Computational Models for Integrative and Developmental Biology

Jean-Louis Giavitto[1], Christophe Godin[2], Olivier Michel[1] and Przemyslaw Prusinkiewicz[3]

[1] LaMI, umr 8042 du CNRS,
Tour Evry2, Genopole - Université d'Evry
523 place des terasses de l'Agora
91000 Evry, France
giavitto,michel@lami.univ-evry.fr

[2] AMAP, umr CIRAD-CNRS-INRA-Université Montpellier II,
TA 40/PS2
34398 Montpellier cedex 5, France
godin@cirad.fr

[3] Department of Computer Science, University of Calgary
2500 University Drive N.W.
Calgary, Alberta, Canada T2N 1N4
pwp@cpsc.ucalgary.ca

1 Introduction

The relation between biology and computation has a long history reviewed by Langton [LIL89]. In this paper, we classify the interactions between computer science and biology in three areas:

1. *Bioinformatics* develops the automated management and analysis of biological data.

2. *Computational Biology* looks at biological entities as information processing systems with the final goal of a better understanding of nature using computer science notions.

3. *Biological Computing* goes in the reverse direction and studies how biological techniques can help out with computational problems.

Bioinformatics consists of developing software tools to support and help the biologist in the analysis and comprehension of biological systems. A good example is the development of data-bases supporting the genome project [Kan00].

Biological Computing imports some *biological metaphors* [Pat94] to develop new way of computing and to design new algorithms. From the beginning of computer sciences, biological processes have been abstracted to produce new computational models: formal neural networks inspired by natural neurons, evolutionary algorithm inspired by Darwinian evolution (see the "Parallel Problem Solving from Nature" (PPSN) conference series), parallel computer architecture (e.g. cellular automata) inspired by biological tissues (see for example the "Information Processing in Cells and Tissues" (IPCAT) conference series), DNA computing abstracted from biochemistry [Pau98a], cooperative distributed algorithm (e.g. multi-agents) motivated by ethological behaviors or social interactions, ...

Computational Biology. Here we are mainly interested in computer modeling and simulation of biological processes. The computer simulation of a biological process implies the definition of a model sufficiently rigorous to lead to a program. With such a formal model, it is possible to systematically explore the system's behavior and sometimes to make predictions. This kind of study is part of the more general idea of *simulated experiments* (also called *in silico* experiment by biologists and numerical experiment by physicists). These experiments are required when in-vivo or in-vitro experiments are out of reach for economical, practical or ethical reasons. Note however that the simulation of a computer model is only one of its possible use: because it is formal, it is possible to reason about it and for example to infer some properties (existence of steady state, stability, phase changes, etc.) that can be checked against the natural phenomena.

More generally, formal models can have a pedagogical, normative, constructive or ideological role:

- pedagogical and heuristic: the model is used to share knowledge about a given system or to illustrate a set of complex relationships involved in a biological process.

- normative: the model is used as a reference between scientists or to compare several systems.

- constructive: the model is used as a blueprint in the design of a new biological entity. Biology has reached the point where in addition to the study of already existing natural entities, it has to design new biologicals artifacts (drug design, metabolic pathways, genetically modified organisms, ...).

- ideological: a model illustrates some biological paradigm and constraints furthermore the investigated schemes. Biology has imported a number of notions developed in computer science, for instance the notion of programs, memory, information, control, etc. [Ste88, Kel95], that have then structured biological theories.

The transfer of concepts and tools between biology and computer science is not a one-way process and often, a computing model inspired initially by a biological phenomena, leads to a formalism used later in simulation of some (other) biological processes. A good example is given by the history of cellular automata (CA): initially developed by J. Von Neuman [VN66], they abstract the idea of a tissue of cells, to investigate the notion of self-reproducing programs. The CA formalism has then been largely used in biological simulation, for example to model the growth of tumor (Eden's models) or in ecology (it has been also successful in numerous other application domains, like in physics).

The contributions of Computational biology in the area of molecular dynamics or ecological modeling, are now well established. They are largely centered around the notion of *dynamical systems*. What appears now, is that this kind of computational models can make connections between molecular mechanisms and the physiological properties of a cell. The theme

gene expression $\longrightarrow$ *system dynamics* $\longrightarrow$ cell physiology

is an emerging paradigm [JTN00] that becomes increasingly more important as we try to integrate the exponential knowledge of all the cells components in a true understanding of the cell. However, this schema from biology to dynamical system and back to biology, has long been advocated in the more general domain of the development [Smi99, Kau95].

2 Dynamical systems

2.1 Basic definitions

Many natural phenomena can be modeled as *dynamical systems*. At any point in time, a dynamical system is characterized by its *state*. A state is represented by a set of *state variables*. For example, in the description of planetary motions around the sun, the set of state variables may represent positions and velocities of the planets. Changes of the state over time are described by a *transition function*, which determines the next state of the system (over some time increment) as a function of its previous state and, possibly, the values of external variables (input to the system). This progression of states forms a *trajectory* of the system in its *phase space* (the set of all possible states of the system).

Mathematical objects with diverse properties can be considered dynamical systems. For instance, state variables may take values from a continuous or discrete domain. Likewise, time may advance continuously or in discrete steps. Examples of dynamical systems characterized by different combinations of these features are listed in Table 1.

Table 1: Some formalisms used to specify dynamical systems according to the discrete or continuous nature of time and state variables.

`C`: continuous, `D`: discrete.	ODE	Iterated Mappings	Finite Automata
Time	`C`	`D`	`D`
State	`C`	`C`	`D`

In simple cases, trajectories of dynamical systems may be expressed using mathematical formulas. For example, the ODE (ordinary differential equation) describing the motion of a mass on a spring has an analytical solution expressed by a sine function (linear spring, in the absence of friction and damping). In more complex cases, analytic formulas representing trajectories of the system may not exist, and the behavior of the system is best studied using computer simulations.

By their nature, simulations operate in discrete time. Models initially formulated in terms of continuous time must therefore be discretized. Strategies for discretizing time in a manner leading to efficient simulations have extensively been studied in the scope of simulation theory, *e.g.* [Kre86].

Dynamical systems with apparently simple specifications may have very complex trajectories. This phenomenon is called *chaotic behavior*, *c.f.* [PJS92], and is relevant to biological systems, for example populations models [May75, May76].

2.2 Structured dynamical systems

Many biological systems are structured, which means that they can be decomposed into parts. The advancement of the state of the whole system is then viewed as the result of the advancement of the state of its parts. For example, the operation of a gene regulation network can be described in terms of the activities of individual genes.

Formally, we use the term *structured dynamical system* to denote a dynamical system divided into component subsystems (units). The set of state variables of the whole system is the Cartesian product of the sets of state variables of the component subsystems. Accordingly, the state transition function of the whole system can be described as the product of the state transition functions of these subsystems. Similarly to non-structured systems, structured dynamical systems can be defined assuming continuous or discrete state variables and time. In addition, the components can be arranged in a continuous or discrete manner in space. Some of the formalisms resulting from different combinations of these features are listed in Table 2.

Table 2: Some formalisms used to specify structured dynamical systems according to the continuous or discrete nature of space, time, and state variables of the components. The heading "Numerical Solutions" refers to explicit numerical solutions of partial differential equations and systems of coupled ordinary differential equations.

`C`: continuous, `D`: discrete.	PDE	Coupled ODE	Numerical Solutions	Cellular Automata
Space	`C`	`D`	`D`	`D`
Time	`C`	`C`	`D`	`D`
States	`C`	`C`	`C`	`D`

Time management is an important issue in the modeling and simulation of structured systems [Lyn96]. For example, state transitions may occur *synchronously* (simultaneously in all components) or *asynchronously* (in one component at a time). Furthermore, efficient simulation techniques may assume different rates of time progression in different components [Jef85].

In many cases, the transition function of each subsystem depends only on a (small) subset of the state variables of the whole system. If the components of the system are discrete (i.e., excluding partial differential equations, or PDEs), these dependencies can be depicted as a *directed graph*, with the nodes representing the subsystems and the arrows indicating the inputs to each subsystem. We say that this graph defines the *topology* of the structured dynamical system, and call *neighbors* the pairs of subsystems (directly) connected by arrows.

The topology of a structured dynamical system may reflect its *spatial organization*, in the sense that only physically close subsystems are connected. A dynamical system with this property is said to be *locally* defined. Locality is an important feature of systems that model physical reality, because physical means of information exchange ultimately have a local character (e.g., transport of signaling molecules between neighboring cells). On the other hand, physically-based models need not to be rigorously local. For example, when modeling plants, it may be convenient to assume that higher branches cast shadow on lower branches without simulating the local mechanism of light propagation through space.

When the number of components in a structured dynamical systems is large, the exhaustive listing of all connections between the components becomes impractical or infeasible. This limitation can be overcome in several ways. For example, if the components are arranged in a regular pattern, the neighbors of each component need not to be listed explicitly. This is the case of *cellular automata* (*e.g.* [TM87], in which cells are arranged in a square grid). *Group-based fields* [GM01b] are a generalization of this idea, allowing for a wider range of connection patterns. Large structures can also be defined by *simulated development*, discussed next.

2.3 Dynamical systems with a dynamic structure

A developing multicellular organism can be viewed as a dynamical system in which not only the values of state variables, but also the *set* of state variables and the state transition function change over time. These phenomena can be captured using an extension of structured dynamic systems, in which the set of subsystems and/or the topology of their connections may dynamically change. We call these systems *dynamical systems with a dynamic structure* [GM01b], or $(\mathrm{DS})^2$-systems in short.

For example, let us consider a model of a multicellular organism, defined at the level of individual cells. When a cell divides, the subsystem that represents it is replaced by two subsystems that represent the daughter cells. Furthermore, the topology of the whole system is adjusted to:

- remove connections (neighborhood relations) between the mother cell and the rest of the organism,
- create connections between the daughter cells,
- insert connections between the daughter cells and the rest of the system.

These operations make it possible to gradually create a large network of interconnected cells.

2.4 A Taxonomy of Formalisms

From a computer science (or a mathematical) point of view, the problem raised by the simulation of dynamical systems with a dynamical structure is that of the programming paradigm (or the modeling language) well fitted to the specification of such systems. For instance, the PDE formalism is not a relevant solution because it prescribes an *a priori* given set of relations between an *a priori* given set of variables. Consequently, these two sets, which embed implicitly the structural interaction between the entities or the system parts, cannot evolve jointly with the running state of the system [Mic96, pp 6, 85], [GM01b, chapter 1].

However, there exist several formalisms that can be used. The criteria used to classify the DS formalism in section 2.1 and 2.2 are still valid and the representation of time and state can be discrete or continuous for $(\text{DS})^2$ as for standard DS. Here we propose an additional criterion to distinguish between the topological nature of the system structure. Table 3 presents some formalisms for the discrete time case.

Table 3: Some formalisms used for the modeling of $(\text{DS})^2$, according to the underlying topology of the state.

