# Arbitrary Nesting of Spatial Computations

Antoine Spicher*, Olivier Michel*, Jean-Louis Giavitto†

*LACL, Université Paris-Est Créteil,
61 av. du Général de Gaulle 94010 Créteil, France
Email: {olivier.micher,antoine.spicher}@u-pec.fr

†UMR 9912 STMS, Ircam & CNRS, UPMC, Inria
1 place Igor Stravinsky, 75004 Paris, France
Email: jean-louis.giavitto@ircam.fr

***Abstract*—Modern programming languages allow the definition and the use of *arbitrary* nested data structures but this is not generally considered in unconventional programming models. In this paper, we present arbitrary nesting in MGS, a spatial computing language. By considering different classes of neighborhood relationships, MGS can emulate several unconventional computing models from a programming point of view. The use of arbitrary nested spatial structures allows a hierarchical form of coupling between them. We propose an extension of the MGS pattern-matching facilities to handle directly nesting. This makes possible the straightforward emulation of a larger class of unconventional programming models.**

## I. Introduction

Modern programming languages allow data structure to be nested so that a valid element of a structure can also be in its turn another structure. Generally, this is not considered in unconventional programming models. For instance, the state of a cell in a cellular automata is not (the state of) another cellular automata. Another example: the value labeling a symbol in parametric Lindenmayer system is not (a string representing a derivation in) another Lindenmayer system.

In *chemical computing*, as exemplified in Gamma [1], chemical solutions are abstracted as *multisets* (a generalization of the notion of set in which members are allowed to appear more than once) and a molecule corresponds to an elementary data and not another chemical solution. Nested multisets are considered in *membrane systems*[1] [4] but are studied as a completely different computational model. Indeed, the management of the nesting entails the introduction of new mechanisms (transport rules in the case of membrane systems).

In this paper we consider arbitrary nesting in MGS, a spatial computing language where space is managed through the structure of the data. MGS relies on neighborhood relationships to represent physical (spatial distribution, localization of the resources) or logical constraints (inherent to the problem to be solved) in a computation.

[1]Nested multisets are also considered in High Order Chemical Language [2]. In Structured Gamma [3], elements of the multiset are linked by relations defined by a graph grammar. It is then theoretically possible to encode a given static nest of multisets using relations specified by a specific graph grammar to implement membership test and to make a distinction between elements and nested multisets.

By considering different classes of neighborhood relationships, MGS can emulate several unconventional computing models from the point of view of the programming. The use of arbitrary nested spatial structures allows a hierarchical form of coupling between them. Furthermore, we propose an extension of the MGS pattern-matching facilities to handle nesting explicitly. This makes possible the concise expression of various algorithms as well as the straightforward emulation of a larger class of unconventional programming models.

*Outlines:* This paper is organized as follows. The next section introduces the emerging field of *spatial computing* and the notions of topological collection and transformation developed in MGS. Section III discusses the relevance of nesting in the context of spatial computing and section III-D proposes a new pattern matching construct to make the handling of nested structures mores easier. Section IV exemplifies the use of nested spaces with three direct applications. The first encodes terms used to represents boolean formulae with nested sets. The computation of a disjunctive normal form on this representation is explained. The second example computes *quadtrees*, a recursive data structure for partitioning a two dimensional space. The last one is dedicated to the informal translation of the *fraglet* computational model into MGS. Related and future work concludes this article.

## II. Background

### A. Computing in Space, Space in Computation and Spatial Computing

*Spatial Computing* is an emerging field of research [5] where the computation is structured in term of *spatial relationships* where only “neighbor” elements may interact.

For example, the elements of a physical computing system are spatially localized and when a locality property holds, only elements that are neighbor in space can interact directly. So the interactions between parts are structured by the spatial relationships of the parts.

Even for non physical system, usually an element does not interact with all other elements in the system. For instance, from a given element in a data structure, only a limited number of other elements can be accessed [6]: in a simply linked list, the elements are accessed linearly (the second after the first, the third after the second, etc.); from a node in a tree, we can access the father or the sons; in arrays, the accessibility relationships are left implicit and implemented through incrementing or decrementing indices (called "Von Neumann" or "Moore" neighborhoods if one or several changes are allowed).

More generally, if an element $e$ in a system interacts during a computation with a subset $E = \{e_1, \ldots, e_n\}$ of other elements, it also interacts with any subset $E'$ included in $E$. This closure property induces a topological organization: the set of elements can be organized as an *abstract cellular complex* which is a spatial representation of the interactions in the computation [7]. This abstract space instantiates a neighborhood relationship that represents *physical* (spatial distribution, localization of the resources) or *logical* (inherent to the problem to be solved) constraints.

In addition, space can be an input to computation or a key part of the desired result of the computation, *e.g.* in computational geometry applications, amorphous computing [8], claytronics [9], distributed robotics or programmable matter... to cite a few examples where notions like position and shape are at the core of the application domain.

