Spatial Organization of the Chemical Paradigm and the Specification of Autonomic Systems

Jean-Louis Giavitto[1], Olivier Michel[1,2], and Antoine Spicher[2,3]

[1] IBISC FRE 3190 CNRS, Université d'Evry, Genopole,
523 place des Terrasses de l'Agora, 91000 Evry, France
{giavitto,michel}@ibisc.univ-evry.fr
http://mgs.ibisc.univ-evry.fr
[2] LACL EA 4213 Université Paris 12 (Paris Est),
61 Av. du Géneral de Gaulle, 94010, Créteil, France
{michel,spicher}@univ-paris12.fr
[3] LORIA UMR 7503 INRIA, CNRS, INPL, UHP, Nancy 2,
Campus Scientifique - BP 239, 54506 Vandoeuvre-lès-Nancy Cedex, France
Antoine.Spicher@loria.fr

Abstract. The *chemical paradigm* is an unconventional programming paradigm well fitted to the *high-level specification of parallel systems*. Based on the fixed point iterations of local rules, its use has been advocated for the programming of *autonomic* and *amorphous systems*. However, this model lacks an explicit handling of *spatial relationships*.

In this contribution, we first show how the chemical paradigm can be extended beyond multisets to arbitrary *topological collections*. Topological collections handle in a uniform way sophisticated data structures required in algorithmics as well as distributed data structures needed for the programming of autonomic or amorphous systems. Then we adapt a well-known result on multiset ordering to the more general case of topological collections. Well-founded ordering on topological collection can be used to prove the termination of the fixed point iteration of local rules.

1 Introduction

1.1 Gamma and the Chemical Paradigm

Introduced by the Gamma language, the chemical reaction metaphor [BCM88] describes computations in terms of reactions between molecules representing data, in a chemical solution represented as a multiset. A multiset is a generalization of a set that allows several occurrences of the same element. Computation proceeds by rewriting elements of a multiset according to conditions and transformation rules. The result of a chemical program is obtained when a stable solution is reached, i.e. when no reaction can take place anymore. For example, the reaction

$$\mathit{convex\_hull} = \textbf{replace } x, y, z, u \textbf{ by } x, y, z \textbf{ if } \mathrm{inside}(u; x, y, z)$$

M. Wirsing et al. (Eds.): Software-Intensive Systems, LNCS 5380, pp. 235–254, 2008.

replaces four points x, y, z and u by the first three points (i.e. u is removed) if u is inside the triangle x, y, z [BL90]. These replacements are repeated until a stable state is reached, that is to say, when no quadruple (x, y, z, u) can be found. The final stable solution contains exactly the elements defining the convex hull of the multiset of points specified in the initial solution.

1.2 Gamma and the Autonomic Computing Challenge

The goal of *Autonomic Computing* [Hor01] is to realize self-managing computer and software relying on properties of:

- *self-organization*: autonomous configuration of the components into a dynamic architecture dedicated to the satisfaction of the defined requirements;
- *self-healing*: autonomous detection and correction of hardware and software faults; and
- *self-optimization*: autonomous monitoring, control of resources and reconfiguration to ensure an optimal functioning.

The chemical paradigm has been claimed well suited to express autonomic properties: the reaction rules correspond to the local actions to be taken to react to a perturbation. Several convincing examples have been developed [BRF04].

We believe that the relevance of the chemical paradigm for the specification and the high-level programming of large autonomic and parallel/distributed systems comes from two fundamental characteristics:

1. the multiset data structure and the multiset rewriting device suitably represent the orderless interactions (reactions) between elements that occur in large parallel or open systems;
2. the computation of a stable state such that self-$*$ behaviors can been seen as the stabilization of the system on a fixed point after a transient perturbation.

However, these two general statements must be refined:

- The direct interactions of arbitrary elements in a system are not always allowed nor desirable. The system may exhibit some data organization and only "neighbor" elements may interact. The neighborhood relationship may represent physical (spatial distribution, localization of the resources) or logical constraints (inherent to the problem to be solved).
- A multiset stable w.r.t. the reactions represents a solution computed by the program or an admissible state of an autonomic system. This state is best characterized by global properties (e.g. the extremal points in a multiset of points in the convex hull computation) while the reactions represent local changes (e.g. the removal of one point fulfilling some conditions). Therefore, the real difficulty of chemical programming lies in the relation between the local changes and the desired global property.

In this paper, we present some concepts and tools in the field of algebraic topology that can be used to build more structured chemical solutions (section 2). For

the second point, we adapt a well-known result on multiset ordering that can be used to establish the convergence of local iterations of reactions (section 3). This result is a first step in the development of a toolbox of theoretical tools that can be used to link the local changes of elements to the global behavior of a system.

2 Introducing Space in the Chemical Paradigm

2.1 From Multisets to Sequences and Beyond

Multisets are a "loose organization" where more structured data require some encoding to be represented. For example, a sequence of elements can be encoded into a multiset M of pairs $[i, x]$ where i is the index of the value x. With this encoding, the reaction

$$sort\_bag = \textbf{replace } [i,x],[j,y] \textbf{ by } [i,y],[j,x] \textbf{ if } (j = i+1) \wedge (x > y)$$

replaces a couple of consecutive out-of-order pairs by the couple of consecutive ordered ones. These replacements go on until a stable state is reached, that is to say, when no orderless couple remains. Thus, the final stable solution corresponds to the sorting of the sequence encoded in M.

A more straightforward approach is desirable and possible. A multiset of values in V can be formalized as an element of the free associative and commutative monoid $(V^*, +)$ where $+$ is the operation that merges two multisets. Then, a multiset is a formal sum and a reaction rule is a rewriting rule on a term in $(V^*, +)$ modulo associativity and commutativity [DJ90]. In this framework, the comma between multiset elements in the pattern of the rule[1] is another notation for the $+$ operator.

From this point of view, it is easy to adapt the chemical paradigm to handle sequences: a sequence is an element of a monoid which is only associative. We can use term rewriting modulo associativity to formalize reaction rules on sequences. Thus, reaction:

$$sort\_sequence = \textbf{replace } x, y \textbf{ by } y, x \textbf{ if } x > y$$

applied on a sequence S directly corresponds to a kind of bubble sort. In this rule, the comma in the pattern represents the associative operator of the monoid and is interpreted as the concatenation of sequences. The rule is at the same time more readable because there is no artificial encoding of the sequence data structure into a multiset of pairs, and potentially more efficient because only consecutive elements are matched.