Topology	Multiset	Sequence	Uniform	Combinatorial
Formalism	multiset rewriting	L-systems	GBF	map L-systems, Graph-grammars, MTG, MGS

In this table, the first line gives the type of the topology used to connect the subcomponents of a system. In a *multiset*, all elements are considered to be connected to each other. In a *sequence*, elements are ordered linearly; this case includes lists and extends also to tree-like structures. *Uniform* structures represents a regular neighborhood: for example, in a rectangular lattice (Von Neumann neighborhood), each element has exactly four neighbors. *Combinatorial* structures are used to define arbitrary connections between the components.

Considering solely the type of the topology underlying the structure of a state is only a partial caracterization that does not emphasize other several important points. Let us mention some of them.

- The relationship between the components can take place in an *a priori* structure. This approach is also known as the Newtonian conception of space where phenomena take place in a predefined scene. The other approach, which has been promoted by Leibniz, considers the topology as the result of the connection between the existing entities. In this point of view, the topology results from the dynamic connection between the system elements. This distinction is found in biology with the notions of *space oriented* or *structure oriented* models. For instance, accretive growth (growth on the boundaries) is an example of a space oriented process and intercalary growth (growth from the inside) is an example of a structure oriented process.

- There are several degrees in the dynamic of the structure. In the simplest case, the type of the topology remains the same during the evolutions of the system. An example is the growth of *Anabaena* filaments (Cf. section 4.2) where the system is always described as a sequence of cells. In addition, once a cell is connected with two neighbors, these connections remain the same. On the other hand, during the development of an embryo, several domains of cells change dramatically their shapes. For instance, the neural tube is formed dorsally in the embryonic development of Vertebrates by the joining of the 2 upturned neural folds formed by the edges of the ectodermal neural plate, giving rise to the brain and spinal nerve cord. In this process, which implies cell migration, the connections of a cell change over time and the global shape changes from a sheet to a tube.

- We have assumed that the interaction between the system parts can be described by a graph. Implicitly, this implies that elements interact two by two, which is not always the case. More elaborated interaction may imply more participants (e.g. a chemical reaction between two chemicals that requires also a catalyst; or the many-to-one relation between a subsystem and its decomposition). An interaction between n participants can be modeled by an n-edge in an *hypergraph*. An alternative representation is to use a n-simplex in a *simplicial complex* [GV01]. In the last case, the dimension of the simplex is directly linked with the number of participants.

- The notion of dimension also appears in the interactions between components in the following way. Often, the components of a system have a physical nature and the logical neighborhood established by the component interaction is the same as the spatial neighborhood implied by the physical structure of the system. For example, the topology implied by the representation of the cell sub-structures is tridimensional (compartments), bidimensional (membranes) and zero-dimensional (molecules). Obviously, the interactions that must be described depend of the dimension of the invoked entities: for instance, a flow of molecules can be conceived only through a membrane boundary between two compartments, not between a filament and another molecule; conservation laws depend on the topological nature of the entities, etc. From this point of view, multiset corresponds to a trivial topology (two points are always neighbors), L-systems corresponds to one-dimensional topologies and a GBF described by n fundamental generators (cf. below, section 5) describe n-dimensional topologies.

2.5 Outline

Following table 3, the next sections and chapters presents some formalisms usable for $(\mathrm{DS})^2$ Modeling:

- Section 3 reviews the use of multisets to model biological state and multiset rewriting to specify the evolution function.
- Section 4 sketches the L-system formalism. This formalism is an effective approach for the modeling of linear and branching structure. For instance, it as largely been applied in the field of plant growing.
- Section 5 presents a general framework, instantiated in a programming language, that is able to unify several approaches by using a topological point of view.
- The chapter **??** "Cellular automata and multi-agent" in this document gives some examples of the use of the computational device in the field of biological modeling.
- "Neural networks" are a special kind of dynamical systems. A large part of the considerations presented here, apply. Their importance has motivated numerous investigations and a lot of results are available. They are presented in **??**.

3 Multiset Rewriting and the Modeling of Biological Systems

3.1 Basic Concepts

Consider a simple chemical system of two molecules types A and B. We suppose that only deterministic second-order catalytic reactions are allowed, that is: a collision of two molecules will catalyze the formation of a specific third molecule and the two colliding molecules are regarded as catalysts. The possible reaction rules are given explicitly as follows:

$$\begin{array}{rrcl} r_1: & A+A & \longrightarrow & A+A+B \\ r_2: & A+B & \longrightarrow & A+B+B \\ r_3: & B+B & \longrightarrow & B+B+A \end{array}$$

A simulation in which every molecule is explicitly stored and every single collision is explicitly performed can easily be implemented if the chemical reactor is abstracted as a *multiset*. Unlike a set, an element can occur several times in a multiset. In the following, we denote a multiset using braces: $\{A, C, A, D, B, C\}$ is a multiset m with elements A and C occurring twice, and elements B and D occurring only one time. To simulate the chemical reaction, we simply interpret each rule as a transformation of the multiset. For instance, the rule r_1 specifies that two molecules A taken in the multiset have to be replaced by the three molecules A, A and B. For example, if reaction r_1 occurs in m at a given time step t_0, then m is transformed in $\{A, C, A, D, B, C, B\}$ (one additional B is produced). See figure 1.

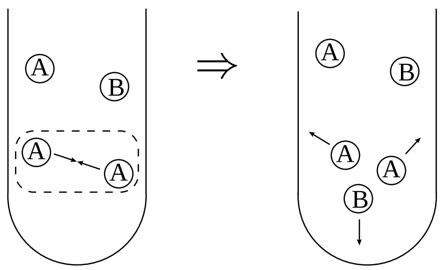

Figure 1: Illustration of one occurence of a reaction r_1 occuring in a test tube considered as a multiset of molecules.

Because several chemical reactions can occur *in parallel* (which means that several reactions involving different elements occur in the same time step), the strategy is to apply in parallel as many transformations as possible to the multiset. Such transformations are iterated to model the evolution of the state of the reactor. However, several competing rules may apply at the same time step: for instance consider a chemical reactor described by $\{A, A, B\}$ at time t_0 and subject to the two reactions r_1 and r_2. If r_1 occurs, then there is no longer A at t_0 to proceed with r_2 and vice-versa. The two reactions cannot occur together because there are not enough resources. In this case, we consider that one of the two rules is chosen in a non-deterministic manner. No assumption is made on the order on which the reactions occur.

The "+" sign that appears in the left and right hand side of the rules means that the linked molecules are present together in the chemical reactor. Thus, the left hand side of rule r_2 can also be equivalently written $B + A$. From a mathematical point of view, it is very convenient to consider $+$ as a formal commutative-associative operator used to construct multisets: a multiset $\{A, C, A, D, B, C\}$ is simply a formal sum $A+C+A+D+B+C$. The associativity and the commutativity properties are simply the expression that the elements of this last sum can be rearranged in any order. Then, rules like the r_i rules can be interpreted as rules for rewriting such formal expression. Abstractly, we can say that a chemical reaction can be modeled as a multiset rewriting system.

This modeling paradigm can be extended from this chemical example to other situations and its biological relevance is advocated in several recent papers [Man01, FMP00]. To quote[1] Fisher *et al.* [FMP00]: "A biological system is represented as a term of the form $t_1 + t_2 + \cdots + t_n$ where each term t_i represents either an entity or a message [or signal, command, information, action, etc.] addressed to an entity. [The simulation of the physical evolution of the biosystem] is achieved through term rewriting, where the left hand side of a rule typically matches an entity and a message addressed to it, and where the right hand side specifies the entity's updated state, and possibly other messages addressed to other entities. The operator $+$ that joins entities and messages is associative and commutative, achieving an ' associative commutative soup ', where entities swim around looking for messages addressed to them."

3.2 Division, Growth and Diffusion Processes

To illustrate this paradigm in a biological situation, we consider the multiplication of a mono-cellular organism in a test tube. A cell exists in one of two forms A or B. Type A and B can be used to characterize a phase of the life cycle of the cell, or as a cell polarity, etc. The division of a cell of type A produces one cell of type A and one of type B. In contrast, a cell of type B does not divide but evolves to give a cell of type A. This can be

[1]with adaptations in the terminology, brackets are our comments

summarized by the two rules:

$$\begin{array}{llll} r_1: & A & \longrightarrow & A+B \\ r_2: & B & \longrightarrow & A \end{array}$$

Starting from a test tube with three initial cells, abstracted as a multiset $m_0 = \{A, B, B\}$, the first three evolutions are:

$$m_0 \to \{A, B, A, A\} \to \{A, B, A, B, A, B, A\} \to \{A, B, A, B, A, B, A, B, A, A, A\} \to \dots$$

There exists several software environments that support multiset rewriting (see next paragraph). So the previous two rules *directly turn to a computer program* that simulates the growing and division processes of this hypothetic mono-cellular organism. In fact, these rules fit well the development of Anabaena, which is described more in details in the next section, if we neglect the sequential organization of the cells. However, this model admit also other interpretations. For example, Fibonacci studied (in the year 1202) about how fast rabbits could breed under some ideal circumstances. Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits are able to mate after one month so that at the end of its second month a female can produce another pair of rabbits. We simplify the model assuming that rabbits never die and that a female always produces one new pair (one male, one female) every month from the second month on. We model by symbol B a newly-born pair of rabbits and by symbol A a mature pair of rabbits. Then the rule r_1 expresses that a mature pair produces a newly-born pair and survive and rule r_2 specifies the maturation of a new pair.

The simulation of this process can be used to determine, for example, the relative ratio of A and B types in a population after some time. However, as mentioned in the introduction, the use of a formal model is not restricted to simulation and can be used to prove formal properties of the system without looking at the results of the simulation (e.g.: Fibonacci was able to prove that the ratio between B and A converges to the golden section as the time goes).

In the previous examples, each entity (a molecule, a cell or a pair of rabbits) is represented as an element of a multiset. In addition, the multiset structure allows objects to interact in a rather unstructured way, in the sense that an interaction between two objects is enabled simply by virtue of both being present in the multiset. In other word, there is no *localization* of the entities. Here is an example of another approach, where multiset rewriting is used in another way to take into account a geometric information. The problem is to model the diffusion of a set of particles on a line. The line is discretized as a sequence of small boxes, indexed by a natural integer, each containing zero or many particles. At each time step, a particle can choose to stay in the same box, or to jump to a neighboring box, with the same probability. See figure 2. The state of a particle is the index of the box where it resides. The entire state of the system is represented as a multiset of indices. The evolution of the system is then specified as three rules:

$$
\begin{aligned}
r_1 : \quad & n \longrightarrow n \\
r_2 : \quad & n \longrightarrow n-1 \\
r_3 : \quad & n \longrightarrow n+1
\end{aligned}
$$

where n is an integer and the operations "+" and "−" that appear in the right hand side are the usual arithmetic operators. Rule r_1 specifies the behavior of a particle that stay in the same box; rule r_2 corresponds to a particle that jumps to the box at the left; and rule r_3 defines a particle jumping to the right. Another solution is to factorize the three rules into one:

$$
r : \quad n \longrightarrow n + \mathrm{Random}(-1, 0, 1)
$$

where the function Random$(\dots)$ returns randomly one of its arguments. In the case of three competing rules, we must assume that there is some fairness in the choice of the rules r_1 to r_3 to be applied, i.e., they have the same probability of being chosen. If there is more chance to stay in a box than to leave it, then the underlying formalism must be able to express some finer control over the rule application. As a matter of fact, specifying an application strategy of the rules that respect the symmetries of the system can be very difficult.

