Continuation Semantics for Dynamic Hierarchical Systems

Gabriel Ciobanu
Institute of Computer Science
Romanian Academy, Iaşi
Email: gabriel@iit.tuiasi.ro

Eneia Nicolae Todoran
Department of Computer Science
Technical University of Cluj-Napoca, Romania
Email: eneia.todoran@cs.utcluj.ro

***Abstract*—We present a denotational semantics designed with metric spaces and continuations for a simple concurrent language $\mathcal{L}_{MB}$ embodying a representative set of features encountered in membrane computing. $\mathcal{L}_{MB}$ is a multiset rewriting language. In $\mathcal{L}_{MB}$ multisets of objects are encapsulated in hierarchical structures of compartments, or regions, delimited by *membranes*. The behaviour of each membrane is specified by means of *multiset rewriting rules*. The semantics of parallel composition in $\mathcal{L}_{MB}$ is based on the concept of *maximal parallelism*. Computations proceed according to the multiset rewriting rules, nondeterministically choosing the rules and the objects. Membranes can be grouped into classes based on the rewriting rules that they encapsulate; $\mathcal{L}_{MB}$ also provides a primitive for membrane creation, or instantiation. In this sense, $\mathcal{L}_{MB}$ is similar to an object oriented language. We use continuations and a powerdomain construction to represent nondeterministic behavior. An element of a powerdomain is a collection of sequences of observables representing dynamic membrane structures. Our continuation semantics describes in a compositional manner the behavior of an $\mathcal{L}_{MB}$ program as a dynamic hierarchical system. As far as we know, this is the first paper that presents a metric denotational semantics for the combination of features embodied in $\mathcal{L}_{MB}$.**

***Keywords*-metric semantics; continuations; parallel multiset rewriting; dynamic hierarchical system; membrane computing**

I. Introduction

Membrane computing [11] is a research area within computer science, which investigates computing patterns inspired by the functioning of living cells. In this paper we investigate the domain by using methods in the tradition of programming language semantics. We consider a simple concurrent language $\mathcal{L}_{MB}$ in which computations are specified by means of multiset rewriting rules distributed into membrane-delimited compartments, or regions. The language is partly inspired by object oriented programming. In $\mathcal{L}_{MB}$ there is a notion of *membrane declaration*, which introduces a type of membranes. Membranes are grouped into classes based on the rewriting rules that they encapsulate. Also, $\mathcal{L}_{MB}$ provides a primitive for membrane creation, or *instantiation*. Each membrane instance is given a *unique* label. For illustration purposes, in Section I-A we present a small $\mathcal{L}_{MB}$ example program.

By using metric spaces and continuations we design a denotational semantics for $\mathcal{L}_{MB}$. We use continuations and a powerdomain construction to represent nondeterministic behavior. An element of a powerdomain is a collection of sequences of observables representing dynamic membrane structures. We employ a fixed point construction to express the semantics of multiset rewriting computations. Our continuation semantics describes in a compositional manner the behavior of an $\mathcal{L}_{MB}$ program as a dynamic hierarchical system.

A *membrane system*, also called a P-system [11], comprises a hierarchical structure of nested regions, delimited by *membranes*. The outermost membrane is called the *skin*. Each membrane can contain a multiset of *objects*, evolution *rules*, and zero or more nested membranes. The behavior of each membrane is specified by means of *multiset rewriting rules*. The computing model is parallel and nondeterministic. The behavior of objects in a membrane system can be described based on the concept of *maximal parallelism*. All applicable rewriting rules are applied in parallel, without using interleaving. Also, when several combinations of rules are applicable the selection of (the combination of) rules is nondeterministic.

The membrane computing paradigm involves a whole class of models, and there is a large panoply of options in devising a membrane system. There are various features that we ignore in this paper. For example, trans-membrane communication is not included in $\mathcal{L}_{MB}$. However, we are confident that most (even all) membrane computing concepts that are investigated in the literature can be given a denotational semantics by using the techniques that we employ in this paper. Continuations for concurrency [15], [7] and classic continuation passing style represent essential tools in this semantic investigation, because they can be used to describe in a purely compositional manner the behavior of dynamic hierarchical systems.

Denotational semantics (initially known as mathematical semantics or Scott-Strathchey semantics) is an important approach to formalizing the meanings of languages. The most important principle in denotational semantics is *compositionality*: the meaning of a compound construction is

determined solely on the basis of of the meanings of its components. In general, denotational semantics assigns to every construction of a language a certain formal *meaning*, which is a value from a suitably chosen mathematical model. Following [3], we choose to use the mathematical framework of *complete metric spaces* for our semantic description. In this aproach to semantics one can define mathematical objects and prove their properties by making use of Banach's theorem which states that contracting functions on complete metric spaces have *unique* fixed points.

A. The language $\mathcal{L}_{MB}$

The aim of this paper is to offer a semantic investigation in the area of membrane computing. For this purpose we use methods and tools consecrated in the tradition of programming language semantics. Hence, we find convenient to use terms which are usually employed in the context of programming languages. In particular, the elements of $\mathcal{L}_{MB}$ are syntactic constructions that we call *statements*, or *programs*. Also, when we describe the behavior of an $\mathcal{L}_{MB}$ program we use the term *execution*.

An $\mathcal{L}_{MB}$ program comprises a list of membrane declarations, which is similar to a list of class declarations in an object oriented language. The list of membrane declarations is followed by an $\mathcal{L}_{MB}$ statement, which is executed in the skin membrane. We illustrate the concepts embodied in $\mathcal{L}_{MB}$ by a brief example.

$$
\begin{array}{l}
\mathsf{membrane}\ M_0\,\{ \\
\quad [o_1, o_3] \Rightarrow o_2 \parallel o_4; \\
\quad [o_2] \Rightarrow o_5 \parallel \mathsf{new}(M_1, o_1 \parallel o_5); \\
\quad [o_2] \Rightarrow o_4; \\
\quad [o_5] \Rightarrow o_4 \\
\}; \\
\mathsf{membrane}\ M_1\,\{ \\
\quad [o_1] \Rightarrow o_2; \\
\quad [o_2] \Rightarrow o_3 \\
\}; \\
o_1 \parallel o_3
\end{array}
$$

This program comprises two membrane declarations, and the statement $o_1 \parallel o_3$. Computations are specified by lists of rewriting rules included in the membrane declarations.

