete dynamical systems. Technical Report LaMI-99-2004, LaMI, May 2004.

[R9] Jean-Louis Giavitto, Antoine Spicher and **Olivier Michel**. Topological Rewriting and the Geometrization of Programming. Technical Report IBISC-XX-2007, IBISC, September 2007.

II.5.8 Other Publications

[O1] **Olivier Michel**. Une plateforme logicielle pour l'expérimentation d'algorithmes de simulation parallèle. application à Time-Warp, September 1992. Rapport de stage du DEA MISI de l'UPMC Paris VI.

[O2] **Olivier Michel** and Dominique De Vito. 8,5 un environnement de développement pour le langage 81/2. In *Journées du GDR Programmation*, Lille, 22–23 September 1994. GDR Programmation du CNRS.

[O3] **Olivier Michel** and Jean-Louis Giavitto. Typer une collection par la présentation d'un groupe. In *Journées du GDR Programmation*, Grenoble, 23–24 November 1995. GDR Programmation du CNRS.

[O4] **Olivier Michel**. The 81/2 reference manual. December 1995.

[O5] Jean-Paul Sansonnet, Jean-Louis Giavitto, **Olivier Michel**, Abderhamane Mahiout, and Dominique De Vito. Rapport d'activité du thème 81/2. rapport final d'activité à destination du G.D.R. de Programmation, (10p.), January 1996.

[O6] Jean-Paul Sansonnet, Jean-Louis Giavitto, **Olivier Michel**, Abderhamne Mahiout, and Dominique De Vito. Rapport d'activité du thème 81/2– 81/2: Modèles et outils pour les grandes simulations. rapport interne (45p.), January 1996.

[O7] **Olivier Michel**. Les amalgames : un mécanisme pour la structuration et la construction incrémentielle de programmes déclaratifs. In *Journées du GDR Programmation*, Orléans, 20–22 September 1996. GDR Programmation du CNRS.

[O8] Jean-Louis Giavitto, **Olivier Michel**, and Julien Cohen. *Une présentation du langage MGS*. LaMI, université d'Évry, May 2002. (tutoriel).

[O9] **Olivier Michel**, Jean-Pierre Banâtre, Pascal Fradet, and Jean-Louis Giavitto. The Unconventional Programming Paradigms home page (UPP04). `http://upp.lami.univ-evry.fr`, 2004. International workshop for "Challenges, Visions and Research Issues for New Programming Paradigms", 15 - 17 September 2004, Mont Saint-Michel, France.

[O10] Antoine Spicher and **Olivier Michel**. Integration and pattern-matching of topological structures in a functional language. In *International Workshop on Implementation and Application of Functional Languages (IFL04)*, Lübeck, September 2004.

[O11] **Olivier Michel** and Jean-Louis Giavitto. Incremental extension of a domain specific language interpreter. In *International Workshop on Implementation and Application of Functional Languages (IFL07)*, Freiburg, Germany, September 2007. The draft proceedings will appear as a technical report of the Computing Laboratory of the University of Kent.

II.5.9 Theses

[T1] **Olivier Michel**. *Représentations dynamiques de l'espace dans un langage déclaratif de simulation*. Ph.D. thesis, Université de Paris-Sud, centre d'Orsay, December 1996. N°4596, (in French).

II.5.10 Masters and Ph.D. Theses under my Supervision

[MT1] Jean-Vincent Segard. Modèles de morphogénèse biologique dans un langage déclaratif de simulation. Masters thesis, D.E.A. de Sciences Cognitives du L.I.M.S.I, 1997.

[MT2] Sami Outerbah. Introduction d'une notion de référence distante dans un formalisme de capture de noms et implémentation d'une plate-forme d'expérimentation. Masters thesis, DEA Informatique d'Évry, 1998.

[MT3] Emmanuel Delsinne. Structures de données indexées par un groupe, isomorphismes de gbf abéliens et extensions aux structures automatiques. Masters thesis, E.N.S. Cachan et Université de Rennes-I, 2002.

[MT4] Valérie Larue. Structures de données indexées par un groupe : représentation graphique et extension au cas non abélien. Masters thesis, DEA Informatique d'Évry, 2002.

[MT5] Antoine Spicher. Typage et compilation de filtrage de chemins dans des collections topologiques. Masters thesis, DEA AMIB Université d'Évry, 2003.

[MT6] Fabien Thonnérieux. Réalisation d'une interface graphique pour le traitement des sorties du programme MGS. Masters thesis, IIE, 2003.

[MT7] Julien Cohen. *Intégration des collections topologiques et des transformations dans un langage fonctionnel*. Ph.D. thesis, Université d'Évry, 2004.

[MT8] Nicolas Mann. Hyperstructures et modélisation de chimie artificielle dans le langage `MGS`. Masters thesis, DEA AMIB Université d'Évry, 2004.

[MT9] Damien Boussié. Simulation en `MGS` du déplacement du spermatozoïde du nématode *Ascaris Suum*. Masters thesis, DEA AMIB Université d'Évry, 2004.

[MT10] Lionel Perret. Intégration des types de données algébriques dans MGS. Masters thesis, École Centrale Paris, 2005.

[MT11] Yann Jullian. Conception et développement d'un éditeur graphique de filtre pour MGS. Masters thesis, IIE, 2005.

[MT12] Fabien Gaubert. Simulation stochastique et modélisation de chimie artificielle dans le langage `MGS`. Masters thesis, DEA AMIB Université d'Évry, 2005.

[MT13] Antoine Spicher. *Transformation de collections topologiques de dimension arbitraires. Application à la modélisation de systèmes dynamiques*. Ph.D. thesis, Université d'Évry, 2006.

[MT14] Christoph Kaleta. Outils de visualisation pour la simulation de systèmes dynamiques à structure dynamique. Masters thesis, Master Informatique, Universität Jena & Université d'Évry, 2007.

Chapter III

Declarative Unconventional Languages for the Modelling and the Simulation of Dynamical Systems

III.1 Introduction: Why Designing New Programming Languages

Every now and then we hear the funeral oration of the programming languages: everything has been said on the subject, the only evolutions being either at a cosmetic level or at the implementation level. Every new proposition (and there are a lot of them, see [Leo] for example) is in one of the main family (imperative, functional, logic) which have been thoroughly studied. It might appear that all these propositions only differ by the syntactic sugar wrapping well-known fundamental mechanisms already described in [Lan66, Str67, Bac78] and in some few other old landmark papers.

This situation can be founded on the historical analysis lead by P. Landin in [Lan66]. P. Landin splits a programming language into two independent parts:

- the part devoted to the data and their primitive operations supported by the language, and
- the part devoted to the expression of the functional relations amongst them and the way of expressing things in terms of other things (independently of the precise nature of these things).

An example of the latter is the notion of identifier and the rule about the contexts in which a name is defined, declared or used. Another example are the control structures used to organize the set of computations that must be performed to achieve a given task. A good choice of these features can make a language flexible, concise, expressive, adaptable, reusable, general. The appropriate choice

of data and primitive yields an "API"[1], a "problem-oriented", a "domain-specific" or a "dedicated" language.

If we follow this point of view, research in programming languages would be reduced to the development of specialized libraries or to the refinement of techniques allowing only more efficient implementation of mechanisms already described by our visionary ancestors.

Since my scientific work was mainly concerned with the design and the development of new data and control structures (even if it is with the goal to provide a programming language well suited to some application domain), nobody should be surprised that I do not share this closed point of view. Indeed, if I agree with L. Wittgenstein when he says that "[...limits of my language mean the limits of my world [...]" [Wit21, 5.6][2] I strongly disagree with "Whereof one cannot speak, thereof one must be silent [...]" [Wit21, 85][3]. On the contrary, I believe that facing new needs, and there are plenty of new needs!, we *have* to invent new means of expression, new *unconventional* programming languages. We should not twist over and over already existing languages that have proved to fail in that purpose. From the confrontation of many alternative proposals will come the next programming languages.

III.1.1 Unconventional Programming Languages

Unconventional approaches of programming have long been developed in various niches and constitute a reservoir of alternative avenues to face the needs of present and future computation-intensive applications and the new computing media (see Section VI.3). The field is experiencing a renewed period of growth with the creation of journals like the International Journal of Unconventional Computing [IJU05] or Natural Computation [Nat02] and series of general conferences and workshops like [UMC98, Com04, BFGM05, Uni05].

Unconventional programming languages provide new abstractions and new notations or develop new ways of interacting with programs. They are implemented by embedding new and sophisticated data structures in a classical programing model (API), by extending an existing language with new constructs (to handle concurrency, exceptions, open environments...), by conceiving new software life cycles and program execution (aspect weaving, dynamic linking, run-time compilation, staged compilation, organic computing [ORG04]) or by relying on an entire new paradigm to specify computations. They are inspired by theoretical considerations (*e.g.*, topological, algebraic or logical foundations), driven by the domain at hand (domain-specific languages like PostScript, LaTeX, HTML, XML, musical notation, animation, signal processing, embedded systems, etc.) or by metaphors taken from various area: quantum theory[4], molecular biology[5], cell biology and physiology[6], ethology and social science[7], and more generally, problem solving from nature[8].

[1] An "Application Programming Interface" is a library making available in a language an abstraction of some objects and functions operating on these objects.

[2] The original quote being "[...] Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt [...]". The common English translation is somehow misleading since the notion of "limit" does not account for something that would be *after* that limit, while the German word "Grenzen" means literally "border" (as between two countries) which accounts for some (unknown?) land after the considered border.

[3] The original quote being "[...] Wovon man nicht sprechen kann, darüber muß man schweigen [...]".

[4] See the list of numerous conferences of the domain at `http://qserver.usc.edu/confs/`

[5] See the DNA Computing conference series [DNA95] and for other approaches [BFL01, Ghe05].

[6] See the Information Processing In Cells and Tissues (IPCAT) conference series [IPC95] and the WMC workshop on membrane computing series [PSY02].

[7] See the Simulation of Adaptive Behaviour (SAAB) conference series [SAB90].

[8] See the Parallel Problems Solving from Nature (PPSN) conference series [PPS90] and the journal "Natural Computing" [Nat02].

III.1.2 8½ as an Unconventional Programming Language

The 8½ project has been initially motivated by the programming and the exploitation of data-parallel architectures for the simulation of massive dynamical systems. The concepts and notations offered by the language try to mimic as closely as possible the basic objects of dynamical system theory: state, trajectory, definition by equations.

The representation of a state structured by a uniform spatial organization has led to the development of the notion of Group Based data Field (GBF), a notion investigated in the papers [GMS95, GM01a, GM02b, GMC02a].

The notion of a trajectory, that is, an infinite temporal sequence of values, has been captured by the notion of streams. The implementation of stream is sketched in the papers [GDVM97, Mic96d, Gia99] but we do not elaborate too much, as this domain is not central in my work.

Programming by solving equations, not really any kind of equations but recursive definitions, leads to a declarative programming style and is developed in the papers [Gia92, Mic96d, GDVM97, MG98, GM01a, Coh04b, Spi06b].

III.1.3 MGS as an Unconventional Programming Language

Initiated in the 8½ project, the idea to use spatial relationships to structure the computer representation of a state, has been considerably deepened in the MGS project up to the point where a data structure is understood as a field on a topological space. This approach is investigated in papers [GMS95, GMC02b, GM02b, GMC02a, GM01a].

The resulting language is based on basic objects introduced in combinatorial topology: cellular complexes and chains. These objects are embedded in a functional language and a new construction for the rule-based specification of functions, called transformation, is introduced to facilitate their management.

Transformations together with the unified topological view on data structure, enable the unification of several bio-inspired programming models. This point is developed in Section III.2.

Transformations are a kind of non-standard rewriting systems [DJ90]: they are non-standard because they apply beyond tree-shaped structures (terms) to more general data organizations. We want to stress that we are not particularly interested in theoretical properties usually investigated in rewriting systems like confluence or normal forms. We are mainly interested by the rewriting approach in biological simulations, see [GMM04] and Section IV.7.1. Lindenmayer systems [PLH+90] give a beautiful example of the use of rewriting systems in modelling and simulation.

As in the 8½ project, the application domain motivating MGS is still the modelling and simulation of dynamical systems, but with a focus on dynamical systems encountered in biology. This kind of system often presents a dynamical structure, especially in morphogenesis. The related notions are exposed in Chapter IV.

III.1.4 Plan of the Chapter

In the next section, we sketch four bio-inspired paradigms that have been influential in the design of MGS. These paradigm can be unified for programming purposes, using a topological point of view: a data structure is a field on a topological space. This viewpoint is sketched in Section III.3. Both 8½ and MGS language enjoy a declarative flavour. This notion is presented in Section III.4. The last section of this chapter, Section III.5, gives a short introduction to each papers of this chapter.

III.2 Bio-Inspired Programming Languages: the Roots of MGS

One of the initial motivations of the MGS project is to unify different computation models[9] initially inspired by biological processes. From the topological point of view taken in the MGS project, these languages are all based upon a substitution mechanism on a data structure for a specific space structure. These different computation models are described in the next paragraphs.

Gamma. Gamma proposes a formalism with no notion of sequentiality. Using multi-set rewriting as the sole control structure, a program can be described following a chemical metaphor where the reactions correspond to rewriting rules and where the chemical solution is a multi-set. The chemical metaphor allows a simple and direct description of parallel processes and non-determinism. Extensions have been added to the initial formalism: among them, we can name some structuration that was missing in the original formalism, a notion of neighborhood relation between the data. Nevertheless, the multi-set structure remains essential (it is actually a structured multi-sets), the neighborhood being encoded using an addressing mechanism and type definitions [BLM86, BFL01]. Recently, the handling of higher-order chemical solutions where programs can now be molecules [BFR06a] has been investigated and such features have been proved very useful to specify autonomic systems [BRF04, BFR06b].

P systems. The P systems are based on the metaphor of the behaviour of biological cells to organize a set of chemical computations. P systems correspond to nested multi-sets, or *compartments*, hosting various elementary objects. Each compartment is characterized by a set of localized rewriting rules specifying how the objects are interacting. Various additional operators allow the transport of objects between compartments and the creation and destruction of compartments. Such systems are parallel and non deterministic. P systems are powerful mechanisms to define new classes of formal languages and the study of P systems was initially focused on calculability problems. Their use has recently shifted to the modelling and simulation of real biological processes [Pău00, Pău01a].

L-systems. Lindenmayer's systems have been initially proposed in 1968 by the biologist A. Lindenmayer to give a formal description of the cellular development of filamentous plants [Lin68]. The formalism developed is very powerful: the system is represented by a list of strings, each string meaning a (sub-)part of the organism and contiguity in the string means neighborhood in the (sub-)parts. The evolution of the system is captured by the parallel rewriting rules selecting a sub-string to be replaced by a new string. This framework has been developed following two different directions. First, the L-systems give a mean to produce words and have been used to define a formal language hierarchy (in a similar way to Chomsky's hierarchy). The generative approach of L-systems has also met a huge success in the modelling and simulation of growth and developmental processes, especially in botany [PLH+90, RS92, PRL06, SGM+06, PEL+07].

Cellular Automata. Cellular automata have been introduced by S. Ulam [Ula62] and J. Von Neumann [VN66] to compare the notion of living organism with the notion of machine with a focus on self-reproduction. A cellular automaton can be described as a predefined network of sites

[9] We are interested in following the programming language point of view: to each of these computation model corresponds a class of programming languages and we are not interested in the study of the complexity of the algorithms written in these computation models but to the expressiveness brought by the constructions only found in the languages related to that computation model.

(for example a NEWS neighborhood), each site having a state taken from a finite set. At a given instant $t+1$, the state of each site is updated following a predefined evolution rule that combines its state together with the state of the neighboring sites at instant t. The behaviour of the automaton corresponds to the update, in discrete steps, of the state of the sites.

A Unifying Point of View

One Observation. All four examples of computation models that we have briefly sketched have been inspired by chemical or biological processes. They lead to fruitful formal developments but have also been used as foundations for *unconventional programming languages* that have been then widely used in the modelling and simulation of natural and artificial processes.

Despite their differences (they have been created on a 50 years time scale with very different motivations), we can observe that those four paradigms have the common feature of expressing control as evolution rules that can be described as specific kind of *rewriting*. By rewriting, we mean here the mechanism consisting in the substitution of a part of an object by another in that same object.

Multi-sets and lists (strings) are *monoïdal* data structures [Man01, GM01b] and their associated rewriting techniques are well-known: it is rewriting *modulo* associativity (for the lists) and *modulo* associativity and commutativity (for the multi-sets).

It is not so common to consider cellular automata as a form of array rewriting, essentially because there is no free inductive definition of the notion of array (an array is not a term). Moreover, rewriting only concerns one site (even if the right-hand side of the rule refers to the neighbors of that site). Nevertheless, some variations on that model, like lattice-gas automata [TN87], are indeed specifying the rewriting of a set of neighboring sites.

The Topological Approach. It is quite tempting to embrace all these computation models in the same framework allowing to define them as specific cases of a more general substitution mechanism on parts of a data structure. For that, we need to have a general notion of object and part, to be able to generalize multi-sets, words, arrays...

The MGS project proposes to rely on notions from algebraic topology to define that framework. The idea is to see a chemical solution, a set of nested membranes, a list or a lattice as an abstract space where computation takes place. We call that space a *topological collection*. The substitution mechanisms and the rules appearing in Gamma, the P systems, the L-systems or the cellular automata will be seen as *transformations* of that space.

It is still too early to claim that the theoretical framework that we are defining in our work to formalize the notions of topological collections and of transformations are well suited to the formal study of these different computation mechanisms. Nevertheless, the examples developed throughout our work show that these notions offer, at least at the syntactic level and as far as programming is concerned, a uniform and expressive framework that allows the simple integration of these various paradigms in the same language.

III.3 Data Structures as Topological Spaces

Our starting point is the following intuitive meaning of a data structure: a data structure s is an *organization* o performed on a data set D. It is customary to consider the pair $s = (o, D)$ and to say

that s is a structure o of D (for instance a *list* of *int*, an *array* of *float*, etc.) and to use set theoretic constructions to specify o. However, here, we want to stress the structure o as a set of *places* or *positions*, independently of their occupation by elements of D [BLL97, Chap. 1]. Following this perspective, a data structure in [Gia00b] is a function from a set of positions to a set of values: this is the point of view promoted by the *data fields* approach. Data fields have been mainly focused on arrays and therefore on $\mathbb{Z}^n$ as the set of positions [HL93]. One of our motivations was to define in the same framework the set of positions representing a tree, an array or a multi-set independently of the set of values.

III.3.1 Data Structure and Neighborhood

One interest of the data field approach is to give the set of places a structure. To define a data organization, we adopt a *topological* point of view: a data structure can be seen as a field over a *space*, the set of positions between which *the computation moves*. This topological approach relies on the notion of *neighborhood* to specify a move from one position to one of its neighbor. Although speaking of neighborhood in a data structure is not usual, the relative accessibility from one element to another is a key point considered in a data structure definition: in a simply linked list, the elements are accessed linearly (the second after the first, the third after the second, etc.), from a node in a tree, we can access the sons, two usual neighborhood are considered for arrays, "Von Neumann" or "Moore" neighborhoods, etc.

III.3.2 Elementary Shifts and Paths

This accessibility relation defines a logical neighborhood. The concept of logical neighborhood in a data structure is not only an abstraction perceived by the programmer and vanishing at the execution, but it does have an actual meaning for the computation. Very often the computation indeed complies with the logical neighborhood of the data elements. For example, the recursive definition of the `fold` function on lists propagates an action to be performed along the list. More generally, recursive computations on data structures respect so often the logical neighborhood that standard higher-order functions (*e.g.*, *primitive recursion*) can be automatically defined from the data structure organization (think about catamorphisms and others polytypic functions on inductive types [MFP91]).

These considerations lead to the idea of *path*: in a sequential computation, elements of the data structure are visited one after the other. We assume that if element e' is visited just after element e in a data structure s, then e' must be a neighbor of e. The succession of visited elements makes a path in s. The idea of sequential path can be extended to include parallel modes of computations: multi-dimensional paths must be used instead of one-dimensional paths [GJ92].

III.3.3 Paths and Computations

We can summarize our topological approach: we assume that a computation induces a path in a space defined by the neighborhood relationship between the elements of a data structure. At each shift, some elementary computation is done. Each topological operation used to build a path can then be turned into a new control structure that composes program fragments.

This schema is presented in an imperative setting but can be easily rephrased into the declarative programming paradigm (see the next section) by just specifying the linking of computational actions with path specifications. When a path specification matches an actual path in a data structure,

then the corresponding action is triggered. It is very natural to require that the results of the computational actions be *local*: the corresponding data structure transformation is restricted to the value of the elements involved in the path and eventually to the organization of the path elements and their neighborhood relationships. Such transformation is qualified as local.

This declarative schema induces a rule-oriented style of programming: a rule defines a local transformation by specifying the path to be matched and the corresponding action. A program run consists in the transformation of a whole data structure by the simultaneous application of local transformations to non-intersecting paths. Obviously, such *global* transformation can then be iterated.

III.4 The Declarative Framework

Whatever application domain and programming paradigm is considered, the work of the last 40 years demonstrate the requirement to have a clear (and the simplest possible) mathematical viewpoint on a program to allow reasoning, validation, proofs and tests of properties on those programs. This statement has been advocated again and again, see for example [TE68, Bac78].

Declarative programming focuses on *what* should be computed instead of *how* it must be done. Thus, a declarative program is an executable specification not burdened by the implementation details. Its objects and constructions are close to the mathematical standards which enable an easier mathematical reasoning on programs. Generally, declarative programming languages offer, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

This statement can be made more specific, in parallel with the statement of [Eng90] on the importance of algebra in programming: in the declarative paradigm, programs have always the form "*find an object (a value) which has this or that property*". In this approach, algebra is central. The algebraic tools enter in two ways: first in the specification of what an admissible program is, and second in specifying the search space for the solution. Admissible programs take the form of equations (for the unknown object to be computed), using an adequate supply of operations to write out the equations. The search space of the solution(s) is chosen as an algebraic structure, typically an extension of the structure in which the parameters that enter the program (the equations) are elements.

A classical example is functional programming [Hug89] where the programs are recursive definitions of functions (a definition is an equation of form $f = \varphi(\dots)$ where f is the unknown[10]) and where the solution space is an extension of the fundamental domain Λ characterized by $\Lambda = \Lambda \to \Lambda$ [GS90] (or, using an alternative formalism, a *Cartesian closed category* [AL91]).

III.4.1 The Declarative Handling of Time

8½ is a declarative language where a program is a static set of equations on the stream data type thus allowing the handling of time. Streams are infinite sequences of values and are found in LUCID [AW77], one of the first programming language defining equations between streams. 8½ streams are very different from those of LUCID: 8½ streams are (timed) sequences of values that change instantaneously at some given instant and whose value is, between any two instants, the value taken by the stream at the previous change.

[10] In a recursive definition, the unknown may appear in the right hand side, like in $f = \varphi(\dots, f, \dots)$.

Time in 81/2 corresponds to a logical time related to the semantics of the application as in any discrete-event simulation. The set of equations in 81/2 states what each stream is equal to: the definition $C = A + B$ means the (current) value of stream C is always equal to the sum of the values of stream A and B. Here, always means "at any time instant". The changes of values is assumed to be propagated instantaneously: when A (or B) changes, so does C at the same logical instant.

III.4.2 The Declarative Incremental Construction of Programs

The set of equations in an 81/2 program has a static structure. To lift this restriction, *amalgams* have been defined as a formalism allowing to define open sets of equations involving free references. Such sets are called *systems* and two operators (merge and restriction) allow to *close* the systems with additional definitions. The data-flow point of view on amalgams is a higher-order dynamic completion of incomplete graphs (with pending edges) with operators similar to the regular parallel and sequential operators.