The path followed to extend the chemical paradigm on sequence cannot be easily generalized: rewriting modulo some theory is usually hard and needs *ad hoc* developments. For instance, at this point there is no satisfactory theory

[1] The pattern of the rule is the term between **replace** and **by**.

for rewriting on arrays. However, an alternative framework, focusing on the topological relationships in the data structure, can be developed. This framework encompasses multisets and sequences to include arbitrary data structures.

A Topological Approach. The idea is to consider the comma that appears in the pattern of a rule, not as a *data structure constructor*, but as a *neighborhood relationship* that depends on the data structure on which the rule is applied [GM02a]. In a multiset, all elements are neighbor, which accounts for the associativity and commutativity that enables arbitrary rearrangements of the term that represents the multiset. All other data organizations arise as a restriction of this "universal neighborhood relationship". For instance, in a sequence, the neighborhood relationships are restricted to a linear graph.

By considering various topologies, one may recover well-known computation models: nested multisets correspond to membrane systems [Pău02], constraining the universal topology provided by multisets to nested sequences leads to Lindenmayer systems [RS92]; restriction to discrete lattice corresponds to cellular automata and more general crystalline computations [TM87]. And topologies with higher dimensions can be used to give a direct finite formulation of field equations in classical physical theories [Ton01] with obvious interests for numerical applications [PS93].

2.2 A Short MGS Presentation

The MGS project [GM02b, Gia03] is based on the previous idea: data structures are defined relying on topological notions to specify their neighborhood relationships. In the rest of this section, we show how the notion of data structure can be identified with the notion of field on some underlying space. Such objects can be rewritten, leading to a novel form of case-based definition of function. These notions are illustrated through some simple but informative examples.

Data Structures as Discrete Fields. In MGS, a data structure is handled as a *field* that associates a scalar value to each point of an *underlying space* [GS08]. The structure of this underlying space is of interest for the computation at hand. For example, a multiset of n elements is a field over a space composed of n points. More specifically, the underlying space may represent some meaningful entity for the problem. For instance, in a simulation of the temperature distribution throughout a room, the underlying space is the room. In a distributed computation, the underlying space represents the connectivity between the processing elements.

The spatial structure of the underlying space is used to record the neighborhood relationships needed by the computation. If the computation of the value v associated with a point σ requires the value v' associated with an other point σ', then σ and σ' must be, somehow, neighbors in the underlying space. For example, in a simply linked list, the elements are linearly accessed: the second after the first, the third after the second, etc., inducing an oriented linear space.

In the context of a programming language, the topology of the underlying space must be algebraically defined to avoid the handling of untractable continuous objects. For technical reasons, it is more convenient and more general to associate values with some subspaces of the underlying space rather than with points only (a point being an elementary subspace). Moreover, in this paper we are only interested in the topological properties of the underlying space: the properties related to a metric will not be considered here.

These constraints can be satisfied using *abstract cellular complexes* to specify the underlying space. Abstract cellular complexes are a variant of cellular complexes developed in algebraic topology [Hen94]: a particular class of topological spaces that are partitioned into pieces of elementary space called *topological cells*. Each cell is homeomorphic to an open ball in $\mathbb{R}^d$. By the term *abstract*, we mean here that only the combinatorial structure of cellular complexes is preserved while the geometric characteristic functions mapping cells to open balls are left apart [Kov01].

The corresponding notion of data structure is called *topological collection* in MGS. Topological collections are formalized by *topological chains*, a notion developed in homology theory [Mun84]. Chains are functions that associate values with the cells of abstract cellular complexes. In the following, we will often drop the term "abstract": we only consider abstract cellular complexes and abstract topological cells.

Transformations of Topological Collection. In the chemical paradigm, multiset transformations are defined using rules and can be formalized by associative-commutative rewriting [DJ90]. This schema can be extended to topological collections, relying on the notion of *chain rewriting* defined in [GS08].

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 that specifies the changes of a subcollection. An MGS transformation T is accordingly specified by a set of rules:

$$\textbf{trans}\ T = \{\ \ldots\ ;\quad \mathit{rule};\quad \ldots\ \}$$

A rule is a basic transformation that takes the following form:

$$\mathit{pattern}\ \Rightarrow\ \mathit{expression}$$

where *pattern* in the left hand side (lhs) of the rule matches a subcollection B of the collection A on which the transformation is applied. The subcollection B is substituted in A by the collection C computed by the evaluation of the right hand side (rhs) *expression* of the rule.

The Pattern Language. Several pattern languages have been developed in MGS. In this paper, we only consider a subset of the *path patterns*. The grammar of this fragment of pattern expressions is:

$$\beta ::=\ x\ \ |\ \ p, p'\ \ |\ \ p*\ \ |\ \ p\ \textbf{as}\ x\ \ |\ \ p/exp$$

where p, p' are patterns, x ranges over the pattern variables and exp is an expression evaluating to a boolean value. Such patterns can be used to match a *path*: a finite sequence of elements e_i where e_{i+1} is a neighbor of e_i. The explanations below give an informal semantics for these patterns.

variable: a pattern variable x matches exactly one element of the topological collection, that is, a cell σ and its associated value v. A variable can only be defined once in a pattern (patterns are linear) but it may be used elsewhere in the expressions of the rule where it denotes the value v.

neighbor: p, p' is a pattern that matches two connected paths p and p'. The connection relationship depends on the topology of the collection. For example, x, y matches two elements such that y is a neighbor of x.

repetition: pattern $p*$ matches a subcollection of connected elements matched by p.

binding: a binding p **as** x gives the name x to the path matched by p. This name can be used anywhere in the rest of the rule. E.g., the pattern (x, y) **as** d matches two connected elements and the corresponding sequence of two elements can be referred through the variable d.

guard: p/exp matches the collections matched by p such that exp holds. For instance, $y \,/\, y > 3$ matches an element y whose associated value is greater than 3.

The Right Hand Side of a Rule. In a rewriting rule, the lhs and the rhs of the rule denote objects of the same type. For instance, in multiset rewriting, the lhs matches a multiset which is replaced by the multiset computed in the rhs. In term rewriting, the lhs matches a tree which is replaced by the tree computed by the rhs. In graph rewriting, the lhs of a rule matches a graph and the rhs computes a graph to be inserted in place of the matched one. Etc.