### B. The MGS approach to Spatial Computing

The MGS project recognizes that space is not an issue to abstract away but that computation is performed distributed across space and that space, either physical or logical, serves as a mean, a resource, an input and an output of a computation.

*1) Topological Collections:* In MGS, the notion of space is handled through a slight generalization of the notion of *field*. In physics, a field assigns a quantity to each point of a spatial domain [10].

MGS handles spatial domains defined by *abstract cellular complex* [11]. An abstract cellular complex is a formal construction that builds a space in a combinatorial way through more simple objects called *topological cells*. Each topological cell abstractly represents a part of the whole space: points are cells with dimension 0, lines are cells with dimension 1, surfaces are 2 dimensional cells, etc. The structure of the whole space, corresponding to the partition into topological cells, is considered through *incidence relationships*, relating a cell and the cells in its boundary.

In this approach a field is a finite labeling of a cellular complex: a cellular complex may count an infinite number of cells but MGS restricts itself on fields labeling only a finite number of these cells. Such fields are called *topological collections* to stress the importance of the neighborhood relationships induced by the incidence relationships. Topological collections are a weakening of the notion of *topological chain* developed in algebraic topology [12] and have been introduced in [13] to describe arbitrary complex spatial structures that appear in biological systems [14] and other dynamical systems with a time varying structure [15], [16]. Topological collections generalize fields because they associate a quantity with 0-cells (points in space) but also with arbitrary $n$-cells.

Graphs are examples of one dimensional cellular complexes: they are made of only 0- and 1-cells. In this paper, we will stick to topological collections where the underlying complex is a graph. In [6] it has been showed how usual data structures (sets, multisets, lists, trees, arrays...) can be seen as one dimensional topological collections: the elements in a data structure are the quantities assigned by the field to the nodes of a graph.

A specific neighborhood relationship has also a special importance in the rest of this paper: the *full relation*. With this relation, every node is neighbor of every other nodes. This corresponds to a node-labeled complete graph and to the multiset data structure.

*2) Transformations:* Usually in Physics, fields and their evolution are specified using differential operators. MGS generalizes these operators in a rewriting mechanism, called *transformation*. A transformation is the application of some local rules following some strategy. The application of a local rule $a \Longrightarrow b$ in a collection $C$:

1) selects a subcollection $A$ that matches the pattern $a$;
2) computes a new subcollection $B$ as the result of the evaluation of the expression $b$ instantiated with the collection $A$;
3) and substitutes $B$ for $A$ in $C$.

A local rule specifies a local evolution of the field: the left hand side (lhs) of the rule typically matches elements in interaction and the right hand side (rhs) computes local updates of the field.

Patterns $pat$ are expressions defined inductively starting from the *pattern variables* "$x$" (matching exactly one element in a collection) and several operators used to compose more complex patterns: the *neighborhood* between two subcollections "$pat, pat'$", the *repetition* "$pat\star$", the *guard* "$pat/exp$" and the *typing* "$pat : T$" (elements matched by $pat$ have to be of type $T$).

Transformations are a powerful means to define functions on topological collections complying with the underlying spatial structure. For instance, discrete analog of differential operators can be defined using transformations [17]. For multisets, transformations reduce simply to associative-commutative rewriting [18] also called multiset rewriting.

## III. Nested Spaces

In classical Physics, a field can be classified as a scalar field or a vector field according to whether the value of the field at each point is a scalar or a vector. However, it is not usual to consider "field valued fields". From the MGS perspective, this notion simply corresponds to the idea of nested topological collections. Such a feature is relevant and valuable in at least three areas: for the representation of and the computation on hierarchical or inductive data structures; in the modeling and the simulation of multiscale systems; and in the emulation of "stratified" computational models.

### A. Inductive Data Structures

The possibility to nest arbitrarily data structures is now pervasive in modern programming languages. Its usefulness to represent hierarchical data (*e.g.*, XML) or inductive structure (*e.g.*, list, trees) is well established. We give an example of the use of nested spaces in an algorithmic application relying on multisets in section IV-A.

### *B. Multiscale Systems*

The modeling of a natural system often implies entities appearing on distinct temporal and spatial scales: each level addresses a phenomenon over a specific window of length and time. These scales appear for logical reasons (at a particular scale, the system exhibits uniform properties and can be modeled by homogeneous rules acting on objects relevant at this scale) or for efficiency reasons (*e.g.*, the reductionist simulation of the whole system from first principles is computationally not tractable while we are only interested in coarse-grained description).