A rewriting rule is a construction of the form $w \Rightarrow x$, where w is a multiset, and x is an $\mathcal{L}_{MB}$ statement. If we compare $\mathcal{L}_{MB}$ with an object oriented (OO) language we can make the following analogy. An $\mathcal{L}_{MB}$ rewriting rule $w \Rightarrow x$ corresponds to an OO method declaration. The multiset w is the name of the 'method', and the statement x is the body of the 'method'. A membrane declaration $\mathsf{membrane}\ M\,\{r\}$ corresponds to an OO class declaration, i.e., it introduces a new type of membranes with name M. $\mathcal{L}_{MB}$ also provides a primitive for membrane creation $\mathsf{new}(M, x)$, which creates a new instance of a membrane of the type M to execute statement x.

We assume given a (countably infinite) alphabet of symbols with typical elements o, o_i. To continue the analogy with object oriented programming, we must state clearly that an object is *not* an instance of a membrane. In $\mathcal{L}_{MB}$ an object is just an elementary statement, a symbol taken from some given alphabet. An $\mathcal{L}_{MB}$ statement may be either an object, or a membrane creation statement of the form $\mathsf{new}(M, x)$, where M is a membrane name and x is a statement, or a parallel composition of two $\mathcal{L}_{MB}$ statements of the form $x_1 \parallel x_2$. Semantically, the parallel composition of two $\mathcal{L}_{MB}$ constructions is also handled by using the concept of a multiset.[1] Whence, a construction $w \Rightarrow x$ specifies a multiset rewriting rule. We use the notation $[o_1, \ldots, o_n]$ to describe the multiset containing the objects $o_1, \ldots, o_n$.

In the example given above, an instance of the membrane named M_0 is created which executes the statement $o_1 \parallel o_3$; this membrane instance becomes the skin membrane of the system. By convention, the skin is always an instance of the first membrane in the membrane declarations list (in this case M_0). The structure of a membrane system can be represented by a tree structure, or equivalently, by a string of correctly matching parentheses. We represent the contents of a membrane between parentheses '$\langle$' and '$\rangle$'. The system behavior is described below by two sequences of membrane configurations, representing the possible executions of the system. We use the symbol '$\Rrightarrow$' to describe each complex computation step. Each such step may involve applications of several rewriting rules in all nested membranes.

$$
\begin{array}{l}
\langle M_0, l_0 \mid [o_1, o_3]; \rangle \\
\Rrightarrow \langle M_0, l_0 \mid [o_2, o_4]; \rangle \\
\Rrightarrow \langle M_0, l_0 \mid [o_4, o_4]; \rangle
\end{array}
$$

The first execution sequence corresponds to the application of two rewriting rules inside the skin membrane, namely: $[o_1, o_3] \Rightarrow o_2 \parallel o_4$, followed by $[o_2] \Rightarrow o_4$. At this point a halting configuration is reached, where no further rule can be applied. The execution model is both parallel and non-deterministic. In this particular case two alternative execution traces are possible; the second one is given below.

$$
\begin{array}{l}
\langle M_0, l_0 \mid [o_1, o_3]; \rangle \\
\Rrightarrow \langle M_0, l_0 \mid [o_2, o_4]; \rangle \\
\Rrightarrow \langle M_0, l_0 \mid [o_5, o_4]; \langle M_1, l_1 \mid [o_1, o_5]; \rangle \rangle \\
\Rrightarrow \langle M_0, l_0 \mid [o_4, o_4]; \langle M_1, l_1 \mid [o_2, o_5]; \rangle \rangle \\
\Rrightarrow \langle M_0, l_0 \mid [o_4, o_4]; \langle M_1, l_1 \mid [o_3, o_5]; \rangle \rangle
\end{array}
$$

Note that, in general, all rules (in all nested membranes) are applied in a maximally parallel manner. This phenomenon occurs in this second execution sequence. The first two steps of the second execution sequence correspond to the application of the following rules in the skin membrane: $[o_1, o_3] \Rightarrow o_2 \parallel o_4$, followed by $[o_2] \Rightarrow o_5 \parallel \mathsf{new}(M_1, o_1 \parallel o_5)$. The second rule creates an inner membrane of the type M_1, with label l_1. In the next step two rules are applied in parallel:

[1] A *multiset* is a generalization of a set, a collection in which an element may occur more than once; a formal definition is provided in Section II.

rule $[o_5] \Rightarrow o_4$ in membrane l_0 (of the type M_0) and rule $[o_1] \Rightarrow o_2$ in membrane l_1 (of the type M_1). Rule $[o_2] \Rightarrow o_3$ is applied in the last step in the inner membrane l_1.

A membrane may contain several nested membranes of the same type, but each membrane has a *unique* label. Labels are generated automatically at execution time.

B. Semantic prototype interpreter

The denotational (mathematical) specification given in this paper was developed following a prototyping approach. We used the functional language Haskell [13] as a prototyping tool (and as a metalanguage) for denotational semantics. An implementation of the semantic interpreter described in this paper is available from [17]. The interpreter contains a couple of $\mathcal{L}_{MB}$ test programs, including the $\mathcal{L}_{MB}$ program presented in Section I-A.

C. Contribution

In previous work we have defined the operational semantics for membrane systems [2], [4], and proved certain semantic properties. Also, in [5], [6] we applied continuation semantics for concurrency [15], [7] in designing a denotational semantics for a multiset rewriting concurrent language. The language described in [5], [6] is a strict subset of the language $\mathcal{L}_{MB}$ that we investigate in this paper.

To the best of our knowledge, this is the first paper that presents a denotational semantics for a language where computations are specified by rules for parallel rewriting of multisets, and rules are distributed to membrane-delimited regions. These are core concepts of the membrane computing paradigm [11]. The dynamic structure of membrane delimited compartments was not investigated in our previous work [5], [6]. Our denotational model is designed with metric spaces and continuations. We show that continuations for concurrency can be used to describe in a purely compositional manner the behavior of dynamic hierarchical systems. We think that continuations can also be used to describe compositionally (all) other concepts that are studied in the membrane computing literature, including parallel communication and parallel dissolving [12], [4]. To model parallel dissolving one can use a variant of failure continuations as in [16].

II. Mathematical preliminaries

The notation $(x \in)X$ introduces the set X with typical element x ranging over X. A *multiset* is a generalization of a set. Intuitively, a multiset is a collection in which an element may occur more than once. One can represent the concept of a multiset of elements of type X by using functions from $X \to \mathbb{N}$, or partial functions from $X \to \mathbb{N}^+$, where $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ ($\mathbb{N}^+$ is the set of natural numbers without 0).