In addition to the papers [Mic96b, Mic96c, MG98], amalgams have also been presented in [Mic96c]. A pure calculus of amalgams (*i.e.*, an algebra considering only systems, merge and restriction) has been investigated in [MG98] using an operational semantics. A distributed implementation (on a network of workstations) embedded in the Ocaml language has been implemented [Out98].

III.4.3 The Intensional Handling of Topological Collections

In the MGS project, the objects are topological collections (they are formalized in [GM02c, Spi06b]), but we do not define them by recursive equations. Nevertheless, we define functions to manipulate those objects, the simplest possible way, by local rules that are iterated over the data structure.

Application Strategies. The way the rules are iterated is called the rule *application strategy*. Several application strategies have been developed in MGS, ranging from the maximal parallel application strategy (similar to the application strategy used in L-systems) to the stochastic application strategy. When a topological collection represents the state of a dynamical system, the evolution function can be specified by a transformation and the application strategy captures the handling of time (synchronous or asynchronous).

Executing an MGS program corresponds most of the time to iterate a transformation starting from an initial topological collection. Iterating forever computes an object that can be expressed as the solution of a fixed-point equation but considering the elementary steps of the iteration allows us to have a finer control on the evolution of this object.

Intensional vs. Extensional Expressions. For the programmer, the application strategy turns transparently a set of local rules into a global computation over the entire data structure. Moreover, the local rules do not explicitly denote the elements involved: these elements are selected through pattern-matching. So, for the programmer, a transformation handles a topological collection *as a whole*. Operations that handle the data as a whole are *intensional* expressions, see [OA95, AFJW95]. On the opposite, operations that explicitly enumerate the elements of a data structure are qualified as *extensional*.

An example will clarify this idea: iterating a computation over a collection of values can be done using a `for` loop. The body of the `for` construct must explicitly denotes the current element using

a pointer or an index that gives access to it. A for-loop is an extensional control structure. The iteration can also be done using a higher-order function, a `map` parameterized by the computation to be applied in the form of a function. Accessing the element is completely hidden in the map which is an intensional operation. Moreover, map can be a first-class object (in a functional language), which is definitively not the case of a control structure.

The intensional style is important because it is more concise and makes possible to formalize the data computations as an algebra. Indeed, there is an algebra of intensional operations on lists, Bird-Mertens algebra [Bir87, Bac89, Gib94], which is not the case for imperative loops.

For the programmer, handling a data structure as a whole presents several advantages:

- global operations on the data structures may hide optimized implementation (this is for example the key of the abstract expression of the *data parallelism*, see [HL93]);
- managing a data structure without referring to the data elements leads to the concise expression of the computation;
- the automatic analysis of the programs are simplified because the compiler is not required to "reconstruct" the semantic meaning of the computations from the description of the low-level operations;
- the expression of the algorithms is more abstract and the algebraic style favors an abstract reasoning unifying the various computation patterns appearing in different programs.

Algebraic Topology. The declarative style of transformation is well illustrated by a recent achievement done in A. Spicher's Ph.D. thesis [Spi06b]. Transformations are used to manipulate topological collections, that is chains in the language of algebraic topology[11]. Amongst the mathematical functions that operates on a chain to return a chain, some are of special interest: those that preserve the chain structure. These chains homomorphisms are called *topological cochains*. And topological cochains can be linked to some notions developed in the differential calculus.

Operators like the *boundary operator* or the *exterior derivative*, but also *coboundary*, *gradient*, *divergence*, *curl* and *Laplacian* can be specified as transformations. These transformations give a discrete counterpart of the differential operators [Hir03, DKT06] that appear in the handling of continuous fields. This analogy reinforces the understanding of data structures (here represented by topological collections) as "physical" fields defined on exotic spaces.

For the programmer these differential operators represent fundamental ways of iterating over topological collections and we want to develop an algebra analogous to the Bird-Merteens algebra encompassing the different kind of data structure that can be represented as a topological collection.

III.5 Presentation of the Papers

III.5.1 Directions for Future Developments in Unconventional Programming Languages

[MBFG06] **Olivier Michel**, Jean-Pierre Banâtre, Pascal Fradet, and Jean-Louis Giavitto. Challenging questions for the rationales of non-classical programming languages.

[11] Combinatorial topology is the part of algebraic topology concerned only by combinatorial properties. The notion of abstract simplicial complex (in combinatorial topology) used to formalize the notion of topological collection is a specialization of the more general notion of cellular complex developed in algebraic topology.

International Journal of Unconventional Computing, 2006.
URL: http://www.ibisc.fr/~michel/PUBLIS/2006/ijuc.pdf
Booklet page: 3

In this paper, we try to draw some lines along wich we detail questions articulating the current developments in the domain of unconventional programming languages. This paper was written after the numerous discussions that we had in the organization of the Unconventional Programming Paradigm (Mont St Michel - 2004) and The Grand Challenge in Non-Classical Computation (York - 2005). We do not give solutions but sketch a landscape by describing the motivations as a list of questions that can be spread into 4 classes:

1. Metaphors for Computations,
2. Programming in the Small and Programming in the Large,
3. The Future of Syntax, Semantics, etc.,
4. New applications, New Opportunities.

III.5.2 The 81/2 Language

[Mic96b] **Olivier Michel**. Introducing dynamicity in the data-parallel language 81/2. In Luc Bougé, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, *EuroPar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Science*, pages 678–686. Springer Verlag, August 1996.
URL: http://www.ibisc.fr/~michel/PUBLIS/1996/w21.pdf
Booklet page: 17

Chronologically, my work has started with the 81/2 project. The work in this project can be understood as the introduction of data parallelism in a data-flow computation model. This paper sketches the resulting language. Traditionally (that is in the data-flow architecture developed in the 70's), a data-flow program exhibits a static structure as a graph of computation tasks defined *a priori* (the graph can introduce the notion of iteration and as a consequence, the model becomes Turing-complete). One of the issue that I have adressed is how to have a dynamic data-flow graph, that is, as the result of some computation. This lead to consider "abstract" data-flow graphs that can be later instanciated into "concrete" data-flow graphs in a way similar to a function that is applied to an argument to "trigger" an effective computation. This mechanism is briefly described in that paper.

[GDVM97] Jean-Louis Giavitto, Dominique De Vito, and **Olivier Michel**. Semantics and compilation of recursive sequential streams in 81/2. In H. Glaser and H. Kuchen, editors, *Ninth International Symposium on Programming Languages, Implementations, Logics, and Programs (PLILP'97)*, volume 1292 of *Lecture Notes in Computer Science*, pages 207–223, Southampton, 3–5 September 1997. Springer Verlag.
URL: http://www.ibisc.fr/~michel/PUBLIS/1997/plilp97.pdf
Booklet page: 29

This paper defines the semantics and compilation of 81/2 streams. These streams differ from classical data-flows in the sense that they are synchronous and also differ from Lustre/Signal streams by a different model of time: in Lustre/Signal, observations and events must be simultaneous while observations are decorrelated in 81/2. This paper illustrates also the use of fixed-points techniques as an evaluation engine for declarative languages.

III.5.3 Data Structures as Topological Spaces

[GM02a] Jean-Louis Giavitto and **Olivier Michel**. Data structure as topological spaces. In *Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02*, volume 2509 of *Lecture Notes in Computer Science*, pages 137–150, Himeji, Japan, October 2002.
URL: http://www.ibisc.fr/~michel/PUBLIS/2002/umc02.pdf
Booklet page: 49

All the papers in this section develop the motto "data structures as topological spaces". This idea is introduced in the paper [GM02a]. We introduce also the distinction between *Leibnizian* and *Newtonian* space: in the former, the space is a consequence of its elements while in the latter the space exists *a priori* and elements are just filling the available positions. Examples of Leibnizian collections are (multi-)sets, Delaunay graphs; examples of Newtonian collections are arrays.

[GMS95] Jean-Louis Giavitto, **Olivier 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 *Lecture Notes in Computer Science*, pages 209–215, Beaune (France), 2–4 October 1995. Springer Verlag.
URL: http://www.ibisc.fr/~michel/PUBLIS/1995/psls.pdf
Booklet page: 65

The paper [GMS95] concretizes the motto by using uniform spaces to define regular structures like trees and arrays, called GBF. We advocate that having in the same theoretical framework structures as different as trees and arrays is a major theoretical (and practical) achievement.

[GM01a] Jean-Louis Giavitto and **Olivier Michel**. Declarative definition of group indexed data structures and approximation of their domains. In *PPDP '01: Proceedings of the 3rd ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 150–161, New York, NY, USA, 2001. ACM Press.
URL: http://www.ibisc.fr/~michel/PUBLIS/2001/ppdp01.pdf
Booklet page: 73

Paper [GM01a] develops the declarative definition of such data structures and the problems that are raised: has a GBF equation a well defined solution and is this solution total or partial (that is, is each element of the aggregate well defined or are we facing "arrays" with "holes")?

[GM02c] Jean-Louis Giavitto and **Olivier Michel**. The topological structures of membrane computing. *Fundamenta Informaticae*, 49:107–129, 2002.
URL: http://www.ibisc.fr/~michel/PUBLIS/2002/FI.pdf
Booklet page: 87

The last paper in this section formalizes arbitrary data structure with cellular complexes. These developments take place in the domain of membrane computing where it is shown that structures with dimensions higher than 1 are useful for example to model interactions between non-hierarchical organisations. Even if quite focused to the domain of P systems, this paper presents ideas of broader interest.

Chapter IV

Modelling and Simulation of Dynamical Systems – Applications

IV.1 Introduction

It is not only important to shape and develop new programming languages, it is also of major importance to effectively use the language against real problems. Claiming that expressiveness is central in programming languages is vacuous without large developments supporting that claim.

The interaction between the language and its applications must be a real two-ways interplay: the issue is not to fine-tune ad-hoc libraries to fit a given problem more and more precisely (and to make a biologist or physicist happy), but to extract from an application domain some specific mechanisms that can be generalized, applied and *in fine* proved fruitful in other domains. A good example is the ML language, developed initially to define proof search strategies in an automatic theorem proving system (LCF). Features like the specification of functions by cases have then been used in other areas and have proved their general utility.

In my work, I have focused on designing languages for the modelling and simulation of dynamical systems, and more specifically, a general class of dynamical systems: dynamical systems with a dynamical structure, or $(DS)^2$. A $(DS)^2$ is a dynamical system whose structure evolves in time [GM01d, GM01b, GM02b, GGMP02a]. This includes morphogenesis processes and lead to fruitful collaborations with C. Godin and P. Prusinkiewicz [GGMP02b, GGMP02a, SMC+07]. In this area, one of the key ideas is that such a complex system can be described by its generative process described by a set of local rules (like for example in L-systems).

We detail in the rest of the chapter the problems raised by the modelling of (DS)[2]. This kind of dynamical system is quite general[1] even if we often focused our attention on applications in biology.

IV.2 Simulation Needs in Integrative Biology

Biology is both a growing application domain (for example with all the computing tools required for sequencing in the *Human Genome Project* and the analysis of all post-genomic available data) and a source of metaphors for computer science [Pat94]: neuronal networks [MP43], cellular automaton [Ula62, VN66], genetic algorithms [Hol73], evolutionary algorithms [Hol75], artificial life [Gar70], DNA computing [Adl94], chemical computing [DZB01, BLM96, Ada01], membrane computing [Pău01b, Car04]...

Among all these domains where the interaction between biology and computer science is fertile, modelling and simulation have a key position. In the domain of *system biology*[2] or *integrative biology*[3], computer simulation is an extraordinarily efficient tool to analyze and verify the correctness of a formal model against hypotheses made from experimental data, to seek for specific properties of certain sets of interactions, to study the decomposition into integrated sub-systems, to predict the result of small perturbations or new situations (mutants, modification of the environment, perturbation of the metabolism, etc.), and to discover new regulation pathways.

Simulation is not the only tool that computers can produce to help progress in system biology: logical modelling [TK01], rewriting systems [GMM04, EKL+02], model-checking [Ric06], test and validation techniques of systems [FMP00], concurrent systems specification [Car04], optimization, etc. are examples of fruitful approaches bringing important contributions. In the MGS project, we only consider simulation.

These new simulation applications are raising very difficult problems to computer science and require new concepts for the representation, the modular and incremental construction of simulation programs, their validation and the interpretation of the results. A possible strategy to face all these problems is to develop a new *domain-specific language*.

The development of a domain-specific language is justified by readily available features that will enable better reuse and capitalization of programs to face the new problems raised by the targeted application domain. We can hope for more productivity, safety, maintainability, upgradability and flexibility than for a general-purpose programming language.

A domain-specific language dedicated to the requirements of simulation in the domain of integrative biology has to allow the analysis of the huge amount of data produced by the methods of large-scale biology to model, relate and integrate the many interaction networks at the intra-cellular

[1] In [BL06, pp 125-127], the authors recognize the importance of this class of dynamical systems and call it "dynamicité auto-constituante" (which could be translated to "self-producing dynamicity"), a distinctive feature of living organisms.

[2] System biology is the "iterative and integrative study of biological systems as systems, in answer to perturbations" [AIRRH03]. This domain relies on mathematics and computer methods to build a biochemical and physiological integrated model from incomplete and heterogeneous knowledge on the functions and interactions involved, generally produced by functional genomics.

[3] The word *integrative* refers to the wish to synthesize all available biological knowledge to understand the interactions taking place between an organism and its environment. Following this standpoint, integrative biology is part of system biology. More specifically, the "integrative" term refers to the "horizontal" integration of a large amount of data or knowledge produced by functional genomics. The organization level that is investigated can be one of the "ome" domain (transcriptome – that is the set of all messenger RNA –, proteome – the set of proteins – or metabolome – the set of all metabolites: hormones, signalling molecules...). In addition, it is possible to integrate different levels of organizations, which in that case would be a "vertical" integration.

level with the cellular and super-cellular structures (tissues, blood compartments, organs...) in order to give a better account and to understand and manage the various biological and physiological functions. We can list more precisely the problems arising in this domain:

- There is a combinatorial explosion of the entities to be specified, where each of them exhibits many possible different attributes and behaviours.
- The specification of a biological entity implies heterogeneous aspects that nevertheless may interact: physical structure, chemical and electrical state, specification of its own evolution and interaction with the other entities, geometry (localization and neighborhood)... Moreover, these aspects dynamically depend on the whole state of the biological entity itself.
- The system cannot be globally described in a simple way (for example by a unique numerical model), but only in terms of dynamic local interactions of more elementary entities.
- The description of the system cannot be structured in simple hierarchical terms. Moreover, this structure is usually dynamic and has to be computed together with the evolution of the system.

The first two items require us to develop new programming concepts and techniques that allow the representation of *dynamically structured and spatially distributed* processes. The notion of $(DS)^2$ lies at the heart of the MGS project and has been presented in [GM01b, GGMP02a, GM02c, GM03, Gia03]. A paradigmatic example of a $(DS)^2$ is the development of an embryo, or more generally, all the models of morphogenesis that have the slightest bit of realism (one can find a description of these applications in [GS06a]).

IV.3 $(DS)^2$: Dynamical Systems with a Dynamical Structure

Intuitively, a dynamical system (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 according to its current location (the *evolution function*). Many different existing formalisms are used to describe a DS: ordinary differential equations (ODE), partial differential equations (PDE), discrete coupling of differential equations, iterations of functions, cellular automaton, etc. depending on the discrete or continuous nature of space and time and of the values used in the modelling. Examples of formalisms, sorted according to the nature of time and of the state variables, are given in table IV.1 below.

C: continuous, D : discrete	ODE	Iterations of functions	Finite state automaton
Time	C	D	D
State	C	C	D

Table IV.1: Some formalisms used to specify DS according to the nature (discrete or continuous) of time and of the state variables (taken from [GGMP02b]).

Many DS have a structure, that is, they can be decomposed into multiple parts. Furthermore, the full state s of the system can be expressed as the simple composition of the state of each part. Then, the evolution of the state of the system is seen as the result of the changes occurring in the states of each part. In that case, the evolution function h_i of the state of part o_i depends only on a

subset $\{o_{i_j}\}$ of the state variable of the whole system. In that context, we say that the DS exhibits a *static structure* if:

1. the state of the system is statically described by the state of a finite set of its parts and this set does not change over time;
2. the relations between the states of the parts, specified by the definition of the function h_i between the o_i and arguments o_{i_j}, are also predefined and do not change over time.

Furthermore, we say that the o_{i_j} are the *logical neighbors* of the o_i (and very often, two parts of a system interact when they are neighbors in the physical space). This situation is very common and appears often in elementary physics. For example, the fall of a stone is statically described by a position and a speed, and this set of variables remains the same over time (even if the *value* of the position and the *value* of the speed are changing over time). The computation of speed does not depend of the position while the computation of position depends on the position and speed. These dependencies do not change over time (the evolution function of the system is always the same one).

On the contrary, following the analysis carried in [GGMP02b], we can remark that many biological systems can be seen as dynamical systems where not only the state variable, but also the *set* of these state variables *and/or* the evolution function, change over time. These are $(DS)^2$ following the terminology introduced in [GM01b, GM01c].

A straightforward example of $(DS)^2$ in biology is given by developmental processes. In the field of developmental biology, one current theoretical framework views the developmental process as changes within a dynamical system (DS). This point of view can be traced back at least to W. D'Arcy Thompson, A. Turing, L. von Bertalanffy, C. Waddington, and contrasts with pure genetically programmed and pre-existing plans that await revelation during the developmental process. In the past two decades or so, the concepts and models of nonlinear dynamical systems have been coupled with models of genetic regulations to overcome the genetic/epigenetic debate on the nature of the ontogenetic processes. These models can be seen in the pioneering work of researchers such as F. J. Varela [Var99], H. Meinhardt [Mei82], J. L. Harper, B. R. Rosen and J. White [HRW86], P. Prusinkiewicz [PLH+90], S. Kaufman [Kau95], J. Maynard-Smith [MS99], L. Wolpert [WBL+02].

A developmental process viewed as a dynamical system often presents the distinctive feature of having a dynamic phase space. Consider the example given by the development of an embryo. Initially, the state of the system is described by the egg's sole chemical state o_0 (whatever the complexity of the chemical state is). After many cell divisions, the state of the embryo is no longer specified by the only chemical state o_i of the cells, but also by their spatial arrangements[4]. The number of cells, its spatial organization and their interactions constantly evolve during the development stages and are not the result of a unique static structure $\mathcal{O}$. On the contrary, the phase space $\mathcal{O}(t)$ used to characterize the structure of the state of the system at time t has to be computed *jointly* with the current state of the system. In other words, the phase space has to be defined as an *observable* of the system.

[4]The neighborhood of each cell is very important in the evolution of the system because of the dependency between the global shape of the system and the state of the cells. The shape of the system has an impact on the diffusion of the physico-chemical signals and consequently on the state of the cells. Conversely, the state of each cell determines, for example by initiating a division, the evolution of the shape of the whole system. A very nice example of the retroaction between the cellular and tissue level of organization is given by the formation of the mesoderm which creates a pressure on the anterior region of the gastrula when the *twist* gene is expressed. Without this mechanical pressure, which is a result of the global structure (the tissue), there is no expression of the gene at the level of each cell and the further development of the embryo is compromised [Far03].

The coupling between the global shape and its parts is called *downward causation* by P.-A. Miquel and A. Soto [Sot06] (following D. Campbell original definition [Cam74] on "the higher level system or whole constrains its parts"). We can see in this *feedback between a global level and a local level*, the shape acting as a regulator of the parts, a modern version of *organicism*, a theory developed by embryologists following I. Kant for whom the parts of an organism "[...] bind themselves mutually into the unity of a whole in such a way that they are mutually cause and effect of one another." [GS00].

The notion of $(DS)^2$ is quite obvious in developmental biology but this is also a key notion in all biology and has received various names: *hyper-cycle* in the study of auto-catalytic networks by M. Eigen and P. Schuster [ES79], *autopoiesis* in the work by F. Varela on autonomous systems [Var99], *variable structure system* in control theory [Itk76, HGH93], *developmental grammar* in the work of E. Mjolsness [MSR91] or *organisation* in the work of W. Fontana and L. Buss on the emergence of stable structures in chemical open systems [FB94].

However, this kind of system is not only found in biology: modelling of dynamic networks (Internet, mobile networks, etc.), morphogenesis phenomena in physics (growth in a dynamic medium), mechanics of elastic medium or deformable systems, urban development of a system of cities and social networks are full of examples of $(DS)^2$.

IV.4 The Topological Structure of the Interactions of a System

From a computer science point of view, the specification and simulation of $(DS)^2$ raises a problem: what is the language suited to the definition of an evolution function h having a state as argument, state whose structure cannot be defined *a priori*?

The system cannot be simply described in global terms but only by a set of local *interactions* between more elementary entities composing the system. Our problem is to define those entities together with their interactions. We show below that these interactions exhibit a topological structure that can be used to specify them.

The starting point of this analysis[5] is the decomposition of a system *from its evolution*. At a given time t, we decompose a system S into disjoints $S_1, \ldots, S_n$ sub-systems such that the next state $s_i(t+1)$ of sub-system S_i only depends on the previous state $s_i(t)$. In other words, each sub-system S_i evolves independently between time t and $t+1$. S_i is sometimes called a *box*. A box encapsulates the set of elements in interactions in a local evolution step [Rau03]. This notion of box gives account for a notion of modularity: when the evolution takes place, only what is in the box has to be known.

The decomposition of S into the S_i is a *functional partition* that can correspond, but not necessarily, to a *structural decomposition* of the system into components. We should remark that we are taking here the opposite view of an "object oriented approach" that starts by defining the structural components of a system before defining their interactions; here, our starting point are the activities of the system, and we try to deduce a decomposition.

The functional decomposition of S into S_i has to be a function of time. Indeed, if the partition S_i were not a function of time, we would face a collection of parallel systems, totally autonomous with no interaction. Then, it would not be required to consider them simultaneously into an integrated system. Consequently, we write $S_1^t, S_2^t, \ldots, S_{n_t}^t$ for the decomposition of 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. It is convenient

[5] This analysis has been presented in [GMCS05], a shortened version has appeared in [Gia04] and a reformulation, from which we base most of our present developments, has been developed in [Coh04a] and more recently in [Spi06a].

to suppose that at a given time t, one of the S_i^t represents the part of the system that "does not evolve" (and consequently whose evolution function is the identity).

The "global" state of system $s(t)$ can be recovered from the "local" states of the sub-systems: there is 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:

$$s(t+1) = h(s(t)) = \varphi^t(h_1^t(s_1(t)), \ldots, h_{n_t}^t(s_{n_t}(t)))$$

If we follow this analysis, the specification of a $(\mathrm{DS})^2$ requires the definition of three entities:

1. the dynamic partition of S into S_i^t,
2. the functions h_i^t,
3. the function φ^t.

The description of the successive decompositions $S_1^t, S_2^t, \ldots, S_{n_t}^t$ can be based on the notion of *elementary part* of the system: an arbitrary sub-system S_i^t will be a union of elementary parts. Many different partitions of S into elementary parts are possible; here, we focus on the decomposition naturally induced by the set of the S_i^t.

Any two subs-systems S' and S'' of S are interacting (at time t) if there exists S_j^t such that $S', S'' \in S_j^t$. Two sub-systems S' and S'' are *separable* if there is some S_j^t such that $S' \in S_j^t$ and $S'' \notin S_j^t$ or conversely. This leads to consider the set $\mathcal{S}$, called the *interaction structure* of S, defined as the smallest set closed by intersection that contains the S_j^t (see figure IV.1). The elements of $\mathcal{S}$ are sets. The elements of $\mathcal{S}$ that do not include other elements of S besides themselves correspond to the elementary parts.