Multiscale models and simulations arise when interactions between scales must be considered. For spatial scales, it means that simultaneous spatial representations must be managed as in *adaptive mesh refinement* [19]. This method relies on a sequence of "nested rectangular grids" on which a PDE is discretized. It is important to realize that these subgrids are not patched into the coarse grid but overlaid to track the feature of interest. A simplistic example is presented in section IV-B. Another example, in the area of discrete modeling, is the *complex automata* framework [20] corresponding to a "graph of cellular automata".

Sometimes scales can be separated, meaning that the coupling between scales can be localized at some isolated interaction points in space and time. Then, the resulting computation corresponds to a hierarchical process with a directed flow of information. This is not always the case and we will introduce a dedicated pattern-matching mechanism in section III-D to ease the reference between scales.

### *C. Stratified Computational Models*

Some models of computation exhibit naturally an inductive structure. For instance, the state of a *membrane system* is a multiset of symbols and (inductively) membrane systems. This structure leads directly to a nested organization of "multiset of symbols and multisets".

Some computational models are also best described as a combination of two paradigms: the second being substituted for some generic parts in the first. We list a few examples issued from various compartmentalization devices introduced over a basic chemical framework. In membrane systems, strings have been considered instead of symbols [21]. This leads obviously to "multiset of sequences of symbols and multisets". Nested multisets are restricted to the description of membranes organized by inclusion only. *Tissue P systems* [22] arrange the membranes and their interactions following an arbitrary graph, calling for a "graph of multisets". *Spatial P systems* [23] are a variant of P systems which embodies the concept of space and position inside a membrane. Membranes and objects are positioned in a two-dimensional discrete space. Hence, we have to consider "grids of multisets and grids and symbols".

In section IV-C, we will sketch the encoding of *fraglets*, a molecular biology inspired execution model for computer communications leading to "graph of multisets of sequences".

### *D. Matching Nested Structures*

In the next section, we give examples of the three usages of nested spaces we have identified above section III. The use of nested spaces does not require *a priori* new control structures. For example, if the reactions between symbols of a P systems are coded by a transformation $EvalRule$, then we can defines a function $Apply$ and an auxiliary transformation $ApplyNested$ to thread $EvalRule$ over the nested structure:

```
fun Apply(x) = EvalRule(ApplyNested(x))
and trans ApplyNested = x:bag ⟹ Apply(x)
```

This piece of code is enough to trigger the chemical rules specified by $EvalRule$ through the entire structure. But the transport rules, used for example to expel one molecule from a membrane to the enclosing one, are a little bit heavier to write because they imply the simultaneous matching of two levels in the nested structure.

The usual MGS pattern constructions are "flat": the pattern variables of a transformation $T$ refer to elements of the collection on which $T$ is applied [24]. To make easier the handling of nested collections, we extend the current syntax with a new construction allowing references to elements of a nested collection. The pattern construct

```
[ pat | x ]
```

matches a collection $C$ nested within the current one. The pattern $pat$ must match a subcollection $C'$ in $C$ and the variable $x$ is bound to the collection $C$ deprived of $C'$. For example, the pattern

```
x, [2, 3 | y] as z
```

matches only one occurrence in the sequence

```
(0, (1, 2, 3, 4), 5)
```

and binds $x$ to 0, $z$ to the nested sequence `(1, 2, 3, 4)` and $y$ to the sequence `(1, 4)`. The notation `[ pat | ... ]` can be used to spare a variable if the rest of the subcollection is not used elsewhere.

## IV. Computing with Nested Collections

### *A. Disjunctive Normal Form*

Since the logical conjunction and disjunction operators are associative, commutative and idempotent, a logical formula can be encoded by nested sets. Let consider the following type declaration:

```
type formula = string | Not | And | Or
and record Not = { f:formula }
and collection And = set[formula]
and collection Or = set[formula]
```

In this declaration, `formula` is a sum type: a formula is either a boolean variable (represented by a string value), or the negation of a formula nested in a record with one field `f`, or the conjunction (resp. disjunction) of formulae nested in a set with subtype `And` (resp. `Or`). With these types, the formula $\neg(p \wedge q) \vee r$ can be represented as follows:

```
{ f = "p"::"q"::And:() }::"r"::Or:()
```

where $e$::$S$ inserts an element $e$ in the set $S$ and $C$:() denotes the empty collection of type $C$.