Let $(x \in)X$ be a countable set. We use the notation:

$$[X] \stackrel{not.}{=} \bigcup_{A \in \mathcal{P}_{finite}(X)} \{m \mid m \in (A \to \mathbb{N}^+)\}$$

where $\mathcal{P}_{finite}(X)$ is the power set of all *finite* subsets of X. As X is countable, $\mathcal{P}_{finite}(X)$ is also countable. An element $m \in [X]$ is a (finite) multiset of elements of type X, a function $m : A \to \mathbb{N}^+$, for some finite subset $A \subseteq X$, such that $\forall x \in A : m(x) > 0$. $m(x)$ is called the *multiplicity* (number of occurrences) of x in m. $[X]$ is the set of all finite multisets of elements of type X.

One can represent a multiset $m \in [X]$ by enumerating its elements between parentheses '[' and ']'. The elements in a multiset are *not* ordered; to give yet another intuition, a multiset is an unordered list of elements. For example, [] is the empty multiset, i.e. the function with empty graph. As another example $[x_1, x_1, x_2] = [x_1, x_2, x_1] = [x_2, x_1, x_1]$ is the multiset with two occurences of x_1 and one occurence of x_2, i.e. the function $m : \{x_1, x_2\} \to \mathbb{N}^+, m(x_1) = 2, m(x_2) = 1$.

One can define various operations on multisets $m_1, m_2 \in [X]$. Below, $\mathsf{dom}(\cdot)$ is the domain of function '$\cdot$'.

- *Multiset sum:* $m_1 \uplus m_2 \quad (\uplus : ([X] \times [X]) \to [X])$
  $\mathsf{dom}(m_1 \uplus m_2) = \mathsf{dom}(m_1) \cup \mathsf{dom}(m_2)$
  $(m_1 \uplus m_2)(x) =$
  $$\begin{cases} m_1(x) + m_2(x) & \text{if} \quad x \in \mathsf{dom}(m_1) \cap \mathsf{dom}(m_2) \\ m_1(x) & \text{if} \quad x \in \mathsf{dom}(m_1) \setminus \mathsf{dom}(m_2) \\ m_2(x) & \text{if} \quad x \in \mathsf{dom}(m_2) \setminus \mathsf{dom}(m_1) \end{cases}$$
- *Multiset difference:* $m_1 \setminus m_2$ $(\setminus : ([X] \times [X]) \to [X])$
  $\mathsf{dom}(m_1 \setminus m_2) = (\,\mathsf{dom}(m_1) \setminus \mathsf{dom}(m_2)\,) \cup \{x \mid x \in \mathsf{dom}(m_1) \cap \mathsf{dom}(m_2)\,, \; m_1(x) > m_2(x)\}$
  $(m_1 \setminus m_2)(x) =$
  $$\begin{cases} m_1(x) & \text{if} \quad x \in \mathsf{dom}(m_1) \setminus \mathsf{dom}(m_2) \\ m_1(x) - m_2(x) & \text{if} \quad x \in \mathsf{dom}(m_1) \cap \mathsf{dom}(m_2) \\ & \quad\; m_1(x) > m_2(x) \end{cases}$$
- *Submultiset:* $m_1 \subseteq m_2$ $(\subseteq : ([X] \times [X]) \to Bool)$
  $m_1 \subseteq m_2 =$
  $(\,\mathsf{dom}(m_1) \subseteq \mathsf{dom}(m_2)\,) \wedge$
  $(\forall x \in \mathsf{dom}(m_1) : m_1(x) \leq m_2(x))$

We write $m_1 = m_2$ to express that the multisets m_1 and m_2 are equal. $m_1 = m_2$ iff $\mathsf{dom}(m_1) = \mathsf{dom}(m_2)$ and $\forall x \in \mathsf{dom}(m_1) : m_1(x) = m_2(x)$.

If $f : X \to X$ and $f(x) = x$ we call x a *fixed point* of f. When this fixed point is unique (see Theorem 2.1) we write $x = fix(f)$.

The denotational semantics given in this paper is built within the mathematical framework of *1-bounded complete metric spaces*. We work with the following notions which we assume known: *metric* and *ultrametric* space, *isometry* (distance preserving bijection between metric spaces, denoted by '$\cong$'), *complete* metric space, and *compact* set. For details the reader may consult, e.g., the monograph [3].

We recall that if (X, d_X), (Y, d_Y) are metric spaces, a function $f : X \to Y$ is a *contraction* if $\exists c \in \mathbb{R}$, $0 \leq c < 1$, $\forall x_1, x_2 \in X : d_Y(f(x_1), f(x_2)) \leq c \cdot d_X(x_1, x_2)$. In metric semantics it is customary to attach a contracting factor of $c = \frac{1}{2}$ to each computation step. When $c = 1$ the function f

is called *non-expansive*. In what follows we denote the set of all nonexpansive functions from X to Y by $X \stackrel{1}{\to} Y$. The following theorem is at the core of metric semantics.

Theorem 2.1 (Banach): Let (X, d_X) be a complete metric space. Each contraction $f : X \to X$ has a *unique* fixed point.

If $(x, y \in) X$ is any nonempty set, one can define the *discrete metric* on X ($d : X \times X \to [0, 1]$) as follows: $d(x, y) = 0$ if $x = y$, and $d(x, y) = 1$ otherwise. (X, d) is a complete ultrametric space. Other composed metric spaces can be built up using the composite metrics given in Definition 2.2.

Definition 2.2: Let $(X, d_X), (Y, d_Y)$ be (ultra) metric spaces. On $(x \in) X$, $(f \in) X \to Y$ (the function space), $(x, y) \in X \times Y$ (the Cartesian product), $u, v \in X + Y$ (the disjoint union of X and Y, which can be defined by: $X + Y = (\{1\} \times X) \cup (\{2\} \times Y)$), and $U, V \in \mathcal{P}(X)$ (the power set of X) one can define the following metrics:

(a) $d_{\frac{1}{2}\cdot X} : X \times X \to [0, 1]$
$d_{\frac{1}{2}\cdot X}(x_1, x_2) = \frac{1}{2} \cdot d_X(x_1, x_2)$

(b) $d_{X \to Y} : (X \to Y) \times (X \to Y) \to [0, 1]$
$d_{X \to Y}(f_1, f_2) = \sup_{x \in X} d_Y(f_1(x), f_2(x))$

(c) $d_{X \times Y} : (X \times Y) \times (X \times Y) \to [0, 1]$
$d_{X \times Y}((x_1, y_1), (x_2, y_2)) =$
$max\{d_X(x_1, x_2), d_Y(y_1, y_2)\}$

(d) $d_{X+Y} : (X + Y) \times (X + Y) \to [0, 1]$
$d_{X+Y}(u, v) =$
if $(u, v \in X)$ then $d_X(u, v)$
else if $(u, v \in Y)$ then $d_Y(u, v)$ else 1

(e) $d_H : \mathcal{P}(X) \times \mathcal{P}(X) \to [0, 1]$:
$d_H(U, V) = max\{\sup_{u \in U} d(u, V), \sup_{v \in V} d(v, U)\}$
where $d(u, W) = \inf_{w \in W} d(u, w)$ and by convention $\sup \emptyset = 0$, $\inf \emptyset = 1$ (d_H is the *Hausdorff* metric).