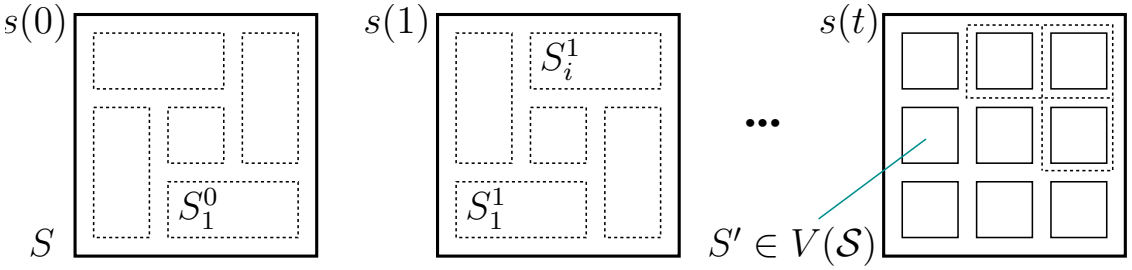


Figure IV.1: The interaction structure of a system S resulting from the subsystems of elements in interaction at a given time step.

Set $\mathcal{S}$ has a natural *topological structure*: $\mathcal{S}$ corresponds to an *abstract simplicial complex*. This notion is a specialization of the notion of cellular complex. An abstract simplicial complex [Hen94] is a collection $\mathcal{C}$ of finite non-empty set such that if A is an element of $\mathcal{C}$, then it is also the case for any non-empty sub-set of A. Element A of $\mathcal{C}$ is called a *simplex* of $\mathcal{C}$; its dimension is equal to the number of its elements minus one. The dimension of $\mathcal{C}$ is the largest dimension of its simplices. Any non-empty sub-set of A is called a *face*. We also define the *set of vertices* $V(\mathcal{C})$, as the union of the sets with one element of $\mathcal{C}$.

The correspondence between the functional decomposition $\mathcal{S}$ and a complex $\mathcal{C}$ is the following: an element of $\mathcal{S}$ is a simplex and the elementary parts of $\mathcal{S}$ correspond to the vertices of $\mathcal{C}$. So, we identify $\mathcal{S}$ with an abstract complex. The elements of $\mathcal{S}$ that are not a face of another element,

called maximal simplices, correspond to the S_i^t. The dimension of a maximal simplex corresponds to the number (minus one) of the elementary parts involved in an interaction.

Topology is the study of connexity: two objects are isomorphic from a topological point of view (and we speak of *homeomorphism*) if there exists a bijection between their parts while maintaining the connectivity relations between these parts. Here, two parts are connected if they interact during (at least) one evolution step of the system: this very abstract notion of neighborhood is captured by the topological structure of $\mathcal{S}$. By definition, only "functionally near" entities are interacting. However, very often, the abstract space of interactions corresponds to the concrete physical space: there are no distant interactions. It is therefore not very surprising that the logical neighborhood exhibited by $\mathcal{S}$ is the same as the spatial neighborhood of the physical parts of the system, as we will see in many examples in the papers.

IV.5 Data and Control Structures for $(DS)^2$

The previous analysis shows that it is possible to specify a $(DS)^2$ S by specifying

- the S_i^t *as a composition of simplices* of $\mathcal{S}$,
- by associating to each S_i^t a function h_i^t,
- by combining the applications of each h_i^t through a function φ^t.

This approach might appear as being unnecessarily abstract but it can be easily interpreted in programming terms:

- The key idea is to directly define the set $\mathcal{S}$ as a data structure. A data structure must therefore be characterized by the neighborhood relationship organizing its elements (the elementary parts of $\mathcal{S}$).
- Then, a function h_i^t allows one to define the evolution of a sub-part of a data structure. The association of a S_i^t and a local evolution function h_i^t can naturally be written by a rule:

  $$S_i^t \Rightarrow h_i^t(S_i^t)$$

- To be generic, the left hand side of the rule must not correspond to a fixed part of the system, but must specify the elementary parts in interactions (a box) to evolve following h_i^t.
- The different applications of rules at a given time have to be controlled and the different results have to be recombined to build the new state of the system. From this point of view, the function φ^t corresponds to both the rule application strategy and to the substitution notion used to apply the rules.

We recover the notions of topological collections and transformation previously introduced: a topological collection corresponds to a structure $\mathcal{S}$ and a topological transformation is the definition of the h_i^t, their domain of application and the substitution function φ^t.

We should note that in a transformation, the specification of the patterns and the functions h_i^t do not change over time. However, the matched parts S_i^t may change. Furthermore, the evolution of one elementary time step corresponds to the application of one transformation and it is possible to use another transformation for the next evolution.

IV.6 The MGS Approach for the Simulation of $(DS)^2$

We can now summarize the MGS approach for the simulation of $(DS)^2$. A topological collection represents the state of a dynamical system at a given time. The elements of the collection can represent entities (a sub-system or an atomic part of a dynamical system) or messages (signals, commands, informations, actions...) addressed to other entities. A sub-collection represents a sub-set of interacting entities together with the messages of the system.

The evolution of the system is specified by a transformation, where the left-hand side of a rule matches an entity together with its message and where the right-hand side specifies the change of the state of an entity, and possible additional messages for other entities.

The neighborhood relationship allows one to take into account many different kind of interactions. For example, if a multi-set organization is taken for the collection, the entities are in a non-structured way of interactions: any element of the collection is able to interact with any other element. Therefore, a multi-set realizes a sort of "chemical soup". More structured topological collections are used to represent spatial organizations and more sophisticated interactions (see the examples in the presented papers below).

More generally, many mathematical models of objects and processes are based on the notion of state that specifies the object or the process by setting data to each point of a physical or abstract space. The MGS programming language facilitates this approach by having many mechanisms allowing one to construct complex evolving spaces and to handle the relations between these spaces and these data.

IV.7 Presentation of the Papers

[GMD03] Jean-Louis Giavitto, **Olivier Michel**, and Franck Delaplace. Declarative simulation of dynamicals systems : the $8_{1/2}$ programming language and its application to the simulation of genetic networks. *BioSystems*, 68(2–3):155–170, Feb/March 2003.
URL: http://www.ibisc.fr/~michel/PUBLIS/2003/biosystem02bis.pdf
Booklet page: 115

The paper that starts this section describes the modelling of a simple dynamical system in $8_{1/2}$. The notion of stream in $8_{1/2}$ allows the direct representation of the trajectory of a dynamical system. Here, the dynamical system described is a regulation network abstracted as a Boolean automata following R. Thomas's approach [Tho78].

[GMCS05] Jean-Louis Giavitto, **Olivier Michel**, Julien Cohen, and Antoine Spicher. Computation in space and space in computation. In Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel, editors, *Unconventional Programming Paradigms (UPP'04)*, volume 3566 of *LNCS*, pages 137–152. ERCIM– NSF, Springer Verlag, 2005.
URL: http://www.ibisc.fr/~michel/PUBLIS/2005/upp04.pdf
Booklet page: 133

Most programming languages are weak at taking into account the modelling problems raised by dynamical systems whose main feature is to have a dynamical structure. The paper [GMCS05] presents the analysis of those systems through the explicit exposure of the interactions between the

parts, as presented in Section IV.4 above. This paper also proposes the idea that many unconventional languages rely on the approach that identifies computation with the simulation of a virtual world abstracted from a physical/biological model (neuronal networks, cellular automata, chemical computing...) and shows how MGS allows the modelling of these physical/biological dynamical models using topological collections and transformations:

Modelling		Unconventional Computing		The MGS Approach
description of the world's laws	↔	program	↔	transformation
state of the world	↔	data of the program	↔	topological collection
parameters of the description	↔	input of the program	↔	input
simulation	↔	the computation	↔	iterative application

The correspondence above must be understood as being wrapped on a cylinder, so that the "description of the world's laws" are related to "transformation", etc.

IV.7.1 Biological Applications

[GMM04] Jean-Louis Giavitto, Grant Malcolm, and **Olivier Michel**. Rewriting systems and the modelling of biological systems. *Comparative and Functional Genomics*, 5:95–99, February 2004.
URL: http://www.ibisc.fr/~michel/PUBLIS/2004/CFG04.pdf
Booklet page: 151

The declarative approach to modelling and simulation of biological systems is presented in the paper [GMM04]. Here, the declarative approach considered consists in "classical" rewriting systems. We show that, even restricted to terms (or trees), rewriting systems are poweful enough to model complex objects, from biochemical interaction networks to plant growth, if the structure of the system is restricted to a complete graph, a linear or a tree-like organization (the MGS project is partly motivated by going further than those algebraic data types).

[GM03] Jean-Louis Giavitto and **Olivier Michel**. Modeling the topological organization of cellular processes. *BioSystems*, (70):149–163, 2003.
URL: http://www.ibisc.fr/~michel/PUBLIS/2003/biosystem02.pdf
Booklet page: 159

The paper [GM03] illustrates the application of these ideas in the MGS project on biological processes at the cellular level. This paper also presents a first informal introduction to MGS and details various examples.

[SM05] Antoine Spicher and **Olivier Michel**. Using rewriting techniques in the simulation of dynamical systems: Application to the modeling of sperm crawling. In *Fifth International Conference on Computational Science (ICCS'05)*, volume I, pages 820–827, 2005.
URL: http://www.ibisc.fr/~michel/PUBLIS/2005/iccs.pdf
Booklet page: 177

One of the questions that naturally arises with this approach is: (1) how to deal with non tree-like structures and (2) is it possible to go beyond discrete structures. Paper [SM05] shows that the MGS approach can address non tree-like structures like planar graphs that are changing (with non-local update rules) over time. It also shows how a discrete formalism can deal with a continuous object by "labelling"a discrete organization with continuous values (floating point values). This idea originates from the L-systems with the notion of *module*. As a matter of fact, we are no longer dealing with simple discrete structures theoretically studied by formal languages theoricians or in the rewriting community. We show that the mechanisms described allow to solve partial differential equations by an explicit method.

[SMC+07] Antoine Spicher, **Olivier Michel**, Mikolaj Cieslak, Jean-Louis Giavitto, and Przemyslaw Prusinkiewicz. Stochastic P systems and the simulation of biochemical processes with dynamic compartments. *BioSystems*, 2007.
URL: http://www.ibisc.fr/~michel/PUBLIS/2007/biosystem07.pdf
Booklet page: 187

The previous paper is an example of how a "complex" state can be handled but the dynamics itself can also be more complex than the iteration of a maximal parallel rule application strategy (as exemplified in L-systems). The MGS idea is to handle sophisticated dynamics as various rule application strategies. The problem described in this paper [SMC+07] is how to take into account the exact kinetics of chemical reactions (in nested compartments). The work presents how Gillespie's algorithm [Gil77] used for the simulation in chemistry can be expressed as a stochastic rewriting strategy.

IV.7.2 Non-Biological Applications

If biology is an obvious domain for the applications of our language concepts, it is important not to ignore the adequacy of these concepts to more classical computer science domains. We have, for example, described how to implement in MGS classical examples (graph algorithms: Hamiltonian path, maximum flow); optimization (*à la* Gamma: knapsack, maximal segment sum...); algorithmics (computation of prime numbers...); and less classical examples (bead sorting, CAD surface subdivision algorithms...).

[MJ05] **Olivier Michel** and Florent Jacquemard. *An Analysis of a Public-Key Protocol with Membranes*, pages 283–302. Natural Computing Series. Springer Verlag, 2005.
URL: http://www.ibisc.fr/~michel/PUBLIS/2005/nspk-05.pdf
Booklet page: 205

The paper [MJ05] relies on the topological collections and the transformations to analyse the Needham-Schroder public key protocol following three different strategies. The underlying general problem is the generation and the exploration of a large state space. This work is done in the framework of the P systems but keeps all its generality.

[SMG05] Antoine Spicher, **Olivier Michel**, and Jean-Louis Giavitto. Algorithmic self-assembly by accretion and by carving in MGS. In *7th International Conference on Artificial Evolution*, 2005.
URL: http://www.ibisc.fr/~michel/PUBLIS/2005/ea05.pdf
Booklet page: 227

The last paper [SMG05] in this section concerns the construction of several fractal objects using some advanced operations of "topological surgery". The fractal objects are usually built by iterative addition, *ad libitum* (following a *self-assembly* process). Here, we construct some of the objects by removing parts in excess (following a *carving* mechanism) starting by a larger envelope of the final surface or volume. This example is important since it shows the capacity of MGS to express, using rules, how to shape complex multi-dimensionnal structures.

Chapter V

Elements of Implementation

V.1 Introduction

A huge implementation effort was required so that the concepts introduced in Chapter III could be validated by the examples of Chapter IV. We give in this chapter some elements on that implementation work. The reader can figure out the amount of this implementation through the tools and coding volume that it represents. All code is written under the GPL licence and all sources are freely available.

The implementation effort has been done by J.-L. Giavitto, J. Cohen (Ph.D. student), A. Spicher (Ph.D. student) and myself.

V.1.1 The 81/2 Language

The language was written in Ocaml with a data-parallel virtual machine written in C; about 36k lines of total code. Available here `http://www.ibisc.univ-evry.fr/pub/Otto/`

V.1.2 The Amalgams Formalism Interpreter

The interpreter was written in Ocaml and C; about 10k lines of total code. The distributed version allowed to run a nameserver on client/server interpreters running on various machines; the source code is available at `http://www.ibisc.univ-evry.fr/~michel/amal_d.tar.bz2`

V.1.3 The MGS Programming Language

Two versions of the interpreter have been written:

MGSv_1 written in Ocaml with a virtual machine based on Landin's SECD machine; about 9k lines of total code. Available as a Debian package here http://www.ibisc.univ-evry.fr/~michel/1_MGS/

MGSv_2 written in Ocaml, C and C++ with many external libraries (gsl, qhull, nauty...) and a virtual machine based on higher-order abstract syntax; about 50k lines of total code. Available here http://www.ibisc.univ-evry.fr/~michel/2_MGS.tar.bz2

V.1.4 The Imoview Vizualizing Tool

All the pictures shown on the MGS website, presentations and papers correspond to output produced by the language and further interpreted by the Imoview vizualizing tool. Imoview has been developed in two versions:

Imoviewv_1 written in Ocaml and C code; about 11k lines of total code [Tho03]. Available here http://mgs.ibisc.univ-evry.fr/ImoView05/index.html

Imoviewv_2 written in Ocaml and C code; about 15k lines of total code [Kal07]. Available here http://www.ibisc.univ-evry.fr/~michel/Imoview2.tar.bz2

V.1.5 The PatchGen Patch Pattern Editor

The writing of patch patterns for complex topological modifications requires to be very precise since it is easy to produce a wrong pattern by mismatching the variables names between the left hand side and the right hand side of a rule. To ease the writing of highly symmetrical patterns as in the example of surface subdivision [SM07], a pattern editor [Jul05] has been developed. It is available here http://www.ibisc.univ-evry.fr/~michel/PatchGen.tbz2

V.2 Presentation of the Papers

[Mic96a] **Olivier Michel**. Design and implementation of 81/2, a declarative data-parallel language. *Computer Languages*, 22(2/3):165–179, 1996. special issue on Parallel Logic Programming.
URL: http://www.ibisc.fr/~michel/PUBLIS/1996/ComputerLanguages.pdf
Booklet page: 243

The first paper [Mic96a] in this section presents the design and implementation of the first language I have been working on: the 81/2 language. The notions of streams and collections are presented, together with a new data structure combining those two: the fabric. A fabric allows the representation of space-time phenomena. Examples of 81/2 programs are given, involving the dynamical creation of spatial structures and some elements of implementation are sketched at the end of the paper.

[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.

URL: http://www.ibisc.fr/~michel/PUBLIS/2001/entcs01.pdf
Booklet page: 261

The second paper [GM01c] gives a general view of the first elaboration of the second language that I have been working on: the MGS language. The motivations for a new programming language are recalled, asking for the development of a new data structure and its associate control structure: *topological collections* allow the uniform representation of spatial phenomena and *transformations* their modifications as the system evolves. Many examples are given and a comparison with similar formalisms and languages (Gamma and the CHAM, P systems and L-systems, and cellular automata) are detailed.

[GMC02] Jean-Louis Giavitto, **Olivier Michel**, and Julien Cohen. Pattern-matching and rewriting rules for group-indexed data structures. *ACM SIGPLAN Notices*, 37(12):76–87, December 2002.
URL: http://www.ibisc.fr/~michel/PUBLIS/2003/sigplan03.pdf
Booklet page: 283

MGS transformations are sets of rules. The paper [GMC02a] presents the pattern matching language and a generic algorithm applied to the specific case of group-indexed data structures (GBFs). One of the main achievements of the MGS project is to have defined a generic matching procedure that is used for all topological collections (we currently have 12 different topological collections types in MGS).

[MG07] **Olivier Michel** and Jean-Louis Giavitto. Incremental extension of a domain specific language interpreter. In *International Workshop on Implementation and Application of Functional Languages (IFL07)*, Freiburg, Germany, September 2007.
URL: http://www.ibisc.fr/~michel/PUBLIS/2007/ifl07.pdf
Booklet page: 297

The last paper [MG07] in this section is not a major result but nevertheless it had a great impact on the development of the MGS language. Indeed, when the project started in 2000, a first version of the language based on the SECD virtual machine was quickly developed: it involved 3 collection types and consisted in 9k lines of Ocaml, C and C++ source code. Seven years later, the language now has 12 different topological collections types and 12 scalar types, consists in 50k lines of Ocaml source code and many libraries, is scattered over 75 files and has 225 user functions. To keep such a growing project maintainable, we had to define some strategies for allowing easy and fast incremental construction, the revision and the fast prototyping of the interpreter. The resulting software architecture is presented in that paper.

We do not detail further our implementation work, but additional elements can be found in [Seg97, Out98, Del02, Lar02, Spi03, Man04, Bou04, Coh04b, Per05, Jul05, Gau05, Spi06b, Kal07].

Chapter VI

Programming the Small and Programming the Large

The unconventional languages 81/2 and MGS have been motivated by the modelling and simulation of dynamical systems. More generally, unconventional languages can be motivated by a dissatisfaction in conventional programming languages, new demanding application domains or new opportunities brought by new computing media. To conclude this document, we want to convince the reader that nowadays these three driving forces converge to the *same problem*: the *engineering of a population of entities*.

The following three sections will sketch the current dissatisfaction in conventional software developments, some of the new problems that arise with ubiquitous large-scale applications[1] and the new computing opportunities offered by the advances in molecular biology.

We postulate that these three domains lead to the same problem: how to design and control a large dynamical population of unreliable entities to obtain a global coherent and stable behaviour. This observation meets the vision brought by H. Abelson and G. Sussman on the programming of *amorphous media* [AAC+00].

Unconventional programming languages are vehicles to investigate this field and offer fundamental tools for the simulation and analysis of such systems as well as for their design or their exploitation.

[1] We do not want to enter into the debate of the differences between "ubiquitous", "pervasive" and "ambient" computing. Here we refer to the shift from regular desktop computing running monolithic programs to distributed computing resources available everywhere at anytime.

VI.1 Facing the Software Crisis

The imperative, functional, object and logical paradigms still do not have met the high expectations of the 80's and the 90's in terms of *programmability* (ease of programming, modularity, expressiveness, encapsulation...), *quality of service* (correctness, validation, security, trustability...), and *evolution* (deployment, reusability, adaptation, portability...). For example, there is still no clear agreement on a model for parallel programming or on a programming model best suited to develop provably correct code[2].

G. Sussman, in 1999, has illustrated the problem as follows [Sus99]

> Computer Science is in deep trouble. Structured design is a failure. Systems, as currently engineered, are brittle and fragile. They cannot be easily adapted to new situations. Small changes in requirements entail large changes in the structure and configuration. Small errors in the programs that prescribe the behaviour of the system can lead to large errors in the desired behaviour. Indeed, current computational systems are unreasonably dependent on the correctness of the implementation, and they cannot be easily modified to account for errors in the design, errors in the specifications, or the inevitable evolution of the requirements for which the design was commissioned. (Just imagine what happens if you cut a random wire in your computer!) This problem is structural. This is not a complexity problem. It will not be solved by some form of modularity. We need new ideas. We need a new set of engineering principles that can be applied to effectively build flexible, robust, evolvable, and efficient systems.

See also the notes of the debate "Object have failed" organised by R. Gabriel at OOPSLA 2002 [Gab02].

At the same time, developers have to face a new situation: the proliferation of the hardware and software environments, the increasing demands of the users, the multiplication of the programs, the integration of the functions within the same interfaces and the sharing of information, competences and services thanks to the generalisation of data-bases and communication networks.

Furthermore, a program is no longer a monolithic entity conceived, produced and finalised by a tightly coupled development team before being used [Fis01]. A program is now developed by hundreds of programmers distributed around the world, connected through repositories like `SourceForge`[3]. *Open source software* asks for new concepts and tools to solve the problems associated with its distributed development, management and distribution[4].

The current situation is not adequately supported by the traditional life-cycle of programs like the waterfall or the iterative software development processes.

[2] This section has been written initially in the preparation of the UPP Workshop [GMCS05]; some elements come also from the file prepared for the Dagstuhl Workshop [DHGG06] and from the technical report [GSM07].

[3] `http://sourceforge.net/`

[4] Cf. for instance the European research project EDOS `http://www.edos-project.org`. To give a quantitative idea of the problems, the full `Debian` Linux distribution represents around 19000 packages that must work together. The various versions of these packages represent 40000 software units. Expressing the dependencies as a logical formula, a package like `KDE` implies more than 30000 variables [BCD+06].

VI.2 New Massive Software-Intensive Systems

Ubiquitous large-scale applications, managing several thousands of processing elements, are already common: think of grid systems like `SETI@home`[5], peer-to-peer systems like the `Mule` [Ora01], popular Internet applications like `Google` [BP98] or the management of the mobile phones in an urban area. Computing with thousands of distributed entities provides challenges asking for the development of new paradigms. One important challenge is to ensure, under changing environments, global properties of the network as a whole: "how do we obtain coherent behaviour from the cooperation of large numbers of unreliable parts that are interconnected in unknown, irregular, and time varying ways?" [Mac03].

If the new applications can rely on, or must cope with, an unbounded number of computing resources, they become also *unbounded in time*: a program is no more a monolithic function designed, produced and finalised before being used, run on an input to give, after some time, an output. The `Google` portal cannot stop, the `mule` network must ensure its functionality despite the constant connections and disconnections and despite the evolution of communication protocols (*e.g.*, the current `MLdonkey` program manages more than a dozen of successive versions for more than five different P2P protocols [FP07]). In the context of highly decentralised and incremental development and deployment practices, a program is now seen as an open and adaptive frame, implementing an extremely long-lived reactive system which, for example, can dynamically incorporate services not foreseen by the initial designer. Such systems are required to be future-proof, able to preserve and update their original functionality in a machine-independent way, and ultimately by being self-sustaining and evolving.

To engineer such systems, the *autonomic computing initiative* [Hor01] proposes to relies on properties like

- *self-organisation* (autonomous configuration of the components into a dynamic architecture dedicated to the satisfaction of the specified requirements),
- *self-healing* (autonomous detection and correction of hardware and software faults),
- and *self-optimisation* (autonomous monitoring, control of resources and reconfiguration to ensure an optimal functioning),

to achieve self-managing computers and software. Systems powered by this model would offer the ability to adapt themselves to the surrounding environment in a totally unsupervised but consistent way, ensuring robustness, fault-tolerance and scalability, while responding to increasing expectation for trustworthy, dependable and long-lasting systems.

The autonomic approach is seducing and can be summarised by "let the system take care of itself" but this does not give any clue to ensure some global properties from the local changes: how to engineer self-∗ properties?

VI.3 New Computing Media: Computing at the Nanoscale Level

The failure of the traditional approaches of software life cycle and the emergence of new applications are not the only motivations to look for new programming paradigms. Our computing machines may change drastically in a near future.

[5] `SETI@home`, `http://setiathome.berkeley.edu/`, is currently the largest distributed computing effort with over 3 millions CPUs.