The computation of the disjunctive normal form can be achieved by iterating until a fixpoint is reached, the following transformation:

```
trans DNF = {
   (* Simplifying unaries *)
   [[x|...]:Not|...]:Not ⟹ x
   x:And / size(x)==1 ⟹ choose(x)
   x:Or / size(x)==1 ⟹ choose(x)

   (* Flattening nested ops *)
   [f:And|g]:And ⟹ join(f,g)
   [f:Or|g]:Or ⟹ join(f,g)

   (* De Morgan's laws *)
   [x:Or|...]:Not ⟹
     fold(::, And:(), map(λf.{f=f}, x))
   [x:And|...]:Not ⟹
     fold(::, Or:(), map(λf.{f=f}, x))

   (* Distributivity *)
   [x:Or|s]:And ⟹ map(λf.f::s, x)

   (* Induction *)
   x:And ⟹ DNF(x)
   x:Or ⟹ DNF(x)
   x:Not ⟹ DNF(x)
}
```

Each rule is a straightforward translation in MGS of a well known transformation of a boolean formula into an equivalent one. In this program, the `map` and `fold` are the usual map and fold functions: `map(`$f$`,`$s$`)` applies the function $f$ to each elements of the collection $s$ and returns the collection of results; `fold(`$f$`,`$z$`,`$s$`)` reduces the elements of the collection $s$ using the binary function $f$ and starting from $z$. The function `join` is used to append two collections and `choose` is used to pick-up one element in a collection. Finally, the expression $\lambda f.f$::$s$ is a lambda expression that appends its argument (here a formula) to the collection $s$.

### *B. A Simple Space Subdivisions Scheme*

This example shows the building of a quadtree that partitions a set of points in 2D space. Quadtrees recursively subdivide a rectangular spatial domain into four regions. The subdivision is recursively iterated until there is less than $n$ points in each region (we take $n = 2$ in the following). Figure 1 gives an example where the points are more or less aligned along a curve.

This adaptive mesh is described by the following type definitions:

```
type QuadTree = Grid[QuadTree] | Cloud
and gbf Grid = <n,e; 2e=0, 2n=0>
and collection Cloud = set[Point2D]
and record Point2D = { x:real, y:real }
```

GBF are collections with a regular neighborhood whose topology is specified though a group presentation. Here, the GBF

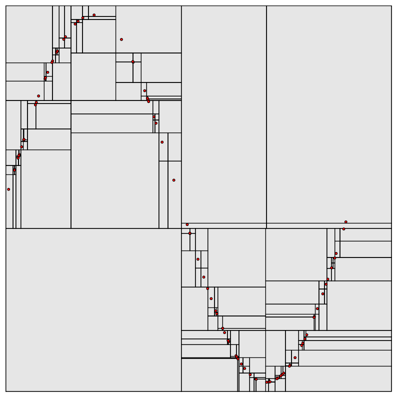
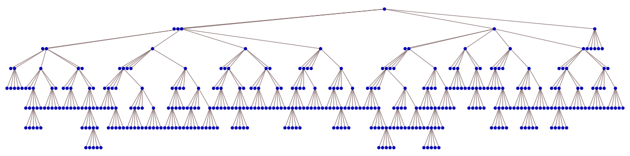
Figure 1. *Top:* Adaptive mesh refinement using quadtrees on a set of 100 points. *Bottom:* The corresponding nested structure pictured as an inclusion tree.

`Grid` specifies a $2 \times 2$ torus with two directions n and e. Such grid describe a partition in four adjacent regions.

The recursive subdivision is computed by the following transformation:

```
trans MakeQuadTree =
  c:Cloud / size(c)>2 ⟹
    MakeQuadTree(SplitCloud(c))
```

The function *SplitCloud* makes the real work:

```
fun SplitCloud(c:Cloud) =
    let g = barycenter(c) in
    let c_0, c_1 = split(λp.p.x<g.x, c) in
    let c_00, c_01 = split(λp.p.y<g.y, c_0) in
    let c_10, c_11 = split(λp.p.y<g.y, c_1) in
        Grid:(c_00@0, c_01@e, c_10@n, c_11@(n+e))
```

Function *SplitCloud* divides a cloud of points $c$ into four subclouds $c_{ij}$ depending on the positions of the points compared to the barycenter (above or below, on the left or on the right). Then it builds a new `Grid` collection where the four cells are labeled by the clouds $c_{ij}$. Function `barycenter` computes the center of mass of a cloud of points (it is easily expressed using a fold over the elements of the set.) The function `split` takes a predicate and a collection, and returns two collections: the elements of the first one satisfy the predicate and the second one gathers the remaining elements. The syntactic construction $T$`:(`$\dots v_i$@$c_i \dots$`)` builds a collection of type $T$ where the value $v_i$ is associated with the cell $c_i$.

The process is illustrated on figure 1. There is no need of the new pattern matching construct because there is no need to "mix" the elements of a top level collection with the elements of a nested one. It is the recursive calls of *MakeQuadTree* that create the nested structure from a flat cloud of points.

### C. *Fraglet*

Fraglets are tiny computation fragments or sequences of tokens that flow and react through a computer network. They have been introduced in [25] as an execution model for computer communications inspired by molecular biology. They have been designed to lay the ground for automatic network adaption and optimization processes as well as the synthesis and evolution of protocol implementations. Table I sketches the core instructions.