We use the abbreviation $\mathcal{P}_{nco}(\cdot)$ to denote the power set of *non-empty and compact* subsets of '$\cdot$'. Also, we often suppress the metrics part in domain definitions, and write, e.g., $\frac{1}{2} \cdot X$ instead of $(X, d_{\frac{1}{2}\cdot X})$.

Remark 2.3: Let $(X, d_X), (Y, d_Y), d_{\frac{1}{2}\cdot X}, d_{X \to Y}, d_{X \times Y}$, d_{X+Y} and d_H be as in Definition 2.2. In case d_X, d_Y are ultrametrics, so are $d_{\frac{1}{2}\cdot X}, d_{X \to Y}, d_{X \times Y}, d_{X+Y}$ and d_H. Moreover, if $(X, d_X), (Y, d_Y)$ are complete then $\frac{1}{2} \cdot X$, $X \to Y$, $X \stackrel{1}{\to} Y$, $X \times Y$, $X + Y$, and $\mathcal{P}_{\mathrm{nco}}(X)$ (with the metrics defined above) are also complete metric spaces [3].

We also use the abbreviation $\mathcal{P}_{finite}(\cdot)$ to denote the power sets of *finite* subsets of '$\cdot$'. In general, the construction $\mathcal{P}_{finite}(\cdot)$ does not give rise to a complete space. When necessary, we use it to define structures that we endow with the discrete metric. Any set endowed with the discrete metric is a complete ultrametric space.

III. Syntax of $\mathcal{L}_{MR}$ and informal explanation

Let $(o \in) O$ be an alphabet of *objects*. We assume that O is a (finite or a) countable set. Let $(w \in) W = [O]$ be the set of all finite multisets of O objects; the construction $[\cdot]$ was defined in Section II. Let also $(M \in) Mname$ be a set of *membrane names*. The abstract syntax of $\mathcal{L}_{MB}$ is given in Definition 3.1.

Definition 3.1: (Syntax of $\mathcal{L}_{MB}$)

(a) (Statements) $x(\in X) ::= o \mid \mathsf{new}(M, x) \mid x \| x$

(b) (Rules) $r(\in R) ::= r_\epsilon \mid w \Rightarrow x; r$

(c) (Membrane declarations)
$d(\in MD) ::= \mathsf{membrane}\ M\,\{r\}$
$D(\in MDs) ::= d \mid d; D$

(d) (Programs) $\rho(\in \mathcal{L}_{MB}) ::= D; x$

An $\mathcal{L}_{MB}$ statement may be either an object, or a membrane creation statement of the form $\mathsf{new}(M, x)$, where M is a membrane name and x is a statement, or a parallel composition of two $\mathcal{L}_{MB}$ statements of the form $x_1 \parallel x_2$.

An $\mathcal{L}_{MB}$ program $D; x$ consists of a list $D \in MDs$ of membrane declarations and a statement $x \in X$. A membrane declaration introduces a type of membranes, which can be instantiated. In $\mathcal{L}_{MB}$ we speak of *membrane types*, and *membrane instances*. Each membrane instance has a (unique) label; the set of labels and the set of membranes will be defined in the next Section. The execution of the program $D; x$ starts with the creation of an instance of the first membrane type in the declarations list D, which becomes a *skin* membrane; the skin membrane starts the execution of the program by executing the statement x.

A membrane declaration $\mathsf{membrane}\ M\,\{r\}$ indicates the name $M \in Mname$ of the membrane (type) and a (possibly empty) list of rules $r \in R$, which specify the behavior of objects inside any instance of a membrane of the type M. A rule $w \Rightarrow x$ is composed of two elements: a multiset $w \in W$ and a statement $x \in X$. An $\mathcal{L}_{MB}$ statement is a concurrent composition of objects, which behave as a multiset. In this interpretation $w \Rightarrow x$ is a multiset rewriting rule, specifying that w is rewritten as x.

Intuitively, a rule $w \Rightarrow x$ is like a 'procedure' definition, with 'name' w and 'body' x. The objects $o_1, \cdots, o_n$ in the multiset $w = [o_1 \cdots o_n]$ are 'fragments' of the procedure name. This intuition is based on the Join calculus [9], where procedure names are also composed of several fragments. Only when all the 'fragments' of such a 'procedure name' are prepared for interaction a rewriting rules is applied, which replaces the 'name of the procedure' with its 'body'. The 'body' (the right hand side) of a rule is a statement.

If we compare $\mathcal{L}_{MB}$ with an object oriented language a rule $w \Rightarrow x$ corresponds to an OO method declaration, the multiset w is the name of the 'method', and the statement x is the body of the 'method'. A membrane declaration $\mathsf{membrane}\ M\,\{r\}$ corresponds to an OO class declaration, and a statement $\mathsf{new}(M, x)$ a creates a new instance of a membrane of the type M to execute statement x.

Remarks 3.2:

(a) If we consider the analogy with object oriented programming, we must state clearly that in $\mathcal{L}_{MB}$ an object

$o \in O$ is *not* an instance of a membrane. In $\mathcal{L}_{MB}$ an object is just an elementary statement, a symbol taken from the alphabet O.

(b) The reader may wonder why we use the semantic notion of a *multiset* in the syntax definition of $\mathcal{L}_{MB}$. It would be easy to make a complete separation between syntax and semantics. For example, in definition 3.1 we could use rules of the form $j \Rightarrow x$, where $j ::= o \mid j \,\&\, j$ is the set of 'procedure names' or 'method names' (this syntax is also inspired by the Join calculus [9]). But we use multisets (as 'method names') because the order in which fragments occur in such a 'method name' is irrelevant.

In Section I-A we have presented a simple $\mathcal{L}_{MB}$ program, and we described its beaviour informally, using an ad-hoc notation. In the next Section we present a denotational semantics for $\mathcal{L}_{MB}$ designed with continuations and a power-domain construction [14]. The behavior of an $\mathcal{L}_{MB}$ program is described by a collection of execution traces. Each trace is a sequence of membrane configurations, showing the effect of the parallel rewriting steps. Our continuation semantics describes in a compositional manner the behavior of an $\mathcal{L}_{MB}$ program as a dynamic hierarchical system. For example, we model the semantics of the program presented in Section I-A by a collection of two execution traces; see Example 4.7.