Research for building and manipulating objects at the nanoscale level in physics, chemistry and molecular biology reveals promising resources *almost* available for unconventional computing [MS03, SBC+06], as foreseen by R. Feynman [Fey60] and later popularised by E. Drexler [Dre81]. In the field of molecular biology, harnessing molecules to compute can be traced back at least to [Hea87] and research in this field explodes with the landmark experiment of L. Adleman [Adl94]. Molecules can be used for their physical interactions or their chemical reactions [HH98, Mac02, RPW03] or, in a biological context, using the gene regulation machinery of a cell to achieve some computations [BHS04, KKA+04, SSZW06]. These computations can be designed and implemented directly "by hand" or using directed evolution [YWA02, FH04]. At last but not least, *synthetic biology* [WK00, WBH+03, Gib04] emerges as a new engineering discipline at the convergence of genetic engineering and computer science, for the design and the implementation of complex artificial biological systems for a variety of applications[6]. In this domain, the pace of the technological changes recalls the glorified Moore's law [Car03].

All these new possible carriers for computation have in common a large number of unreliable elementary entities that interact and cooperate dynamically and randomly. It is nevertheless necessary to ensure a global emergent[7] behaviour, robust and persistent in time, that can serve as a foundation for computing.

VI.4 A New Playground

I really want to stress that the challenge raised by computing at the nanoscale (the small) *meets* the challenge raised by the programming of the new massive and software intensive applications (the large), even if in one case entities are molecules and in the other case entities are software units and running processes spawned around the world. In both cases, the medium to inform or to program is *amorphous* in the sense introduced by H. Abelson and G. Sussman: a highly redundant, massively parallel, asynchronous system, without assumptions about the precise interconnection or the precise geometrical arrangement of its parts, exhibiting a dynamic topology and a varying logical organisation, constrained to local interactions between its parts and by the short relative lifetime of the basic components. The structure is dynamic because elements can join or leave the medium and they can fail, their interactions are variable in time following the available resources, the current goal of the system and the evolution of each of its part.

The Programming Language Strategy

Obviously, an amorphous medium can be seen as a $(DS)^2$ (see Section IV.3). In other words, the design of self-managed amorphous systems requires the development of a constructive theory of *distributed $(DS)^2$ without a global time or a global state*. And I advocate that unconventional programming languages have a key role to play in this program.

[6] The web site `http://syntheticbiology.org/` is a good introduction. A French resource (under construction) is `http://www.ibisc.univ-evry.fr/pub/pmwiki/pmwiki.php`. The web site of **iGEM**, the international Genetically Engineered Machine competition, gives an outline of the envisioned applications: `http://parts.mit.edu/wiki`.

[7] I am quite reluctant to use a word that has, over the years, received so much attention with the only result of confusing things to a point where no clear definition *emerges*. I stick with the characterisation given by S. Stepney at the "Grand Challenge in Non-Classical Computation International Workshop" that took place in 2005 in York (quoting from memory – the focus on the word "language" is mine): "[...] emergence is when a phenomenon taking place at a given level cannot be described with the **language** used for the entities at the origin of the phenomenon [...]".

In the following, I will outline some thoughts that have emerged inside the MGS project in front of the problems of synthetic biology. They are very preliminary and speculative, but they suggest a new and attractive playground that prompts research into the expressiveness, semantics and implementation of alternative programming languages and systems architectures, as well as into the algorithmic complexity and optimisation of unconventional programs.

Simulation. More than 90% of the parts designed for the iGEM competition are non functional. One of the main reason advocated is the lack of time: testing and tuning the design *in vivo* takes too much time. Simulation may shorten the validation of the design of new biological parts. That is not to say the *in silico* approach will magically solve the problem: biological processes are largely undetermined by the available biological data, these processes are highly dynamic and their simulation can be very difficult. But it is easier to change the parameters of a simulation and to restart it than to redo a wet lab experiment.

The MGS project is an attempt to provide some solutions for the simulation of $(DS)^2$ like those arising in a cell. Other approaches are certainly possible, *e.g.*, in the field of multi-agent systems. In any case, additional efforts are needed, *e.g.*, to face the amount of entities to be represented. For example, a quorum sensing between few bacteria takes minutes in MGS and the simulation of the growth of the meristem takes a few hours for the evolution of a few thousands cells. In contrast, the migration of a population of *E. Coli* in a Petri dish in response to a chemical gradient is out of reach. Such a simulation requires new concepts to represent efficiently (*e.g.*, symbolically) gradients and population mixing discrete and continuous approaches. This problem was the subject of the work started during the visit of D. Coore [Coo99], this summer in Évry.

Algorithmics. Self-$*$ behaviours can been seen as the *stabilisation* of the system on a *fixed point* after a transient perturbation. Such idea is already investigated in domain like self-stabilising algorithms [Tix06] or parallel asynchronous associative computations [Duc99].

To infer that the asynchronous parallel application of local rules leads to stable points exhibiting some required properties is a difficult problem, but some theoretical tools already exists like induction principles on multi-sets and other topological collections [DM79], asynchronous iterations [Ber83], or termination techniques developed in rewriting systems [Cha90].

One of the lessons of MGS is the importance of the topological structure induced by the interactions in the system. The direct interactions of arbitrary elements in a system are not always allowed nor desirable. The corresponding neighborhood relationship is a constraint that can be used to control the global behaviour of the system. For instance, 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 minimise the computation time.

In addition, the evolution of the system creates by itself new constraints that shape the further possible element's future (downward causality as described Section IV.3). We are still missing a useful theory of the feedback between a form and the processes that take place within this form. Such theory will be central in the design of self-organising processes. However, some "design patterns" can be gathered in the literature on developmental biology [Mei82, WSJ+98, Pru00].

Compilation. The previous remarks aim at identifying tools that can be used to validate a system. On the contrary, we can imagine that the properties required for an amorphous system are not proved *a posteriori* on a design coming "out of the blue" but preserved through an iterative

compilation chain going from the high-level global specification of the system down to the definition of the local behaviour of each of its elements.

Low-level "assembly" languages are depending of the final target. In synthetic biology, this role can be played by the *BioBricks* [KRC+07] currently developed.

At a global level, the programming languages allowing the observation, the organisation, the control and the use of a collection of elementary spatially located entities whose behaviour and structure are changing in time are still to be imagined. We believe that, to be successful, a candidate will have the ability to express (1) interaction (local influence of an entity over another), (2) cooperation (necessary interactions required to meet a global goal), (3) regulation (global constraints on the behaviour of the entities) and (4) development (evolution of the structure of the system).

Such mechanisms link a global object (*e.g.*, a gradient, a wave) or a global behaviour (a stabilisation, a cyclic trajectory) to some local objects and behaviours. This problematics is encountered for instance in differential geometry: a differential form mixes local objects (the derivatives of a function) with global ones (the function itself). Existence theorems for a differential equation or a partial differential equation ensure the existence of a global object from the local properties. The "wave equation" and the "diffusion equation" can be seen as abstract programs and "integration" is the principle used to build (compute) the global object from the local constraints. This parallel has motivated the investigation of a notion of discrete differential form in MGS [Spi06b] and we want to pursue the work undertaken.

VI.5 An Plea for Further Interdisciplinary Dialogues

To conclude this document we want to acknowledge the necessity of interdisciplinary *dialogues* between computer science and other scientific fields.

Computation is the notion studied by computer scientists and unconventional languages are different viewpoints used to capture this subtle notion. This notion has revealed a conceptual importance in other sciences, as acknowledged by W. Fontana [Fon06]

> Over the past half-century, the idea has taken hold that physical processes, particularly in biological systems, can be understood as computation. A back-and-forth between transparent experimental models of molecular computation and the development of formal tools for reasoning about concurrent behaviour might lead to a better appreciation of what it means for cells to "compute", "organise", or "process" information and, perhaps, evolve.

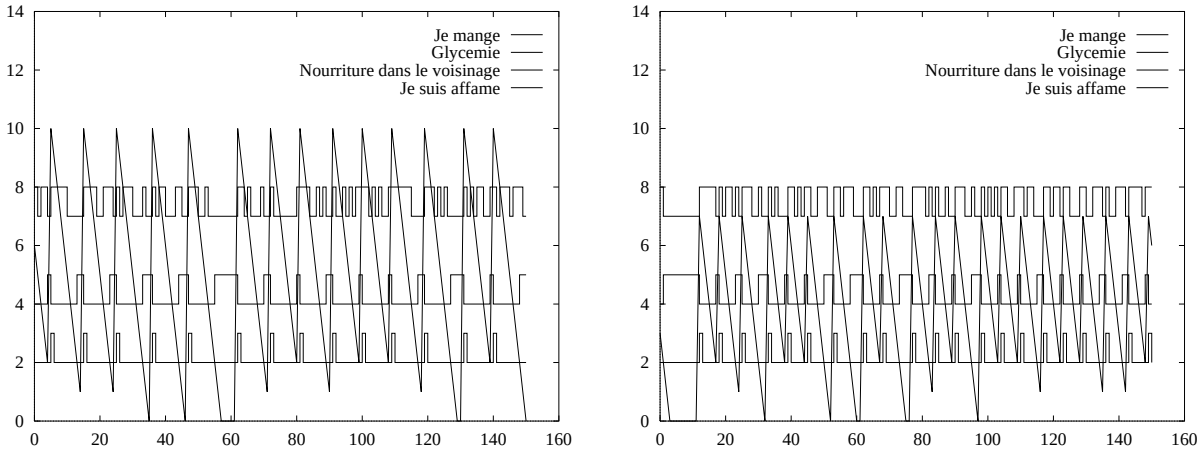
At the same time, the travelling of concepts between sciences is not one way and computer science is fertilised by notions developed in other fields. In the landscape sketched in this chapter, statistical physics, dynamical system theory, developmental biology, evolution theory, etc. will certainly change profoundly the way we evaluate and compile our programs and will expand our understanding of what a computation is. And we believe that synthetic biology will be a place for these dialogues.

Appendix A

Graphic Gallery

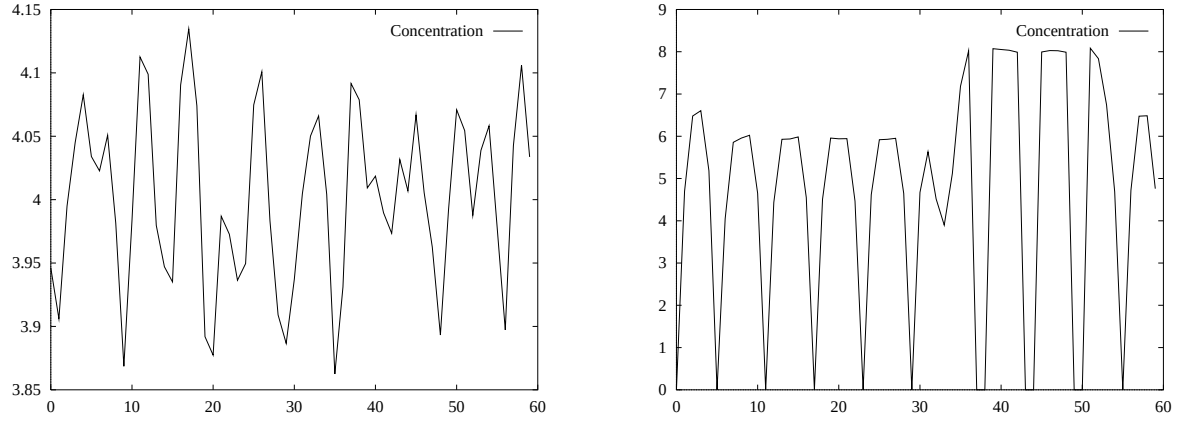
In this appendix are given a set of graphical illustrations that do not appear in cited papers. We hope that these elements will help the reader to appreciate the scope of this work. For further elements, the interested reader can refer to the MGS home page at http://mgs.ibisc.univ-evry.fr

A.1 Illustrations related to the $8_{1/2}$ project



(a) A first evaluation (b) An evaluation starting with different intial values

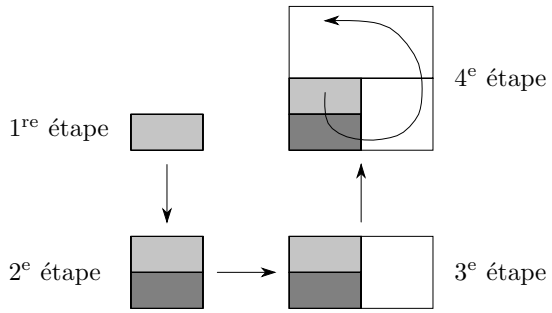
Figure A.1: Hybrid dynamical systems in $8_{1/2}$. A *wlumf* is an "animat" whose behavior (eating) is triggered by the level of some internal state [Mae91]. More precisely, a wlumf is *hungry* when its *glycaemia* is under some level. It is able to eat when food is available in its environment. Its metabolism is such that when it eats, the glycaemia goes up to some maximum value and then decreases to zero at a rate of one unit per time step. Essentially, the wlumf is an hybrid dynamical system made of counters and flip-flop triggered and reset at different rates. Two chronograms of a wlumf behavior are given. They correspond to different initial states and different food in the environment. The corresponding $8_{1/2}$ program is less than 10 lines of code.



(a) State of the system after 100 steps of simulation (b) State of the system after 1000 steps of simulation

Figure A.2: Numerical resolution of a diffusion/reaction in a torus, following A. Turing [Tur52]. The result of the simulation is represented after 100 time steps (sub-figure (a)) and after 1000 time step, once that the phenomenon described by the dynamical system has stabilized. The variations correspond to the quantity of morphogenes and can be interpreted as periodically located spots on the torus.

(a) A "real" ammonite shell.



(b) Sketch of the four first steps of the development of the ammonite shell..

Figure A.3: Modelling of the growth process of an ammonite shell. An ammonite grows by building a new shell leaned to the previous one. This process corresponds to the construction of a *gnomon* which indicates in geometry a shape that can be added to another shape to produce a larger shape similar to the original [Mic96c].

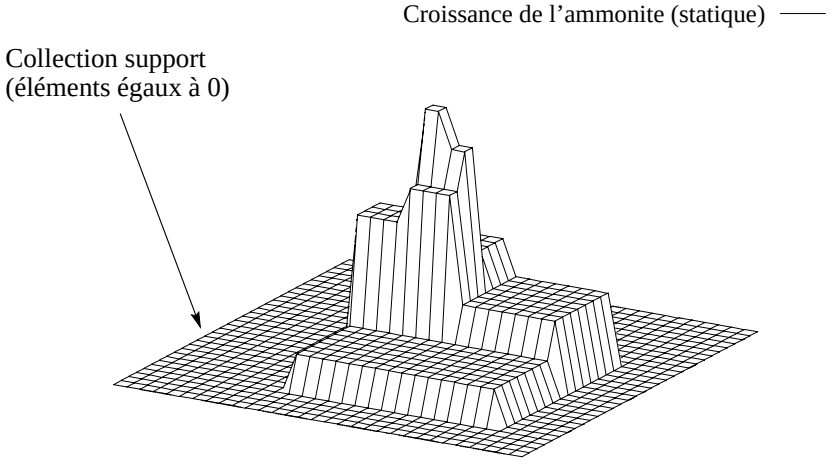


Figure A.4: Simulation in $8_{1/2}$ of the growth process of the ammonite. Cells corresponding to the same generation in the shell are located at the same height.

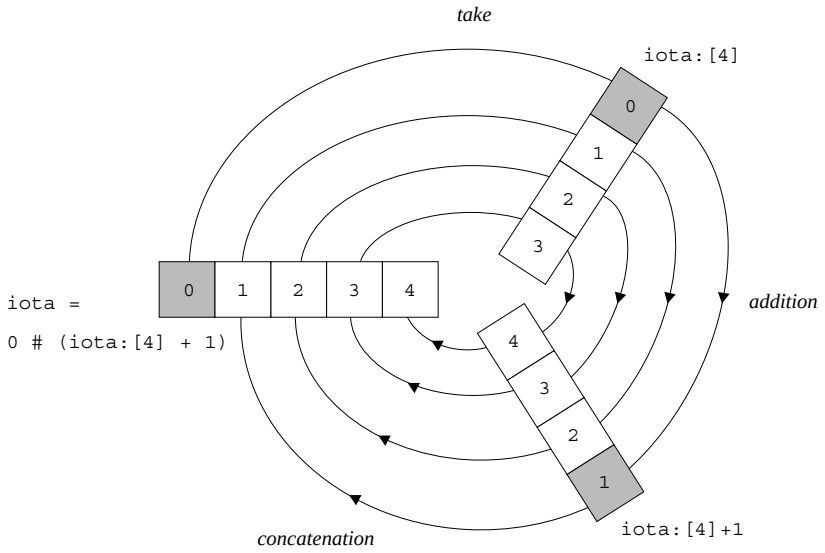
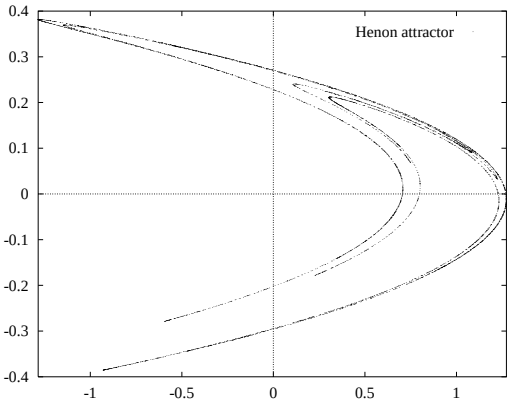
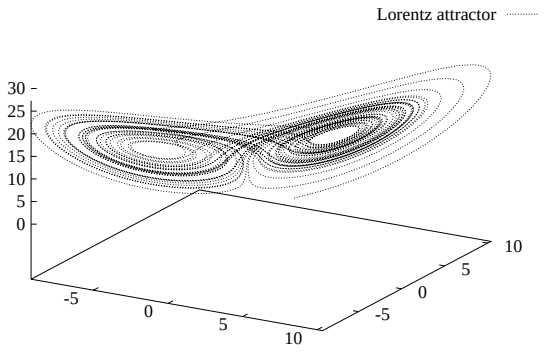


Figure A.5: Resolution of a recursive equation on vectors: `iota` $= 0\#(1 +$ `iota` $: [4])$ by an iterative method (# denotes vector concatenation, : [4] truncation to 4 elements, scalar constant are overloaded and represents constant vectors). Static analysis [Gia92] is used to infer that `iota` has 5 elements. In the diagram, the index corresponds to the time generation of the value of the collection `iota`. At the initialization, only the value of the first element is known; then, the computation propagates to the right and the set of values of the collection is computed. This method [Gia00, GM01] is implemented in the $8_{1/2}$ interpreter [Mic96a].

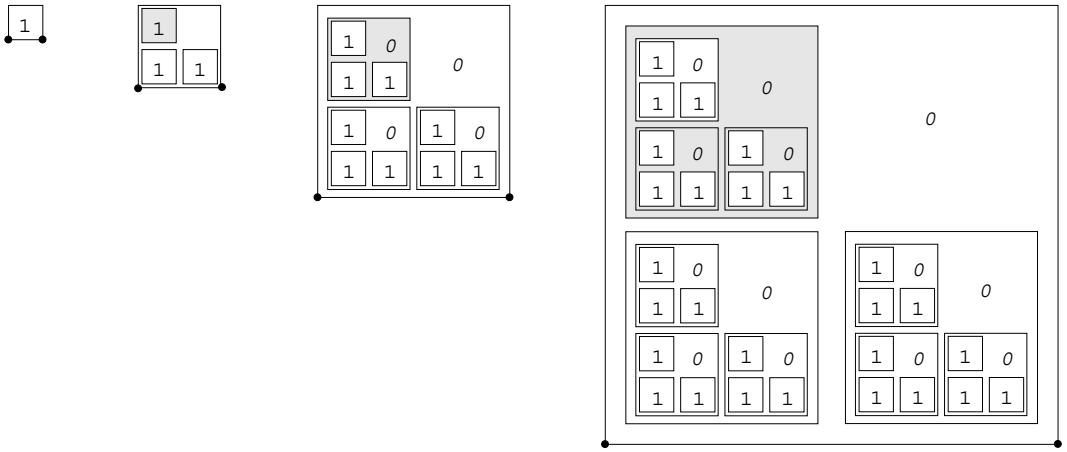


(a) Hénon's attractor in 2D.

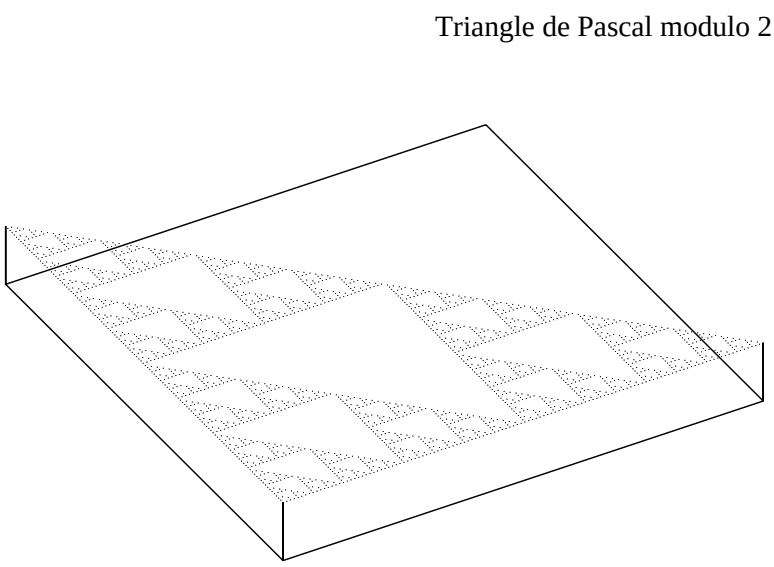


(b) Lorentz's attractor in 3D.

Figure A.6: Plots of Henon and Lorentz attractors [PJS92]. A $8_{1/2}$ stream is used to represent the dynamical system trajectory.



(a) Recursive construction of Pascal's triangle modulo 2. Points correspond to the vertices where the triangle is repeated. The initial triangle is the construction in grey. We have sketched four successive steps of the construction.



(b) Result of the 7 first steps of the computation of Pascal's triangle modulo 2. The dump is made using an interface to Gnuplot.

Figure A.7: Recursive construction of Pascal's triangle modulo 2 [Ste95]. Method and results of the execution of the $8_{1/2\mathcal{D}}$ program.