For example, the fraglets below together with a fraglet `[length tail]` located on the same node of the network, will compute the length of *`tail`* by generating the fraglet `[total n]` (where $n$ is the size of *`tail`*):

```
[counter 0]
[matchp length empty stop cnt]
[matchp stop match counter total]
[matchp cnt pop cnt1]
[matchp cnt1 split match counter
                incr counter * length]
[matchp incr exch sum 1]
```

In this program, the fraglets can be interpreted as follows: fraglet `[counter 0]` defines a local variable with initial value 0; fraglets starting with `matchp` define functions; finally fraglet `[length tail]` is the application of function `length` on the list *`tail`*.

In the following we encode the fraglet formalism by implementing a fraglet interpreter in MGS. For the sake of simplicity, we do not consider here the localization of the fraglets on the nodes of a communication network and the communication rules between nodes[2]. Let consider the following MGS type declaration:

```
type Token = int | `nul | `exch | ...
and collection Fraglet = seq[Token]
and collection State = bag[Fraglet]
```

The state of the system is represented by a multiset inhabited by a population of fraglets; fraglets are sequences of tokens (symbols or integers). For each fraglet operator, Table I gives the formal fraglet instruction, its informal semantics and its translation into an MGS transformation rule. For example, the split instruction consists in extracting the subsequence of tokens in the fraglet located between the operator (the first element in the sequence) and the first occurrence of the special token $\star$. This operation is straightforwardly translated in MGS: the pattern matches in a fraglet the operator `` `split`` (the syntactic construction `@0` checks that the operator is located at the first position in the sequence) followed by a subsequence terminated by the special token `` `time``. The subsequence is specified by `(`$x/x$` != `time)*` that matches a repetition of elements different from `` `time``.

[2]Nevertheless the reader is invited to pay attention that this restriction is done for the sake of the simplicity: the whole formalism can be specified in MGS using an additional level of nesting by considering a graph labeled by multisets of fraglets.

## V. Related Work

Topological collections are reminiscent of *Data-fields*, studied *e.g.* by B. Lisper [26]. Data-fields are a generalization of the array data structure where the set of indices is extended to all $\mathbb{Z}^n$ (see also [27]). We have introduced the concept of *group based fields*, or GBF [28], [29], to extend data-fields towards more general regular data structures. Topological collections emphasize data structures as a set of *places* independently of their occupation by values. This approach is also shared by the theory of *species of structures* [30]. Motivated by the development of enumeration techniques for labeled structures, the emphasis is put on the transport of structures along bijections while spatial computing focuses on topological relationships.

Disentangling the elements in a data structure from their organization has several advantages. In [31], B. Jay develops a concept of *shape polymorphism* where a data structure is also a pair (*shape*, *set of data*). The shape describes the organization of the data structure (restricted to tabular organizations) and the set of data describes the content of the data structure. This separation allows the development of *shape-polymorphic functions* and their typing: the shape of the result of a shape-polymorphic function application depends only on the shape of the argument, not of its content. The same line is developed in the field of *polytypic programming* for algebraic data type [32]. MGS transformations are naturally polytypic and extend far beyond arrays and algebraic data type. Polytypism in MGS relies on a generic implementation of pattern matching [24] not on overloading or *ad-hoc* polymorphism.

Transformations are a kind of rewriting that differs in many ways from graph rewriting. Their formalization in [33] is not based on the usual graph morphisms and pushouts like in [34] but is inspired by the approach of J.-C. Raoult [35] where graph rewriting based on a (multi-)set point of view is developed. The proposed model is close to term rewriting modulo associativity and commutativity (where the left hand side of a rule is removed and the right hand side is added). This kind of approach also allows to extend results from term rewriting to topological rewriting (as we did for termination in [36]). Note that the notions of topological collection and topological rewriting are more general than labeled graphs and graph rewriting, and may handle higher dimensional objects, a feature relevant in a lot of application areas [37].

Nested data structure are now widespread in programming languages but are less natural in the context of data bases. The importance of organizing the accesses to the element in a complex structure trough primitive operations related to the type constructor is stressed in [38]. In MGS, accesses rely on pattern matching, and the pattern matching constructs reflect the spatial structure underlying a collection. Nevertheless, structural recursion, advocated in [38], is straightforward as showed by the programs in sections IV-A and IV-B.