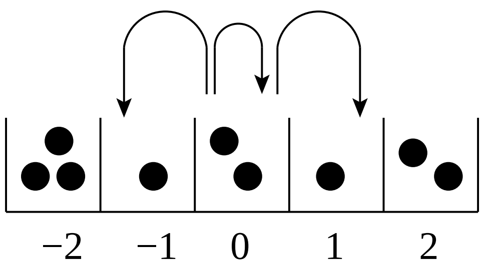

Figure 2: Diffusion of a particle along a line

3.3 Applications, Theories and Tools for Multiset Rewriting

Multiset rewriting has inspired several applications leading to the emergence of a new field: *Artificial Chemistry*. The home page [Dit00] and reference [DZB00] are a good introduction to this new area. There is a growing body of applications in artificial life, chemical and biological modeling, information processing and optimization. More specifically, Artificial Chemistry has been advocated as a productive framework for the study of pre-biotic and bio-chemical evolution, and for the study of the evolution of organization in general.

Multiset rewriting has also been used to extend other formalisms. For example, a multiset of L-systems is used to model an ecosystem (a multiset) of individual plants (modeled using L-system), see [LP02].

From the computer science point of view, the use of the chemical metaphor as a *computing model* has been investigated by Gamma [BM86, BCM87] in the middle of the eighties. A good review of the research done about Gamma can be found in [BFM01]. The CHemical Abstract Machine (CHAM) formalism extends these ideas with a focus on the expression of semantic of non deterministic processes [BB90]. The CHAM is an elaboration on the original Gamma formalism introducing the notion of sub-solution enclosed in a membrane. It is shown that models of algebraic process calculi can be defined in a very natural way using a CHAM: the fact that concurrency (between rule application) is a primitive built-in notion makes proof far easier than in the usual process semantics. The motivations of Gamma and the CHAM are the development of a formalism to support the specification and the programming of parallel and non deterministic programs. Multiset rewriting lies at the core of the formalism.

From the point of view of *term rewriting* [DJ90], multiset rewriting is the special case where the operators considered are both associative and commutative. In this domain, the perspective is more logical and directed towards the concepts of rewriting calculus and rewriting logic. The applications considered are the design of theorem provers, logic programming languages, constraint solvers and decision procedures. Several frameworks provide efficient and expressive environments to apply rewrite rules following dedicated strategies. It is worth mentioning ELAN [ela02] and MAUDE [mau02].

At last but not least, in the domain of formal language theory and computational complexity, P systems [Pau98b, Pau00] are a new distributed parallel computing model based on the notion of a membrane structure. This paradigm extends standard multiset rewriting introducing the notion of membrane. A membrane structure is a nesting of compartments represented, e.g, by a Venn diagram without intersection and with a unique superset: the skin. Objects are placed in the regions defined by the membranes and evolve following various transformations: an object can evolve into another object, can pass through a membrane or dissolve its containing membrane. In the initial definition of the P systems, each region defined by a membrane corresponds to a multiset of atomic objects which can evolve following evolution rules very similar to Gamma's (the right hand side of each rule is augmented to specify the destination of the results of the reaction). The membrane structure enables the specification of some localization of the processes. For an example, see section 5. Several alternatives have been devised and a region can be equipped with various computational mechanisms: string rewriting, splicing systems (DNA computing), etc. From the calculability point of view, several variants of such computing devices can compute all recursively enumerable sets of natural numbers. When an enhanced parallelism is provided, by means of membrane division (and, in certain variants where one works with string-objects, by means of object replication), NP-complete problems can be solved in linear time (of course, making use of an exponential space).

4 L-systems

4.1 Basic notions

L-systems were introduced in 1968 in the landmark paper by A. Lindenmayer, *Mathematical models for cellular interaction in development* [Lin68]. They provide a well developed and flexible tool for modeling and simulating a restricted but biologically important class of dynamic systems with a dynamic structure: linear and branching structures.

Originally, Lindenmayer described his formalism in terms of cellular automata, in which — in contrast to the standard definition — the cells could divide. Subsequently he observed that L-systems can be formulated in a simpler and more elegant manner in terms of formal language theory [Lin71]. That theory was originally proposed by Chomsky [Cho56, Cho57] to describe the syntax of natural languages. Its fundamental notion is that of a (generative) *grammar*, which consists of *productions* or *rewriting rules.* In general, a production replaces a symbol by zero, one, or several new symbols. They may represent words in a sentence, as in the original interpretation by Chomsky, but they also may represent cells or other components of a living organism, as was proposed by Lindenmayer. The use of related formalisms in the description of such apparently distant notions as languages and biological structures may seem surprising at first. In fact, it reflect the common dynamic nature of sentences under construction and developing organisms.

Applications of L-systems to modeling have an extensive literature, last reviewed in [Pru98] and [Pru99]. Below we outline one variant, called *parametric L-systems* [Han92, PH90, PL90] Within this formalism, the individual subsystems are called *modules.* Each module is represented by a symbol (letter) with optional parameters. This letter and parameters jointly characterize the module's state. For instance, the letter may represent a cell type, while the parameters may represent quantitative attributes of the cell, such as its dimensions and concentrations of chemicals that it contains.

The assumption that the organism forms a filament makes it possible to represent it at any moment of time as a string of modules, called a *parametric word.* For example, the string

$$A(2.5)B(3.14,0.2)CA(1.3) \tag{1}$$

may represent an organism that consists of four cells. The first cell has type A and is characterized by one parameter, the value of which is equal to 2.5. The remaining symbols have an analogous interpretation.

An L-system model describes the development of the entire structure by operating on individual modules. A production specifies the fate of a unit over a given time interval as a function of its current state and, optionally, the states of its neighbors. For example, the production

$$A(x) < B(y,z) > C \to CB(x+y,z/2) \tag{2}$$

operates on a module B that appears in the *context* of a module A to its left and module C to its right. The left and right contexts are separated from the *strict predecessor* B by the metasymbols (i.e., the symbols that do not represent modules) $<$ and $>$, respectively. In this example, module B divides into a module C and a new module B. The arithmetic expressions in the production's successor determine new parameter values. Hence, when applied to string (1), production (2) will yield the string

$$A(2.5)CB(5.64, 0.1)CA(1.3). \tag{3}$$

Simultaneous application of productions to all modules advances the state of the whole structure. If the set of module types is finite, the corresponding finite set of productions provides a mechanism for advancing the state of the entire structure independently of its size (the number of modules).

4.2 A sample model

We will illustrate the notion of genetic L-systems by constructing a model of heterocyst differentiation in a growing filament of the cyanobacterium *Anabaena*. The following description is adapted from [HP96].

The cells of *Anabaena* are organized into filaments which consist of sequences of *vegetative cells* separated by *heterocysts*. The vegetative cells divide into two cells of unequal length and, in some cases, differentiate into heterocysts which do not further divide. The organism maintains an approximately constant spacing between heterocysts: whenever the distance between two heterocysts becomes too large due to the division and elongation of vegetative cells, a new heterocyst emerges.

What mechanisms is responsible for the differentiation of heterocysts and the maintenance of the approximately constant spacing between them? Baker and Herman [BH70, BH72] (see also [dL87, HR75, Lin74] proposed the following simulation model. The heterocysts produce a substance that diffuses along the filament and is used by the vegetative cells. This substance inhibits the differentiation of vegetative cells into heterocysts. When its level in a cell drops below a threshold value, the cell detects that it is no longer inhibited and differentiates into a heterocyst.

Although the model of Baker and Herman is capable of reproducing the observed pattern of heterocyst spacing, it is very sensitive to parameter values. Small changes in these values easily result in filaments with pairs of heterocysts appearing almost simultaneously, close to each other. This is not surprising, considering the operation of the model. The gradient of the concentration of the inhibitor may be too small near the middle of a sequence of vegetative cells to precisely define the point in which a new heterocyst should differentiate. Consequently, the threshold value may be reached almost simultaneously by several neighboring cells, resulting in the differentiation of two or more heterocysts close to each other.

The above model can be improved assuming that the prospective heterocysts compete until one "wins" and suppresses the differentiation of its neighbors. This "interactive" model was originally proposed by Wilcox *et al* [WMS73]. It can be formalized using the framework of the *activator-inhibitor* class of reaction-diffusion models [Mei82]. In addition to the substance that inhibits the differentiation, the cells are assumed to carry a substance called the activator. The concentration of the activator is the criterion that distinguishes the vegetative cells (low concentration) from the heterocysts (high concentration). The activator and inhibitor are antagonistic substances: the production of the activator is suppressed by the inhibitor unless the concentration of the inhibitor is low. In that case, production of the activator drastically increases through an autocatalytic process (an increased concentration of the activator promotes its own further production). High concentration of the activator also promotes the production of the inhibitor, which diffuses to the neighboring cells. This establishes a ground for competition in which activator-producing cells attempt to suppress production of the activator in the neighboring cells. For proper values of parameters that control this process, only individual, widely spaced cells are able to maintain the high-activation state.

