Metric Denotational Semantics for Parallel Rewriting of Multisets

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 an abstract concurrent language embodying two new features: the semantics of parallel composition is based on the concept of *maximal parallelism*, and computations are specified by means of *multiset rewriting* rules. To the best of our knowledge, this is the first paper that presents a metric denotational semantics for this combination of features, encountered also in membrane computing.**

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

I. Introduction

In this paper we present a denotational semantics of an abstract concurrent language $\mathcal{L}_{MR}$ embodying two features: the semantics of parallel composition is based on the concept of *maximal parallelism*, and computations are specified by means of *multiset rewriting* rules.

In $\mathcal{L}_{MR}$, all the applicable multiset rewriting rules are applied in parallel. Parallel computations proceed simultaneously, without using interleaving. When several combinations of rules are applicable, the selection of (the combination of) rules is nondeterministic.

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 construct is determined solely on the basis of of the meanings of its components. In general, denotational semantics assigns to every construct 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 [4] which states that contracting functions on complete metric spaces have *unique* fixed points.

In classic (order-theoretic) domains [9] one typically works with *least* fixed points of continuous functions. Compared to classic domains, metric spaces employ additional information. One can (compare and even) measure the distance between any two elements of a metric space. In classic domains the order is partial (not all elements are comparable), and there is no quantitative measure of how much two elements differ. In semantics, the extra information may be both a strength and a weakness of metric techniques. Metric spaces may be more appropriate for models that are naturally characterized by unique fixed points. On the other hand, the use of metric structures may make certain optimizations more difficult to achieve. For example, in some applications it may be difficult to avoid using *hiatons* or *silent steps* to achieve the contractiveness of the semantic operators [3].

In the design of the denotational semantics of $\mathcal{L}_{MR}$ we use the continuation semantics for concurrency (CSC) technique [13]. We need the theory developed in [1] for solving reflexive domain equations in a category of complete metric spaces as continuations in the CSC approach are elements of a complete space which is the solution of a domain equation where the domain variable occurs in the left-hand side of a function space construction. The denotational semantics of $\mathcal{L}_{MR}$ produces as meaning a pair consisting of a multiset and a mapping from continuations to collections of sequences of multisets. For a given $\mathcal{L}_{MR}$ statement x, the first component (the multiset) is a semantic representation of the interaction capability specified by the statement x, and the second component describes the behavior of the language construct in continuation semantics. Our denotational model captures the meaning of an $\mathcal{L}_{MR}$ program as a collection of sequences of multisets with *no silent steps* intersperesed.

A. Continuation semantics for concurrency

The distinctive characteristic of the continuation semantics for concurrency (CSC) technique is the modeling of continuations as application-specific structures of computations (denotations). Intuitively, a denotational model designed with CSC is a semantic formalization of a process scheduler; the scheduling policy is given by the structure of continuations. In the particular case of $\mathcal{L}_{MR}$ a continuation is a multiset of computations that are evaluated in parallel.

The evaluation by maximal parallel rewriting in $\mathcal{L}_{MR}$ allows to express repetitive (including non-terminating) computations. In this paper it is shown that by using continuations and classic fixed-point constructions, the concepts of maximal parallelism and multiset rewriting can be handled denotationally using techniques from metric semantics. The CSC technique can be used to express tree-structured parallelism [13]. Based on ongoing research we

believe the CSC technique provides sufficient flexibility for expressing in a denotational manner the nested structure and the complex interactions of some biologically-inspired models of computation.

B. Semantic prototype interpreter

The denotational (mathematical) specification given in this paper was developed following a prototyping approach. We used the functional language Haskell [11] as a prototyping tool (and as a metalanguage) for denotational semantics.