Sophisticated data structure make harder the definition of the the rhs and of the associated replacement operation. However, the structure of a path pattern can be used to drastically simplify the rhs of an MGS rule. A *path pattern* matches a path, that is, a sequence. Therefore, the rhs of the rule can evaluate to a sequence, no matter how complex the considered data structure is. The substitution of the matched path by the sequence computed in the rhs is done element-wise. Having a matched path and a rhs sequence of the same length is a constraint that can be relaxed for some collection types. For example, if a transformation is applied on a monoidal collection (i.e. a set, a multiset or a sequence), the rhs can be of arbitrary length. From now on, the expression in the rhs of a rule must be interpreted as a sequence construction.

2.3 The MGS Programming Language

MGS is an experimental programming language that embeds the idea of topological collections and their transformations into the framework of a functional language. Collections are just new kinds of values and transformations are functions acting on collections and defined by a specific syntax using rules. The set

of values has a rich type structure used in the definition of pattern-matching, rule and transformations. The collection types in MGS range from totally unstructured with sets and multisets to more structured with sequences, trees, Voronoï diagrams, Cayley graphs, arbitrary graphs, Generalized Maps [Lie94]... and abstract cellular complexes which subsume all other collection types.

A transformation T is a function on collections and a *first-class* value. For instance, a transformation can be passed as an argument to another function or be returned as a result. This feature allows to sequence and compose transformations very easily.

The expression $T(c)$ denotes the application of one transformation step to the collection c. As said above, a transformation step consists in the application of the rules (modulo the rule application's strategy). A transformation step can be easily iterated:

$$T[n]\,(c) \text{ denotes } n \text{ iterations of the application of } T \text{ on } c$$
$$T[\textbf{fixpoint}]\,(c) \text{ denotes the application of } T \text{ until a fixpoint is reached}$$

Several rule application strategies and transformation application strategies have been defined, including asynchronous and stochastic ones [SMC+08]. Synchronous rule application strategies (several rules are applied in parallel in one application of a transformation) are non-deterministic: only non-intersecting paths are rewritten and these paths are non-deterministically chosen (priorities or probabilities can be used to have a finer control). For example, the *maximal parallel application strategy* apply as many rules as possible in parallel that is, when rewriting occurs, there is no path in the remaining elements that can be matched by the lhs of a rule. This strategy is similar to the one used in Lindenmayer systems [RS92].

2.4 A Few Examples

Obviously, all Gamma chemical programs can be translated easily in MGS. We give below some examples that take advantage of the spatial structure of the topological collection. After two simple examples on sequences, we focus on computations on a regular grid and on a graph because such data structures are not algebraic data structures, and therefore the usual approach based on term rewriting is not applicable.

MGS has been involved in sophisticated simulation applications in biology, like neurulation [SM07], cell mobility [SM05], growth of plant meristem at a cellular level [BdR+06], or simulations at various scales of a synthetic multicellular bacterium [IGE07]. Classical algorithms (various sorting procedures, graph algorithmics, optimization processes, mesh refinement algorithms, etc.) have also been easily developed, see [Mic07].

Bubble Sort. In the MGS syntax, the *sort_sequence* program in section 2.1 is specified by the following transformation:

$$\textbf{trans}\ \mathit{sort\_sequence} = \{\ x, y\ /\ x{>}y\ \Rightarrow\ y, x\ \}$$

As mentioned above, the rhs of the rule is a sequence construction: the comma is then interpreted as the sequence constructor. The transformation must be iterated until a fixed point is reached. Note that the fixed point is reached regardless the rule application strategy. This result can be established by the tools presented in section 3.

Duplicate Removal. It is easy to remove the contiguous duplicated elements in a list using the iteration of the transformation:

$$\textbf{trans}\ \mathit{remove\_duplicate} = \{\ x, y\ /\ x{=}y\ \Rightarrow\ x\ \}$$

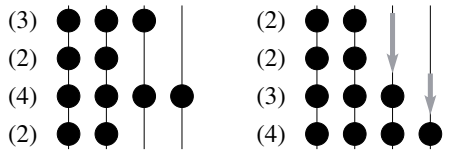
The pattern $x, y\ /\ x{=}y$ selects two contiguous elements labeled by the same value. Such occurrences are replaced by only one element. Note that the rhs must be interpreted as a sequence of only one element. The conversion between an element and the corresponding singleton is implicit.

Bead Sort. Various meshes can be described in MGS. For example, the NEWS lattice is specified by the following type declaration:

$$\textbf{gbf}\ \mathit{NEWS} = \langle \mathit{North}, \mathit{East}, \mathit{West}, \mathit{South}; \mathit{North} + \mathit{South} = 0, \mathit{East} + \mathit{West} = 0\rangle$$

This declaration introduces a "group based data-field" or GBF [GM01]. The underlying space of a GBF is the Cayley graph of the abelian group presentation specified by the right hand side of the declaration. The vertices of the Cayley graph are linked by edges labeled by the group generators *North*, *East*, etc.

The bead-sort is an original way to sort positive integers proposed by [ACD02]. This sorting algorithm considers a column of numbers written in unary basis. The first schema below pictures the numbers 3, 2, 4 and 2 where the beads stand for the digits. The sorting is done by letting the beads fall down as shown on the second schema. The numbers can be implemented by a regular grid of booleans where *true* stands for a digit and *false* for the absence of digit as shown on the third and fourth schema.



The bead-sort is achieved by iterating the application of the following transformation until a fixpoint is reached:

$$\textbf{trans}\ \mathit{bead\_sort} = \{x/(x = \mathit{false})\quad |\mathit{North}{>}\quad y/(y = \mathit{true})\ \Rightarrow\ y, x\}$$

The construction $|\mathit{North}{>}$ refines the comma operator, constraining the element y to be a *North*-neighbor of x.

Hamiltonian Path. A graph is an MGS topological collection. Expressing in MGS the search of an Hamiltonian path in a graph is straightforward:

```
exception Not_Found, Found;;
trans H = { x* as path/size(path) = size(self) ⇒ raise Found(path) };;
fun hamiltonian(g) = try
                       H(g); raise Not_Found
                     with Found(path) → path;;
```

Transformation H uses an iterated pattern $x*$ that matches a path. The keyword **self** refers to the collection the transformation is applied on. The size of a graph returns the number of its vertices. So, if the length of *path* is the same as the number of vertices in the graph, then *path* is an Hamiltonian path (patterns are linear without repeated matched element). The rhs raises an exception which is trapped in function *hamiltonian*. The normal return of H is followed by the raising of the *Not_Found* exception in function *hamiltonian*.