## VI. Conclusions and Future Work

In this paper, we have presented the advantages of nesting in a spatial model of computation, the MGS experimental language. The language MGS, has been used in the context

| *Op* | *Input* | *Output* |
|---|---|---|
| nul | [nul *tail*] | [] |
| | destroy a fraglet | |
| | [ ‘null@0 $\mid$ *tail* ]:Fraglet $\Longrightarrow$ Fraglet:() | |
| dup | [dup t a *tail*] | [t a a *tail*] |
| | duplicate a single symbol | |
| | [ ‘dup@0, $t$, $a \mid$ *tail* ]:Fraglet $\Longrightarrow$ $t$::$a$::$a$::*tail* | |
| exch | [exch t a b *tail*] | [t b a *tail*] |
| | swap two tags | |
| | [ ‘exch@0, $t$, $a$, $b \mid$ *tail* ]:Fraglet $\Longrightarrow$ $t$::$b$::$a$::*tail* | |
| split | [split s1 $\star$ s2] | [s1] [s2] |
| | break a fraglet into two at the first occurrence of * | |
| | [ ‘split@0, ($x/x$ != ‘time) $\star$ as $s_1$, ‘time $\mid s_2$ ]:Fraglet $\Longrightarrow$ $s_1$, $s_2$ | |
| pop | [pop h a *tail*] | [h *tail*] |
| | pop the “head” element of the list “a, *tail*” | |
| | [ ‘pop@0, $h$, $a \mid$ *tail* ]:Fraglet $\Longrightarrow$ $h$::*tail* | |
| empty | [empty yes no *tail*] | [yes] *or* [no *tail*] |
| | test for empty *tail* | |
| | [ ‘empty@0, $y$, $n \mid$ *tail* ]:Fraglet $\Longrightarrow$ if size(*tail*) == 0 then $y$:: Fraglet:() else $n$::*tail* | |
| sum | [sum t $n_1$ $n_2$ *tail*] | [t $(n_1 + n_2)$ *tail*] |
| | arithmetic addition | |
| | [ ‘sum@0, $t$, $n_1$, $n_2 \mid$ *tail* ]:Fraglet $\Longrightarrow$ $t$::($n_1$+$n_2$)::*tail* | |
| match | [match a *tail1*],[a *tail2*] | [*tail1* *tail2*] |
| | two fraglets react, their tails are concatenated | |
| | [ ‘match@0, $a \mid t_1$ ]:Fraglet, [ $b$@0 $\mid t_2$ ]:Fraglet $\Longrightarrow$ join($t_1$,$t_2$) | |
| matchP | [matchP a *tail1*],[a *tail2*] | [*tail1* *tail2*] |
| | *idem* as match but the rule persists | |
| | [ ‘matchp@0, $a \mid t_1$ ]:Fraglet as $f$, [ $b$@0 $\mid t_2$ ]:Fraglet $\Longrightarrow$ $f$, join($t_1$,$t_2$) | |

Table I
SUBSET OF THE FRAGLETS CORE INSTRUCTIONS (FROM [25]) AND THEIR MGS TRANSLATION.

of P systems [13] and in several large modeling projects in systems biology [39], [14], [40].

One interest of the spatial paradigm *à la* MGS is its ability to subsume several computational models in a single uniform formalism, as long as one focuses on programming [41], [42]. We showed the benefits of considering nested spatial computing through three kind of examples: in algorithmic, in simulation of multiscale phenomena and in the emulation of other programming models.

The management of nested collections is achieved through three kinds of devices:

1) collections are first-citizen values and can be used as the values of another collection;
2) a specific pattern construction [ $p$ | ... ] makes possible, within the current pattern, to refer to the elements matched by a pattern $p$ in a nested collection;
3) recursive type declarations generate predicates used to constrain the nesting and to control the pattern matching facilities.

These three features together enable a very concise and readable programming style, as exemplified in section III-D. All the presented examples are actual MGS programs, at the exception of some slight syntactic sugar.

The work presented in this paper may be enriched and extended in several directions. The pattern matching we have presented can be seen as operating at an “horizontal level” on the elements of a collection and at a “vertical level” when descending to match some elements of a nested collection. The constructions dedicated to the horizontal level are very expressive, allowing for example the matching of an unknown number of elements. The handling of the vertical level is actually restricted to the [ *pat* | ... ] operator. Other construction can be designed, by analogy with the vertical level. For example, an operator to allow references through an unknown number of nesting, in a manner analog to the iteration operator “$\star$”, would be interesting to mimic path queries in XML. Note however that the distinction between horizontal and vertical level is questionable. An alternative approach would be to unify the nested collection by looking for the spatial relationships holding in the whole structure, irrespectively of the horizontal or the vertical view. The topology of this “flat whole structure” can be build as the topology of a fiber space over the top collection. The investigation of this framework remains to be done.

## ACKNOWLEDGMENTS

The authors would like to thanks H. Klaudel, F. Pommereau, F. Delaplace and J. Cohen for many questions, encouragements and sweet cookies. This research is supported in part by the ANR projects SynBioTIC.

## REFERENCES

[1] J. Banâtre, P. Fradet, and D. Le Métayer, “Gamma and the chemical reaction model: Fifteen years after,” *Multiset processing: mathematical, computer science, and molecular computing points of view*, vol. 2235, pp. 17–44, 2001.
[2] J. Banâtre, P. Fradet, and Y. Radenac, “Programming self-organizing systems with the higher-order chemical language,” *International Journal of Unconventional Computing*, vol. 3, no. 3, p. 161, 2007.
[3] P. Fradet and D. Le Métayer, “Structured gamma,” *Science of Computer Programming*, vol. 31, no. 2-3, pp. 263–289, 1998.