A variant of the language $\mathcal{L}_{MR}$ was introduced in our previous work [7]. In [7] we have presented a compositional interpreter for $\mathcal{L}_{MR}$ - also designed with continuations and implemented in Haskell - which served as an initial prototype in the development of the denotational model given in this paper. Unfortunately, a straight formalization using metric spaces of the Haskell interpreter given in [7] leads to a denotational model that produces (unnatural) silent steps with no operational counterpart, that are only needed to obtain the contractiveness of the semantic operators.

We had to further adapt the model given in [7] to obtain a semantic interpreter that may serve as an accurate prototype for the denotational (mathematical) semantics given in this paper. This new Haskell prototype is available at [14]. The reader may (download and) run the interpreter to see the execution of a couple of $\mathcal{L}_{MR}$ programs, including the two example programs given in section III. The final yield of our denotational semantics models the intended behavior of an $\mathcal{L}_{MR}$ program as a collection of sequences of multisets with no silent steps interspersed.

C. Contribution

To the best of our knowledge, this is the first paper that presents a (metric) denotational semantics for a language incorporating the concepts of maximal parallelism and multiset rewriting. The denotational model is designed with metric spaces and continuation semantics for concurrency.

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. Within set theory, 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 ']'. Notice that 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 occurrences of x_1 and one occurrence 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:* $m_1 \setminus m_2$ $(\setminus : ([X] \times [X]) \to [X])$
  $\mathsf{dom}(m_1 \setminus m_2) = (\, \mathsf{dom}(m_1) \setminus \mathsf{dom}(m_2) \,) \cup$
  $\{x \mid x \in \mathsf{dom}(m_1) \cap \mathsf{dom}(m_2)\,, m_1(x) > m_2(x)\}$
  $(m_1 \setminus m_2)(x) =$
  $$\begin{cases} m_1(x) & \text{if} \quad x \in \mathsf{dom}(m_1) \setminus \mathsf{dom}(m_2) \\ m_1(x) - m_2(x) & \text{if} \quad x \in \mathsf{dom}(m_1) \cap \mathsf{dom}(m_2) \end{cases}$$
- *Submultiset:* $m_1 \subseteq m_2$ $(\subseteq : ([X] \times [X]) \to \mathit{Bool})$
  $m_1 \subseteq m_2 =$
  $(\, \mathsf{dom}(m_1) \subseteq \mathsf{dom}(m_2) \,) \wedge$
  $(\forall x \in \mathsf{dom}(m_1) : m_1(x) \leq m_2(x))$

The free commutative monoid on a set X can be taken to be the set of finite multisets with elements drawn from X.

Let $f \in X \to Y$ be a function. The function $(\, f \mid x \mapsto y \,) : X \to Y$, is defined (for $x, x' \in X, y \in Y$) by: $(\, f \mid x \mapsto y \,)(x') = \text{ if } x' = x \text{ then } y \text{ else } f(x')$. We also use the notation $(\, f \mid x_1 \mapsto y_1 \mid \cdots \mid x_n \mapsto y_n \,)$ as an abbreviation for $(\, (\, f \mid x_1 \mapsto y_1 \,) \cdots \mid x_n \mapsto y_n \,)$. 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].

1) Examples.: The following metrics are frequently used in semantic applications.

1) 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: $d(x, y) = \text{ if } x = y \text{ then } 0 \text{ else } 1$. (X, d) is a complete ultrametric space.
2) A central idea in metric semantics is to state that two computations have distance 2^{-n} whenever the first difference in their behaviors appears after n

computation steps. Let $(a \in)A$ be a nonempty set, and let $(x, y \in)A^\infty = A^* \cup A^\omega$, where $A^*(A^\omega)$ is the set of all finite (infinite) sequences over A. One can define a metric over A^∞ as follows: $d(x,y) = 2^{-\sup\{\, n \mid x(n)=y(n)\,\}}$, where $x(n)$ denotes the prefix of x of length n, in case $length(x) \geq n$, and x otherwise (by convention, $2^{-\infty} = 0$). d is a Baire-like metric. (A^∞, d) is a complete ultrametric space.

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 $X \xrightarrow{1} Y$. 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.