3 From Local Changes to Global Specifications

The notions introduced in MGS for spatial organization can be used to handle directly: (a) highly organized data structures used in algorithms (like trees, arrays, etc.), (b) semi-structured data like those managed in XML applications (XML schema are well represented by nested topological collections) or (c) to take into account the data distribution over a network. The network architecture can be static and regular (e.g., as in a tightly coupled parallel architecture) or more fuzzy and dynamic (e.g., as in a grid on the Internet, in a P2P system or in an amorphous medium [AAC+00]). In either case, the communication cost between processing elements induces a neighborhood relationship. The states of the processing elements together with this neighborhood relationship constitute a topological collection. If any point-to-point communication exhibits the same uniform cost, the corresponding topology is the topology of a multiset.

3.1 Self-* Properties and Fixed Point Iterations of Local Rules

In this point of view, the state of an autonomic system is a (distributed) topological collection. The evolution of an autonomic system is specified through local evolution rules that define the (local) evolution of a small subpart of the system. A topological collection stable w.r.t. the reactions represents an admissible state of the autonomic system. A perturbation or an interaction with the environment corresponds to a change in the topological collection: the addition or the removal of some elements *or* a modification of the neighborhood relationship. This framework is illustrated in Fig. 1.

Thus, an autonomic system can be defined in this framework if one can infer that the asynchronous parallel applications of local rules lead to stable points exhibiting some required properties. Some theoretical tools can be used in this

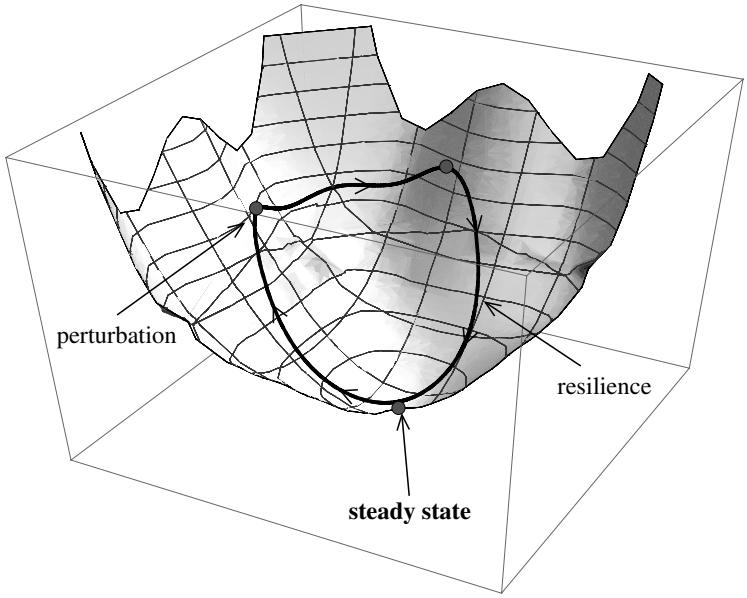


Fig. 1. Autonomic computing via trajectory stabilization. An autonomic system can be seen as a distributed dynamical system. In this diagram, the states of the dynamical system are figured as a plane and the system's evolutions are given by a trajectory. The surface represents some potential, for instance a quantitative evaluation of the divergence of the system from a desired behavior. When some transient perturbations make the system leave its steady state, the local transformations triggered by the matching of some rules eventually lead to the return of the system's state to an admissible state.

difficult task. In the rest of this presentation, we develop several topological collection orderings based on *multiset orderings* [DM79]. Such orderings can be used to prove the convergence of an autonomic system towards a fixed point.

The framework pictured in Fig. 1 can be made more precise in the following way. We suppose that the state of an autonomic system is described by a topological collection $c \in \mathcal{C}$ where $\mathcal{C}$ is the state space of the system. The autonomic program driving the autonomic system is specified by a unique transformation T. The admissible states s of the corresponding autonomic system are fixed points of T: $s = T(s)$. A perturbation of the system corresponds to a state s' such that $s' \neq T(s')$. The trajectory of the system after the perturbation is given by the sequence $s_0 = s', s_1 = T(s_0), \ldots, s_{n+1} = T(s_n)$. The problem is then to know whether the sequence s_n converges towards a fixed point in a finite number of steps. A classical approach is to exhibit a well-founded ordering [2] $\prec$ of the state space $\mathcal{C}$ such that $s_{n+1} \prec s_n$.

[2] An ordering is well-founded if it contains no infinite descending sequences of elements. For countable sets, a well-founded ordering $\prec$ can be specified by giving a mapping τ onto the positive integers such that $c \prec c' \Leftrightarrow \tau(c) < \tau(c')$. The "potential" surface in Fig. 1 figures such a function τ.

In this context, it would be very useful to exhibit various well-founded orderings on arbitrary topological collections. To this end, we can adapt the well-known multiset ordering introduced in [DM79].

3.2 Multiset Ordering

For a partially-ordered set of values $(V, <)$, the multiset ordering $\ll$ on $\mathcal{M}(V)$ the multisets over V, is defined as follows: $A \ll B$ if for some $X, Y \in \mathcal{M}(V)$ such that $X \neq \emptyset$ and $X \subset B$,

$$A = (B - X) \cup Y \qquad \text{and} \qquad \forall y \in Y, \exists x \in X, y < x$$

In other words, A is obtained from B by removing some elements (those in X) and their replacement with a finite number of elements (those in Y) that are smaller than one of the element of X. The definition of the relation $\ll$ relies on the relation $<$ and we will write $\ll_<$ when we need to make such dependence explicit.

In the previous definition, set operators denote their multiset analogs: the equality $A = B$ of two multisets, for example, means that any element occurring exactly n times in A, also occurs exactly n times in B. The union of two multisets $A + B$ is a multiset containing $m + n$ occurrences of any element occurring m times in A and n times in B. $A \subset B$ means that for any element occurring n times in A, this element occurs m times in B with $n < m$. If $A \subset B$, then $B - A$ is a multiset where any element occurring n times in A and m times in B, occurs $m - n$ times. Union and difference between multisets are extended to the addition and the removal of elements: $A + x = A + \{x\}$ and $A - x = A - \{x\}$.

The result demonstrated in [DM79] is that $\ll_<$ is well-founded iff $<$ is well-founded. This result can be generalized in two ways to topological collections more general than multisets.

3.3 Forgetting the Spatial Structure

The first approach simply consists in forgetting the additional spatial structure of a topological collection. Thus, we can associate to each topological collection c the multiset $\mathbf{m}(c)$ of the values associated with the cells of c. Assuming that $\ll$ is well-founded, the order $\prec$ defined by $c \prec c'$ iff $\mathbf{m}(c) \ll \mathbf{m}(c')$, is also well-founded. We write $\prec_\ll$ when we want to make explicit the underlying multiset ordering.