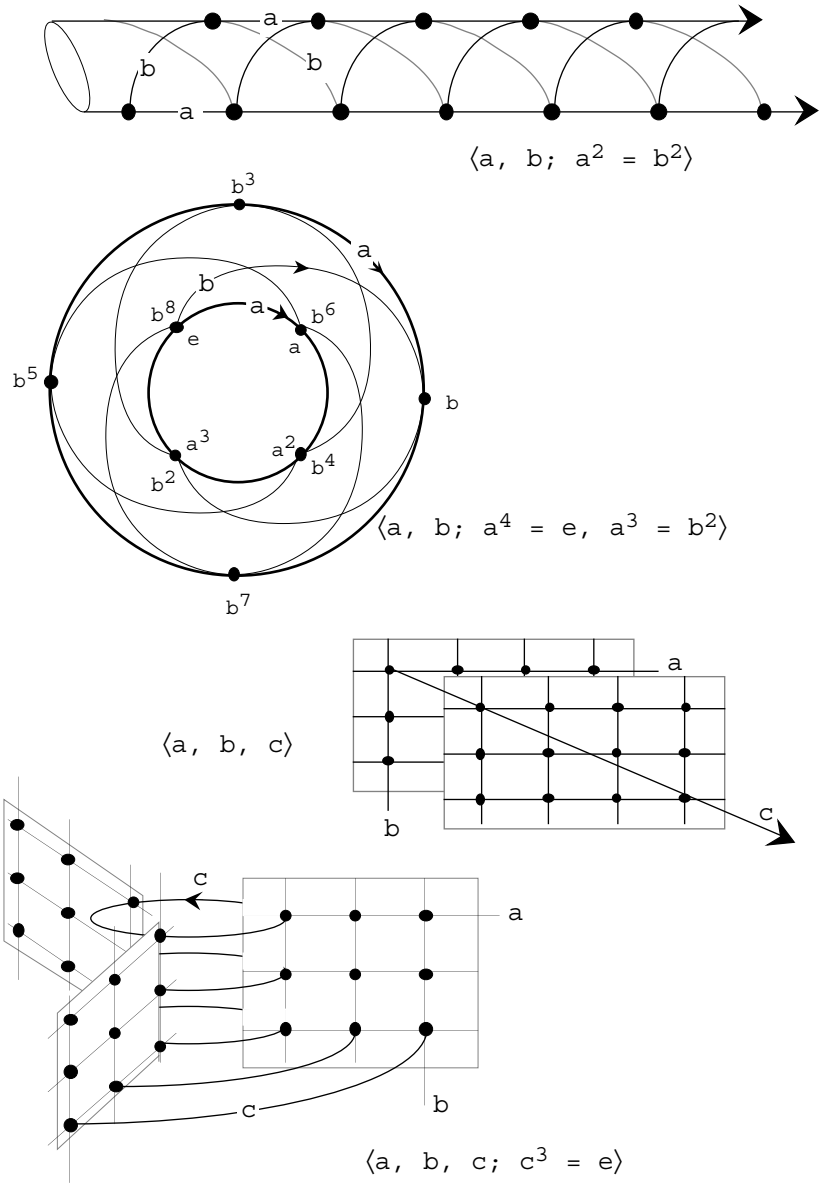


Figure A.8: Four presentations of abelian groups and their associated Cayley graph [GMS95, Mic96c, GM01]. Such presentation can be used to define a GBF type. Abelian GBF have been implemented in $8_{1/2}$ and MGS.

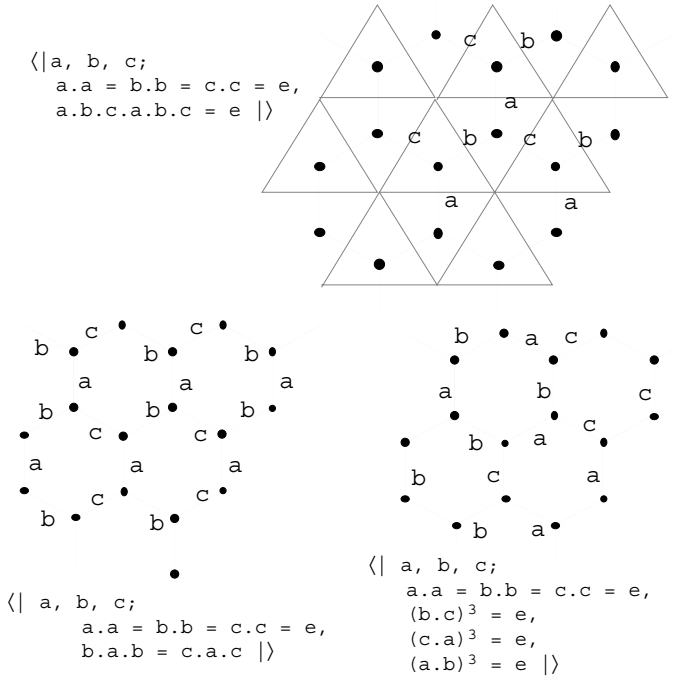


Figure A.9: Three examples of shapes with three neighbours; these are non-abelian shapes.

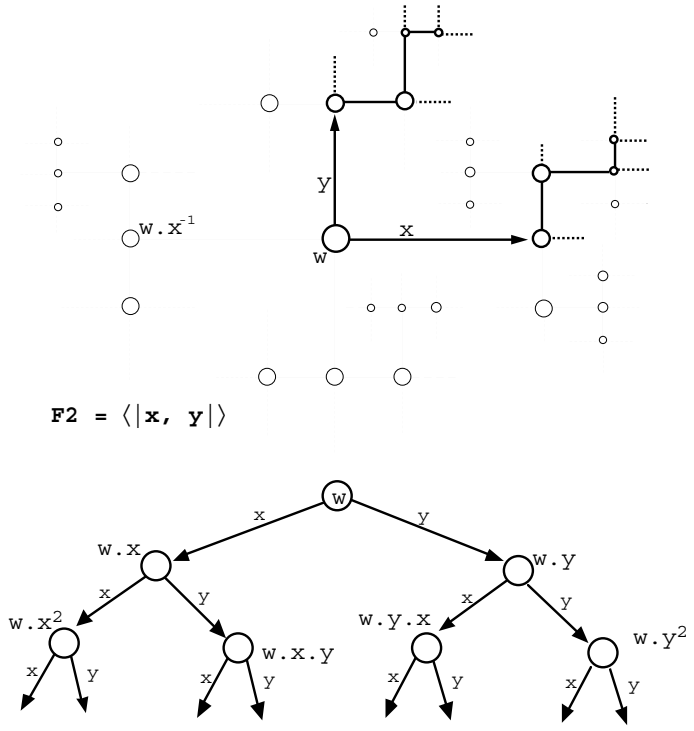
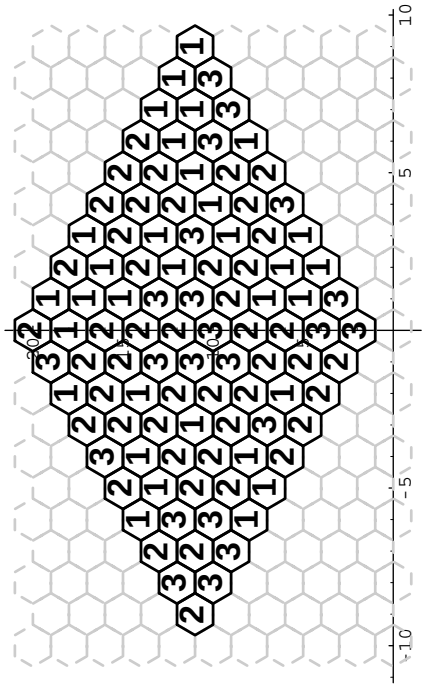
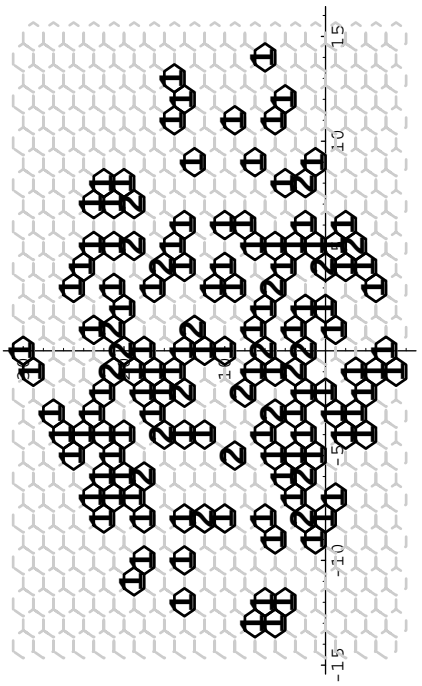


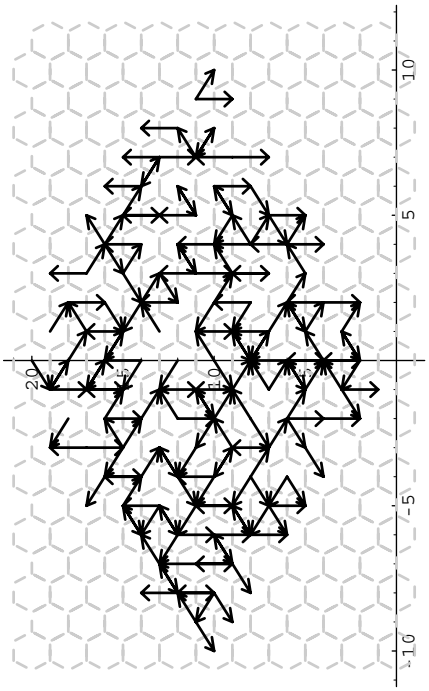
Figure A.10: A free non-abelian group with two generators. The bold lines correspond to vertices that can be reached from vertex w by following elementary moves x and y.



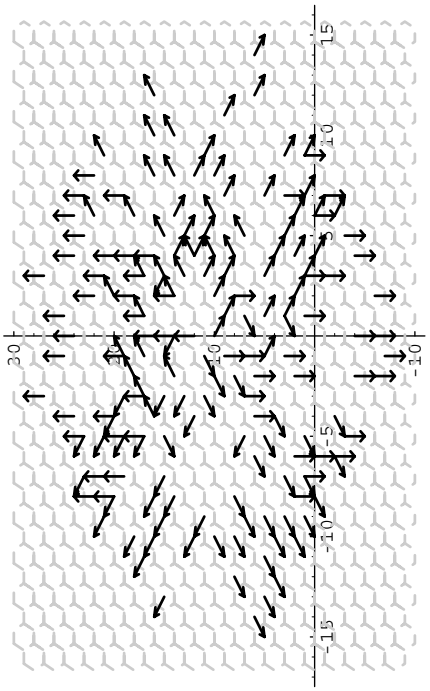
(a) Initialization



(b) After 5 steps of evaluation



(c) Initialization



(d) After 5 steps of evaluation

Figure A.11: Two examples of the evolution, with the same set of rules, but starting from two different initial situations, of a lattice-gas simulation [TN87] performed on an hexagonal lattice (using a GBF topological collection) in $8_{1/2}$. At instant 0, the particles are concentrated in a small domain. The pressure leads to the expansion of the gas ball. The numbers in the cells correspond to the number of particles on each site. The vectors correspond to the speed vector of the particles [Mic96c].

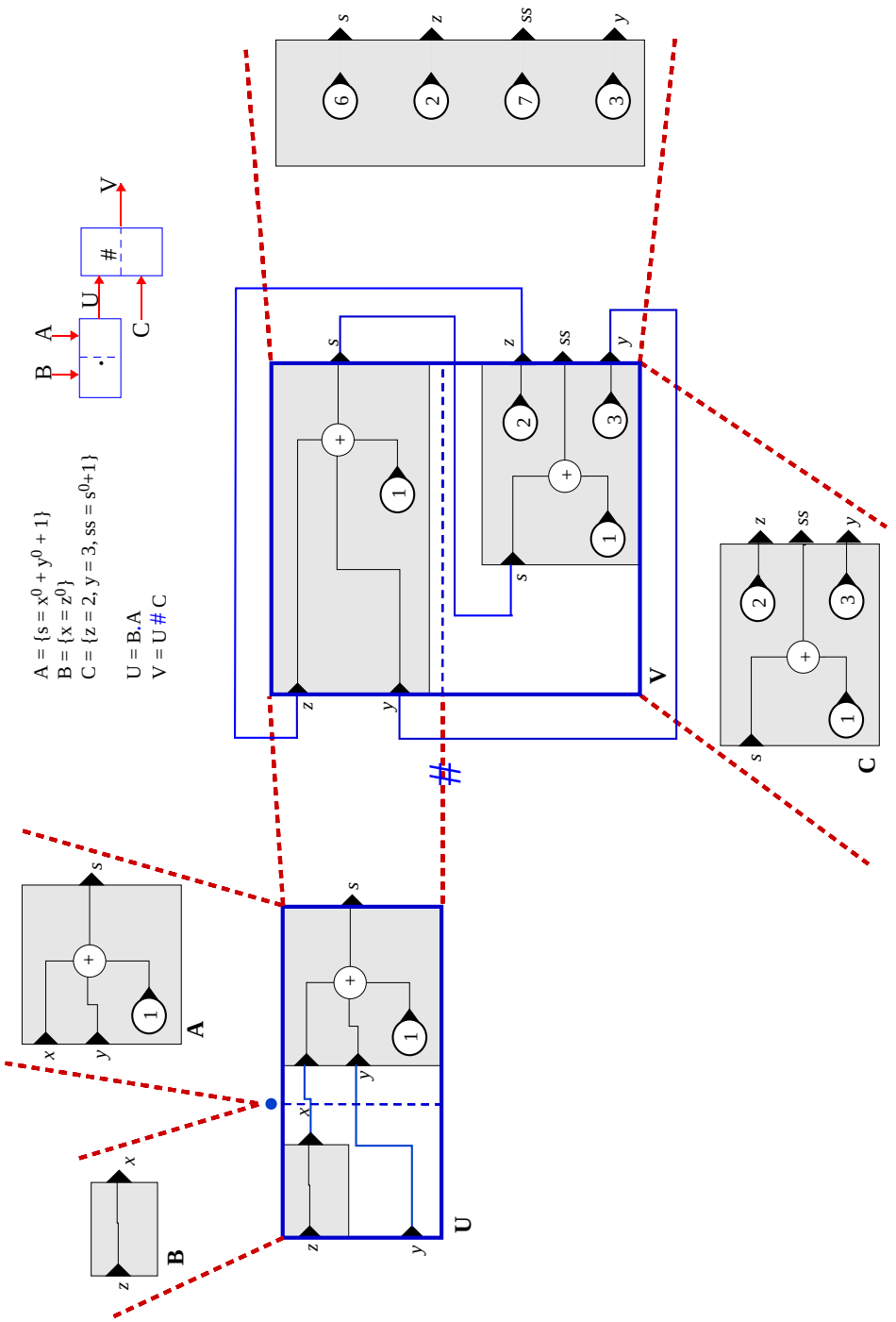


Figure A.12: Example of amalgam computation. Amalgams are higher-order data-flow graphs [Mic96b, MG98, Out98]: the values moving on the edges are systems, that is, data-flow graphs. The edges have been "widened" so that the structure of the manipulated values can bee seen. The object represented here is a graph of graphs. System U represents the evaluation of system A in the context of the definition provided by B. System U and C are then joined and their free references completed to give the resulting system V.

A.2 Illustrations related to the MGS project

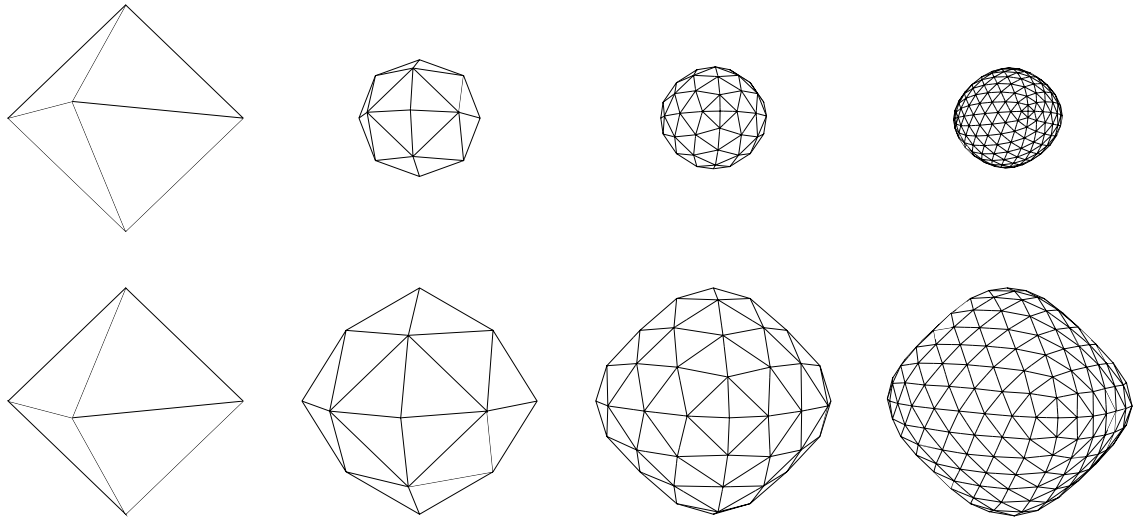


Figure A.13: Loop and Butterfly surface subdivision. Surface subdivision algorithms [SZ98] are algorithms used to refine a mesh. They are used to generate a smooth surface (at the limit) starting from a few points of control (the initial mesh). These algorithms are very intuitively described by local rules. However, their implantation in an imperative settings is diffcult due to sensible management of the indexed points. In [PSSK03] the authors ask if a declarative framework, comparable to the L-systems used for curve subdivision, can be developped for surfaces and higher-dimensionnal objects. MGS gives a positive answer to this question.

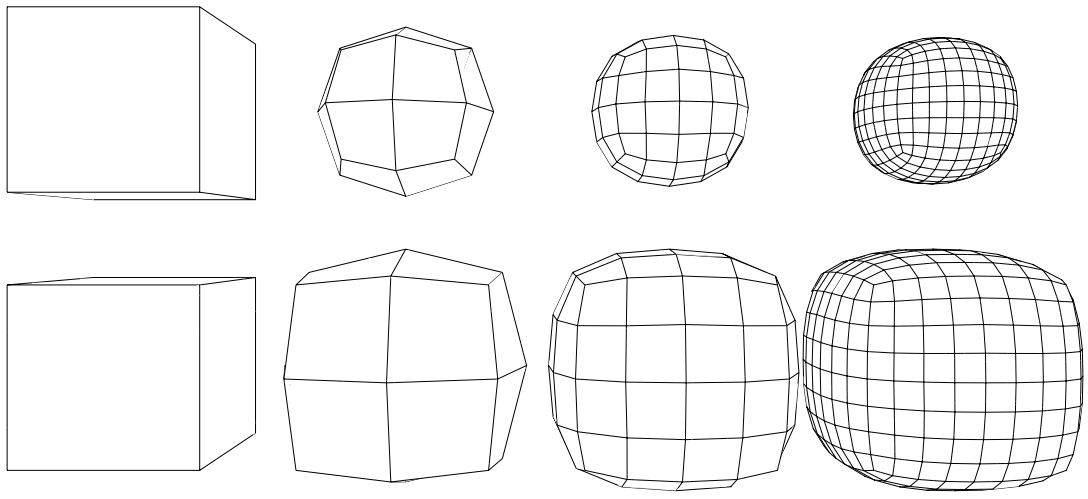


Figure A.14: Catmull-Clark and Kobbelt surface subdivision. There are a lot of surface subdivision algorithms [SZ98]. In this figure, the result of the first iterations of two other algorithms are presented.

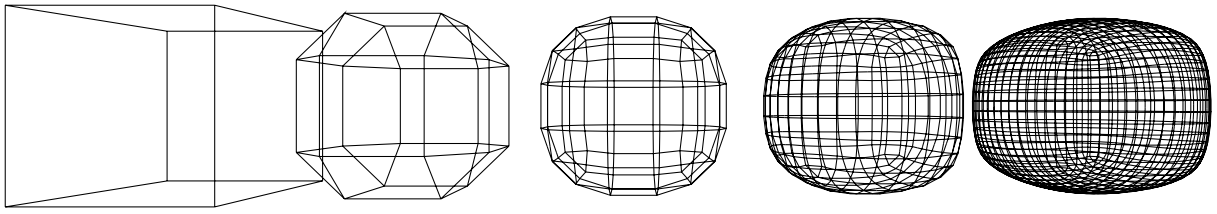


Figure A.15: Doo-Sabin surface subdivision [SZ98]. This surface subdivision can also be seen as the production of tapers [Led02] on the edges of a volume.

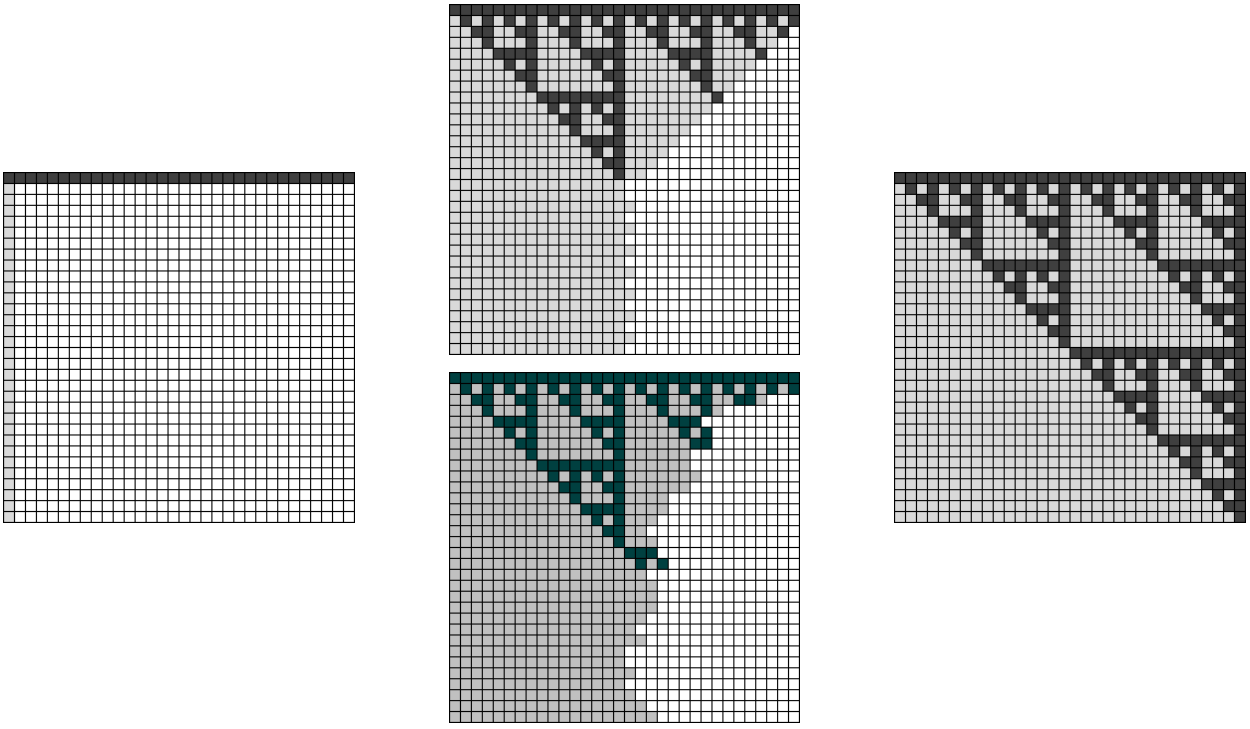


Figure A.16: Growth of the Sierpinski triangle on a GBF. The process mimics the self-assembly of DNA sierpinski triangles [RPW03]. The left figure is the initial state while the figure at the right is the final state. The top figure represents and intermediary step of a computation with a parallel maximal rule application strategy while the figure at the bottom is the result of a computation using a stochastic rule application strategy. On each figure, white boxes correspond to the `<undef>` value while the light (resp. dark) grey boxes are the values `0` (resp. `1`).

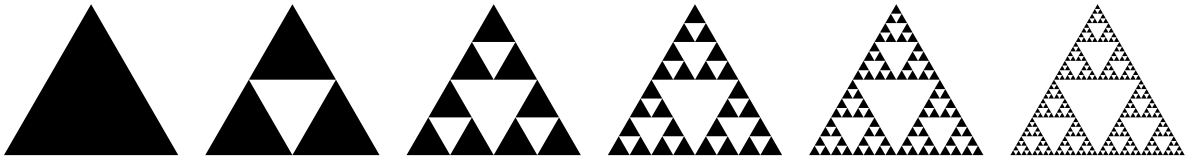


Figure A.17: Construction of the Sierpinski's triangle by a carving process. The iteration of the carving process leads to the final result [GS06].