IV. Denotational Semantics

In this section we introduce the metric domains that we need to express the behaviour of $\mathcal{L}_{MB}$ programs and we design a continuation-based denotational semantics for $\mathcal{L}_{MB}$. The final yield of the denotational semantics is an element of a linear time domain [3].

A. Final semantic domain

We assume given a (countably) infinite set $(l \in)L$ of *membrane labels*, together with a function $\nu : \mathcal{P}_{finite}(L) \to L$, such that $\nu(I) \notin I$, for any $I \in \mathcal{P}_{finite}(L)$. We obtain a possible example of such a set L and function ν by putting $L = \mathbb{N}$, and $\nu(I) = 1 + max\{n \mid n \in I\}$.

Following [2] we define the set $(\mu \in)Mb$ of *membranes* inductively.

- If $M \in Mname$ is a membrane name, $l \in L$ is a label and $w \in W = [O]$ is a multiset of O objects then $\langle M, l \mid w; \rangle \in Mb$; $\langle M, l \mid w; \rangle$ is called a (*simple* or) *elementary membrane*.
- If $M \in Mname$ is a membrane name, $l \in L$ is a label, $w \in W$ is a multiset of O objects, and $\mu_1, \ldots, \mu_n \in Mb$ then $\langle M, l \mid w; \mu_1, \ldots, \mu_n \rangle \in Mb$; $\langle M, l \mid w; \mu_1, \ldots, \mu_n \rangle$ is called a *composite membrane*.

Remark 4.1: The inner membranes $\mu_1, \ldots, \mu_n \in Mb$ of a membrane $\langle M, l \mid w; \mu_1, \ldots, \mu_n \rangle$ are not ordered. The membranes of the same level can float around [12]. We could represent this behavior by using the concept of (a multiset or) a set of sibling membranes indexed with unique labels. Alternatively, we could represent a membrane by its structure together with the multisets of objects and the rules associated with the membrane structure [12], [4]. In this paper we do not consider in detail these alternative options, which can all be easily handled within the basic set theory. We consider a particular representation of the set of membranes Mb. In the present metric setting Mb is essentially a set endowed with the discrete metric (see below).

We design a continuation-based denotational semantics for $\mathcal{L}_{MB}$. The domain of continuations is introduced in Section IV-B. The final yield of our denotational semantics is an element of the linear time domain $\mathbf{P}$.

$$(p \in)\mathbf{P} = \mathcal{P}_{nco}(\mathbf{Q})$$

$$(q \in)\mathbf{Q} \cong \{\epsilon\} + (Mb \times \frac{1}{2} \cdot \mathbf{Q})$$

In this domain equation the set Mb is endowed with the discrete metric, which is an ultrametric. The composed metric spaces are built up using the composite metrics given in Definition 2.2. As explained in [3], the above domain equation has a *unique* solution (up to isomorphism). The solution is a complete ultrametric space.

An element of the type $\mathbf{P}$ is a non-empty and compact collection of $\mathbf{Q}$ sequences. $\mathbf{Q}$ is a domain of finite and infinite sequences over Mb. ϵ models the empty sequence. Instead of $(\mu_1, (\mu_2, \ldots (\mu_n, \epsilon) \ldots))$ and $(\mu_1, (\mu_2, \ldots))$, we write $\mu_1\mu_2 \ldots \mu_n$ and $\mu_1\mu_2 \ldots$, respectively.

Also, we use the following notations: $\mu \cdot q = (\mu, q)$ and $\mu \cdot p = \{\mu \cdot q \mid q \in p\}$, for any $\mu \in Mb, q \in \mathbf{Q}, p \in \mathbf{P}$.

B. Semantics of $\mathcal{L}_{MB}$ statements

We define the domain $\mathbf{D}$ of *computations* (*denotations*) and the domain $\mathbf{F}$ of *continuations* by:

$$(\varphi \in)\mathbf{D} = Mb \to \mathbf{F} \stackrel{1}{\to} \mathbf{P}$$

$$(f \in)\mathbf{F} = Mb \to \mathbf{P}$$

In the above domain definitions the set Mb (of membranes) is endowed with the discrete metric.

Nondeterministic behaviour in $\mathcal{L}_{MB}$ is defined by using the operator $+ : (\mathbf{P} \times \mathbf{P}) \to \mathbf{P}$

$$p_1 + p_2 = \{q \mid q \in p_1 \cup p_2, q \neq \epsilon\} \cup \{\epsilon \mid \epsilon \in p_1 \cap p_2\}$$

It is easy to check that $+$ is well-defined, non-expansive, associative and commutative [3].

Definition 4.2: (Semantics of parallel composition)

(a) We define $\|: (\mathbf{D} \times \mathbf{D}) \stackrel{1}{\to} \mathbf{D}$ as follows:

$$\varphi_1 \parallel \varphi_2 = \lambda\mu \,.\, \lambda f. \,((\varphi_1 \lfloor \varphi_2)(\mu)(f) + (\varphi_2 \lfloor \varphi_1)(\mu)(f))$$

where $\lfloor : (\mathbf{D} \times \mathbf{D}) \stackrel{1}{\to} \mathbf{D}$ is given by

$$(\varphi_1 \lfloor \varphi_2) = \lambda\mu \,.\, \lambda f. \,\varphi_1(\mu)(\lambda\mu_1.\varphi_2(\mu_1)(f))$$

(b) For any $n \in \mathbb{N}$ we define $\|^n\ (\cdot) : \mathbf{D}^n \to \mathbf{D}$ ($\mathbf{D}^n = \mathbf{D} \times \cdots \times \mathbf{D}$ - n times, $n \geq 1$) by:

$\|^1\ (\varphi) = \varphi$

$\|^{n+1}\ (\varphi_1, \ldots, \varphi_{n+1}) = \varphi_1 \parallel (\|^n\ (\varphi_2, \ldots, \varphi_{n+1}))$

For easier readability, instead of $\|^n\ (\varphi_1, \ldots, \varphi_n)$ we write $\varphi_1 \parallel \cdots \parallel \varphi_n$, for any $n \in \mathbb{N}$.

The semantics of parallel composition is modeled in continuation semantics as a non-deterministic choice between two alternative computations: one starting from the first statement and another starting from the second [7], [16].