An L-system implementation of these mechanisms (a variant of the L-system from [HP96]) is given below.

$$
\begin{array}{rl}
\omega: & M(0.5, 0.1, 200, \text{right}) M(0.5, 0.1, 100, \text{right}) M(0.5, 0.1, 100, \text{right}) \\
p_1: & M(s_l, a_l, h_l, p_l) < M(s, a, h, p) > M(s_r, a_r, h_r, p_r) : \\
 & \qquad s < s_{max} \;\&\; a < a_{th} \rightarrow M(s', a', h', p) \\
p_2: & M(s_l, a_l, h_l, p_l) < M(s, a, h, p) > M(s_r, a_r, h_r, p_r) : \\
 & \qquad s \geq s_{max} \;\&\; a < a_{th} \;\&\; p = \text{left} \rightarrow \\
 & \qquad M(ks', a', h', \text{left}) M((1-k)s', a', h', \text{right}) \\
p_3: & M(s_l, a_l, h_l, p_l) < M(s, a, h, p) > M(s_r, a_r, h_r, p_r) : \\
 & \qquad s \geq s_{max} \;\&\; a < a_{th} \;\&\; p = \text{right} \rightarrow \\
 & \qquad M((1-k)s', a', h', \text{left}) M(ks', a', h', \text{right}) \\
p_4: & M(s_l, a_l, h_l, p_l) < M(s, a, h, p) > M(s_r, a_r, h_r, p_r) : \\
 & \qquad a \geq a_{th} \rightarrow M(s, a', h', p)
\end{array}
$$

where

$$
\begin{aligned}
s' &= s(1 + r\Delta t), \\
a' &= a + \left(\tfrac{\rho}{h}\left(\tfrac{a^2}{1+\kappa a^2} + a_0\right) - \mu a\right)\Delta t, \\
h' &= h + \left(\rho\left(\tfrac{a^2}{1+\kappa a^2} + h_0\right) - \nu h + D_h \tfrac{h_l + h_r - h}{sw}\right)\Delta t.
\end{aligned}
$$

The cells are specified as modules M, where parameter s stands for cell length, a is the concentration of the activator, h is the concentration of the inhibitor, and p denotes polarity, which plays a role during cell division. All

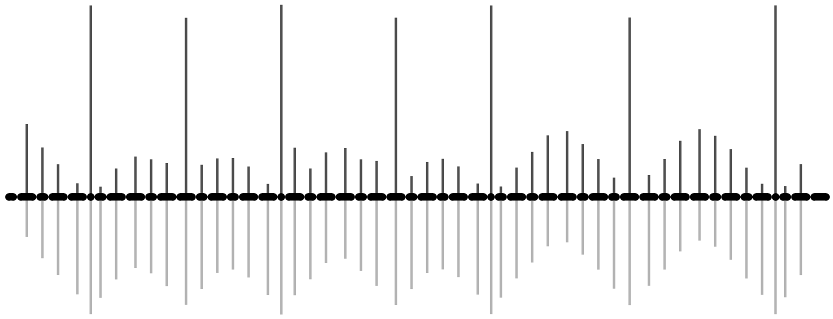

Figure 3: Fragment of a simulated filament of *Anabaena.* Vertical lines indicate the concentrations of the activator and inhibitor (above and below the cells, respectively). Notice the sharp peaks of the activator concentration that define the heterocysts, and high levels of the inhibitor concentration in the neighboring vegetative, which prevent their differentiation. The parameters used in the simulation were: $\rho = 3$, $\kappa = 0.001$, $a_0 = 0.01$, $\mu = 0.1$, $h_0 = 0.001$, $\nu = 0.45$, $D_h = 0.004$, $a_{th} = 1$, $k = 0.38196$, $s_{max} = 1$, $r = 0.002$, and $w = 0.001$.

productions are context-sensitive to capture diffusion of the activator and inhibitor. It is assumed that the main barrier for the diffusion are cell walls of width w. Production p_1 characterizes growth of vegetative cells ($a < a_{th}$), controlled by the growth rate r. A cell that reaches the maximum length of s_{max} divides into two unequal daughter cells, with the lengths controlled by constant $k < 0.5$. The respective positions of the longer and shorter cells depends on the polarity p of the mother cell, as described by productions p_2 and p_3. Increase of the concentration of the activator a to or above the threshold value a_{th} indicates the emergence of a heterocyst. According to production p_4, a heterocyst does not further elongate or divide. The equations for s', a', and h' govern the exponential elongation of the cells and the activator-inhibitor interactions [Mei82].

The operation of the model is illustrated in Figure 3. The vertical lines indicate the concentrations of the activator (above the filament) and inhibitor (below the filament) associated with each cell.

It is interesting from the historical perspective that the interactive model of Wilcox *et al.* [WMS73] and its subsequent L-system implementation [HP96] predicted the essential structure of the gene regulation network that controls the development of *Anabaena* filaments in nature [Ada00]. The activator corresponds to the protein HetR, which plays a key role in the maintenance of the heterocyst state, whereas the inhibitor corresponds to the protein PatS (or a fragment of it), which diffuses across the filament and maintains the spacing between the heterocysts. The character of interactions captured by the simulation model is consistent with the postulated structure of the gene regulation network, in which HetR upregulates its own production as well as the production of PatS, whereas PatS downregulates production of HetR.

We believe that models of similar nature, integrating the action of genes into developmental models of multicellular structures, will become more widely used in the future, offering insights into developmental processes that are difficult to obtain through observations and qualitative reasoning alone.

5 The MGS Approach

5.1 Motivations and Background

The previous examples of formalisms do not fully address issues of structural interactions between entities or system parts because of the *lack of topological organization.* The need to represent more structured organizations (than sequence or multiset) of entities and their interactions has been already stressed [FMP00] and motivates several extensions of rewriting (see for one example amongst others [BH00]). However, a general drawback with these extensions is that they work with a fixed topology of entities, and it is not obvious at all how to extend this to systems where the relationships between entities are drastically changing. This is precisely one of the main motivations of the MGS research project[2].

MGS is aimed at the representation and manipulation of local transformations of entities structured by *abstract topologies* [GM01b, GM02]. A set of entities organized by an abstract topology is called a *topological collection.* Topological means here that each collection type defines a neighborhood relation specifying both the notion of *locality* and the notion of *sub-collection.* The collection types can range in MGS from totally unstructured with sets and multisets to more structured with sequences and GBFs [GMS95, Mic96, GM01a] (other topologies are currently under development and include Voronoï partitions and arbitrary combinatorial neighborhoods).

The *global transformation* of a topological collection C consists in the parallel application of a set of *local transformations.* A local transformation is specified by a rewriting rule r that specifies the change of a sub-collection. A rewrite rule r:

1. selects a sub-collection A in C,
2. computes a new collection B as a function f of A and its neighbors,
3. and specifies the insertion of B in place of A into C.

These steps are summarized in figures 4 and 5. The topology of B depends on f and can be different from the topology of A. For example, a set in a sequence can be replaced by a sequence. Moreover, the topological structure of C can be changed through the application of transformations. These features enables the modeling of (DS)^2: states of a DS are represented by collections and transformations are used to model transition functions on these structured states.

[2] MGS is the acronym of "*(encore) un Modèle Géneral de Simulation (de système dynamique)*" (yet another General Model for the Simulation of dynamical systems). The MGS home page is located at url `www.lami.univ-evry.fr/mgs` where additional informations are available.

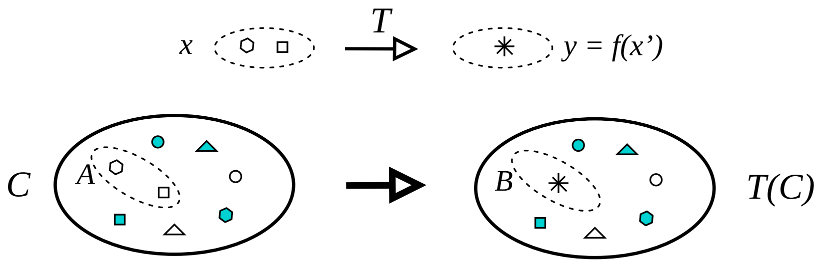

Figure 4: A basic transformation of a topological collection. Collection C is of some kind (set, sequence, array, cyclic grid, tree, term, etc). A rule T specifies that a sub-collection A of C has to be substituted by a collection B computed from A. The right hand side of the rule is computed from the sub-collection matched by the left hand side x and its possible neighbors x' in the collection C.

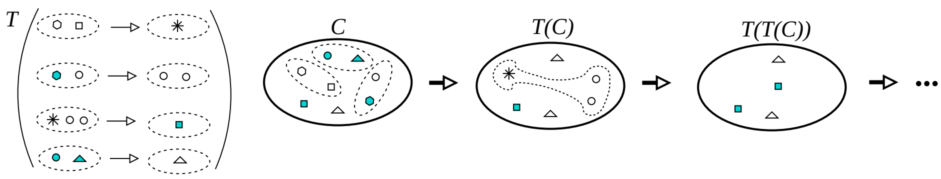

Figure 5: Transformation and iteration of a transformation. A transformation T is a set of basic transformations applied synchronously to make one evolution step. The basic transformations do not interact together. A transformation is then iterated to build the successive states of the system.

As a programming language based on topological concepts, MGS integrates the idea of topological collections and their transformations into a general high-level functional programming language: topological collections are just new kinds of values and transformations are functions acting on collections. The approach is purely declarative: operators acting on values combine values to give new values, they do not act by side-effect.

5.2 Biological Examples in MGS

In this subsection, we sketch several examples in various domains to exemplify the versatility of the MGS formalism.

The Eden Model

We start with a simple model of growth sometimes called the Eden model (specifically, a type B Eden model) [Ede58]. The model has been used since the 1960's as a model for such things as tumor growth and growth of cities. In this model, a 2D space is partitioned in empty or occupied cells (we use the white-space character and the `C` letter). We start with only one occupied cell. At each step, occupied cells with an empty neighbor are selected, and the corresponding empty cell is made occupied.

The corresponding `MGS` model starts by defining the 2D partition using a *group based field* (GBF in short). A GBF is an extension of the notion of array, where the elements are indexed by the elements of a group, called the *shape* of the GBF [GMS95, GM01a]. This kind of collection can be used to describe uniform and regular topologies. For example:

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

defines a shape called *Grid2*, corresponding to the Von Neuman neighborhood in a classical array (a cell above, below, left or right – not diagonal). The two names `north` and `east` refer to the directions that can be followed to reach the neighbors of an element. These directions are the *generators* of the underlying group structure. The list of the generators can be completed by giving equations that constraint the displacement in the shape:

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

defines an hexagonal lattice that tiles the plane, see. figure 6. Each cell has six neighbors (following the three generators and their inverses). The equation `east + north = northeast` specifies that a move following `northeast` is the same has a move to `east` followed by a move to `north`.

The Eden's aggregation process is simply described as the following transformation:

```
trans Eden = {
    x,y / (x = "C") & (y = " ")  ⇒  x,"C";
}
```

the keyword `trans` introduce the rules of a transformation. A rule takes the following form:

pattern $\Rightarrow$ *expression*

where *pattern* in the left hand side of the rule matches a sub-collection A of the collection C on which the transformation is applied. The sub-collection A is substituted in C by the collection B computed by the *expression* in the right hand side of the rule. Here, the pattern "x,y" filters an element y neighbor of an element x such that the value of x is occupied and the value of y is empty. The conditions on the elements matched are given by the expression after the "/" operator and the comma operator "," means that x and y must be neighbors. The right hand side specifies that the couple x,y matched by the left hand side must be replaced by a couple x,"C".