A Toy Example. To illustrate the use of a well-founded topological collection ordering $\prec$, we consider a wireless sensor network targeted at environmental monitoring, like for example the one described in [SKHH06]. Each node reacts to environmental changes and, to fix the idea, the goal of the system is to record the maximal temperature over the covered area. For this purpose, each node i stores a current maximal temperature t_i and updates it by comparison with its neighbors. A perturbation is the change of one t_i to record a new local maximum.

The state of the autonomic system is a topological collection $c \in \mathcal{C}$ where $\mathcal{C}$ can be for instance the set of vertex-labeled undirected graphs (a vertex represents

a node, an edge between two vertices corresponds to nodes able to interact, and the label of the vertex i is the current maximal temperature t_i). We further suppose that there is only one strongly connected component. The behavior of the system is specified as the iteration of the following transformation:

$$\textbf{trans}\ \mathit{propagate} = \{\ t, t'/t' < t\ \Rightarrow\ t, t\ \}$$

(the local change of a t_i is considered as an external change and not modeled here).

It is straightforward to prove that in absence of a perturbation, the system will reach a state where all t_i are equal. This property is a global one and must be deduced from the application of the local rule. Although this result is obvious, its proof illustrates the use of a well-founded topological collection ordering.

Constant Fields are the Fixed Points. First, the topological collections c where each cell has the same value t, are the only fixed points of the *propagate* transformation. Indeed, if c is a fixed point, we cannot find in c a pair of cells σ and σ' labeled by the values t and t' such that $t < t'$ (the relation $<$ is the classical numerical comparison over the integers).

Exhibiting a Well-Founded Ordering. We may suppose that the temperature is bounded by a sufficiently big positive number $t_{\max}$. Thus the set $V = \{-\infty, \ldots, t_{\max}\}$ with the order $\lessdot$ defined by $(t \lessdot t') \Leftrightarrow (t' < t)$ is well-founded. Consequently, the order on $\mathcal{M}(V)$ defined by $\ll_{\lessdot}$ is well-founded, as well as $\prec_{\ll_{\lessdot}}$ on $\mathcal{C}$.

The System's Trajectory is Decreasing. As a matter of fact, if $c' = \mathit{propagate}(c)$ and $c' \neq c$, we have $c' \prec_{\ll_{\lessdot}} c$ because $\mathbf{m}(c') = \mathbf{m}(c) - t + t'$ with $t, t' \in \mathbf{m}(c)$ and $t' \lessdot t$, and so $\mathbf{m}(c') \ll_{\lessdot} \mathbf{m}(c)$. This just shows that the sequence $T^n(c)$ is decreasing and since the order is well-founded, it implies that it converges in a finite number of steps to some fixed point.

3.4 Topological Collection as Generalized Multisets

Forgetting the spatial structure is useful only if the fixed point does not depend on the spatial structure of the topological collection, which is generally not true.

It is possible at the same time to keep the spatial structure of a topological collection c and to consider it as a kind of generalized multiset. To see the connection between both notions, we need to introduce some formal definitions. The reader is warned that the following definitions are truncated to only focus on the multiset structure of a topological collection. The algebraic structure required to represent the spatial organization of a topological collection (dimension of a cell, boundary operator, the neighborhood relationships, etc.) is ignored. Complete definitions can be found in [GM02b, GS08].

Definition 1 (Cells, Abstract Cellular Complexes and Chains). *Let* S *be a set of symbols called the* universal set of cells. *An abstract cellular complex* K *is a partially ordered subset of* S. *We write* $\mathbb{K}$ *the set of all abstract cellular complexes.*

Let K *be a complex and* G *be an abelian group. The set of functions from* K *to* G*, null almost everywhere, is called the set of* topological chains *of* K *to* G*, and is written* $\mathcal{C}_\mathsf{K}(G)$*.*

The elements of $\mathcal{C}_\mathsf{K}(G)$ are easily representable by finite formal sums:

$$\forall c \in \mathcal{C}_\mathsf{K}(G), \quad c = \sum_{\sigma \in \mathsf{K}} c(\sigma).\sigma$$

where $c(\sigma).\sigma$ is the formal product that represents the association of the value $c(\sigma) \in G$ with the cell $\sigma \in \mathsf{S}$. Thanks to the abelian group structure imposed on G, the set $\mathcal{C}_\mathsf{K}(G)$ is an abelian group considering the addition $+_{\mathcal{C}_\mathsf{K}(G)}$ defined by:

$$\forall c_1, c_2 \in \mathcal{C}_\mathsf{K}(G), \quad c_1 +_{\mathcal{C}_\mathsf{K}(G)} c_2 = \sum_{\sigma \in \mathsf{K}} (c_1(\sigma) +_G c_2(\sigma)).\sigma$$

where $+_G$ denotes the group operation in the group G. The proof is straightforward. If the context is clear, the subscript in the notation of the group operation will be dropped.

While they seem useless in our context, the group structure on the set of values and the induced group structure on chains are really meaningful to deal with topological collections:

- $0_G.\sigma$ means that no value is associated with σ (0_G is the neutral element in the group G),
- the neutral element of $\mathcal{C}_\mathsf{K}(G)$ represents the empty data structure,
- the operator $+_{\mathcal{C}_\mathsf{K}(G)}$ adds a new association of a value $v \in G$ with a cell $\sigma \in \mathsf{K}$ in a data structure c: $c + v.\sigma$,
- the opposite $-_{\mathcal{C}_\mathsf{K}(G)}$ removes an association: $c - v.\sigma = c + (-v).\sigma$.

We have mentioned above that a topological collection c can be represented as a finite formal sum $\sum_{\sigma \in \mathsf{K}} c(\sigma).\sigma$. We interpret this sum as a set of pairs

$$\mathbf{mp}(c) = \{(\sigma, c(\sigma)) : \sigma \in \mathsf{S} \text{ and } c(\sigma) \neq 0_V\}$$

A set is a special kind of multiset and so can be ordered using a multiset ordering based on the ordering of the set elements. Hence the desired definition and theorem:

Theorem 1. *Let* $(V, <)$ *a partially-ordered set of values, and* $(\mathsf{S}, \lessdot)$ *a partially-ordered universal set of cells. Then the* topological collection ordering *is the partial-order* $\prec$ *defined on* $\mathcal{C}_\mathsf{K}(V)$ *by:* $c \prec c'$ *iff* $\mathbf{mp}(c) \prec_{(\lessdot \times <)} \mathbf{mp}(c')$ *where* $(\lessdot \times <)$ *is the lexicographic or the element-wise ordering on* $\mathsf{S} \times V$*. The relation* $\prec$ *is well-founded iff* $\lessdot$ *and* $<$ *are well-founded.*

In words, a collection c is smaller than a collection c' if c can be obtained from c' by replacing some valued cells by some other valued cells, in arbitrary number, provided that the introduced cells and/or the associated values are smaller.