[4] G. Paun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 1, no. 61, pp. 108–143, 2000.

[5] A. De Hon, J.-L. Giavitto, and F. Gruau, Eds., *Computing Media and Languages for Space-Oriented Computation*, ser. Dagsthul Seminar Proceedings, no. 06361. Dagsthul, http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=2006361, 3-8 sptember 2006.

[6] J.-L. Giavitto and O. Michel, "Data structure as topological spaces," in *Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02*, vol. 2509, Himeji, Japan, Oct. 2002, pp. 137–150, lecture Notes in Computer Science.

[7] J.-L. Giavitto, O. Michel, J. Cohen, and A. Spicher, "Computation in space and space in computation," in *Unconventional Programming Paradigms (UPP'04)*, ser. LNCS, vol. 3566. Le Mont Saint-Michel: Spinger, Sep. 2005, pp. 137–152.

[8] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss, "Amorphous computing," *CACM: Communications of the ACM*, vol. 43, 2000.

[9] B. Aksak, P. S. Bhat, J. Campbell, M. DeRosa, S. Funiak, P. B. Gibbons, S. C. Goldstein, C. Guestrin, A. Gupta, C. Helfrich, J. F. Hoburg, B. Kirby, J. Kuffner, P. Lee, T. C. Mowry, P. Pillai, R. Ravichandran, B. D. Rister, S. Seshan, M. Sitti, and H. Yu, "Claytronics: highly scalable communications, sensing, and actuation networks," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, SenSys 2005, San Diego, California, USA, November 2-4, 2005*, J. Redi, H. Balakrishnan, and F. Zhao, Eds. ACM, 2005, p. 299. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098964

[10] G. T. Leavens, "Fields in physics are like curried functions or physics for functional programmers," Iowa State University, Department of Computer Science, Tech. Rep. TR94-06b, May 1994.

[11] A. Tucker, "An abstract approach to manifolds," *The Annals of Mathematics*, vol. 34, no. 2, pp. 191–243, 1933.

[12] J. Munkres, *Elements of Algebraic Topology*. Addison-Wesley, 1984.

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

[14] ——, "Modeling the topological organization of cellular processes," *BioSystems*, vol. 70, pp. 149–163, 2003.

[15] J.-L. Giavitto, "Topological collections, transformations and their application to the modeling and the simulation of dynamical systems," in *14th International Conference on Rewriting Technics and Applications (RTA'03)*, ser. LNCS, vol. 2706. Valencia: Springer, Jun. 2003, pp. 208–233.

[16] J.-L. Giavitto and A. Spicher, "Topological rewriting and the geometrization of programming," *Physica D*, vol. 237, no. 9, pp. 1302–1314, jully 2008.

[17] ——, "Topological rewriting and the geometrization of programming," *Physica D*, vol. 237, no. 9, pp. 1302–1314, jully 2008.

[18] N. Dershowitz, J. Hsiang, N. Josephson, and D. Plaisted, "Associative-commutative rewriting," in *Proceedings of the Eighth international joint conference on Artificial intelligence-Volume 2*. Morgan Kaufmann Publishers Inc., 1983, pp. 940–944.

[19] M. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of computational Physics*, vol. 53, no. 3, pp. 484–512, 1984.

[20] A. Hoekstra, E. Lorenz, J. Falcone, and B. Chopard, "Towards a complex automata framework for multi-scale modeling: Formalism and the scale separation map," *Computational Science–ICCS 2007*, pp. 922–930, 2007.

[21] J. Castellanos, G. Paun, and A. Rodriguez-Paton, "Computing with membranes: P systems with worm-objects," in *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*. IEEE, 2000, pp. 65–74.

[22] C. Martín-Vide, G. Paun, J. Pazos, and A. Rodríguez-Patón, "Tissue p systems," *Theoretical Computer Science*, vol. 296, no. 2, pp. 295–326, 2003.

[23] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, G. Pardini, and L. Tesei, "Spatial p systems," *Natural Computing*, vol. 10, no. 1, pp. 3–16, 2011.

[24] J.-L. Giavitto and O. Michel, "Pattern-matching and rewriting rules for group indexed data structures," in *ACM Sigplan Workshop RULE'02*. Pittsburgh: ACM, Oct. 2002, pp. 55–66.

[25] C. Tschudin, "Fraglets-a metabolistic execution model for communication protocols," in *Proc. 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS), Menlo Park, USA*, 2003, pp. 1–6.

[26] B. Lisper, "On the relation between functional and data-parallel programming languages," in *Proc. of the 6th. Int. Conf. on Functional Languages and Computer Architectures*, ACM. ACM Press, Jun. 1993.