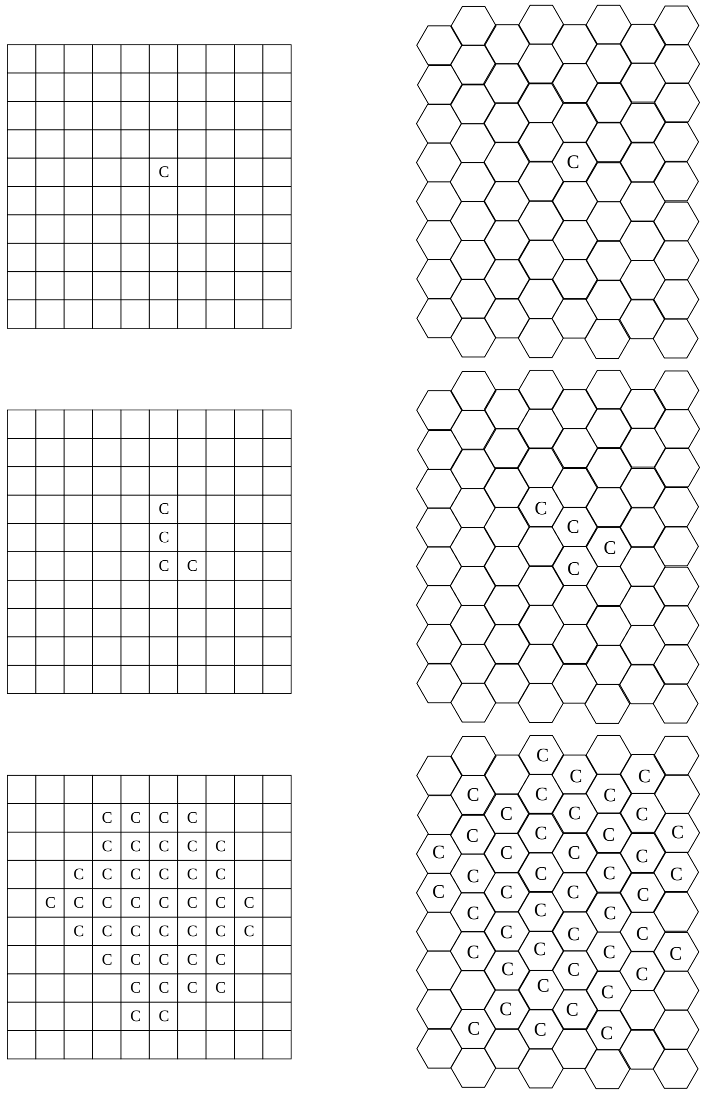

Figure 6: Eden's model on a grid and on an hexagonal mesh (initial state, and states after the 3 and the 7 time steps). The same transformation is used for both cases.

The transformation *Eden* defines a function that can then be applied to compute the evolution of some initial state. One of the advantages of the `MGS` approach, is that this transformation can apply indifferently on grid or hexagonal lattices (or *any* other collection kind). The meaning of the neighborhood operator "`,`" in the pattern of a rule depends on the collection on which the transformation is applied.

It is interesting to compare transformations on GBFs with the genuine cellular automata (CA) formalism (see the corresponding chapter). There are several differences. The notion of GBF extends the usual square grid of CA to more general Cayley graphs. The pattern in a rule may match arbitrary domain, not only one cell as it is usually the case for CA. Moreover, the value of a cell can be arbitrary complex (even another GBF) and is not restricted to take a value in a finite set.

Restriction Enzymes

This example shows the ability to nest different topologies to achieve the modeling of a biological organization. We want to represent the action of a set of restriction enzymes on the DNA. The DNA structure is simplified as a sequence of letters `A`, `C`, `T` and `G`. The DNA strings are collected in a multiset. Thus we have to manipulate a multiset of sequences (this kind of nested structures has been proved useful in other areas, e.g. [LP02]).

A restriction enzyme is represented as a rule that splits the DNA strings; for instance a rule like:

```
EcoRI =  x+ as X,
         (cut+ as CUT / CUT = "G","A","A","T","T","C"),
         y+ as Y
       ⇒ (X, "G")::("A","A","T","T","C", Y)::seq:()
```

corresponds to the *EcoRI* restriction enzyme with recognition sequence `G^AATTC` (the point of cleavage is marked with `^`). The x`+` pattern filters the part of the DNA string before the recognition sequence and the result is named X (the `+` operator denotes repetition of neighbors). Identically, Y names the part of the string after the recognition sequence. The right hand side of the rule constructs the two resulting parts as a sequence of two sequences (the `::` operator indicates the construction of a nested sequence).

We assume that all restrictions enzyme rules are collected into one transformation. We need an additional rule, called `Void` for specifying that a DNA string without recognition sequence must be inserted as such:

```
trans Restriction = {
    EcoRI = ...;
    ...;
    Void = x+ as X ={flat=false}=> X
}
```

The attribute "`flat=false`" in the body of the arrow of rule `Void` indicates that the X (which is a sequence) must be inserted in the resulting multiset as one single entity. This contrasts with the rule `EcoRi` whose right hand side computes a sequence of elements to be inserted in the enclosing multiset.

The transformation *Restriction* can then be applied to the DNA strings floating in a multiset using the simple transformation:

```
trans Apply = { dna ⇒ Restriction(dna) }
```

A Localized Signaling Network

At last but not least, we want to sketch the modeling of a spatially distributed biochemical network in MGS. We rely on a model proposed by A. E. Bugrim [Bug00]. The example focuses on a small signaling network that consists of cAMP and calcium signaling. See figure 7 for a more complete description.

The corresponding topological structure mimics the spatial organization of the cell using nested multisets, see figure 8. The MGS declarations:

```
collection Volume = bag;
collection Membrane = bag;

collection Environment = Volume;
collection Plasma = Membrane;
collection Cytosol = Volume;
collection EndoRetic = Membrane;
```

are used to introduce some new kinds of multisets (the `bag` keyword). This kinds are used here mainly do describe the hierarchy of localization and compartments and can be used, if necessary, to discriminate between multisets.

The main part of the corresponding MGS program consists in defining the ontology of this application domain: there exists several molecules, each have a name; some exists in two state: active or inactive; some are characterized as receptors; etc. Such ontology is described in MGS using *subtyping*. These subtypes are then used in pattern-matching to select entities with or without some properties. For example, a molecule is described as a record having or not some fields. Record type in MGS may specify the presence or the absence of a field, or the value of a specific field. For instance:

```
state Molecule = {name};
state Activity = {activation};
state Activated = {activation = 1};
state Inactivated = {activation = 0};
state ATP = Molecule + {name = "atp"};
```

define five record types. The record type declaration is introduced by the keyword `state`. *Molecule* is the type of any record having at least a field named `name`. *Activated* is the type of a record having at least a field named `activation` and with value `1`. This type is a subtype of *Activity* which only requires the presence of the field `activation`. The type *ATP* corresponds to a molecule named `"atp"`.

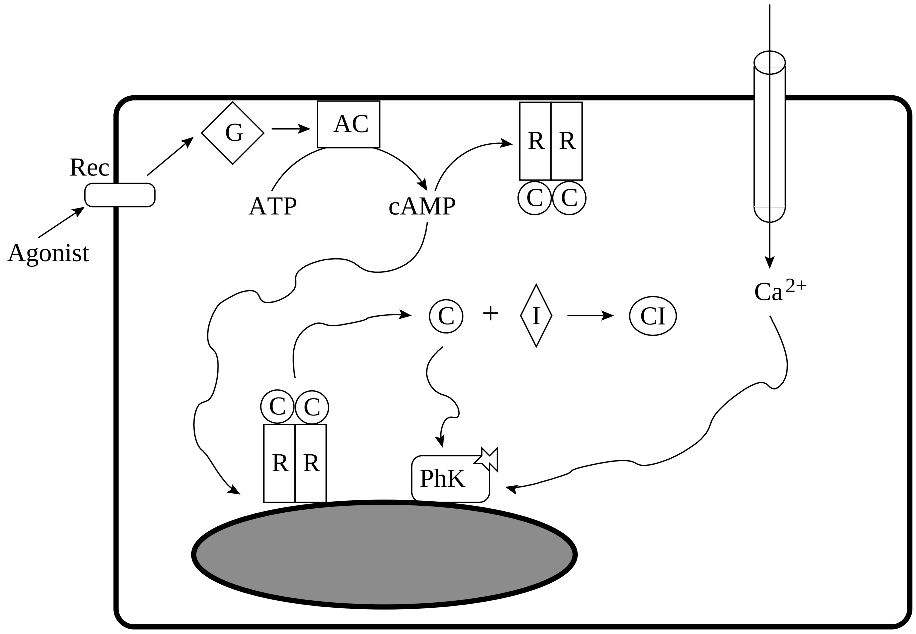

Figure 7: *cAMP and calcium signaling pathways* (this schema is reprinted from [Bug00]). The different components of the two pathways are localized at various places within the cell.
The first steps of the cAMP pathway occur at the plasma membrane, starting with the activation of adrenegric receptors. Then, the cAMP molecules bind to a regulatory sub-unit of the protein kinase A, with the effect of dissociating a catalytic sub-unit C. The localization of PKA depends of a family of anchoring proteins AKAPs that target this kinase to different compartments. In this example, two localizations are considered: the plasma membrane and an internal compartment (e.g., nucleus or ER).
The calcium pathway starts by the activation of a channel in the plasma membrane. The fraction of PhK associated to the internal compartment is the target of both pathways. A possible inhibitor I of PKA is also considered.

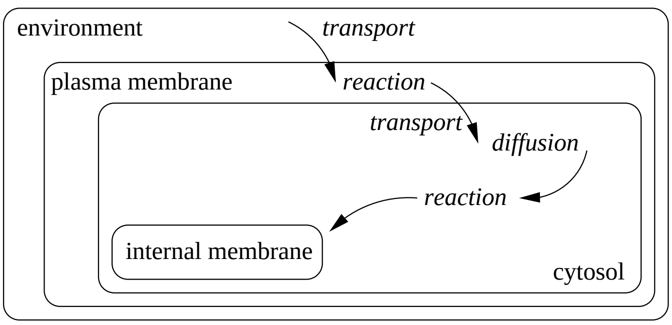

Figure 8: The reaction, diffusion and transport processes described in figure 7 are modeled as multiset transformations taking place in a nest of multisets. This is reminiscent of the P system approach, see section 3.

Three kinds of transformations are used to define the processes of the Bugrim's model. The first class corresponds to some ancillary transformations. For example

```
trans ActivateReceptor = { r:Receptor → r + {activation=1} }
```

is a rule that updates to `1` the field `activation` of an entity r of type *Receptor*. This kind of transformations is triggered by a rule of the sole transformation of the second class. This transformation summarize all the rule corresponding of the description of the biochemistry (they are about 10 reactions in this pathway):

```
trans Biochemistry = {
    R1 = a:ActiveAgonist, p:Plasma
       ⇒ a+{activation=0},ActivateReceptor(p);
    ...
}
```

For example, rule `R1` specifies that an active agonist and a plasma membrane interact to inactivate the agonist and to transform the plasma with transformation *ActivateReceptor* (this transformation turn on all the activation fields of the receptors anchored in the plasma membrane).