3.5 Correction of the Eratosthenes's Sieve

To illustrate the previous result, we consider the fixed point iteration of the following transformation[3]:

$$
\begin{aligned}
&\textbf{trans } \mathit{prime} = \{\\
&\quad x/(x>0),\ 0 \ \Rightarrow\ 0,-x;\\
&\quad x/(x\geq 0),\ y/(y<0),\ 1/(-y>x) \ \Rightarrow\ x,-y,1;\\
&\quad x/(x\geq 0),\ y/(y<0),\ 1/(-y<x) \ \Rightarrow\ x,1;\\
&\quad x/(x\geq 0),\ y/(y<0),\ z/(z>0)/(-y>x)\wedge(-y<z) \ \Rightarrow\ x,-y,y,z;\\
&\quad x/(x<0),\ y/(y>0)\wedge(y\leq -x)\wedge(x\%y\neq 0) \ \Rightarrow\ y,x;\\
&\quad x/(x<0),\ y/(y>0)\wedge(y\leq -x)\wedge(x\%y= 0) \ \Rightarrow\ y;\\
&\quad x/(x<0),\ y/(y>0)\wedge(y> -x)\wedge(y\%x\neq 0) \ \Rightarrow\ y,x;\\
&\quad x/(x<0),\ y/(y>0)\wedge(y> -x)\wedge(y\%x= 0) \ \Rightarrow\ x;\\
&\}
\end{aligned}
$$

We assume that this process is applied on a sequence that begins with a zero and ends with a one. This transformation maintains a sorted sequence (sorted at the exception of the ending one) of relatively prime positive integers. An external perturbation consists in introducing an arbitrary integer $x > 0$ at the beginning of the sequence. If all the integers up to $m > 1$ are introduced, *in any order*, the sequence stabilizes on the sorted sequence of prime numbers up to m.

In the rules, the operator $\wedge$ is the boolean conjunction operator and % is the modulo operator. The fourth rule increases the length of the sequence while some other rules shrink it. So the convergence is not obvious. The idea behind this program is similar to the sieve of Eratosthenes but adapted in order to admit the orderless introduction of the integers. The perturbation x "travels" along the sequence, from the beginning to the end of the sequence. This "traveling number" is distinguished from the other numbers in the sequence using a negative number (note that there is no constraint in the perturbation, so several "traveling numbers" may coexist in the sequence). This number and the next are tested to check if they are relatively prime. The "traveling number" is inserted in the sequence at the correct position in order to maintain a sorted sequence and the propagation continues to check the primality of the rest of the sequence.

The correction of this algorithm is easy to establish with the previous tool. We focus on the convergence, that is: a fixed point is reached in a finite number of steps. It is sufficient to exhibit a well-founded order such that a rule application to sequence c_1 gives raise to a smaller sequence c_2. We take the topological collection order induced by the following order on cells and values:

- Cells are ordered in descending order "from left to right". For instance, the sequence $0, 2, 3, 1$ is represented by the sum $0.\sigma_0 + 2.\sigma_1 + 3.\sigma_2 + 1.\sigma_3$ and

[3] This transformation is derived from a natural implementation of the sieve of Eratosthenes, see [GM02b]. This program is not intended to be readable and has been chosen to illustrate the approach on an arbitrary set of rules and conditions.

$\sigma_3 \lessdot \sigma_2 \lessdot \sigma_1 \lessdot \sigma_0$. We assume that only a finite number of cells has been used, so $\lessdot$ is well-founded. This assumption is satisfied because a cell is created each time a new integer is introduced and we suppose that there is a bounded number of introductions.
- Relative integers are ordered as follows: let N be the greatest integer introduced in the sequence. Then the set of values $V = \{-N, \ldots, N\}$ used in the program is ordered by : $-N > -N+1 > \cdots > -1 > N > N-1 > \cdots > 1 > 0$. This order is well-founded. Note that V is a subset of the abelian group $(\mathbb{Z}, +)$ and this is enough for our purpose.

Finally, we consider the lexicographic order on the pairs (cell, value). It is straightforward to check that each rule applied to a sequence c_1 gives raise to a smaller sequence c_2. We will sketch only three rules for illustration:

- The first rule does not change the sequence length. Its application replaces in a sequence c_1 the submultiset $x.\sigma_0 + 0.\sigma_1$ by $0.\sigma_0 + (-x).\sigma_1$ where $\sigma_1 \lessdot \sigma_0$ and $x > 0$. We have $x.\sigma_0 > 0.\sigma_0$ (because $x > 0$) and $x.\sigma_0 > (-x).\sigma_1$ (because $\sigma_1 \lessdot \sigma_0$). It follows that $c_2 \prec c_1$.
- Rule 4 is the only rule that increases the sequence length. The submultiset:

$$c_1 = x.\sigma_0 + y.\sigma_1 + z.\sigma_2 \qquad \text{where } \sigma_2 \lessdot \sigma_1 \lessdot \sigma_0 \text{ and } 0 \leq x \text{ and } y < 0$$

  is replaced by

$$c_2 = x.\sigma_0 + (-y).\sigma_1 + y.\sigma_2 + z.\sigma_2' \qquad \text{where } \sigma_2' \lessdot \sigma_2$$

  We have $c_2 \prec c_1$ because $y.\sigma_1 > (-y).\sigma_1$ (y is strictly negative and so $-y > y > 0$). We have also $y.\sigma_1 > y.\sigma_2$ and $y.\sigma_1 > z.\sigma_2'$ because $\sigma_2' \lessdot \sigma_2 \lessdot \sigma_1$.
- The application of rule 8, the last rule of the transformation, decreases the sequence since it removes one element.

So, since the successive sequences are decreasing in a well-founded order the sequence must converge in a finite number of steps.

3.6 The Group Structure and the Ordering of the Values

We cannot assume that every set of values V we may consider, carries a group structure. The group operation is useful to manage the association of a value to a cell but it is in some sense external to the proper definition of V. Hopefully, in absence of any "natural" group operation on V, we can use a mathematical trick to turn any set V into an abelian group.