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

(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) $d_{X \times Y} : (X \times Y) \times (X \times Y) \to [0,1]$
$d_{X \times Y}((x_1, y_1), (x_2, y_2)) =$
$max\{d_X(x_1, x_2), d_Y(y_1, y_2)\}$
The functions
$fst : (X \times Y) \to X$ and $snd : (X \times Y) \to Y$
are defined by
$fst(x, y) = x \quad$ and $\quad snd(x, y) = y.$
Both fst and snd are nonexpansive mappings.

(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 $\frac{1}{2} \cdot X$, $X \to Y$, $X \xrightarrow{1} Y$, $X \times Y$, $X + Y$, 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 construct $\mathcal{P}_{finite}(\cdot)$ does not give rise to a complete space. In our study, we use it to create a structure 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 also $(w \in)W$ be the set of all finite multisets of O objects. The abstract syntax of $\mathcal{L}_{MR}$ is given below.

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

(a) (Statements) $\quad x(\in X) ::= o \mid x \| x$

(b) (Rules) $\quad r(\in R) ::= \epsilon \mid w \Rightarrow x \,\square\, r$

(c) (Programs) $\quad (\rho \in)\mathcal{L}_{MR} = R \times X$

An $\mathcal{L}_{MR}$ program is a pair (r, x) consisting of a set of rewriting rules $r(\in R)$ and a statement $x(\in X)$. The rules in a list $r = (w_1 \Rightarrow x_1 \,\square\, \cdots \,\square\, w_n \Rightarrow x_n)$ are assumed to be pairwise distinct. A statement may be either an object or the parallel composition of two statements.

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 [8], 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. In the semantic model that we give in this paper a statement is executed also as a multiset of computations that are evaluated in parallel. The basic idea is that a construct $w \Rightarrow x$ is the specification of a multiset rewriting rule.

The reader may wonder why we use the semantic notion of a multiset in the syntax definition of $\mathcal{L}_{MR}$. It would be easy to make a complete separation between syntax and semantics. For example, in definition 3.1 we could have used rules of the form $j \Rightarrow x$, where $j ::= o \mid j \,\&\, j$ is the set of 'procedure names' (this syntax is again inspired by the Join calculus [8]). But we use multisets (as procedure names) because the order in which fragments occur in such a 'procedure name' is irrelevant.

Let's describe informally the behavior of $\mathcal{L}_{MR}$ programs. Intuitively, in $\mathcal{L}_{MR}$ 'procedure calls' are applications of multiset rewriting rules. Moreover, all rewriting rules that can be used are applied in maximal parallel rewriting steps. We consider the $\mathcal{L}_{MR}$ program given below.

$\rho_1 = (r_1, o_1 \parallel o_2)$
$r_1 = ([o_1] \Rightarrow o_2 \| o_4 \,\square\, [o_1 o_2] \Rightarrow o_2 \,\square\, [o_2] \Rightarrow o_3)$

In general, $\mathcal{L}_{MR}$ programs can give rise to nonterminating computations. ρ_1 is just a trivial $\mathcal{L}_{MR}$ program, whose execution terminates in few steps. There are two possible