One can check that the operator for parallel composition $\parallel$ is well-defined and non-expansive in the both arguments. Also, the definition of $\varphi_1 \parallel \varphi_2$ is symmetric and $+$ is commutative, hence $\parallel$ is commutative. However, other properties, e.g., the associativity of $\parallel$, are more difficult to establish. A formal proof of the fact that $\parallel$ is associative could employ the technique introduced in [7]. In [7], [8] each (nontrivial) semantic property is proved by identifying a corresponding invariant of the computation, as a relation between continuation structures. The identication of semantic properties from the invariants of the computation is common in bisimulation semantics [10]. In [7], [8] this idea is adapted to a denotational framework, by using arguments of the kind $\varepsilon \leq \frac{1}{2} \cdot \varepsilon \Rightarrow \varepsilon = 0$, which are standard in metric semantics [3]. ε is the distance between two behaviorally equivalent continuations, before and after a computation step, respectively. The effect of each computation step is given by the $\frac{1}{2} \cdot$ -contracting factor. Hence $\varepsilon = 0$ and the desired property follows.

In order to define the semantics of $\mathcal{L}_{MB}$ statements we need some auxiliary operators on membranes. Each membrane determines a *compartment*, also called a *region* [11]. Membranes produce a demarcation between regions. For each membrane there is a unique associated region. Because of this one-to-one correspondence we sometimes use the term membrane instead of region. Also, the membranes (and the corresponding regions) are labelled in a one-to-one manner with labels from the given set L. We define operators to generate new (fresh) membrane labels, to add objects to membranes, and to create new nested membranes.

The operator $new_L : Mb \to L$ takes as parameter a membrane μ and generates a new label that is not contained in μ. It uses the mapping ν introduced in Section IV-A and an auxiliary mapping $labels : Mb \to \mathcal{P}_{finite}(L)$, which computes the finite set of labels contained in a membrane.

$$new_L(\mu) = \nu(\, labels(\mu))$$
$$labels(\langle M, l \mid w; \rangle) = \{l\}$$
$$labels(\langle M, l \mid w; \mu_1, \ldots, \mu_n \rangle) = \{l\} \cup labels(\mu_1) \cup \cdots \cup labels(\mu_n)$$

Given a membrane μ, $add(o, l', \mu)$ adds the object o to the multiset stored in the membrane region indicated by label l'. $new_M(M_{new}, l_{new}, l', \mu)$ creates a new membrane region with label l_{new} of the type M_{new} as an inner membrane (a child) of the membrane region with label l'. The operators are defined by induction on the structure of membranes, considering the fact that membranes are labeled in a one-to-one manner with labels from the given set L. We only provide the definitions for elementary membranes, leaving the case of composite membranes to the reader. The Haskell implementation of these operators is available from [17]. In the definition of add, $\uplus$ is the multiset sum operation, presented in Section II.

$$add : (O \times L \times Mb) \to Mb$$
$$add(o, l', \langle M, l \mid w; \rangle) = \begin{cases} \langle M, l \mid [o] \uplus w; \rangle & \text{if } l' = l \\ \langle M, l \mid w; \rangle & \text{if } l' \neq l \end{cases}$$

$$new_M : (Mname \times L \times L \times Mb) \to Mb$$
$$new_M(M_{new}, l_{new}, l', \langle M, l \mid w; \rangle) = \begin{cases} \langle M, l \mid w; \langle M_{new}, l_{new} \mid []; \rangle \rangle & \text{if } l' = l \\ \langle M, l \mid w; \rangle & \text{if } l' \neq l \end{cases}$$

Definition 4.3: (Denotational semantics $[\![\cdot]\!]$ of $\mathcal{L}_{MB}$ statements) We define $[\![\cdot]\!] : X \to L \to \mathbf{D}$ by:

$$\begin{aligned} [\![o]\!](l) &= \lambda\mu\,.\,\lambda f\,.\, f(\, add(o, l, \mu)) \\ [\![\,\mathsf{new}(M', x)]\!](l) &= \lambda\mu\,.\,\lambda f\,.\, [\![x]\!](l')(\, new_M(M', l', l, \mu))(f) \\ &\quad \text{where } l' = new_L(\mu) \\ [\![x_1 \parallel x_2]\!](l) &= [\![x_1]\!](l) \parallel [\![x_2]\!](l) \end{aligned}$$

The denotational mapping $[\![x]\!](l)$ takes as parameter a statement $x \in X$ and a label $l \in L$. The parameter l is the membrane label where the statement x is (to be) executed. If $x = o$, where $o \in O$ is an object, then o is simply added to the membrane μ in the region with label l and the control is transmitted to the continuation (f). If $x = \mathsf{new}(M', x)$, for some $M' \in Mname$ and $x \in X$, then a new membrane region with label l' is created, as an inner element of the membrane with label l. The statement x is executed in this newly created membrane region. The semantics of parallel composition is defined based on the semantic operator $\parallel$ introduced in Definition 4.2. We assume that function application binds stronger than the operator for parallel composition $\parallel$. The expression $[\![x_1]\!](l) \parallel [\![x_2]\!](l)$ reads as follows: $([\![x_1]\!](l)) \parallel ([\![x_n]\!](l))$.

C. Some auxiliary operators

The mapping $appRules(r, w')$ computes a (finite) set of pairs, where each pair consists of a multiset of rewriting rules applicable to w' and an irreducible (sub)multiset (of w'). It may be defined as follows:

$$appRules : (R \times W) \to \mathcal{P}_{finite}(R \times W)$$
$$appRules(r, w) = \text{if } aux(r, w) = \emptyset \text{ then } \{(r_\epsilon, w)\} \text{ else } \{(\overline{w} \Rightarrow \overline{x} \,\square\, r', w'') \mid ((\overline{w}, \overline{x}), w') \in aux(r, w),\ (r', w'') \in appRules(r, w')\}$$
$$aux : (R \times W) \to \mathcal{P}_{finite}((W \times X) \times W)$$
$$aux(r_\epsilon, w) = \emptyset$$

$$aux(w' \Rightarrow x' \,\square\, r, w) =$$
$$\quad \mathsf{if}\ (w' \subseteq w)\ \mathsf{then}\ \{((w', x'), w \setminus w')\} \cup aux(r, w)$$
$$\quad \mathsf{else}\ aux(r, w)$$

The definitions of mappings $appRules(r, w)$ and $aux(r, w)$ can be justified by an easy induction (on the number of elements in the multiset w, respectively by induction on the length of list r). In the definition of aux, $\subseteq$ and $\setminus$ are the operations for multiset inclusion and multiset difference, respectively, introduced in Section II.