Definition 2 (Abelianization of a set). *Let V be an arbitrary set. The $\mathbb{Z}$-*module freely generated by the elements of V, *written $\mathcal{A}(V)$, is the free abelian group generated by the elements of V. An element g of $\mathcal{A}(V)$ can be written has a formal sum: $\sum_{v \in V} z_v.v$ where z_v belongs to $\mathbb{Z}$.*

In the following, we consider only "finite formal sum" where only a finite subset of coefficients z_v are different from zero. The structure of $\mathbb{Z}$-module generalizes the structure of multiset: as a matter of fact, any multiset $m \in \mathcal{M}(V)$ can be represented by a formal sum with positive coefficients z_v. So, $\mathcal{A}(V)$ generalizes $\mathcal{M}(V)$ by allowing a *negative* number of occurrences.

Given $g \in \mathcal{A}(V)$, we can distinguish between the positive and the negative coefficients of g, that is:

$$g = (\sum_{v \in V_1} z_v.v) - (\sum_{v' \in V_2} z_{v'}.v)$$

with $V_1, V_2 \subset V$. Assuming $V_1 \cap V_2 = \emptyset$ and all coefficients $z\_$ strictly positive, the decomposition of g is unique and we write: $g = g^+ - g^-$. Both sums g^+ and g^- are multisets on V and this justify the following definition of the abelian ordering.

Definition 3 (Abelian ordering). *Let $(V, <)$ be a partially-ordered set of values. The* abelian ordering $\ll$ *on $\mathcal{A}(V)$ is defined as follows: $g \ll g'$ iff $(g^+, g^-) < (g'^+, g'^-)$ where $<$ is the lexicographic ordering or the element-wise ordering and where the elements of the pair are compared using the multiset ordering on $\mathcal{M}(V)$.*

Theorem 2. *Let $(V, <)$ a well-founded partially-ordered set of values, and $(\mathsf{S}, \lessdot)$ a well-founded partially-ordered universal set of cells. Let $\prec$ be the* topological collection ordering *defined on $\mathcal{C}_{\mathsf{K}}(Abel(V))$ by: $c \prec c'$ iff $\mathbf{mp}(c) \prec_{(\lessdot \times \ll)} \mathbf{mp}(c')$ where $\ll$ is the abelian ordering induced by $<$ on $\mathcal{A}(V)$ and $(\lessdot \times \ll)$ is the lexicographic or the element-wise ordering on $\mathsf{S} \times \mathcal{A}(V)$. Then, the relation $\prec$ is well-founded.*

4 Conclusion

In this paper we have shown how the declarative chemical programming paradigm can be enhanced to take into account logical and physical spatial organization using some notions developed in algebraic topology. This new framework is investigated in the MGS experimental programming language. At the core of this extension is the idea that a data structure can be conceived as a physical field. This notion is not entirely new and we review below some previous work.

The chemical paradigm has been advocated for the development of amorphous and autonomic systems. In a second part, we have adapted the well-founded multiset ordering, a classical tool used to prove program termination, to show how the convergence of a fixed point iteration can be established for topological collections. The parallel between autonomic systems and self-stabilizing systems has been recently noticed and we sketch some differences.

Data Structure as Field. The notion of data field is an old one in computer science: it already appeared in the development of recurrence equations and

dates at least from [KMW67]. The term "data field" seems to be used for the first time in [CiCL91]. The notions of data field and data parallelism have been explicitly brought together in [Lis93]. This approach is also close to the notion of *pvar* or *xapping* [SH86] in the context of the *Connection Machine*. However, in all these works, the set of points is simply an integer lattice (points are elements of $\mathbb{Z}^n$) and is often left implicit.

Topological collections consider, for the underlying space, more general spaces than integer lattices or even arbitrary graphs, in order to accommodate a large variety of spatial organizations [GM02a]. This generality will ease the development of various applications, for example in simulation by allowing a direct representation of the modeled entities. As a matter of fact, many physical quantities have different values at various points in space (temperature field, velocity field, potential, etc.). In addition, the value associated with a spatial domain often depends on the dimension of the domain [Ton74], e.g. a concentration for a volume and a flux for the surface bounding this volume. Such generality will also facilitate portability by offering a uniform abstraction of arbitrary spatial computing media (e.g.: grids, amorphous computers, chemical reaction diffusion computers, DNA self-assembly, natural or synthetic cellular assemblies, etc.).

Self-Stabilization and the Self-* Paradigms. Recent works in the self-stabilization community [BDHY07, BDH$^+$08] advocate the use of self-stabilization as a provable property to achieve the goal of self-* paradigms for systems. Usually, a self-stabilizing system is designed to start in any possible configuration where processors, processes, communication devices, etc. are in an arbitrary state. The approach exemplified here follows the same line. However, it is more abstract (communications are abstracted by the data movements within a topological collection) and less demanding concerning the initial state. We want to underline that our goal is not to develop algorithmic tools for designing self-stabilizing systems but to show that well-known approaches in *program semantics* can be adapted to new programming paradigms advocated for amorphous and autonomic applications.

Future Work. Our future research directions follow two paths. First, we need to investigate further how classical tools in the semantics of programs can be adapted to the case of perpetual autonomic systems. The other direction tries to import some tools from dynamical system theory to design and study the semantics of autonomic systems. A first step in this direction is the development of a discrete analog of differential operators for topological collections [GS08]. Our idea is to rely on topological and geometrical results (fixed-point theorem, existence of objects defined by differential equations, integration theorem) to design, control and validate global behaviors from the specification of local ones.

Acknowledgments. Pascal Fradet at INRIALPES, Thierry Priol and Jean-Pierre Banâtre at IRISA are gratefully acknowledged for stimulating discussions and interactions on the chemical paradigm an its application to autonomic systems. The authors also wish to thank the organizers of the InterLink workshop series

for making these fertile workshops possible. The work presented here are partially funded by the ANR NanoProg, the ANR AutoChem, a BQR of the University of Evry and the CNRS.