executions, which are selected in a nondeterministic manner. Here, we describe the execution of the program in an operational manner. The execution begins with the evaluation of the statement $o_1 \parallel o_2$ which reduces to the multiset $[o_1 o_2]$. The behavior is nondeterministic, because two combinations of rules can be applied: either $[o_1] \Rightarrow o_2 \| o_4$ and $[o_2] \Rightarrow o_3$, or $[o_1 o_2] \Rightarrow o_2$. If the rules $[o_1] \Rightarrow o_2 \| o_4$ and $[o_2] \Rightarrow o_3$ are selected then the multiset $[o_1 o_2]$ is rewritten as the statement $o_2 \parallel o_3 \parallel o_4$. Notice that the both rules are applied in parallel in a single step; according to the intuition given above, two 'procedure calls' are performed in parallel. The statement $o_2 \parallel o_3 \parallel o_4$ reduces to the multiset $[o_2 o_3 o_4]$. The multiset $[o_2 o_3 o_4]$ reduces to the statement o_3 and the irreducible multiset $[o_3 o_4]$ by using the third rule $[o_2] \Rightarrow o_3$; in the end it is obtained the multiset $[o_3 o_3 o_4]$, which is irreducible. If the rule $[o_1 o_2] \Rightarrow o_2$ is selected then the multiset $[o_1 o_2]$ is rewritten as o_2, which reduces to the multiset $[o_2]$. At this point the 'procedure call' $[o_2] \Rightarrow o_3$ is performed (i.e. the multiset rewriting rule is applied) and in the end o_3 reduces to the multiset $[o_3]$.

The execution of the program can be described by the following two sequences of multiset rewriting steps.

$$[o_1 o_2] \Rightarrow [o_2 o_3 o_4] \Rightarrow [o_3 o_3 o_4]$$
$$[o_1 o_2] \Rightarrow [o_2] \Rightarrow [o_3]$$

In this paper we present a denotational semantics for $\mathcal{L}_{MR}$ designed with continuations and powerdomains [12], within the mathematical framework of metric semantics [3]. The behavior of an $\mathcal{L}_{MR}$ program is described by a collection of execution traces. Each trace is a sequence of multisets, showing the effect of the maximal parallel rewriting steps. For example, we model the semantics of ρ_1 by the following collection of two traces:

$$\{\, [o_1 o_2][o_2 o_3 o_4][o_3 o_3 o_4]\,,\, [o_1 o_2][o_2][o_3]\,\}$$

IV. Continuation structure

In the semantic domain, we use continuation semantics for concurrency to express behavior. In the CSC approach, a continuation is an application specific structure of computations (denotations). It will not come as a surprise that in the particular case of $\mathcal{L}_{MR}$ a continuation is a multiset of computations that are evaluated in parallel. In the domain for the denotational semantics we implement this concept with the aid of a set $(\alpha \in) Id$ of *identifiers*. Let also $(\pi \in)\Pi = \mathcal{P}_{finite}(Id)$. We assume given a function $\nu : \Pi \to Id$, such that $\nu(\pi) \notin \pi$, for any $\pi \in \Pi$. Given a set $\pi \in \Pi$, the function ν yields a new identifier not in π. A possible example of such a set Id and function ν could be obtained by setting $Id = \mathbb{N}$ ($\mathbb{N}$ is the set of natural numbers) and $\nu(\pi) = 1 + max\{\alpha \mid \alpha \in \pi\}$.

Let $(x \in)\mathbf{X}$ be a metric domain, i.e. a complete metric space. We use the following notation:

$$\{\!|\mathbf{X}|\!\} \quad \stackrel{not.}{=} \quad \Pi \times (Id \to \mathbf{X})$$

Let $\alpha \in Id, (\pi, \varpi) \in \{\!|\mathbf{X}|\!\}$ with $\pi \in \Pi$, $\varpi \in Id \to \mathbf{X}$. We define $id : \{\!|\mathbf{X}|\!\} \to \Pi$, $id(\pi, \varpi) = \pi$. We also use the following abbreviations:

$$\begin{array}{lcll} (\pi,\varpi)(\alpha) & \stackrel{not.}{=} & \varpi(\alpha) & (\in \mathbf{X}) \\ (\pi,\varpi) \setminus \pi' & \stackrel{not.}{=} & (\pi \setminus \pi', \varpi) & (\in \{\!|\mathbf{X}|\!\}) \\ x : (\pi,\varpi) & \stackrel{not.}{=} & (\pi \cup \{\alpha\}, (\,\varpi \mid \alpha \mapsto x\,)) & (\in \{\!|\mathbf{X}|\!\}) \\ & & \text{where} \;\; \alpha = \nu(\pi) & \end{array}$$