Let also $rules : (MDs \times Mname) \to R$ be given by:

$$rules(\ \mathsf{membrane}\ M'\,\{r\}, M) = \begin{cases} r & \mathsf{if}\ M' = M \\ r_\epsilon & \mathsf{if}\ M' \neq M \end{cases}$$

$$rules(\ \mathsf{membrane}\ M'\,\{r\}; D, M) = \begin{cases} r & \mathsf{if}\ M' = M \\ rules(D, M) & \mathsf{if}\ M' \neq M \end{cases}$$

We define a scheduler mapping $sched : (Mb \times MDs) \to \mathcal{P}_{finite}(\mathbf{D} \times Mb)$. The function $sched(\mu, D)$ takes as arguments a membrane μ and a list of membrane declarations D. It yields a finite set of pairs, each pair consisting of a computation (denotation) and a corresponding membrane. $sched(\mu, D)$ is defined by induction on the structure of membrane μ. It uses $appRules$ to compute the applicable rules for each membrane region. In Section IV-D we use the mapping $sched$ in a fixed point construction, which we need to define the semantics of parallel rewriting of multisets in a compositional manner.

$$\begin{array}{l} sched(\langle M, l \mid w; \rangle, D) = \\ \quad \{([\![x_1]\!](l) \parallel \cdots \parallel [\![x_n]\!](l) \parallel [\![o_1]\!](l) \parallel \cdots \parallel [\![o_m]\!](l), \\ \qquad \langle M, l \mid []; \rangle) \\ \quad \mid (r', w') \in appRules(rules(D, M), w), \\ \qquad r' = w_1 \Rightarrow x_1; \ldots; w_n \Rightarrow x_n; r_\epsilon, \\ \qquad w' = [o_1, \ldots, o_m]\} \\ sched(\langle M, l \mid w; \mu_1, \ldots, \mu_k \rangle, D) = \\ \quad \{([\![x_1]\!](l) \parallel \cdots \parallel [\![x_n]\!](l) \parallel [\![o_1]\!](l) \parallel \cdots \parallel [\![o_m]\!](l) \parallel \\ \qquad \varphi_1 \parallel \cdots \parallel \varphi_k, \langle M, l \mid []; \mu'_1, \ldots, \mu'_k \rangle) \\ \quad \mid (r', w') \in appRules(rules(D, M), w), \\ \qquad r' = w_1 \Rightarrow x_1; \ldots; w_n \Rightarrow x_n; r_\epsilon, \\ \qquad w' = [o_1, \ldots, o_m], \\ \qquad (\varphi_1, \mu'_1) \in sched(\mu_1, D), \ldots, \\ \qquad (\varphi_k, \mu'_k) \in sched(\mu_k, D)\} \end{array}$$

We also define a mapping $haltMb : (Mb \times MDs) \to Bool$, which (given list of membrane declarations) decides whether the membrane system has reached a halting configuration. $haltMb$ is defined with the aid of an auxiliary mapping $haltM : (Mname \times MDs \times W) \to Bool$, based on the mapping $appRules$.

$$\begin{array}{l} haltMb(\langle M, l \mid w; \rangle, D) = haltM(M, D, w) \\ haltMb(\langle M, l \mid w; \mu_1, \ldots, \mu_n \rangle, D) = \\ \quad haltM(M, D, w) \wedge \\ \quad haltMb(\mu_1, D) \wedge \cdots \wedge haltMb(\mu_n, D) \end{array}$$

$$\begin{array}{l} haltM(M, D, w) = \\ \quad (\, appRules(\, rules(D, M), w) = \{(r_\epsilon, w)\}) \end{array}$$

D. Semantics of $\mathcal{L}_{MB}$ programs

In order to define the semantics of an $\mathcal{L}_{MB}$ program ρ we must take into account the information about membrane declarations and the rewriting rules contained in ρ. We define the semantics of $\mathcal{L}_{MB}$ programs based on a fixed point construction, which is needed to handle the semantics parallel rewriting of multisets in a compositional manner.

Definition 4.4: Let $\Psi : MDs \to \mathbf{F} \to \mathbf{F}$ be given by:

$$\begin{array}{l} \Psi(D)(f)(\mu) = \\ \quad \mu \cdot \{\ \mathsf{if}\ (\, haltMb(\mu, D))\ \mathsf{then}\ \{\epsilon\} \\ \qquad\qquad \mathsf{else}\ + \{\, \varphi(\mu')(f) \mid (\varphi, \mu') \in sched(\mu, D)\} \end{array}$$

For any $D \in MDs$, we define the *initial continuation* $f_0 \in \mathbf{F}$, by $f_0 = fix(\Psi(D))$.

Definition 4.4 is justified by Lemma 4.5, which states that $\Psi(D)$ is $\frac{1}{2}$ contractive, for any $D \in MDs$. Hence, according to Banach's Theorem 2.1, $\Psi(D)$ has a *unique* fixed point, for any $D \in MDs$.

Lemma 4.5: $\Psi(D) \in \mathbf{F} \xrightarrow{\frac{1}{2}} \mathbf{F}$, for any $D \in MDs$, i.e.

$$d(\Psi(D)(f_1)(\mu), \Psi(D)(f_2)(\mu)) \leq \tfrac{1}{2} \cdot d(f_1, f_2)$$

for any $D \in MDs, \mu \in Mb, f_1, f_2 \in \mathbf{F}$.

Proof: If $haltMb(\mu, D) = true$ then

$$d(\Psi(D)(f_1)(\mu), \Psi(D)(f_2)(\mu)) = d(\{\epsilon\}, \{\epsilon\}) = 0$$

Otherwise, if $haltMb(\mu, D) = false$, we have

$$\begin{array}{l} d(\Psi(D)(f_1)(\mu), \Psi(D)(f_2)(\mu)) \\ = \frac{1}{2} \cdot d(+\{\, \varphi(\mu')(f_1) \mid (\varphi, \mu') \in sched(\mu, D)\} \\ \qquad\qquad +\{\, \varphi(\mu')(f_2) \mid (\varphi, \mu') \in sched(\mu, D)\}) \\ [+ \text{ is non-expansive}] \\ \leq \frac{1}{2} \cdot max\{d(\varphi(\mu')(f_1), \varphi(\mu')(f_2)) \\ \qquad\qquad \mid (\varphi, \mu') \in sched(\mu, D)\} \qquad [\varphi \in \mathbf{D}] \\ \leq \frac{1}{2} \cdot d(f_1, f_2) \end{array}$$

■

Definition 4.6: (Semantics of $\mathcal{L}_{MB}$ programs) We define $\mathcal{D}[\![\cdot]\!] : \mathcal{L}_{MB} \to \mathbf{P}$ by:

$$\mathcal{D}[\![D; x]\!] = [\![x]\!](l_0)(\mu_0)(f_0)$$

where $D =$ membrane $M_0\,\{r_0\}; \cdots;$ membrane $M_m\,\{r_m\}$, $f_0 = fix(\Psi(D))$, $l_0 = \nu(\emptyset)$ and $\mu_0 = \langle M_0, l_0 \mid []; \rangle$.