There is also only one transformation in the last class of transformations. It is used to thread the biochemistry rules amongst the nested multisets:

```
fun Run(x) = Thread(Biochemistry(x));
trans Thread = {
    p:Membrane ⇒ Run(p);
    c:Volume ⇒ Run(c);
}
```

The transformation *Thread* applies the function *Run* to each entity of type *Membrane* or *Volume* found in the collection argument. The function *Run* consists in running the biochemistry transformation and then iterating the threading.

The complete `MGS` program is approximatively 150 line long, including the building of the initial system state. It describes 40 molecules in diverse states, uses of 5 auxiliary transformation to define 10 chemical interactions.

6 Multiscale graphs

The previous formalisms have been used to model the changes of structure that arise throughout time. However, biological structures may change also due to a change in the scale of observations.

On the one hand, plants appear as complex structures due to the intrication of many sub-structures at various levels of detail. On the other hand, plants are essentially spatially and temporally periodic structures which gives an overall impression of simplicity. In such a paradoxical situation, the question arises: what mathematical formalisms and what tools are necessary to model plants at several scales ?

In this chapter, we analyse how biological systems, such as plants, can be formally represented with combinatorial formalisms (see section 2). We particularly analyze how this formalism must be designed in order to account for a new dimension, namely the scale dimension. We then briefly describe the types of mathematical and computational tools that must be developed in this context.

6.1 Plants as modular organisms

The growth of a plant can be depicted as the result of two growth processes. This *apical growth process* gives the plant the ability to develop in one direction. During their activity, shoot meristems can give birth to distinct embryogenic cellular areas (always associated with corresponding leaves), called axillary or lateral meristems. This defines the *branching process*. Plants make branching structures if the meristems located at leaf axils enter an apical growth process. Using the branching process, plants can develop shoots in more than one direction. The overall *growth process* is thus the combination of both the apical growth process and the branching process. Growth is a fundamentally repetitive process which creates various forms of patterns repeated as "modules" throughout the plant structure ([HRW86], [Bar91]). Figure 9 illustrates different types of modules that can be observed on plants.

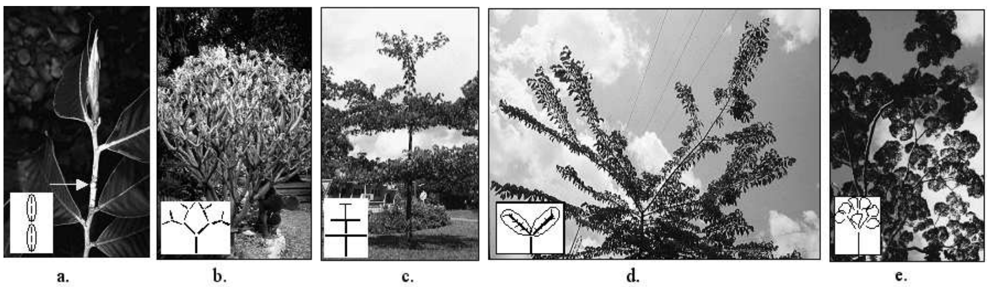

Figure 9: Different types of modularity in plants. a. nodes b. axes c. whorls d. branching systems e. crownlets

For a given type of module, the plant can be split-up into a set of modules of this type. This defines a particular plant modularity. A plant modularity, is caracterised by the type of modules considered and their adjacency within the plant. This information can be represented by a *directed graph.*

A directed graph is defined by a set of objects, called vertices, and a binary relation between these vertices. The binary relation defines a set of ordered pair of vertices, called edges. In plant representations, vertices represent botanical entities and edges adjacency between these entities. Edges are always directed from oldest entities to youngest ones. Given an edge (a, b), we say that a is a *father* of b and b is a *son* of a. Directed graphs representing plants have tree-like structures: every vertex, except one, called the root, has exactly one father vertex. Morevover, in order to identify the different axes of a given plant, two types of connections are distinguished: an entity can either precede (type '$<$') or bear (type '$+$') another entity (Figure 10). In order to describe different characteristics of plant entities, vertices can have attributes, e.g. length, diameter, spatial location, leaf area, number of flowers, type of branched entities, *etc.*

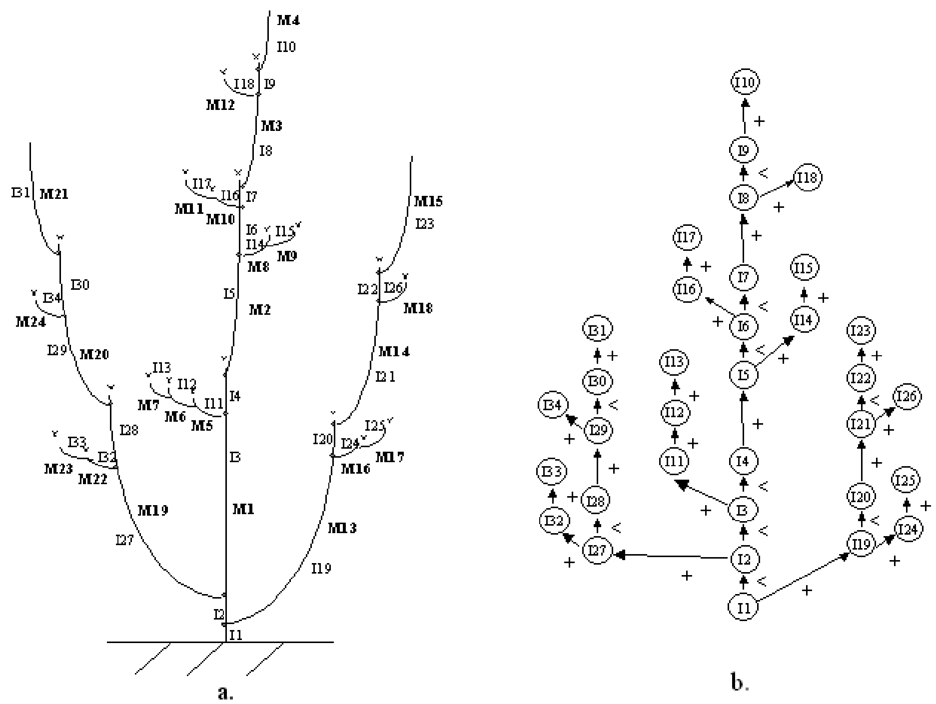

Figure 10: a. A tree b. The tree graph representation of its topology (at node scale)

6.2 Multiscale representations

Many modularities can exist on a single individual. Several types of modularity, stemming from either natural or artificial decomposition of the plant into modules, can exist within a single individual at the same time. For above-ground systems, at least the nodal (the plant is a set of leaves) and the axial modularity (the plant is a set of axes) coexist. If, in addition, the plant reiterates, a modularity by reiteration is superimposed on the previous ones. Thus, there always exist two or three types of modularities expressed in a plant simultaneously. There can be more, depending on the number of regular fluctuations that characterize the plant growth. This is the case, for example, for plants containing growth unit or annual shoot modules. These types of module can exist simultaneously in a plant, such as in apricot tree, evergreen oak or Aleppo pine. For a single plant, there is thus the theoretical possibility of finding numerous types of modularity, each one corresponding to a particular topological interpretation of the plant.

The existence of several modularities on the same plant can be illustrated by *Vochysia guyanensis* [San92]. For this plant, the number of modularities stemming from natural decomposition is relatively high. The highest scale corresponds to the description of the topological structure in terms of internodes. At a lower scale, the rhythmic elongation of stems produces an alternate sequence of cataphylls and developed leaves which enables the observer to define growth unit modules (11.a). The final stopping of stem elongation, due to the death of their apical meristem, makes it possible to group growth units into axes (11.a). The architectural unit of the young tree consists of a stack of such axes (11.b). The plant continues its development by reiterating its architectural unit. The resulting topological structure is described in terms of reiterated complexes. Eventually, at the lowest scale, the crown of the adult tree is a set of crownlets, each of them made of reiterated complexes (11.c). The plant can thus be represented by a specific topological structure for each possible scale. The set of these topological structures defined at every scale and their relations characterizes the overall topological structure of the plant, *i.e. multiscale topological structure* of the plant.

To formally represent the multi-modular structure of plants, extension od directed graphs, called multiscale tree graphs (MTGs) [GC98], are used. The MTG formalism has been designed in order to enable users to express both the modularity and the multiscale nature of plant structures. Each scale of analysis corresponds to a modular structure which can be formally represented by a tree graph. Entities at one scale are decomposed into entities at finer scales. For instance, internodes of Figure 10.a can be grouped into growth units, leading to a more macroscopic description of the plant topology (Figure 12).

A MTG integrates in a homogeneous framework the different tree graphs

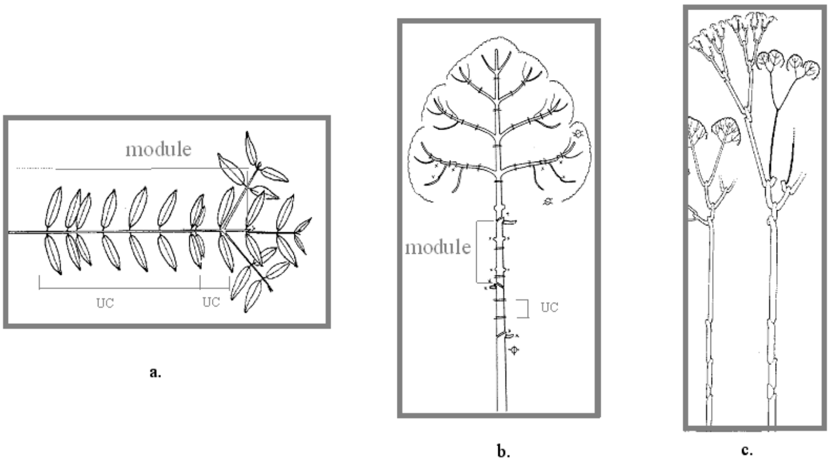

Figure 11: Nested modularities: a. nodes, growth units and axes. b. Architectural unit c. crowlets.

corresponding to plant descriptions at different scales (Figure 13.a). Vertices at one scale are composed of vertices at a higher scale. If an entity a is composed of n entities $x_1, x_2, ..., x_n$, for every $i \in [1, n]$, a is called the *complex* of x_i, and x_i is a *component* of a. The complex of any entity x_i is denoted $\pi(x_i)$. If the scale of a is defined by the integer s, then for every $i \in [1, n]$, the scale of x_i is $s + 1$. The most macroscopic scale s_0 consists of a single vertex, representing the entire plant, and by convention has value 0. In order to maintain coherence between the different tree graph representations of a same individual, MTGs must respect the following consistency constraint : if there exists an edge (x, y) in the tree graph representing the plant structure at scale $s + 1$, and if the complexes of x and y are different, then there necessarily exists a corresponding edge $(\pi(x), \pi(y))$ between these complexes in the tree graph representing the plant at scale s (Figure 13.b) This expresses that the connection between two macroentities results from the connection between two of their components.