The basic idea is that we treat (π, ϖ) as a 'function' with finite graph $\{(\alpha, \varpi(\alpha)) \mid \alpha \in \pi\}$, thus ignoring the behaviour of ϖ for any $\alpha \notin \pi$ (π is the 'domain' of (π, ϖ)). We use this mathematical structure to represent finite bags or multisets or lists of computations. The set Id is used to distinguish between multiple occurrences of a computation in such a bag. We endow both sets Id and Π with discrete metrics; every set with a discrete metric is a complete ultrametric space. The metric on $\{\!|\mathbf{X}|\!\}$ is obtained by using the composite metrics given in Definition 2.2. The operators behave as follows. $id(\pi, \varpi)$ returns the collection of identifiers for the valid computations contained in the bag (π, ϖ), $(\pi, \varpi)(\alpha)$ returns the computation with identifier α, $(\pi, \varpi) \setminus \pi$ removes the computations with identifiers in π. $x : (\pi, \varpi)$ adds the computation x to the bag; a new (fresh) identifier identifier $\alpha(\notin \pi)$ is automatically generated for x.

The reader may wonder why we use two different notations for the same concept. We represent multisets (unordered lists) of objects by using the construct $[\cdot]$ explained in section II. At the same time we use the construct $\{\!|\cdot|\!\}$ to represent (ordered) lists of computations, which we use also to model multisets of computations. $[\cdot]$ is a more abstract representation of multisets, but works only in classic set theory. At the same time we use the construct $\{\!|\cdot|\!\}$ to represent multisets of computations which are elements of a semantic domain (a complete metric space). It is not straightforward to generalize the construct $[\cdot]$ to work with semantic domains rather than plain sets. This may be a subject of future research.

V. Denotational semantics

We present a continuation-based denotational semantics for $\mathcal{L}_{MR}$. The domain definitions are given below.

$$\begin{array}{l} (\psi \in)\mathbf{D} = W \times \mathbf{F} \\ (\phi \in)\mathbf{F} \cong \mathbf{C} \stackrel{1}{\to} \mathbf{P} \\ (\gamma \in)\mathbf{C} = W \times \mathbf{K} \\ (\kappa \in)\mathbf{K} = \{\!|W \times (\tfrac{1}{2} \cdot \mathbf{F})|\!\} \\ (\eta \in)\mathbf{E} = O \to \mathbf{D} \\ (p \in)\mathbf{P} = \mathcal{P}_{nco}(\mathbf{Q}) \\ (q \in)\mathbf{Q} \cong \{\epsilon\} + (W \times (\tfrac{1}{2} \cdot \mathbf{Q})) \end{array}$$

$\mathbf{D}$ is the domain of denotations. A *denotation* $\in \mathbf{D}$ is a pair consisting of a W multiset and a computation of type $\mathbf{F}$. We put $W = [O]$. The construct $[\cdot]$ was explained in

section II. A computation of type $\mathbf{F}$ is a (non-expansive) mapping from continuations to processes of type $\mathbf{P}$. A *process* is a (non-empty and compact) collection of $\mathbf{Q}$ execution traces. A *trace* is a finite or infinite sequence of multisets, representing a possible execution of an $\mathcal{L}_{MR}$ program. $\mathbf{C}$ is the domain of (structured) CSC continuations. A $\mathbf{C}$ continuation is a pair consisting of a W multiset (representing an irreducible multiset with respect to the set of rules of an $\mathcal{L}_{MR}$ program) and a $\mathbf{K}$ multiset of computations of type $(W \times (\frac{1}{2} \cdot \mathbf{F}))$. $\mathbf{E}$ is a domain of *semantic environments* - a standard notion in denotational semantics - mapping O objects to denotations.