The function $\mathcal{D}[\![\rho]\!]$ defines the semantics of an $\mathcal{L}_{MB}$ program $\rho = D; x$, where $D \in MDs$ and $x \in X$. The execution of an $\mathcal{L}_{MB}$ program begins with the creation of the skin membrane, which is an instance of the first membrane type in the declaration list D. The label of the skin membrane is $\nu(\emptyset)$. The statement x is executed in the skin membrane. The initial continuation f_0 contains the information about the multiset rewriting rules that define the behavior of each membrane compartment, based on the membrane declarations. Intuitively, a rewriting rule is treated as a procedure or method declaration, and the execution of a rewriting rule is treated as a procedure or method call. In the spirit of denotational semantics, $f_0 \in \mathbf{F}$ is defined (see Definition 4.4) as fixed point of an appropriate higher-order mapping Ψ.

Example 4.7: We consider again the $\mathcal{L}_{MB}$ program $\rho = D; x$ given in Section I-A, where D is:

$$\begin{array}{l}
\mathsf{membrane}\ M_0\,\{ \\
\quad [o_1, o_3] \Rightarrow o_2 \parallel o_4; \\
\quad [o_2] \Rightarrow o_5 \parallel \mathsf{new}(M_1, o_1 \parallel o_5); \\
\quad [o_2] \Rightarrow o_4; \\
\quad [o_5] \Rightarrow o_4 \\
\}; \\
\mathsf{membrane}\ M_1\,\{ \\
\quad [o_1] \Rightarrow o_2; \\
\quad [o_2] \Rightarrow o_3 \\
\}
\end{array}$$

and $x = o_1 \parallel o_3$.

Let $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_6 \in Mb$

$\mu_1 = \langle M_0, l_0 \mid [o_1, o_3]; \rangle$
$\mu_2 = \langle M_0, l_0 \mid [o_2, o_4]; \rangle$
$\mu_3 = \langle M_0, l_0 \mid [o_4, o_4]; \rangle$
$\mu_4 = \langle M_0, l_0 \mid [o_5, o_4]; \langle M_1, l_1 \mid [o_1, o_5]; \rangle \rangle$
$\mu_5 = \langle M_0, l_0 \mid [o_4, o_4]; \langle M_1, l_1 \mid [o_2, o_5]; \rangle \rangle$
$\mu_6 = \langle M_0, l_0 \mid [o_4, o_4]; \langle M_1, l_1 \mid [o_3, o_5]; \rangle \rangle$

One can check that $\mathcal{D}[\![\rho]\!] = \{\mu_1\mu_2\mu_3, \mu_1\mu_2\mu_4\mu_5\mu_6\}$. The calculations involved are somewhat laborious. In order to compute automatically the meaning of $\mathcal{L}_{MB}$ programs one can use the semantic interpreter available from [17], which is a Haskell implementation of the denotational mapping given in this paper. A couple of $\mathcal{L}_{MB}$ example programs (including the $\mathcal{L}_{MB}$ example discussed above) are provided and can be tested by using the semantic interpreter available from [17].

V. Concluding remarks and future research

We present a denotational semantics for a simple abstract concurrent language $\mathcal{L}_{MB}$, embodying a representative set of features encountered in membrane computing [11]. In $\mathcal{L}_{MB}$ computations are specified by means of multiset rewriting rules distributed into membrane-delimited compartments. The parallel composition operator is interpreted based on the concept of maximal parallelism and computations are specified by means of multiset rewriting rules. In the semantic design we employ continuations for concurrency and the mathematical methodology of metric semantics [3].

In the near future we plan to further develop the semantic model given in this paper in order to obtain metric denotational descriptions for the various computing phenomena encountered in the full model of membrane computing [11].

References

[1] P. America, J.J.M.M. Rutten, "Solving Reflexive Domain Equations in a Category of Complete Metric Spaces," *J. of Comput. System Sci.*, vol. 39, pp. 343–375, 1989.

[2] O. Andrei, G. Ciobanu and D. Lucanu, "A Rewriting Logic Framework for Operational Semantics of Membrane Systems," *Theoretical Computer Science*, vol. 373, pp. 163–181, 2007.

[3] J.W. de Bakker, E.P. de Vink, *Control Flow Semantics*, MIT Press, 1996.

[4] G. Ciobanu, "Semantics of P Systems," *Handbook of Membrane Computing*, Oxford University Press, pp. 413-436, 2009.

[5] G. Ciobanu, E.N. Todoran, "Metric Denotational Semantics for Parallel Rewriting of Multisets," *Proceedings of 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2011)*, pp. 276–283, IEEE Computer Press, 2011.

[6] G. Ciobanu, E.N. Todoran, "Relating Two Metric Semantics for Parallel Rewriting of Multisets," *Proceedings of 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*, pp. 273–280, IEEE Computer Press, 2012.

[7] G. Ciobanu, E.N. Todoran, "Continuation Semantics for Asynchronous Concurrency," *Fundamenta Informaticae*, vol. 131(3-4), pp. 373–388, 2014.

[8] G. Ciobanu, E.N. Todoran, "Continuation Semantics for Concurrency with Multiple Channels Communication," *Proceedings of 17th International Conference on Formal Engineering Methods (ICFEM 2015)*, *Lecture Notes in Computer Science*, vol. 9407, pp. 400–416, Springer, 2015.

[9] C. Fournet, G. Gonthier, "The Join Calculus: a Language for Distributed Mobile Programming," *Lecture Notes in Computer Science*, vol. 25, pp. 68–96, 2002.

[10] R. Milner, *Communicationg and mobile systems: the π calculus,* Cambridge University Press, 1999.

[11] Gh. Păun, *Membrane Computing. An Introduction.* Springer, 2002.

[12] Gh. Păun, "Introduction to Membrane Computing," G. Ciobanu, M.J. Perez-Jimenez, Gh. Paun, Editors, *Applications of Membrane Computing*, Springer, 2006.

[13] S. Peyton Jones, J. Hughes (Eds.), Report on the Programming Language Haskell 98: a Non-Strict Purely Functional Language, 1999. Available at `http://www.haskell.org/`.

[14] G.D. Plotkin, "A Powerdomain Construction," *SIAM Journal of Computing*, vol.5, pp. 452–487, 1976.

[15] E.N. Todoran, "Metric semantics for synchronous and asynchronous communication: a continuation-based approach," *Electronic Notes in Theoretical Computer Science*, vol.28, pp. 119–146, Elsevier, 2000.

[16] E.N. Todoran, N. Papaspyrou, "Continuations for parallel logic programming," *Proceedings of the 2nd ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, pp. 257–267, 2000.

[17] `http://ftp.utcluj.ro/pub/users/gc/synasc2015`