[27] J.-L. Giavitto, D. De Vito, and J.-P. Sansonnet, "A data parallel Java client-server architecture for data field computations over $\mathbb{Z}^n$," in *EuroPar'98 Parallel Processing*, ser. LNCS, vol. 1470, Sep. 1998, pp. 742–??

[28] J.-L. Giavitto, O. Michel, and J.-P. Sansonnet, "Group based fields," in *Parallel Symbolic Languages and Systems (International Workshop PSLS'95)*, ser. LNCS, I. Takayasu, R. H. J. Halstead, and C. Queinnec, Eds., vol. 1068. Beaune (France): Springer-Verlag, 2–4 Oct. 1995, pp. 209–215. [Online]. Available: ftp://ftp.lri.fr/LRI/articles/michel/psls95.ps.gz

[29] J.-L. Giavitto, "Rapport scientifique en vue d'obtenir l'habilitation à diriger des recherches," Ph.D. dissertation, Université de Paris-Sud, centre d'Orsay, May 1998. [Online]. Available: http://www.lami.univ-evry.fr/~giavitto/

[30] F. Bergeron, G. Labelle, and P. Leroux, *Combinatorial species and tree-like structures*, ser. Encyclopedia of mathematics and its applications. Cambridge University Press, 1997, vol. 67, isbn 0-521-57323-8.

[31] C. B. Jay, "A semantics for shape," *Science of Computer Programming*, vol. 25, no. 2–3, pp. 251–283, 1995.

[32] J. Jeuring and P. Jansson, "Polytypic programming," *Lecture Notes in Computer Science*, vol. 1129, pp. 68–114, 1996.

[33] A. Spicher, O. Michel, and J.-L. Giavitto, "Declarative mesh subdivision using topological rewriting in mgs," in *Int. Conf. on Graph Transformations (ICGT) 2010*, ser. LNCS, vol. 6372, Sep. 2010, pp. 298–313.

[34] H. Ehrig, M. Pfender, and H. J. Schneider, "Graph grammars: An algebraic approach," in *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1973.

[35] J.-C. Raoult and F. Voisin, "Set-theoretic graph rewriting," in *Proceedings of the International Workshop on Graph Transformations in Computer Science*. London, UK: Springer-Verlag, 1994, pp. 312–325. [Online]. Available: http://portal.acm.org/citation.cfm?id=647364.725670

[36] J.-L. Giavitto, O. Michel, and A. Spicher, *Software-Intensive Systems and New Computing Paradigms*, ser. LNCS. Springer, november 2008, vol. 5380, ch. Spatial Organization of the Chemical Paradigm and the Specification of Autonomic Systems, pp. 235–254. [Online]. Available: http://www.springerlink.com/content/g1357n85j8301078/?p=a5c6f79393724a9d88f508d110a8bfe2&pi=6

[37] E. Tonti, "On the mathematical structure of a large class of physicial theories," *Rendidiconti della Academia Nazionale dei Lincei*, vol. 52, no. fasc. 1, pp. 48–56, Jan. 1972, scienze fisiche, matematiche et naturali, Serie VIII.

[38] P. Buneman, S. Naqvi, V. Tannen, and L. Wong, "Principles of programming with complex objects and collection types," *Theoretical Computer Science*, vol. 149, no. 1, pp. 3–48, 18 Sep. 1995.

[39] P. Barbier de Reuille, I. Bohn-Courseau, K. Ljung, H. Morin, N. Carraro, C. Godin, and J. Traas, "Computer simulations reveal properties of the cell-cell signaling network at the shoot apex in Arabidopsis," *PNAS*, vol. 103, no. 5, pp. 1627–1632, 2006. [Online]. Available: http://www.pnas.org/cgi/content/abstract/103/5/1627

[40] A. Spicher, O. Michel, and J.-L. Giavitto, *Understanding the Dynamics of Biological Systems: Lessons Learned from Integrative Systems Biology*. Springer Verlag, Feb. 2011, ch. Interaction-Based Simulations for Integrative Spatial Systems Biology.

[41] O. Michel, A. Spicher, and J.-L. Giavitto, "Rule-based programming for integrative biological modeling – application to the modeling of the lambda phage genetic switch," *Natural Computing*, vol. 8, no. 4, pp. 865–889, december 2009, published online: 12 November 2008. [Online]. Available: http://www.lacl.fr/~michel/PUBLIS/2009/naco09.pdf

[42] A. Spicher, O. Michel, and J.-L. Giavitto, *Understanding the dynamics of biological systems*. Springer, 2011, ch. Interaction-based simulations for Integrative Spatial Systems Biology, pp. 195–231. [Online]. Available: http://www.lacl.fr/~michel/PUBLIS/2010/ibss.pdf