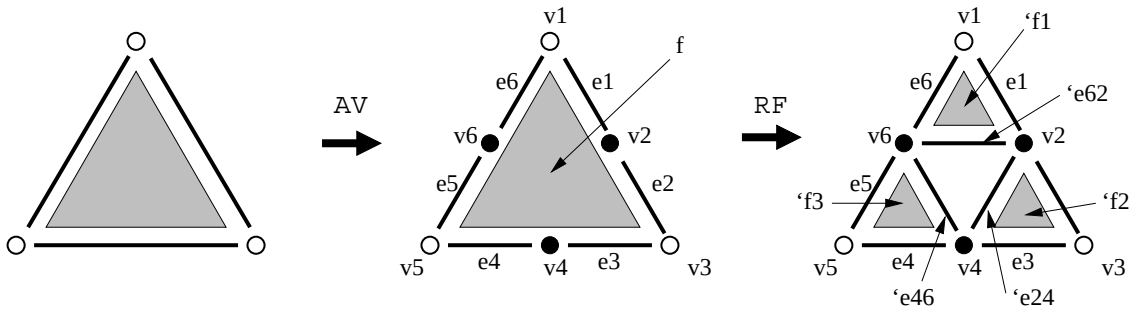


Figure A.18: Carving of a triangle: the first transformation adds a vertex in the middle of each edge; the second refines the hexagon obtained into 3 triangles leaving a triangular hole in the center of the structure [Spi06].

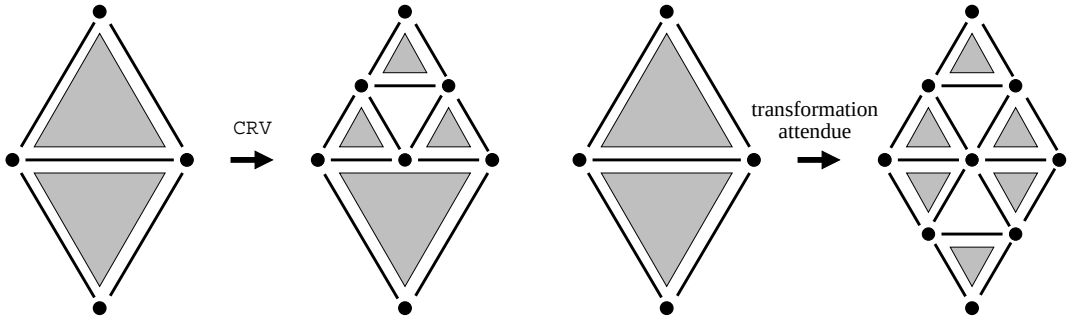


Figure A.19: The left figure sketches the resul of the application of the previous rules. Because occurences of a pattern are disjoint, the result obtained (CRV) is incorrect. The right figure sketches the awaited result. Using a generation number, it is easy to iterate the application CRV to refine only one time each surface triangle [Spi06].

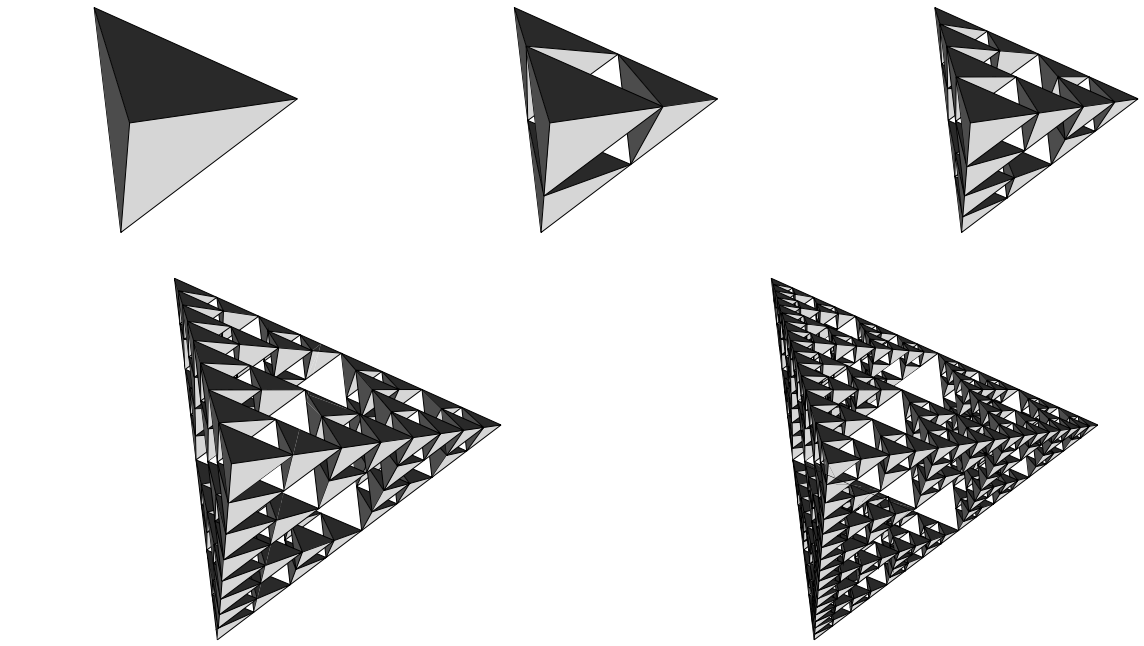


Figure A.20: Construction of the Sierpinski's sponge by carving: initial state and steps 1, 2, 3 and 4.

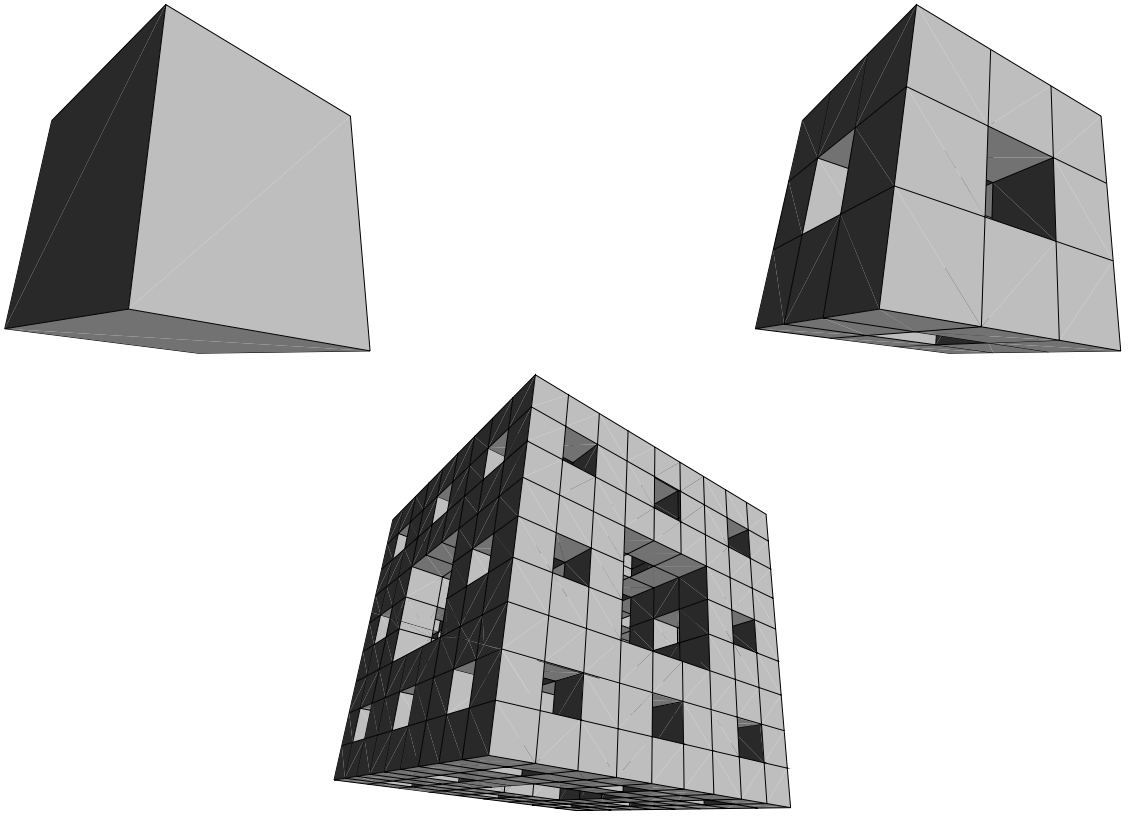


Figure A.21: Construction of the Menger's sponge by carving: initial state and steps 1 and 2.

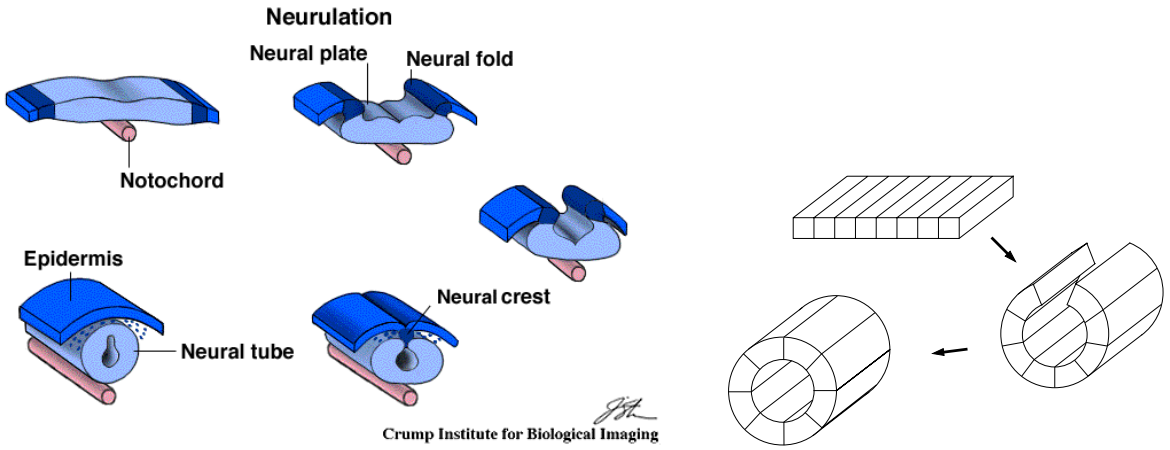


Figure A.22: The left figure describes diagrammatically the neurulation process (from a drawing by Patricia Phelps – `http://www.physci.ucla.edu/research/phelps/index.php` – with permission). The right figure details the three steps of our simplified model [SM06]. Initially, the system is composed of a sheet of cells describing the neural plate. Then it invaginates under the "pressure" of the deformation of the cells. Finally, during the last step, the sheet of cells "closes" to become a true continuous topological cylinder.

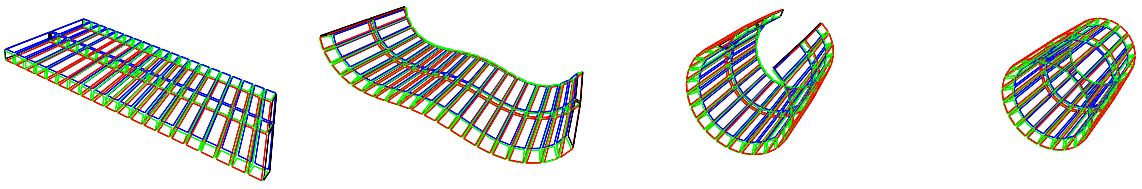


Figure A.23: Simulation of the neurulation process in MGS [SM06]: from the left to the right, a sheet of epithelial cells invaginates until its boundaries are close enough to join and form a tube.

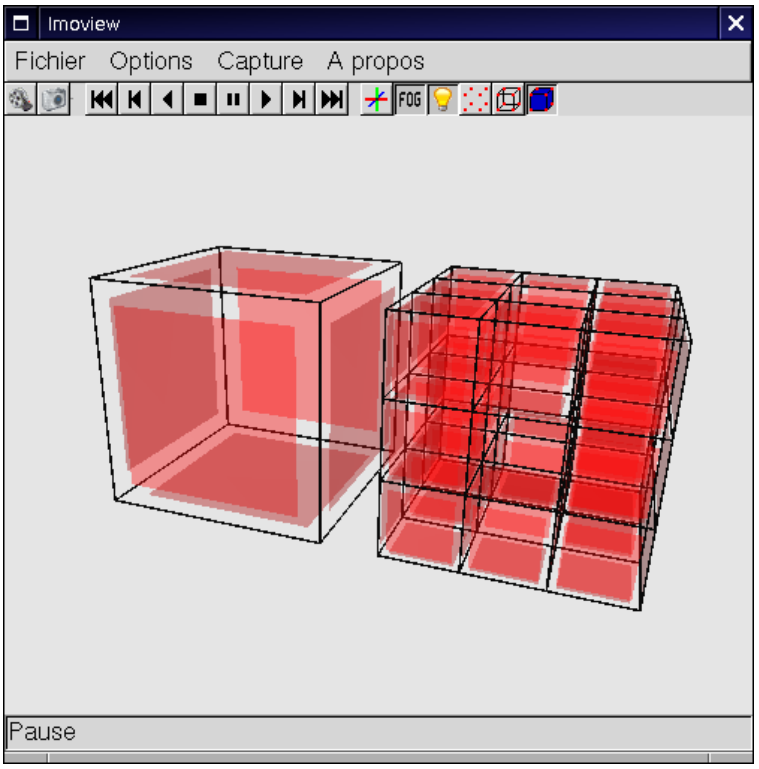


Figure A.24: Screen dump of the Imoview vizualization software (iterations of the construction of the Menger's sponge) [Tho03].

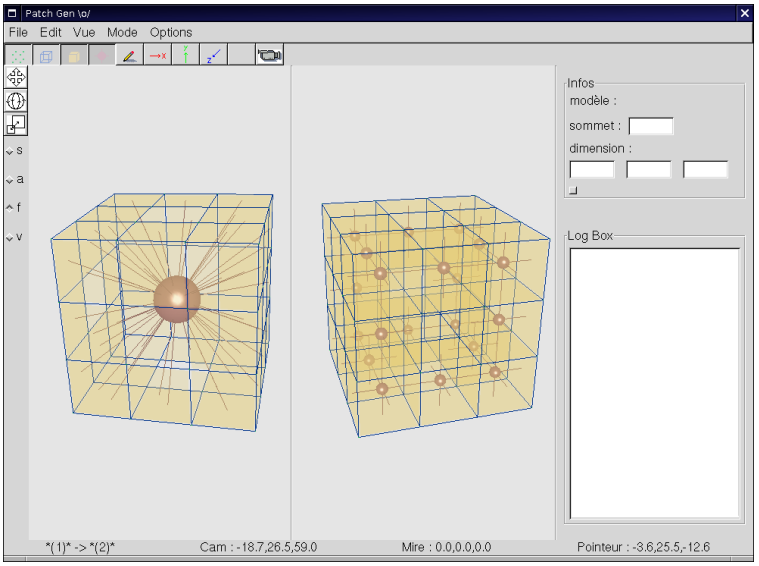


Figure A.25: Screen dump of the PatchGen software [Jul05]. Patchgen is a graphical patch editor for MGS's. The figure illustrates a step in the construction of the rules for carving the Menger's sponge.

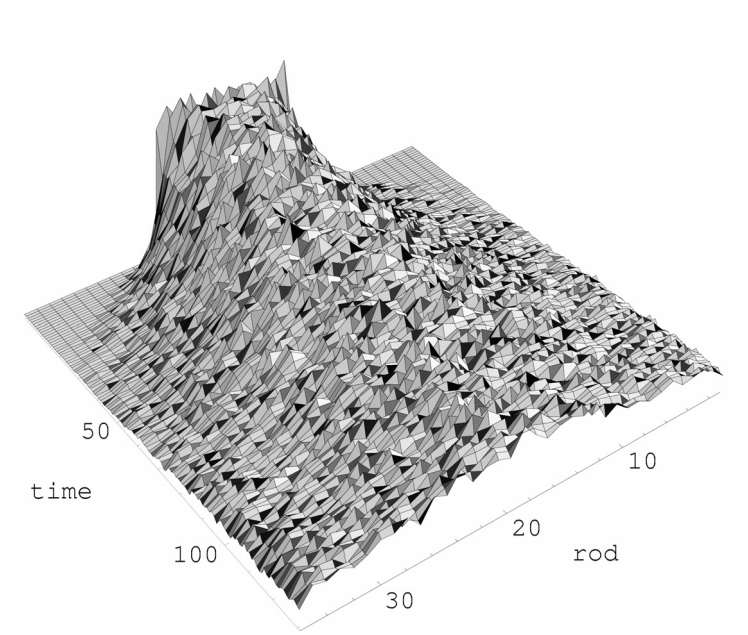


Figure A.26: Heat diffusion on a uniform rod (represented by a multi-set topological collection) with moving quanta of heat [GM04]. Ends are reflexive.

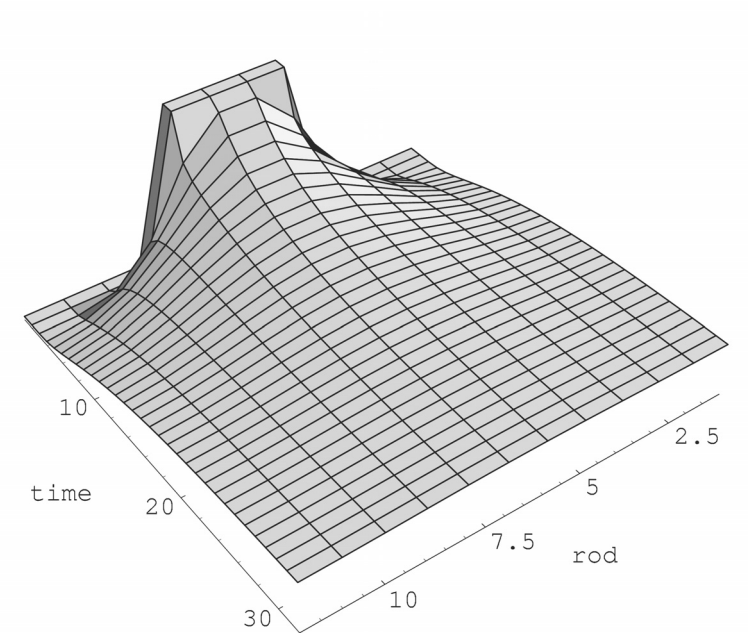


Figure A.27: Result of the simulation of the heat diffusion in a uniform rod by solving numerically the partial differential equation of the diffusion [GM04]. Both ends of the rod are held to a temperature of $0°C$. At initialization, only the middle third is heated. The time axis goes from the back to the front.

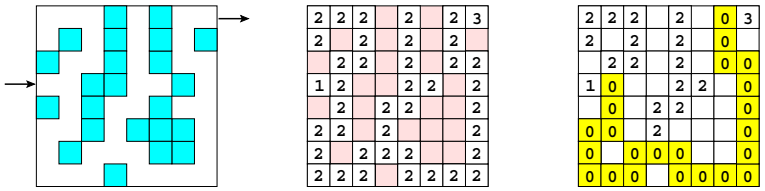


Figure A.28: Description on a NEWS grid (using a GBF topological collection) of a maze: the entrance is coded with the numerical value 1 and the exit with the numerical value 3; possible paths are coded with numerical values 2. A valid path from the entrance to the exit is coded in a one MGS rule. The pattern used `(1, 2*, 3)` describes a succesful path starting at the maze entrance and ending at the exit.

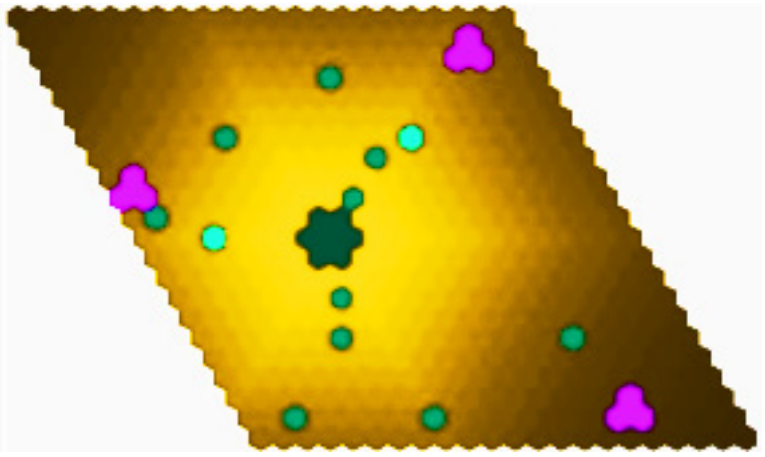


Figure A.29: Ants foraging on an hexagonal lattice (defined by a GBF topological collection) [Coh04a].

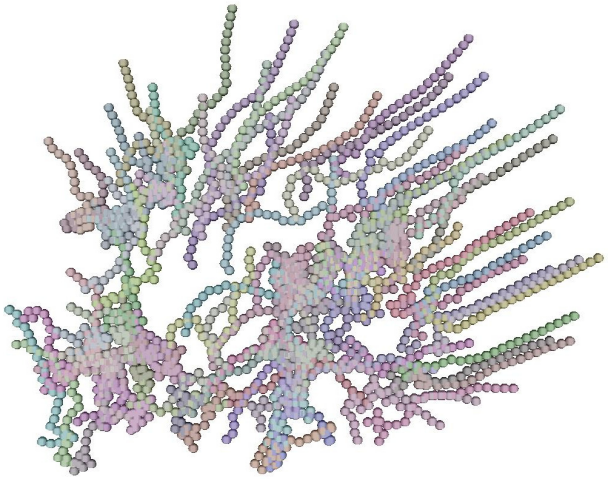


Figure A.30: Movements of a flock of birds represented in a Delaunay topological collection and vizualised using the Imoview software.

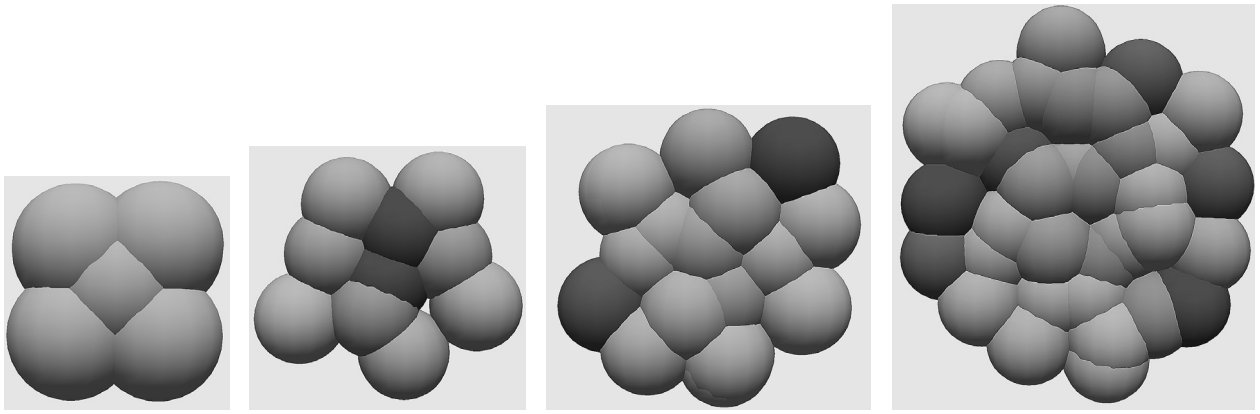


Figure A.31: Four steps in the growth of a sheet of cells. The color of a cell is related to the concentration of the chemicals triggering the cell division (the darkest cells are about to divide). The mechanics of this model is a spring-mass system.