References

[AAC+00] Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight, T.F., Nagpal, R., Rauch, E., Sussman, G.J., Weiss, R.: Amorphous computing. CACM: Communications of the ACM 43 (2000)

[ACD02] Arulanandham, J.J., Calude, C.S., Dinneen, M.J.: Bead-Sort: A natural sorting algorithm. EATCS Bull. 76, 153–162 (2002)

[BCM88] Banâtre, J.-P., Coutant, A., Le Metayer, D.: A parallel machine for multiset transformation and its programming style. Future Generation Computer Systems 4, 133–144 (1988)

[BDH+08] Brukman, O., Dolev, S., Haviv, Y., Lahiani, L., Kat, R., Schiller, E., Tzachar, N., Yagel, R.: Self-stabilization from theory to practice. Bulletin of the EATCS (94), 130–150 (2008)

[BDHY07] Brukman, O., Dolev, S., Haviv, Y., Yagel, R.: Self-stabilization as a foundation for autonomic computing. In: Proc. of the Second IEEE International Conference on Availability, Reliability and Security (ARES 2007), Workshop on Foundation of Fault-tolerance Distributed Computing (FOFDC 2007), pp. 991–998. IEEE, Los Alamitos (2007)

[BdR+06] Barbier de Reuille, P., Bohn-Courseau, I., Ljung, K., Morin, H., Carraro, N., Godin, C., Traas, J.: Computer simulations reveal properties of the cell-cell signaling network at the shoot apex in Arabidopsis. PNAS 103(5), 1627–1632 (2006)

[BL90] Banâtre, J.-P., Le Métayer, D.: The GAMMA model and its discipline of programming. Science of Computer Programming 15(1), 55–77 (1990)

[BRF04] Banâtre, J.-P., Radenac, Y., Fradet, P.: Chemical specification of autonomic systems. In: IASSE, pp. 72–79. ISCA (2004)

[CiCL91] Chen, M., Il Choo, Y., Li, J.: Crystal: Theory and Pragmatics of Generating Efficient Parallel Code. In: Szymanski, B.K. (ed.) Parallel Functional Languages and Compilers. Frontier Series, vol. 7, pp. 255–308. ACM Press, New York (1991)

[DJ90] Dershowitz, N., Jouannaud, J.-P.: Rewrite systems. In: Handbook of Theoretical Computer Science, vol. B, pp. 244–320. Elsevier Science, Amsterdam (1990)

[DM79] Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Communications of the Association for Computing Machinery 22, 465–476 (1979)

[Gia03] Giavitto, J.-L.: Invited talk: Topological collections, transformations and their application to the modeling and the simulation of dynamical systems. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, Springer, Heidelberg (2003)

[GM01] Giavitto, J.-L., Michel, O.: 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 2001). ACM Press, New York (2001)

[GM02a] Giavitto, J.-L., Michel, O.: Data structure as topological spaces. In: Calude, C.S., Dinneen, M.J., Peper, F. (eds.) UMC 2002. LNCS, vol. 2509, pp. 137–150. Springer, Heidelberg (2002)

[GM02b] Giavitto, J.-L., Michel, O.: The topological structures of membrane computing. Fundamenta Informaticae 49, 107–129 (2002)

[GS08] Giavitto, J.-L., Spicher, A.: Topological rewriting and the geometrization of programming. Physica D (2008) (accepted for publication)

[Hen94] Henle, M.: A combinatorial introduction to topology. Dover publications, New-York (1994)

[Hor01] Horn, P.: 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

[IGE07] IGEM. Modeling a synthetic multicellular bacterium. Modeling page of the Paris team wiki at iGEM 2007 (2007), http://parts.mit.edu/igem07/index.php/Paris/Modeling

[KMW67] Karp, R.M., Miller, R.E., Winograd, S.: The organization of computations for uniform recurrence equations. Journal of the ACM 14(3), 563–590 (1967)

[Kov01] Kovalevsky, V.: Algorithms and data structures for computer topology. In: Digital and image geometry: advanced lectures, pp. 38–58. Springer, New York (2001)

[Lie94] Lienhardt, P.: N-dimensional generalized combinatorial maps and cellular quasi-manifolds. International Journal on Computational Geometry and Applications 4(3), 275–324 (1994)

[Lis93] Lisper, B.: On the relation between functional and data-parallel programming languages. In: Proc. of the 6th. Int. Conf. on Functional Languages and Computer Architectures. ACM Press, New York (1993)

[Mic07] Michel, O.: There's plenty of room for unconventional programming languages, or, declarative simulations of dynamical systems (with a dynamical structure). Habilitation Manuscript (December 2007), http://www.ibisc.univ-evry.fr/~michel/Hdr/hdr.pdf

[Mun84] Munkres, J.: Elements of Algebraic Topology. Addison-Wesley, Reading (1984)

[Pău02] Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)

[PS93] Palmer, R.S., Shapiro, V.: Chain models of physical behavior for engineering analysis and design. In: Research in Engineering Design, vol. 5, pp. 161–184. Springer International, Heidelberg (1993)

[RS92] Rozenberg, G., Salomaa, A. (eds.): Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology. Springer, Heidelberg (1992)

[SH86] Steele, G.L., Hillis, D.: Connection machine LISP: Fine grained parallel symbolic programming. In: Proceedings of the 1986 ACM Conference on LISP and Functional Programming, pp. 279–297. ACM, New York (1986)

[SKHH06] Suhonen, J., Kohvakka, M., Hännikäinen, M., Hämäläinen, T.D.: Design, implementation, and experiments on outdoor deployment of wireless sensor network for environmental monitoring. In: Vassiliadis, S., Wong, S., Hämäläinen, T.D. (eds.) SAMOS 2006. LNCS, vol. 4017, pp. 109–121. Springer, Heidelberg (2006)

[SM05] Spicher, A., Michel, O.: Using rewriting techniques in the simulation of dynamical systems: Application to the modeling of sperm crawling. In: Sunderam, V.S., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2005. LNCS, vol. 3514, pp. 820–827. Springer, Heidelberg (2005)

[SM07] Spicher, A., Michel, O.: Declarative modeling of a neurulation-like process. BioSystems 87(2-3), 281–288 (2007)

[SMC+08] Spicher, A., Michel, O., Cieslak, M., Giavitto, J.-L., Prusinkiewicz, P.: Stochastic p systems and the simulation of biochemical processes with dynamic compartments. BioSystems 91(3), 458–472 (2008)

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

[Ton74] Tonti, E.: The algebraic-topological structure of physical theories. In: Glockner, P.G., Sing, M.C. (eds.) Symmetry, similarity and group theoretic methods in mechanics, Calgary, Canada, pp. 441–467 (August 1974)

[Ton01] Tonti, E.: A direct discrete formulation of field laws: The cell method. Computer Modeling in Engineering & Sciences 2(2), 237–258 (2001)