6.3 Space of modularities

From a structural point of view, the relative position of two modularities in a plant can be of two types.

- Firstly, one modularity is a refinement of the other (Figure 14.a). For example, a topological structure represented in terms of growth units can be refined by considering the plant decomposition in terms of internodes. Each growth unit is considered as a set of internodes. Similarly, the axis structure of a plant can be interpreted as a refinement of the plant description in terms of branching systems, since each

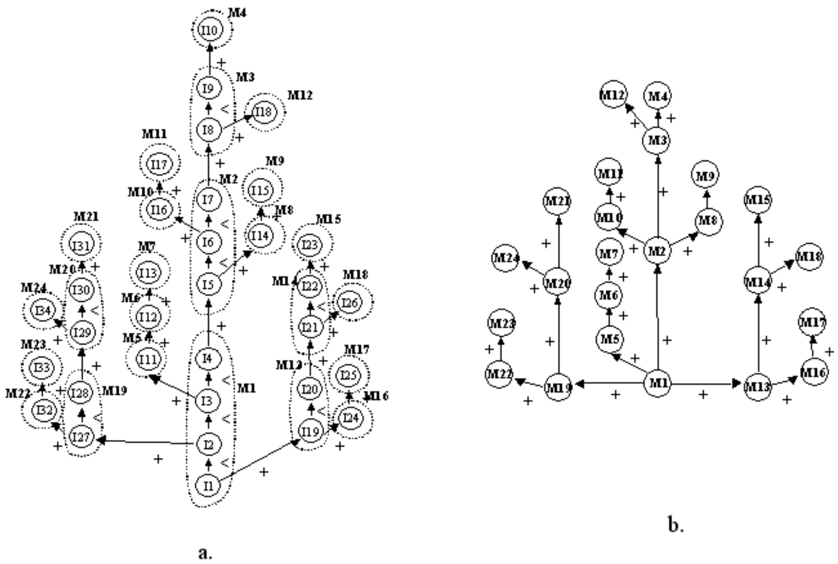

Figure 12: a. Partitionning graph of Figure 10 into growth units (M). b. Topology of the plant at scale M.

branching system can be decomposed into a set of axes. Hence, one modularity is a refinement of another if each module of the second can be decomposed into a set of modules of the first and, reciprocally, each module of the first modularity is a part of a module of the second. These modularities correspond to two topological structures representing the plant at two different *scales*. The highest scale corresponds to the finest modularity, while the lowest scale corresponds to the coarsest modularity. Within a plant representation, the scale of internodes is higher than the scale of growth units which is itself higher than the scale of axes.

- Secondly, the two modularities are not a refinement of eachother : they are overlapping (Figure 14.b). This is the case if at least one module of one modularity shares a common part with one module of the second modularity, whereas there is no inclusion of one into the other. Let us consider for example the topological structure of an apple tree in terms of both annual shoots and axes (14.b). At the beginning of the vegetative period, the apical meristem of some branches produces short shoots terminated by a flower, called "bourse"[CL95]. During a second phase of the vegetative period, a vegetative shoot may develop on some bourses. These are called "bourse shoots". A bourse shoot is part of the same annual shoot as the bourse, since it is created during the same vegetative period. Therefore, some annual shoots are made of a bourse bearing a bourse shoot. Such an annual shoot is thus

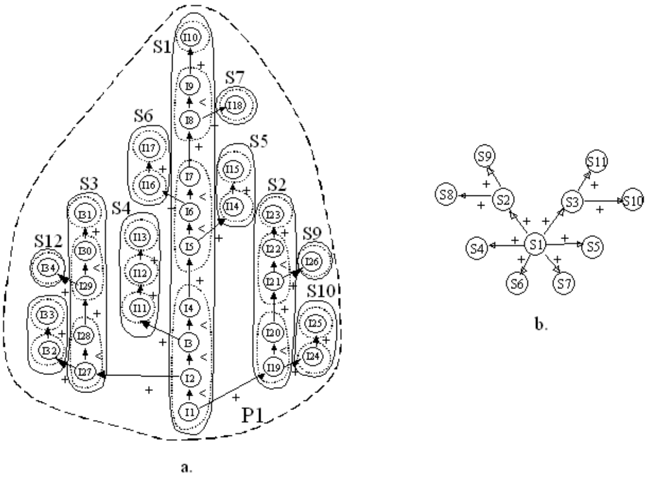

Figure 13: a. Multiscale graph corresponding to tree of Figure 10. b. corresponding topology at S module scale.

straddling two axes: on one side the axis terminated by the bourse and on the other side, the axis which begins with the bourse shoot. Reciprocally, each axis is straddling two annual shoots. The modularities corresponding respectively to axes and annual shoots determine two topological interpretations of the plant which are not a refinement of eachother.

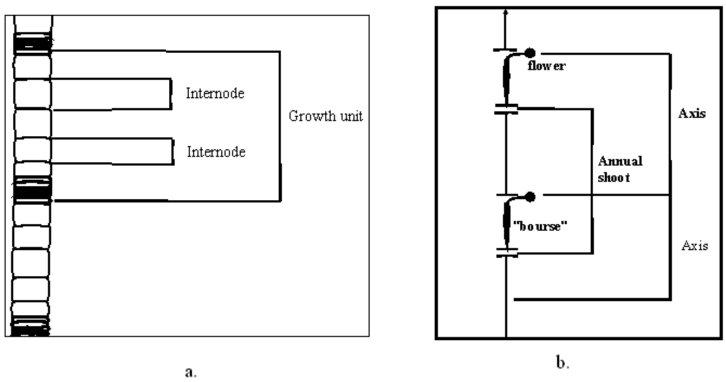

Figure 14: a. nested modularities. b. overlapping modularities

The different types of modularities that can be identified within a given plant define different topological structures. These modularities are *comparable* if they are refinements of each other. The refinement relation expresses the existence of a decomposition relation between the modules of the coarsest modularity and those of the finest. In the opposite case, modularities are *incomparable*, i.e. none of them is a refinement of the other. No decomposition relation exists between the modules of both modularities since they overlap.

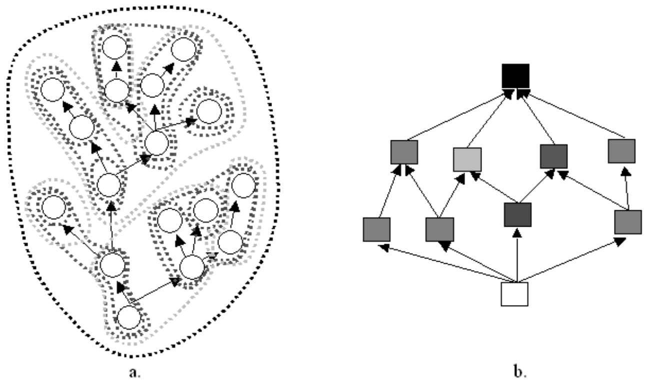

Figure 15: a. a general MTG. b. its corresponding modularity graph.

Now, if we consider a graph g and different partitionning of the vertices of this graph, representing different modularities (Figure 15.a). Let us assume that each modularity is represented by a square element (Figure 15.b), and an edge is drawn from modulatities A to modularity B whenever A is a refinement of B. The graph obtained from this process is a lattice :

> *Let g be a tree graph. Let $L(g)$ be the set of all partitions on g, such that the induced macroscopic graph (quotient graph) is a tree graph.*
>
> *$L(g)$ is a lattice*

This proposition characterizes the space of all modularities that can be potentially defined on a given individual by a remarkable algebraic property : it is a sublattice of the partition lattice (the set of all subsets of a set). A multiscale graph is associated with only a subset of this sublattice. This subset corresponds to the set of modularities that are actually taken into consideration by the observer in the plant description. Multiscale graphs are thus a model of the observer's subjective interpretation of the plant.

6.4 Growing multiscale structures

From a temporal point of view, the analysis of the relations between the different types of modularities is a delicate issue. Indeed, whereas the growth of a topological structure at a given scale seems to be a relatively clear phenomenon, the simultaneous growth of different topological structures representing a given individual, at different scales, raises the problem of understanding how these growth processes are linked to each other [GC98]. Figure 16 illustrates such a problem.

Consider an adult tree bearing a well hierarchized crown (16, date t_1). At a subsequent date t_2, a possible development of the crown may preserve the original hierarchy of branches. Another possible development is that one of the branches starts to compete with the trunk, yielding a reiterated complex (16 dates t_1 and t_2). This phenomenon can be interpreted in terms of MTGs (lower part of 16) if we assume that a component can belong to different complex entities throughout time.

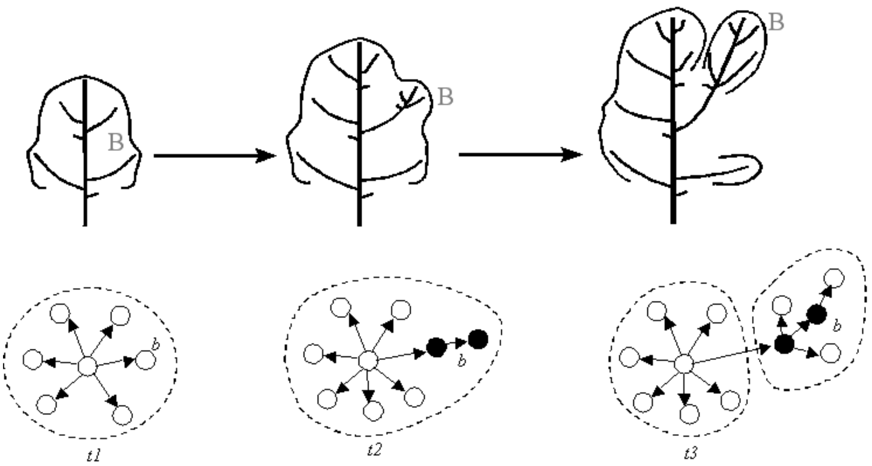

Figure 16: a. (upper part) reiterated complex is produced througout time. b. (lower part) Corresponding MTG interpretation

The growth of a multiscale structure illustrates an important aspect of the model: rather than an objective plant topological structure, defined once and for all, a time-varying multiscale graph actually represents the plant topological structure as a subjective object depending on the observer's goals, knowledge and means of observation.

6.5 Handling plant architecture databases

Multiscale tree graphs are currently used as the backbone of a general methodology for measuring and analyzing plant topological structures, implemented in the AMAPmod software [GGC99]. Real plants are encoded by the observer using a specific coding language designed for this purpose. The multiscale plant topological structure can then be loaded into the computer. A set of dedicated tools, gathered in the AMAPmod software, enable the user to access these virtual plants and to explore them. They provide users with a methodology and corresponding tools to measure plants, create plant databases, analyse information extracted from these databases. This methodology can be depicted as follows (Figure 17).