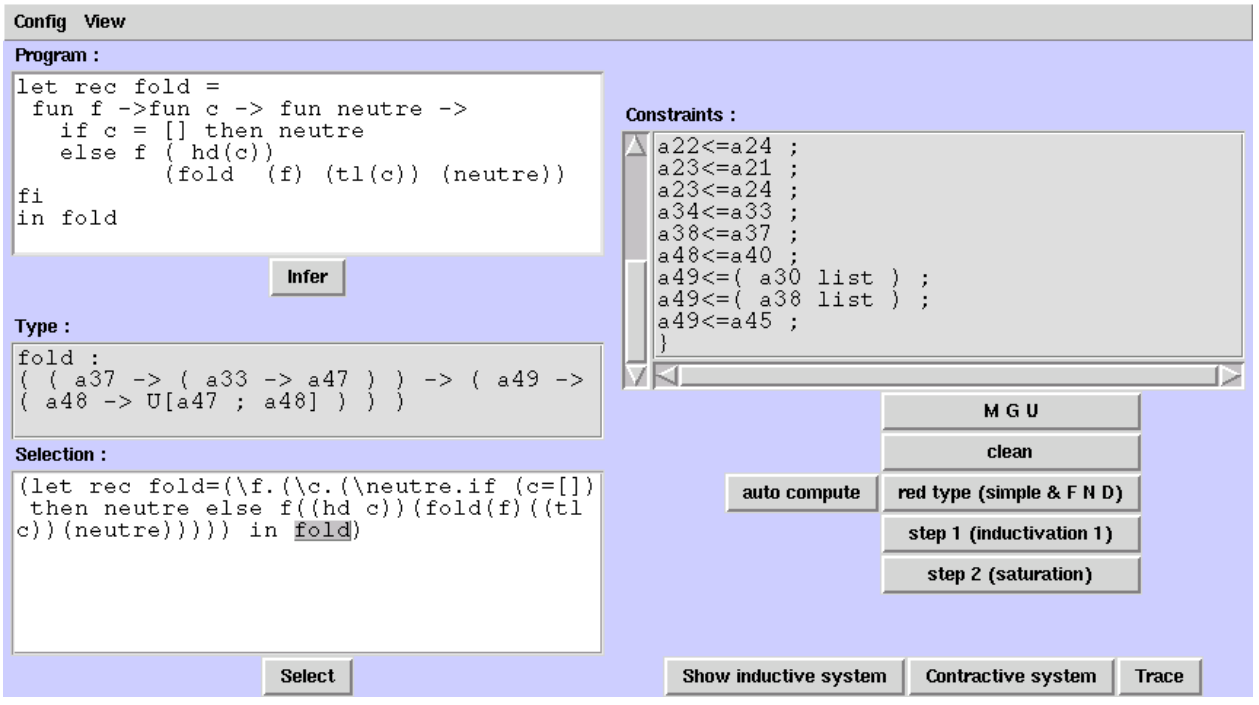


Figure A.32: Screen dump of a type inference tool for a subset of MGS programs [Coh03, Coh04a, Coh04b].

Bibliography

[AAC+00] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight Jr., Radhika Nagpal, Erik Rauch, Gerald J. Sussman, and Ron Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, 2000.

[Ada68] Duane Albert Adams. *A computation model with dataflow sequencing*. PhD thesis, Stanford University, California, 1968.

[Ada01] Andrew Adamatzky. Reaction-diffusion and excitable processors: A sense of the unconventional. *Parallel and Distributed Computing Practices*, 3(2):113–132, 2001.

[Adl94] Len Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–4, November 1994.

[AFJW95] Edward A. Ashcroft, Antony Faustini, Rangaswamy Jagannathan, and William Wadge. *Multidimensional Programming*. Oxford University Press, February 1995. ISBN 0-19-507597-8.

[AG77] Arvind and Kim P. Gostelow. *Some relationships between asynchronous interpreters of a dataflow language*, chapter in: Formal description of programming concepts, E. J. Neuhold editor, pages 95–119. Noth Holland, 1977.

[AIRRH03] Charles Auffray, Sandrine Imbeaud, Magali Roux-Rouquié, and Leroy Hood. From functional genomics to systems biology: concepts and practices. *CR biologies*, 326(10–11):879–892, 2003.

[AL91] Andrea Asperti and Giuseppe Longo. *Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist*. Foundations of Computing Series. The MIT Press, Cambridge, MA, 1991.

[Are06] Aresa. Reddetect - plant for landmine detection. Web, 2006. http://www.aresa.dk/landmine_plant_project_english.html.

[Arn81] André Arnold. Sémantique des processus communicants. *RAIRO*, 15(2):103–140, 1981.

[AW77] Edward A. Ashcroft and William W. Wadge. Lucid, a nonprocedural language with iteration. *Communications of the ACM*, 20(7):519–526, July 1977.

[Bac78] John Backus. Can programming be liberated from the von Neumann style? *Communications of the ACM*, 21(8):613–641, 1978.

[Bac89] Roland Carl Backhouse. *STOP Summer School on constructive algorithmics*, chapter An exploration of the Bird-Merteens formalism. Ameland, September 1989.

[BC90] Gilad Bracha and William Cook. Mixin-based inheritance. *ACM SIGPLAN Notices*, 25(10):303–311, October 1990. *OOPSLA ECOOP '90 Proceedings*, N. Meyrowitz (editor).

[BCD+06] Jaap Boender, Roberto Di Cosmo, Berke Durak, Xavier Leroy, Fabio Mancinelli, Mario Morgado, David Pinheiro, Ralf Treinen, Paulo Trezentos, and Jérôme Vouillon. News from the EDOS project: improving the maintenance of free software distributions. In Olivier Berger, editor, *Proceedings of the International Workshop on Free Software (IWFS'06)*, pages 199–207, Porto Allegre, Brazil, April 2006.

[BdR05] Pierre Barbier de Reuille. *Vers un modèle dynamique du méristème apical caulinaire d'Arabidopsis thaliana*. PhD thesis, Université Montpellier II, 2005.

[BdRBCL+06] Pierre Barbier de Reuille, Isabelle Bohn-Courseau, Karin Ljung, Halima Morin, Nicola Carraro, Christophe Godin, and Jan Traas. Computer simulations reveal properties of the cell-cell signaling network at the shoot apex in Arabidopsis. *PNAS*, 103(5):1627–1632, 2006.

[Ber83] Dimitri P. Bertsekas. Distributed asynchronous computation of fixed points. *Math. Programming*, 27:107–120, 1983.

[BFGM05] Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel, editors. *Unconventional Programming Paradigms*, Revised Selected and Invited Papers of the International Workshop UPP 2004, Le Mont-Saint-Michel, France, 2005. Springer-Verlag, LNCS, Vol. 3566.

[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.

[BFR06a] Jean-Pierre Banâtre, Pascal Fradet, and Yann Radenac. A generalized higher-order chemical computation model. *Electr. Notes Theor. Comput. Sci*, 135(3):3–13, 2006.

[BFR06b] Jean-Pierre Banâtre, Pascal Fradet, and Yann Radenac. Towards chemical coordination for grids. In Hisham Haddad, editor, *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), Dijon, France, April 23-27, 2006*, pages 445–446. ACM, 2006.

[BHS04] Asa Ben-Hur and Hava T Siegelmann. Computation in gene networks. *Chaos*, 14(1):145–151, Mar 2004.

[Bir87] Richard S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design, NATO ASI Series, vol. F36*, pages 217–245. sv, 1987.

[BL06] Francis Bailly and Giuseppe Longo. *Mathématiques et sciences de la nature*. Hermann, 2006.

[BLL97] François Bergeron, Gilbert Labelle, and Pierre Leroux. *Combinatorial species and tree-like structures*, volume 67 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1997. isbn 0-521-57323-8.

[BLM86] Jean-Pierre Banâtre and Daniel Le Metayer. A new computational model and its discipline of programming. Technical Report RR-0566, Inria, 1986.

[BLM96] Jean-Pierre Banâtre and Daniel Le Metayer. *Gamma and the Chemical Reaction Model: Ten Years After*, pages 3–41. Imperial College Press, 1996.

[Boa83] Mary L. Boas. *Mathematical Methods in the Physical Sciences*. John Wiley and Sons, 2nd edition, 1983.

[Bou04] Damien Boussié. Simulation en `MGS` du déplacement du spermatozoïde du nématode *Ascaris Suum*. Master's thesis, DEA AMIB Université d'Évry, 2004.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. 7th Int. World Wide Web Conf.*, 14–18 April 1998.

[BRF04] Jean-Pierre Banâtre, Yann Radenac, and Pascal Fradet. Chemical specification of autonomic systems. In *Intelligent and Adaptive Systems and Software Engineering (IASSE)*, pages 72–79. ISCA, 2004.

[Cam74] Donald Thomas Campbell. *Studies in the Philosophy of Biology: Reductionism and Related Problems*, chapter "Downward causation" in hierarchically organized biological systems, pages 179–186. Macmillan Press, New York, 1974.

[Car03] Robert Carlson. The pace and proliferation of biological technologies. *Biosecur Bioterror*, 1(3):203–214, 2003.

[Car04] Luca Cardelli. Brane calculi. In Vincent Danos and Vincent Schächter, editors, *Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278, Berlin, 2004. Springer.

[CEB+00] Joanne Chory, Joseph R. Ecker, Steve Briggs, Michel Caboche, Gloria M. Coruzzi, Doug Cook, Jeffrey Dangl, Sarah Grant, Mary Lou Guerinot, Steven Henikoff, Rob Martienssen, Kiyotaka Okada, Natasha V. Raikhel, Chris R. Somerville, and Detlef Weigel. National Science Foundation-Sponsored Workshop Report: "The 2010 Project" Functional Genomics and the Virtual Plant. A Blueprint for Understanding How Plants Are Built and How to Improve Them. *Plant Physiol.*, 123(2):423–426, 2000.

[CES71] Edward. G. Coffman, Michael J. Elphick, and Arie Shoshani. System deadlocks. *Computing Surveys*, 3(2):67–78, 1971.

[Cha90] K. Mani Chandy. Reasoning about continuous systems. *Science of Computer Programming*, 14(2–3):117–132, October 1990.

[Che86] Marina Chen. A parallel language and its compilation to multiprocessor machines or VLSI. In *Principles of Programming Languages*, pages 131–139, Florida, 1986.

[CiCL91] Marina Chen, Young il Choo, and Jingke Li. Crystal: Theory and Pragmatics of Generating Efficient Parallel Code. In Boleslaw K. Szymanski, editor, *Parallel Functional Languages and Compilers*, Frontier Series, chapter 7, pages 255–308. ACM Press, New York, 1991.

[Coh03] Julien Cohen. Typing rule-based transformations over topological collections. In Jean-Louis Giavitto and Pierre-Etienne Moreau, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.

[Coh04a] Julien Cohen. *Intégration des collections topologiques et des transformations dans un langage de programmation fonctionnel*. PhD thesis, Université d'Évry, December 2004. http://www.ibisc.univ-evry.fr/~jcohen/THESE/these.officiel.pdf.

[Coh04b] Julien Cohen. *Intégration des collections topologiques et des transformations dans un langage fonctionnel*. PhD thesis, Université d'Évry, 2004.

[Coh04c] Julien Cohen. Typage fort et typage souple des collections topologiques et des transformations. In Valérie Ménissier-Morain, editor, *Journées Francophones des Langages Applicatifs (JFLA 2004)*, pages 37–54. INRIA, 2004.

[Com04] ACM International Conférence on Computing Frontiers, from 2004. Proceedings published from 2004 by the ACM Press. http://www.computingfrontiers.org/.

[Con63] Melvin E. Conway. Design of a separable transition-diagram compiler. *Communication of the ACM*, 6(7):396–408, 1963.

[Coo99] Daniel Coore. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. PhD thesis, MIT, 1999.

[Del02] Emmanuel Delsinne. Structures de données indexées par un groupe, isomorphismes de gbf abéliens et extensions aux structures automatiques. Master's thesis, E.N.S. Cachan et Université de Rennes-I, 2002.

[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 sptember 2006.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter Rewrite systems, pages 244–320. Elsevier Science, 1990.

[DKT06] Mathieu Desbrun, Eva Kanso, and Yiying Tong. Discrete differential forms for computational modeling. In *Discrete differential geometry: an applied introduction*, pages 39–54. Schröder, P, ACM Press, 2006. SIGGRAPH'06 course notes.

[DM79] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the Association for Computing Machinery*, 22:465–476, 1979.

[DNA95] International Meeting on DNA Computing, from 1995. Proceedings published from 1995 to 2000 as AMS DIMACS volume and then published as LNCS volume. http://hagi.is.s.u-tokyo.ac.jp/dna/.

[Dre81] Kim E. Drexler. Molecular engineering: An approach to the development of general capabilities for molecular manipulation. *Proc. Nat. Acad. Sci. USA*, 78(9):5275–5278, 1981.

[DS96] Dominic Duggan and Constantinos Sourelis. Mixin modules. In *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming*, pages 262–273, Philadelphia, Pennsylvania, 24–26 May 1996.

[Duc99] Bertrand Ducourthial. *Les réseaux associatifs – un modèle de programmation à parallélisme de données, pour algorithmes et données irréguliers, à primitives de calcul asynchrones*. PhD thesis, Université Paris-Sud XI, 1999. http://www.hds.utc.fr/~ducourth/BIBLIO/these.ps.gz.

[DZB01] Peter Dittrich, Jens Ziegler, and Wolfgang Banzhaf. Artificial chemistries-A review. *Artificial Life*, 7(3):225–275, 2001.

[EKL+02] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, and Carolyn L. Talcott. Pathway logic: Executable models of biological networks. *Electr. Notes Theor. Comput. Sci*, 71, 2002.

[End05] Drew Endy. Foundations for engineering biology. *Nature*, 438(7067):449–453, Nov 2005.

[Eng90] Erwin Engeler. Combinatory differential fields. *Theoretical Computer Science*, 72(2-3):119–131, 1990.

[ES79] Manfred Eigen and Peter Schuster. *The Hypercycle: A Principle of Natural Self-Organization.* Springer, 1979.

[Far03] Emmanuel Farge. Mechanical induction of Twist in the Drosophila foregut/stomodeal primordium. *Current Biology*, 13(16):1365–1377, August 2003.

[Fau82] Antony A. Faustini. An operational semantics of pure dataflow. In M. Nielsen and E. M. Schmidt, editors, *Automata, languages and programing: ninth colloquium*, volume 120 of *Lecture Notes in Computer Science*, pages 212–224. Springer Verlag, 1982. equivalence sem. op et denotationelle.

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

[Fey60] Richard P. Feynman. There's plenty of room at the bottom – an invitation to enter a new field of physics. *Engineering and Science*, February 1960.

[FH04] Paul François and Vincent Hakim. Design of genetic networks with specified functions by evolution in silico. *Proc Natl Acad Sci U S A*, 101(2):580–585, Jan 2004.

[Fis01] Gerhard Fischer. The software technology of the 21st century: From software reuse to collaborative software design. In *International Symposium on Future Software Technology*, pages 1–8, ZhengZhou, China, November 2001. Software Engineers Association, Japan. http://l3d.cs.colorado.edu/ gerhard/papers/isfst2001.pdf.

[FMP00] Michael Fisher, Grant Malcolm, and Raymond Paton. Spatio-logical processes in intracellular signalling. *BioSystems*, 55:83–92, 2000.

[Fon06] Walter Fontana. Pulling strings. *Science*, 314(5805):1552–1553, Dec 2006.

[FP07] Fabrice Le Fessant and Simon Patarin. MLdonkey, a multi-network peer-to-peer file-sharing program, March 14 2007.

[Gab02] Richard P. Gabriel. Objects have failed – notes for a debate, 18 November 2002. Debate at OOPSLA 2002. http://www.dreamsongs.org/NewFiles/ObjectsHaveFailed.pdf.

[Gar70] Martin Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.

[Gau05] Fabien Gaubert. Simulation stochastique et modélisation de chimie artificielle dans le langage `MGS`. Master's thesis, DEA AMIB Université d'Évry, 2005.

[GDVM97] Jean-Louis Giavitto, Dominique De Vito, and Olivier Michel. Semantics and compilation of recursive sequential streams in 81/2. In H. Glaser and H. Kuchen, editors, *Ninth International Symposium on Programming Languages, Implementations, Logics, and Programs (PLILP'97)*, volume 1292 of *Lecture Notes in Computer Science*, pages 207–223, Southampton, 3–5 September 1997. Springer Verlag. http://www.ibisc.fr/~michel/PUBLIS/1997/plilp97.pdf.

[GGMP02a] Jean-Louis Giavitto, Christophe Godin, Olivier Michel, and Przemyslaw Prusinkiewicz. Computational models for integrative and developmental biology. Technical Report 72-2002, LaMI – Université d'Évry Val d'Essonne, March 2002. http://www.ibisc.fr/~michel/PUBLIS/2002/rt72.pdf.

[GGMP02b] Jean-Louis Giavitto, Christophe Godin, Olivier Michel, and Przemyslaw Prusinkiewicz. *Modelling and Simulation of biological processes in the context of genomics*, chapter "Computational Models for Integrative and Developmental Biology". Hermes, July 2002. http://www.ibisc.fr/~michel/PUBLIS/2002/autran02.ps.gz.

[Ghe05] Marian Gheorghe, editor. *Molecular Computational Models: Unconventional Approaches*. IGI Publishing, 2005. ISBN: 1-59140-333-2.

[Gia92] Jean-Louis Giavitto. Typing geometries of homogeneous collection. In *2nd Int. workshop on array manipulation, (ATABLE)*, Montréal, 1992.

[Gia99] Jean-Louis Giavitto. *Scientific Report for the habilitation*. PhD thesis, LRI, Université de Paris-Sud, centre d'Orsay, September 1999. Research Report 1226.

[Gia00a] Jean-Louis Giavitto. A fixed point approach to the resolution of array equations. In I. Guessarian, editor, *Fixed Points in Computer Science (FICS'2000)*, Paris, July 2000. satellite workshop of LC'2000.

[Gia00b] Jean-Louis Giavitto. A framework for the recursive definition of data structures. In *Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-00)*, pages 45–55. ACM Press, September 20–23 2000.

[Gia03] Jean-Louis 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.

[Gia04] Jean-Louis Giavitto. *Modelling and simulation of biological processes in the context of genomics*, chapter On "The biochemical abstract machine BIOCHAM", pages 175–176. Genopole & Platypus Press, ISBN 2-84704-0374, 2004. Comments on a presentation of François Fages to the thematic school held in Évry (France) in April 2004.

[Gib94] Jeremy Gibbons. An introduction to the Bird-Meertens formalism. In *New Zealand Formal Program Development Colloquium Seminar*, Hamilton, 1994.

[Gib04] Wayt W. Gibbs. Synthetic life. *Scientific American*, 290(5):74–81, May 2004.

[Gil77] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[GJ92] Eric Goubault and Thomas P. Jensen. Homology of higher-dimensional automata. In *Proc. of CONCUR'92*, Stonybrook, August 1992. Springer Verlag.

[GM01a] Jean-Louis Giavitto and Olivier Michel. Declarative definition of group indexed data structures and approximation of their domains. In *PPDP '01: Proceedings of the 3rd ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 150–161, New York, NY, USA, 2001. ACM Press. http://www.ibisc.fr/~michel/PUBLIS/2001/ppdp01.pdf.

[GM01b] Jean-Louis Giavitto and Olivier Michel. MGS: a programming language for the transformations of topological collections. Technical Report 61-2001, LaMI – Université d'Évry Val d'Essonne, May 2001. http://www.ibisc.fr/~michel/PUBLIS/2001/lami-RR61--mgs.pdf.

[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. http://www.ibisc.fr/~michel/PUBLIS/2001/entcs01.pdf.

[GM01d] Jean-Louis Giavitto and Olivier Michel. MGS: Implementing a unified view on four biologically inspired computational models. In *Pre-proceedings of WMC-CdeA 2001 (Workshop on Membrane Computing, Curtea de Arges)*. Research Report 17/01 of the Universitat Rivira I Virgili, Tarragona, Spain, August 2001. http://www.ibisc.fr/~michel/PUBLIS/2001/wmc01.ps.gz.

[GM02a] Jean-Louis Giavitto and Olivier 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. http://www.ibisc.fr/~michel/PUBLIS/2002/umc02.pdf.

[GM02b] Jean-Louis Giavitto and Olivier Michel. Pattern-matching and rewriting rules for group indexed data structures. In *ACM Sigplan Workshop RULE'02*, pages 55–66, Pittsburgh, October 2002. ACM. http://www.ibisc.fr/~michel/PUBLIS/2002/rule02.pdf.

[GM02c] Jean-Louis Giavitto and Olivier Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49:107–129, 2002. http://www.ibisc.fr/~michel/PUBLIS/2002/FI.pdf.

[GM03] Jean-Louis Giavitto and Olivier Michel. Modeling the topological organization of cellular processes. *BioSystems*, (70):149–163, 2003. http://www.ibisc.fr/~michel/PUBLIS/2003/biosystem02.pdf.

[GM04] Jean-Louis Giavitto and Olivier Michel. *Molecular Computational Models: Unconventional Approaches*, chapter Modeling Developmental Processes in MGS, pages 150–189. Idea Group Publishing, 2004. http://www.ibisc.fr/~michel/PUBLIS/2004/MCM.pdf.

[GMC02a] Jean-Louis Giavitto, Olivier Michel, and Julien Cohen. Pattern-matching and rewriting rules for group indexed data structures. *ACM SIGPLAN Notices*, 37(12):76–87, December 2002. http://www.ibisc.fr/~michel/PUBLIS/2003/sigplan03.pdf.

[GMC02b] Jean-Louis Giavitto, Olivier Michel, and Julien Cohen. Pattern-matching and rewriting rules for group indexed data structures. Technical Report 76-2002, LaMI – Université d'Évry Val d'Essonne, June 2002. http://www.ibisc.fr/~michel/PUBLIS/2002/rt76.pdf.

[GMCS05] Jean-Louis Giavitto, Olivier Michel, Julien Cohen, and Antoine Spicher. Computation in space and space in computation. In Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel, editors, *Unconventional Programming Paradigms (UPP'04)*, volume 3566 of *LNCS*, pages 137–152. ERCIM–NSF, Springer Verlag, 2005. http://www.ibisc.fr/~michel/PUBLIS/2005/upp04-version-finale-lncs.pdf.

[GMD03] Jean-Louis Giavitto, Olivier Michel, and Franck 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. http://www.ibisc.fr/~michel/PUBLIS/2003/biosystem02bis.pdf.

[GMM04] Jean-Louis Giavitto, Grant Malcolm, and Olivier Michel. Rewriting systems and the modelling of biological systems. *Comparative and Functional Genomics*, 5:95–99, February 2004. http://www.ibisc.fr/~michel/PUBLIS/2004/CFG04.pdf.

[GMQS89] Pierrick Gachet, Christophe Mauras, Patrice Quinton, and Yannick Saouter. Alpha du Centaur: A prototype environment for the design of parallel regular algorithms. In *Conference Proceedings, 1989 International Conference on Supercomputing*, pages 235–243, Crete, Greece, June 5–9, 1989. ACM SIGARCH.

[GMS95] Jean-Louis Giavitto, Olivier 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 *Lecture Notes in Computer Science*, pages 209–215, Beaune (France), 2–4 October 1995. Springer Verlag. http://www.ibisc.fr/~michel/PUBLIS/1995/psls.pdf.

[Gou00] Eric Goubault. Geometry and concurrency: A user's guide. *Mathematical Structures in Computer Science*, 10:411–425, 2000.

[GS90] Carl A. Gunter and Dana S. Scott. *Handbook of Theoretical Computer Science*, volume 2, chapter Semantic Domains, pages 633–674. Elsevier Science, 1990.

[GS00] Scott F. Gilbert and Sahotra Sarkar. Embracing complexity: Organicism for the 21st century. *Developmental Dynamics*, 219:1–9, 2000.

[GS06a] Jean-Louis Giavitto and Antoine Spicher. *Morphogénèse*, chapter Morphogénèse informatique, pages 178–198. Belin, 2006.

[GS06b] Jean-Louis Giavitto and Antoine Spicher. *Systems Self-Assembly: multidisciplinary snapshots*, chapter Simulation of self-assembly processes using abstract reduction systems. Elsevier, 2006.

[GSM07] Jean-Louis Giavitto, Antoine Spicher, and Olivier Michel. Topological rewriting and the geometrization of programming. Technical Report IBISC-XX-2007, IBISC, September 2007.