In the equations given above the sets O and $W(=[O])$ are endowed with the discrete metric which is an ultrametric. The composed metric spaces are built up using the metrics of Definition 2.2. To conclude that such a system of equations has a solution, which is unique up to isometry, we rely on the general method developed in [1]. The space $\mathbf{F}$ is defined by a recursive domain equation. The solution for $\mathbf{F}$ is obtained as a complete ultrametric space. In [1], the family of complete (ultra)metric spaces is made into a category $\mathcal{C}$. It is proved that a generalized form of Banach's fixed point theorem holds, stating that a functor $\mathcal{F} : \mathcal{C} \to \mathcal{C}$ that is *contracting* (in a sense) has a unique fixed point (up to isometry). Intuitively, in the equation above the relevant functor is contracting as a consequence of the fact that the recursive occurrence of $\mathbf{F}$ is preceded by a $\frac{1}{2} \cdot$ -factor.

We define the denotational mapping $[\![\cdot]\!]$ (with respect to an arbitrary semantic environment η) as follows:

$$
\begin{array}{l}
[\![\cdot]\!] : X \to \mathbf{E} \stackrel{1}{\to} \mathbf{D} \\
[\![o]\!]\eta = \eta(o) \\
[\![x_1 \parallel x_2]\!]\eta = \\
\quad \mathsf{let}\ (w_1, \phi_1) = [\![x_1]\!]\eta \\
\qquad\ (w_2, \phi_2) = [\![x_2]\!]\eta \\
\quad \mathsf{in}\ (w_1 \uplus w_2\ ,\ \lambda(w,\kappa).(\phi_1(w, (w_2, \phi_2) : \kappa) \cup \\
\qquad\qquad\qquad\qquad \phi_2(w, (w_1, \phi_1) : \kappa)))
\end{array}
$$

$\uplus$ is the multiset sum (see section II) and $\cup$ is the standard set union operator. It is easy to check that the multiset sum operation $\uplus : W \times W \to W$ (defined in section II) is nonexpansive. The denotational mapping is designed by using continuations. The second components of the yield of a denotational mapping is a computation of type $\mathbf{F}$, which is a (non-expansive) function from $\mathbf{C}$ continuations to processes of type $\mathbf{P}$. Following the continuation semantics for concurrency technique [13], the semantics of parallel composition is modeled as a non-deterministic choice between two alternative computations: one starting from the first statement and another starting from the second. The following Lemma is easily established.

Lemma 5.1: (Well-definedness of $[\![\cdot]\!]$) For all $x \in X$, $\eta \in \mathbf{E}$, $(w, \kappa) \in W \times \mathbf{K}$:

(a) $[\![x]\!]\eta \in \mathbf{D}$
(b) $(snd([\![x]\!]\eta))(w, \kappa) \in \mathbf{P}$
(c) $(snd([\![x]\!]\eta)) \in \mathbf{C} \stackrel{1}{\to} \mathbf{P}$
(d) $[\![x]\!] \in \mathbf{E} \stackrel{1}{\to} \mathbf{D}$

Proof: The proofs can proceed by structural induction on $x \in X$. We present here only the proof for Lemma 5.1(d). We show that:

$d([\![x]\!]\eta, [\![x]\!]\overline{\eta}) \leq d(\eta, \overline{\eta})$

by structural induction on $x \in X$. We use the symbol '$\equiv$' to express syntactic identity.

Case $x \equiv o$.

$d([\![o]\!]\eta, [\![o]\!]\eta) = d(\eta(o), \overline{\eta}(o)) \leq d(\eta, \overline{\eta})$

Case $x \equiv x_1 \parallel x_2$.