Multiscale representation of plant architectures are described from either field observations or plant growth simulation programs, using a dedicated en-

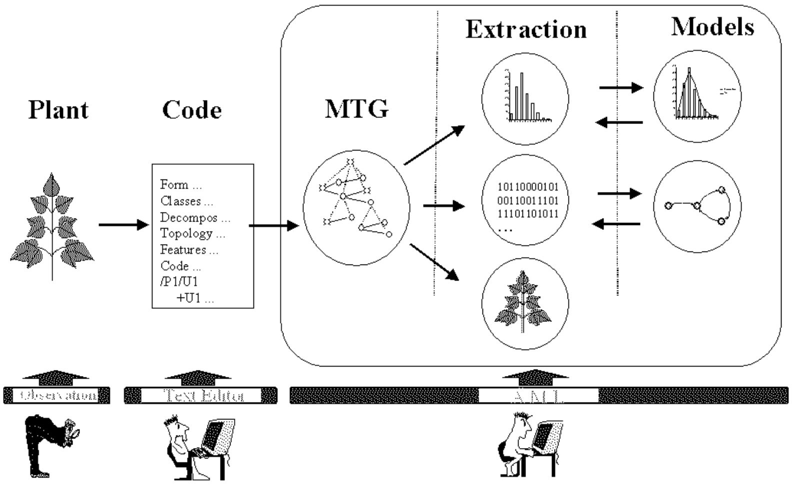

Figure 17: Synopsis of the AMAPmod system.

coding language. The resulting database can then be analysed with various statistical analysis tools (e.g. [GBCC01]). Plants can be graphically reconstructed at different scales and vizualised in 3 dimensions. Various types of data can be extracted and analysed with different viewpoints. Different families of probabilistic or stochastic models are provided in the system. These models are intended to be used as advanced statistical analysis tools for exploring in greater depth the information contained in the database. All these tools are available through a querying language called AML (AMAPmod Modelling Language) which enables the user to work on various objects, i.e. multiscale representation of plants, samples of data or models. AML provides the user with a homogeneous language-based interface to load, display, save, analyse or transform each type of object.

References

[Ada00] D. G. Adams. Heterocyst formation in cyanobacteria. *Current Opinoin in Microbiology*, 3:618–624, 2000.

[Bar91] D. Barthlmy. Levels of organization and repetition phenomena in seed plants. *Acta Biotheoretica*, 39:309–323, 1991.

[BB90] G. Berry and G. Boudol. The chemical abstract machine. In *Conf. Record 17th ACM Symp. on Principles of Programmming Languages, POPL'90, San Francisco, CA, USA, 17–19 Jan. 1990*, pages 81–94. ACM Press, New York, 1990.

[BCM87] J. P. Banatre, A. Coutant, and Daniel Le Metayer. Parallel machines for multiset transformation and their programming style. Technical Report RR-0759, Inria, 1987.

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

[BH70] R. Baker and G. T. Herman. CELIA — a cellular linear iterative array simulator. In *Proceedings of the Fourth Conference on Applications of Simulation* (9–11 December 1970), pages 64–73, 1970.

[BH72] R. Baker and G. T. Herman. Simulation of organisms using a developmental model, parts I and II. *International Journal of Bio-Medical Computing*, 3:201–215 and 251–267, 1972.

[BH00] Ronald Brown and Anne Heyworth. Using rewriting systems to compute left kan extensions and induced actions of categories. *Journal of Symbolic Computation*, 29(1):5–31, January 2000.

[BM86] J. P. Banatre and Daniel Le Metayer. A new computational model and its discipline of programming. Technical Report RR-0566, Inria, 1986.

[Bug00] A. Bugrim. A logic-based approach for computational analysis of spatially distributed biochemical networks. In *ISMB 2000*, San Diego California, August 2000.

[Cho56] N. Chomsky. Three models for the description of language. *IRE Trans. on Information Theory*, 2(3):113–124, 1956.

[Cho57] N. Chomsky, editor. *Syntactic structures.* Mouton & Co., The Hague, 1957.

[CL95] E. Costes and P. L. Lauri. Processus de croissance en relation avec la ramification sylleptique et la floraison chez pommier. In J. Bouchon, editor, *Architecture des Arbres Fruitiers et Forestiers*, volume 74, pages 41–50, Montpellier, France, 1995. INRA Editions.

[Dit00] P. Dittrich. Artificial chemistry page, 2000. `http://ls11-www.cs.uni-dortmund.de/achem`.

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

[dL87] C. G. de Koster and A. Lindenmayer. Discrete and continuous models for heterocyst differentiation in growing filaments of blue-green bacteria. *Acta Biotheoretica*, 36:249–273, 1987.

[DZB00] P. Dittrich, Jens Ziegler, and Wolfgang Banzhaf. Artificial chemistries - a review. *Artificial Life*, 2000. (to be submitted, available from the authors).

[Ede58] M. Eden. In H. P. Yockey, editor, *Symposium on Information Theory in Biology*, page 359, New York, 1958. Pergamon Press.

[ela02] Elan home page, 2002. `http://www.loria.fr/equipes/protheo/SOFTWARES/ELAN/`.

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

[GBCC01] Y. Gudon, D. Barthlmy, Y. Caraglio, and E. Costes. Pattern analysis in branching and axillary flowering sequences. *Journal of Theoretical Biology*, 212:481–520, 2001.

[GC98] C. Godin and Y. Caraglio. A multiscale model of plant topological structures. *Journal of Theoretical Biology*, 191:1–46, 1998.

[GGC99] C. Godin, Y. Gudon, and E. Costes. Exploration of plant architecture databases with the AMAPmod software illustrated on an apple-tree hybrid family. *Agronomie*, 19(03-avr):163–184, 1999.

[GM01a] J.-L. Giavitto and O. Michel. Declarative definition of group indexed data structures and approximation of their domains. In *Proceedings of the 3nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-01)*. ACM Press, September 2001.

[GM01b] J.-L. Giavitto and O. Michel. MGS: A programming language for the transformation of topological collections. Research Report 61-2001, CNRS - Université d'Evry Val d'Esonne, Evry, France, 2001.

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

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

[GV01] J.-L. Giavitto and E. Valencia. *Diagrammatic Representation and Reasonning*, chapter A Topological Framework for Modeling Diagrammatic Reasoning Tasks. Springer-Verlag, 2001.

[Han92] J. S. Hanan. *Parametric L-systems and their application to the modelling and visualization of plants.* PhD thesis, University of Regina, June 1992.

[HP96] M. Hammel and P. Prusinkiewicz. Visualization of developmental processes by extrusion in space-time. In *Proceedings of Graphics Interface '96*, pages 246–258, 1996.

[HR75] G. T. Herman and G. Rozenberg. *Developmental systems and languages.* North-Holland, Amsterdam, 1975.

[HRW86] J. L. Harper, B. R. Rosen, and J. White. *The growth and form of modular organisms.* The Royal Society, "London, UK", 1986.

[Jef85] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[JTN00] K. Chen J.J. Tyson, M.T. Borisuk and B. Novak. *Computational Modeling of Genetic and Biochemical Networks*, chapter Analysis of Complex Dynamics in Cell Cycle Regulation, pages 287–306. MIT Press, 2000.

[Kan00] Minoru Kanehisa. *Post-genome informatics.* Oxford University Press, 2000. ISBN 0-19-850326-1.

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

[Kel95] Evelyn Fox Kelle. *Refiguring Life: Metaphors of Twentieth-century Biology.* Columbia University Press, 1995.

[Kre86] W. Kreutzer. *System simulation: Programming styles and languages.* Addison-Wesley, Sydney, 1986.

[LIL89] C. Langton, L. In, and C. Langton. Artificial life, 1989.

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

[Lin71] A. Lindenmayer. Developmental systems without cellular interaction, their languages and grammars. *Journal of Theoretical Biology*, 30:455–484, 1971.

[Lin74] A. Lindenmayer. Adding continuous components to L-systems. In G. Rozenberg and A. Salomaa, editors, *L Systems*, Lecture Notes in Computer Science 15, pages 53–68. Springer-Verlag, Berlin, 1974.

[LP02] Brendav Lane and Przemek Prusinkiewicz. Specifying spatial distributions for multilevel models of plant communities. In *proc. of Graphics Interface 2002*, 2002.

[Lyn96] N. A. Lynch. *Distributed algorithms.* Morgan Kauffman, Los Altos, CA, 1996.

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

[mau02] Maude home page, 2002. `http://maude.csl.sri.com/`.

[May75] R. M. May. Biological population models obeying difference equations: Stable points, stalbe cycles, and chaos. *Journal of Theoretical Biology*, 51:511–524, 1975.

[May76] R. M. May. Simple mathematical models with very complicated dynamics. *Nature*, 261:459–467, 1976.

[Mei82] H. Meinhardt. *Models of biological pattern formation.* Academic Press, New York, 1982.

[Mic96] O. Michel. *Reprsentations dynamiques de l'espace dans un langage dclaratif de simulation.* PhD thesis, Universit de Paris-Sud, centre d'Orsay, December 1996. N°4596, (in french).

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

[Pau98a] Gheorge Paun, editor. *Computing with Bio-Molecules: Theory and Experiments.* Springer, 1998.

[Pau98b] Gheorghe Paun. Computing with membranes. Technical Report TUCS-TR-208, TUCS - Turku Centre for Computer Science, November 11 1998.

[Pau00] G. Paun. From cells to computers: Computing with membranes (p systems). In *Workshop on Grammar Systems*, Bad Ischl, austria, July 2000.

[PH90] P. Prusinkiewicz and J. Hanan. Visualization of botanical structures and processes using parametric L-systems. In D. Thalmann, editor, *Scientific visualization and graphics simulation*, pages 183–201. J. Wiley & Sons, Chichester, 1990.

[PJS92] H.-O. Peitgen, H. J urgens, and D. Saupe, editors. *Chaos and fractals. New frontiers of science.* Springer-Verlag, New York, 1992.

[PL90] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants.* Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[Pru98] P. Prusinkiewicz. Modeling of spatial structure and development of plants: a review. *Scientia Horticulturae*, 74:113–149, 1998.

[Pru99] P. Prusinkiewicz. A look at the visual modeling of plants using L-systems. *Agronomie*, 19:211–224, 1999.

[San92] E. Sanoja. *Essai d'application de l'architecture vgtale la systmatique. L'exemple de la famille des Vochysiaceae.* PhD thesis, USTL Montpellier France, 1992.

[Smi99] John Maynard Smith. *Shaping Life: Genes, Embryos and Evolution.* Yale University Press, 1999.

[Ste88] Isabelle Stengers. *D'une science l'autre. Les concepts nomades.* Le Seuil, 1988.

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

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

[WMS73] M. Wilcox, G. J. Mitchison, and R. J. Smith. Pattern formation in the blue-green alga, *Anabaena.* I. Basic mechanisms. *Journal of Cell Science*, 12:707–723, 1973.