[Hea87] Tom Head. Formal language theory and dna: an analysis of the generative capacity of specific recombinant behaviors. *Bull. Math. Biology*, 49(6):737–759, 1987.

[Hen94] Michael Henle. *A combinatorial introduction to topology*. Dover publications, New-York, 1994.

[HGH93] John Y. Hung, Weibing Gao, and James C. Hung. Variable structure control: A survey. *IEEE Transactions on Industrial Electronics*, 40(1):2–22, 1993.

[HH98] Tad Hogg and Bernardo A. Huberman. Controlling smart matter. *Smart Materials and Structures*, 7:R1, 1998.

[Hil85] Daniel W. Hillis. *The Connection Machine*. MIT Press, Cambridge, Mass., 1 edition, 1985.

[Hir03] Anil N. Hirani. *Discrete exterior calculus*. PhD thesis, California Institute of Technology, 2003.

[HL93] Per Hammarlund and Björn Lisper. On the relation between functional and data parallel programming languages. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, pages 210–222, New York, NY, USA, June 1993. ACM Press.

[HO96] Masatomo Hashimoto and Atsushi Ohori. A typed context calculus. Technical Report RIMS-1098, Research Institute for Mathematical Sciences, Kyoto University, August 1996.

[Hol73] John H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[Hol75] John H. Holland. *Adpatation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[Hor01] Paul Horn. Autonomic computing: IBM's perspective on the state of information technology. Technical report, IBM Research, October 2001. http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[HRW86] J.L. Harper, B.R. Rosen, and J. White. *The Growth and Form of Modular Organism*. London: The Royal Society, 1986.

[Hug89] John Hughes. Why functional programming matters. *The Computer Journal*, 32(2):98–107, April 1989.

[IJU05] Home page of the "International Journal of Unconventional Computing". http://www.oldcitypublishing.com/IJUC/IJUC.html, from 2005.

[IPC95] IPCAT - Information Processing in Cells and Tissues, from 1995. Proceedings published by World Scientific and as special issues of the Biosystems journal.

[Itk76] Yevgeny Itkis. *Control Systems of Variable Structure*. Wiley, 1976.

[Jam93] Max Jammer. *Concepts of space – the history of theories of space in physics*. Dover, 1993. third enlarged edition (first edition 1954).

[JHS+06] Henrik Jönsson, Marcus G. Heisler, Bruce E. Shapiro, Elliot M. Meyerowitz, and Eric Mjolsness. An auxin-driven polarized transport model for phyllotaxis. *Proceedings of the National Academy of Sciences*, 103(5):1633–1638, 2006.

[Jul05] Yann Jullian. Conception et développement d'un éditeur graphique de filtre pour MGS. Master's thesis, École Centrale Paris, 2005.

[Kah74] Gilles Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, August 1974. North Holland, Amsterdam.

[Kal07] Christof Kaleta. Outils de visualisation pour la simulation de systèmes dynamiques à structure dynamique. Master's thesis, Master Informatique, Universität Jena & Université d'Évry, 2007.

[Kau95] Stuart Kaufman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford: Oxford University Press, 1995.

[KHK+03] N. Kam, D. Harel, H. Kugler, R. Marelly, A. Pnueli, J. Hubbard, and M. Stern. Formal modeling of *C. Elegans* development: A scenario-based approach. In *First International Workshop on Computational Methods in Systems Biology*, volume 2602 of *LNCS*, pages 4–20, Rovereto, Italy, February 2003. Springer.

[KHL98] Hiroaki Kitano, Shugo Hamahashi, and Sean Luke. The perfect c. elegans project: An initial report. *Artificial Life*, 4(2):141–156, 1998.

[KKA+04] Hideki Kobayashi, Mads Kaern, Michihiro Araki, Kristy Chung, Timothy S Gardner, Charles R Cantor, and James J Collins. Programmable cells: interfacing natural and engineered gene networks. *Proc Natl Acad Sci U S A*, 101(22):8414–8419, Jun 2004.

[KM66] Richard M. Karp and Raymond E. Miller. Properties of a model of parallel computations : determinacy, termination and queuing. *SIAM Journal of Applied Mathematics*, 14:1390–1411, 1966.

[KMW67] Richard M. Karp, Raymond E. Miller, and Shmuel Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563–590, July 1967.

[KRC+07] Tom Knight, Randall Rettberg, Leon Chan, Drew Endy, Reshma Shetty, and Austin Che. Idempotent vector design for the standard assembly of biobricks. Technical report, MIT department of biology, 2007. http://web.mit.edu/synbio/release/docs/biobricks.pdf.

[Lan66] Peter J. Landin. The next 700 programming languages. *Communications of the ACM*, 9(3):157–166, March 1966.

[Lar02] Valérie Larue. Structures de données indexées par un groupe : représentation graphique et extension au cas non abélien. Master's thesis, DEA Informatique d'Évry, 2002.

[Led02] Franck Ledoux. *Étude et spécifications formelles de l'arrondi d'objets géométriques*. PhD thesis, Université d'Évry, 2002.

[Leo] Mark Leone. Programming language research. http://www.cs.cmu.edu/~mleone/language-research.html.

[LF93] Shinn-Der Lee and Daniel P. Friedman. Quasi-static scoping: Sharing variable bindings across multiple lexical scopes. In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 479–492, Charleston, South Carolina, January 1993.

[LF96] Shinn-Der Lee and Daniel P. Friedman. Enriching the lambda calculus with contexts: toward a theory of incremental program construction. In *International Conference on Functional Programming*. ACM, May 1996.

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

[Lis96] Björn Lisper. Data parallelism and functional programming. In *Proc. ParaDigme Spring School on Data Parallelism*. Springer-Verlag, March 1996. Les Ménuires, France.

[Mac02] Bruce J. Maclennan. Replication, sharing, deletion, lists, and numerals: Progress on universally programmable intelligent matter, November 23 2002.

[Mac03] MIT Project Mac. The Amorphous Computing Home page, 2003. http://www.swiss.ai.mit.edu/projects/amorphous.

[Mae91] Patti Maes. A bottom-up mechanism for behavior selection in an artificial creature. In Bradford Book, editor, *Proceedings of the first international conference on simulation of adaptative behavior (SAB)*. MIT Press, 1991.

[Man01] Vincenzo Manca. Logical string rewriting. *Theoretical Computer Science*, 264(1):25–51, 2001.

[Man04] Nicolas Mann. Hyperstructures et modélisation de chimie artificielle dans le langage `MGS`. Master's thesis, DEA AMIB Université d'Évry, 2004.

[MBFG06] Olivier Michel, Jean-Pierre Banâtre, Pascal Fradet, and Jean-Louis Giavitto. Challenging questions for the rationales of non-classical programming languages. *International Journal of Unconventional Computing*, 2006. http://www.ibisc.fr/~michel/PUBLIS/2006/ijuc.pdf.

[Mei82] Hans Meinhardt. *Models of Biological Pattern Formation*. Academic Press, New York, 1982.

[MFP91] Erik Meijer, Marteen M. Fokkinga, and Ross 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.

[MG98] Olivier Michel and Jean-Louis Giavitto. Amalgams: Names and name capture in a declarative framework. Technical Report 32, LaMI – Université d'Évry Val d'Essonne, January 1998. http://www.ibisc.fr/~michel/PUBLIS/1996/rapport-amalgame.ps.gz.

[MG07] Olivier Michel and Jean-Louis Giavitto. Incremental extension of a domain specific language interpreter. In *19th International Symposium on Implementation and Application of Functional Languages*, Freiburg, Germany, 27–29 September 2007. http://www.ibisc.fr/~michel/PUBLIS/2007/ifl07.pdf.

[Mic95] Olivier Michel. Design and implementation of 81/2, a declarative data-parallel language. Technical Report 1012, Laboratoire de Recherche en Informatique, UMR 8623 du CNRS, December 1995. http://www.ibisc.fr/~michel/PUBLIS/1995/rt-1012.ps.gz.

[Mic96a] Olivier Michel. Design and implementation of 81/2, a declarative data-parallel language. *Computer Languages*, 22(2/3):165–179, 1996. http://www.ibisc.fr/~michel/PUBLIS/1996/ComputerLanguages.pdf.

[Mic96b] Olivier Michel. Introducing dynamicity in the data-parallel language 81/2. In Luc Bougé, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, *EuroPar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Science*, pages 678–686. Springer Verlag, August 1996. http://www.ibisc.fr/~michel/PUBLIS/1996/w21.pdf.

[Mic96c] Olivier Michel. Les amalgames : un mécanisme pour la structuration et la construction incrémentielle de programmes déclaratifs. In *Journées du GDR Programmation*, Orléans, 20–22 September 1996. GDR Programmation du CNRS. http://www.ibisc.fr/~michel/PUBLIS/1996/gdr96.ps.gz.

[Mic96d] Olivier Michel. *Représentations dynamiques de l'espace dans un langage déclaratif de simulation.* PhD thesis, Université de Paris-Sud, centre d'Orsay, December 1996. http://www.ibisc.fr/ michel/PUBLIS/1996/phd.ps.gz.

[Mic96e] Olivier Michel. A straightforward translation of d0l systems in the declarative data-parallel language 81/2. In *Euro-Par '96: Proceedings of the Second International Euro-Par Conference on Parallel Processing*, pages 714–717. Springer-Verlag, 1996. http://www.ibisc.fr/~michel/PUBLIS/1996/w11.ps.gz.

[MJ05] Olivier Michel and Florent Jacquemard. *An Analysis of a Public-Key Protocol with Membranes*, pages 283–302. Natural Computing Series. Springer Verlag, 2005. http://www.ibisc.fr/~michel/PUBLIS/2005/nspk-05.pdf.

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP71] John P. Mylopoulos and Theodosios Pavlidis. On the topological properties of quantized spaces. i. the notion of dimension. *Journal of the ACM (JACM)*, 18:238–246, 1971.

[MS99] John Maynard-Smith. *Shaping Life: Genes, Embryos and Evolution.* New Haven, CT: Yale University Press, 1999.

[MS03] Robin Milner and Susan Stepney. Nanotechnology: Computer science opportunities and challenges. Technical report, Submission by the UK Computing Research Committee to the Nanotechnology Working Group of the Royal Society and the Royal Academy of Engineering, August 2003.

[MSR91] Eric Mjolsness, David H. Sharp, and John Reinitz. A connectionist model of development. *Journal of Theoretical Biology*, 152(4):429–454, 1991.

[Nat02] Home page of the "Natural Computing" journal. http://www.springerlink.com/content/108905/, from 2002.

[OA95] Mehmet A. Orgun and Edward A. Ashcroft, editors. *Intensional Programming I*, Macquarie University, Sydney Australia, May 1995. World Scientific.

[Ora01] Andrew Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[ORG04] The Organic Computing Page, 2004. http://www.organic-computing.org.

[Out98] Sami Outerbah. Introduction d'une notion de référence distante dans un formalisme de capture de noms et implémentation d'une plate-forme d'expérimentation. Master's thesis, DEA Informatique d'Évry, 1998.

[Pat94] Ray Paton, editor. *Computing With Biological Metaphors*. Chapman & Hall, 1994.

[Pău00] Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 1(61):108–143, 2000.

[Pău01a] Gheorghe Păun. From cells to computers: Computing with membranes (p systems). *Biosystems*, 59(3):139–158, March 2001.

[Pău01b] Gheorghe Păun. From cells to computers: Computing with membranes (P systems). *Biosystems*, 59(3):139–158, March 2001.

[PEL+07] Przemyslaw Prusinkiewicz, Yvette Erasmus, Brendan Lane, Lawrence D. Harder, and Enrico Coen. Evolution and development of inflorescence architectures. *Science*, 316(5830):1452–1456, 2007.

[Per05] Lionel Perret. Intégration des types de données algébriques dans MGS. Master's thesis, École Centrale Paris, 2005.

[PJS92] Heinz-Otto Peitgen, Hartmut J urgens, and Dietmar Saupe, editors. *Chaos and fractals. New frontiers of science*. Springer-Verlag, New York, 1992.

[PLH+90] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, Jim S. Hanan, et al. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[PPS90] PPSN - Parallel Problem Solving from Nature, from 1990. Proceedings published from 1994 as LNCS volumes. http://ls11-www.cs.uni-dortmund.de/PPSN/.

[PRL06] Przemyslaw Prusinkiewicz and Anne-Gaëlle Rolland-Lagan. Modeling plant morphogenesis. *Current Opinion in Plant Biology*, 9:83–88, 2006.

[Pru00] Przemyslaw Prusinkiewicz. *Pattern Formation in Biology, Vision, and Dynamics*, chapter Paradigms of pattern formation: Towards a computational theory of morphogenesis. World Scientific, 2000.

[PSY02] The P Systems web page: http://psystems.disco.unimib.it/, 2002.

[QRW95] Patrice Quinton, Sanjay Rajopadhye, and Doran Wilde. Deriving imperative code from functional programs. In *Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture (FPCA'95)*, pages 36–44, La Jolla, California, June 25–28, 1995. ACM SIGPLAN/SIGARCH and IFIP WG2.8, ACM Press.

[Qui86] Patrice Quinton. An introduction to systolic architectures. In M. Vanneschi P. Treleaven, editor, *Proceedings of the Advanced Course on Future Parallel Computers*, volume 272 of *LNCS*, pages 387–400, Pisa, Italy, June 1986. Springer.

[Rau03] Erik Rauch. Discrete, amorphous physical models. *International Journal of Theoretical Physics*, 42(2):329–348, Februray 2003.

[Ric06] Adrien Richard. *Modèle formel pour les réseaux de régulation génétique & Influence des circuits de rétroaction*. PhD thesis, Université d'Évry, 2006.

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

[Rók95] Zsuzsanna Róka. The firing squad synchronization problem on Cayley graphs. *Lecture Notes in Computer Science*, 969:402–??, 1995.

[RPO+06] Dae-Kyun Ro, Eric M. Paradise, Mario Ouellet, Karl J. Fisher, Karyn L. Newman, John M. Ndungu, Kimberly A. Ho, Rachel A. Eachus, Timothy S. Ham, James Kirby, Michelle C. Y. Chang, Sydnor T. Withers, Yoichiro Shiba, Richmond Sarpong, and Jay D. Keasling. Production of the antimalarial drug precursor artemisinic acid in engineered yeast. *Nature*, 440(940-943), 2006.

[RPW03] Paul W. K. Rothemund, Nick Papadakis, and Eric Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. In *Preliminary Proceedings of DNA Computing, 9th international Workshop on DNA-Based Computers, DNA 2003, Madison, Wisconsin, USA, 1-4 June 2003*, page 125, 2003.

[RS92] Grzegorz Rozenberg and Arto Salomaa. *Lindenmayer Systems*. Springer, Berlin, 1992.

[SAB90] SAB - nternational Conferences on Simulation of Adaptive Behavior, from 1990. http://www.isab.org/confs/.

[SBC+06] Susan Stepney, Samuel L. Braunstein, John A. Clark, Andrew M. Tyrrell, Andrew Adamatzky, Robert E. Smith, Thomas R. Addis, Colin G. Johnson, Jonathan Timmis, Peter H. Welch, Robin Milner, and Derek Partridge. Journeys in non-classical computation II: initial journeys and waypoints. *Parallel Algorithms Appl*, 21(2):97–125, 2006.

[Seg97] Jean-Vincent Segard. Modèles de morphogénèse biologique dans un langage déclaratif de simulation. Master's thesis, D.E.A. de Sciences Cognitives du L.I.M.S.I, 1997.

[SGM+06] Richard S. Smith, Soazig Guyomarc'h, Therese Mandel, Didier Reinhardt, Cris Kuhlemeier, and Przemyslaw Prusinkiewicz. A plausible model of phyllotaxis. *Proceedings of the National Academy of Sciences*, 103(5):1301–1306, 2006.

[SH86] Guy L. Steele and W. Daniel Hillis. Connection machine LISP: Fine grained parallel symbolic programming. In *Proceedings of the 1986 ACM Conference on LISP and Functional Programming, Cambridge, MA*, pages 279–297, New York, NY, 1986. ACM.

[SM04] Antoine Spicher and Olivier Michel. Integration and pattern-matching of topological structures in a functional language. In *International Workshop on Implementation and Application of Functional Languages (IFL04)*, Lübeck, September 2004. http://www.ibisc.fr/~michel/PUBLIS/2004/IFL04.pdf.

[SM05] Antoine Spicher and Olivier Michel. Using rewriting techniques in the simulation of dynamical systems: Application to the modeling of sperm crawling. In *Fifth International Conference on Computational Science (ICCS'05)*, volume I, pages 820–827, 2005. http://www.ibisc.fr/~michel/PUBLIS/2005/iccs.pdf.

[SM06] Antoine Spicher and Olivier Michel. Declarative modeling of a neurulation-like process. *BioSystems*, 87:281–288, February 2006. http://www.ibisc.fr/~michel/PUBLIS/2006/biosystem06.pdf.

[SM07] Antoine Spicher and Olivier Michel. Représentation et manipulation de structures topologiques dans un langage fonctionnel. *Technique et Science Informatique*, 2007. http://www.ibisc.fr/~michel/PUBLIS/2007/tsijfla07.pdf,.

[SMC+07] Antoine Spicher, Olivier Michel, Mikolaj Cieslak, Jean-Louis Giavitto, and Przemyslaw Prusinkiewicz. Stochastic p systems and the simulation of biochemical processes with dynamic compartments. *BioSystems*, page in press, 2007. http://www.ibisc.fr/~michel/PUBLIS/2007/biosystem07.pdf.

[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. http://www.ibisc.fr/~michel/PUBLIS/2004/acri04.pdf.

[SMG05] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. Algorithmic self-assembly by accretion and by carving in MGS. In *7th International Conference on Artificial Evolution*, 2005. http://www.ibisc.fr/~michel/PUBLIS/2005/ea05.pdf.

[SMG06] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. *Rewriting and Simulation - Application to the Modeling of the Lambda Phage Switch*, volume Modélisation de systèmes biologiques complexes dans le contexte de la génomique, chapter Modeling of the Lambda Phage Switch. Genopole, 2006. http://www.ibisc.fr/~michel/PUBLIS/2006/phage-spicher-michel.pdf.

[Sot06] Ana Soto. La causalité biologique : émergence et interaction dans la morphogenèse normale et cancéreuse. Ecole thématique "Logique et interaction : vers une géométrie de la cognition", Cerisy-la-salle, September 2006.

[Spi03] Antoine Spicher. Typage et compilation de filtrage de chemins dans des collections topologiques. Master's thesis, DEA AMIB Université d'Évry, 2003.

[Spi06a] Antoine Spicher. *Transformation de collections topologiques de dimension arbitraire – Application à la modélisation de systèmes dynamiques.* PhD thesis, Université d'Évry, December 2006.

[Spi06b] Antoine Spicher. *Transformation de collections topologiques de dimension arbitraires. Application à la modélisation de systèmes dynamiques.* PhD thesis, Université d'Évry, 2006.

[SS78] Waclaw Szybalski and A. Skalka. Nobel-prizes and restriction enzymes. *Gene*, 1978.

[SSWT83] J. E. Sulston, E. Schierenberg, J. G. White, and J. N. Thomson. The embryonic cell lineage of the nematode *Caenorhabditis elegans. developmental biology*, 100:64–119, 1983.

[SSZW06] Georg Seelig, David Soloveichik, David Yu Zhang, and Erik Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314(5805):1585–1588, Dec 2006.

[Ste95] Ian Stewart. Four encounters with Sierpinski's gasket. *Mathematical Intelligencer*, 17:52–64, 1995.

[Str67] Christopher Strachey. Fundamental concepts in programming languages. In *Notes.* International Summer School in Computer Programming, 1967.

[Sus99] Gerald Jay Sussman. Robust design through diversity (position paper). In *Workshop on Amorphous Computing*, Cambridge, September 1999. DARPA ITO – MIT Lab, Cf. http://swiss.csail.mit.edu/projects/amorphous/workshop-sept-99/.

[SZ98] Peter Schröder and Denis Zorin. Subdivision for modeling and animation. volume SIGGRAPH'98 course notes. ACM Press, 1998.

[TE68] Lawrence G. Tesler and Horace J. Enea. A language design for concurrent processes. In *AFIPS Conference Proceedings*, volume 32, pages 403–408, 1968.

[Tho78] René Thomas. Logical analysis of systems comprising feedback loops. *Journal Theoretical Biology*, 73(4):631–656, 1978.

[Tho03] Fabien Thonnérieux. Réalisation d'une interface graphique pour le traitement des sorties du programme MGS. Master's thesis, IIE, 2003.

[Tix06] Sébastien Tixeuil. Vers l'auto-stabilisation des systèmes à grande echelle. Technical report, Université Paris-Sud XI, Orsay, France, May 2006. Habilitation à Diriger les Recherches. http://tel.archives-ouvertes.fr/tel-00124848.

[TK01] René Thomas and Morten Kaufman. Multistationarity, the basis of cell differentiation and memory. i. structural conditions of multistationarity and other nontrivial behavior. *Chaos*, 11(1):170–179, 2001.

[TN87] Tommaso Toffoli and Margolus Norman. *Cellular Automata Machine.* MIT Press, Cambridge MA, 1987.

[Tur52] Alan M. Turing. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc. of London*, Series B: Biological Sciences(237):37–72, 1952.

[Ula62] Stanislas M. Ulam. On some mathematical problems connected with patterns of growth of figures. *Proc. Symposia Appl. Math.*, 14:215–224, 1962.

[UMC98] UMC - Unconventional Computation, from 1998. Proceedings published at Springer. http://www.cs.auckland.ac.nz/CDMTCS/conferences/uc.

[Uni05] University of York and Microsoft Research. *The Grand in Non-Classical Computation*, York, UK, 18–19 april 2005.

[Var99] Francisco J. Varela. *Principle of Biological Autonomy*. New York: McGrawHill/Appleton and Lange, 1999.

[VN66] John Von Neumann. *Theory of Self-Reproducing Automata*. Univ. of Illinois Press, 1966.

[WBH+03] Ron Weiss, Subhayu Basu, Sara Hooshangi, Abigail Kalmbach, David Karig, Rishabh Mehreja, and Ilka Netravali. Genetic circuit building blocks for cellular computation, communications, and signal processing. *Natural Computing*, 2(1):43–84, 2003.

[WBL+02] Lewis Wolpert, Rosa Beddington, Peter Lawrence, Elliot Meyerowitz, Jim Smith, and Thomas M. Jessell. *Principles of Development (2nd ed.)*. Oxford: Oxford University Press, 2002.

[Wie80] Erwin Wiedmer. Computing with infinite objects. *Theoretical Computer Science*, 10(2):133–155, 1980.

[Wit21] Ludwig Wittgenstein. Logisch-philosophischen Abhandlung. *Annalen der Naturphilosophie*, XIV:185–262, 1921. trad. française de P. Klossowski, Tractatus logico-philosophicus, Gallimard 1961.

[WK00] Ron Weiss and Tom Knight. Engineered communications for microbial robotics. In *DNA: International Workshop on DNA-Based Computers*. LNCS, 2000.

[WSJ+98] Lewis Wolpert, Jim Smith, Tom Jessell, Peter Lawrence, Elizabeth Robertson, and Elliot Meyerowitz. *Principles of development*. Oxford University Press, 1998.

[YC92] J. Allan Yang and Young-il Choo. Data fields as parallel programs. In *Proceedings of the Second International Workshop on Array Structure*, Montreal, Canada, June/July 1992.

[YWA02] Yohei Yokobayashi, Ron Weiss, and Frances H Arnold. Directed evolution of a genetic circuit. *Proc Natl Acad Sci U S A*, 99(26):16587–16591, Dec 2002.

[ZEM+07] Y.-H. Percival Zhang, Barbara R. Evans, Jonathan R. Mielenz, Robert C. Hopkins, and Michael W.W. Adams. High-yield hydrogen production from starch and water by a synthetic enzymatic pathway. *PLoS ONE*, 2(5):e456, mai 2007.