$$
\begin{array}{l}
d([\![x_1 \parallel x_2]\!]\eta, [\![x_1 \parallel x_2]\!]\overline{\eta}) \\
= d(\mathsf{let}\ (w_1, \phi_1) = [\![x_1]\!]\eta \\
\qquad\quad (w_2, \phi_2) = [\![x_2]\!]\eta \\
\qquad \mathsf{in}\ (w_1 \uplus w_2\ ,\ \lambda(w,\kappa).(\phi_1(w, (w_2, \phi_2) : \kappa) \cup \\
\qquad\qquad\qquad\qquad \phi_2(w, (w_1, \phi_1) : \kappa))), \\
\qquad \mathsf{let}\ (\overline{w}_1, \overline{\phi}_1) = [\![x_1]\!]\overline{\eta} \\
\qquad\quad (\overline{w}_2, \overline{\phi}_2) = [\![x_2]\!]\overline{\eta} \\
\qquad \mathsf{in}\ (\overline{w}_1 \uplus \overline{w}_2\ ,\ \lambda(w,\kappa).(\overline{\phi}_1(w, (\overline{w}_2, \overline{\phi}_2) : \kappa) \cup \\
\qquad\qquad\qquad\qquad \overline{\phi}_2(w, (\overline{w}_1, \overline{\phi}_1) : \kappa))) \\
\quad)^{(+)}
\end{array}
$$

By the induction hypothesis:

$d([\![x_1]\!]\eta, [\![x_1]\!]\overline{\eta}) = d((w_1, \phi_1), (\overline{w}_1, \overline{\phi}_1)) \leq d(\eta, \overline{\eta})$

and

$d([\![x_2]\!]\eta, [\![x_2]\!]\overline{\eta}) = d((w_2, \phi_2), (\overline{w}_2, \overline{\phi}_2)) \leq d(\eta, \overline{\eta})$

$d((w_1, \phi_1), (\overline{w}_1, \overline{\phi}_1)) = max\{d(w_1, \overline{w}_1), d(\phi_1, \overline{\phi}_1)\}$, so $d(w_1, \overline{w}_1) \leq d(\eta, \overline{\eta})$ and $d(\phi_1, \overline{\phi}_1) \leq d(\eta, \overline{\eta})$. Similarly, $d(w_2, \overline{w}_2) \leq d(\eta, \overline{\eta})$ and $d(\phi_2, \overline{\phi}_2) \leq d(\eta, \overline{\eta})$. Therefore

$$
\begin{array}{l}
^{(+)} = max\{d(w_1 \uplus w_2, \overline{w}_1 \uplus \overline{w}_2), \\
\qquad\qquad d(\lambda(w,\kappa).(\phi_1(w, (w_2, \phi_2) : \kappa) \cup \\
\qquad\qquad\qquad\qquad \phi_2(w, (w_1, \phi_1) : \kappa)), \\
\qquad\qquad\quad \lambda(w,\kappa).(\overline{\phi}_1(w, (\overline{w}_2, \overline{\phi}_2) : \kappa) \cup \\
\qquad\qquad\qquad\qquad \overline{\phi}_2(w, (\overline{w}_1, \overline{\phi}_1) : \kappa)))\}
\end{array}
$$

[$\uplus$ and $\cup$ are nonexpansive, Definition 2.2(b)]

$$
\begin{array}{l}
\leq max\{d(w_1, \overline{w}_1), d(w_2, \overline{w}_2), \\
\qquad sup_{(w,\kappa) \in W \times \mathbf{K}} \\
\qquad\quad (max\{d(\phi_1(w, (w_2, \phi_2) : \kappa), \\
\qquad\qquad\qquad \overline{\phi}_1(w, (\overline{w}_2, \overline{\phi}_2) : \kappa))^{(++)}, \\
\qquad\qquad\quad d(\phi_2(w, (w_1, \phi_1) : \kappa), \\
\qquad\qquad\qquad \overline{\phi}_2(w, (\overline{w}_1, \overline{\phi}_1) : \kappa))^{(+++)}\})\}
\end{array}
$$

We already know that $d(w_1, \overline{w}_1) \leq d(\eta, \overline{\eta})$ and $d(w_2, \overline{w}_2) \leq d(\eta, \overline{\eta})$. To obtain the desired result it suffices to show that $^{(++)} \leq d(\eta, \overline{\eta})$ for any $(w, \kappa) \in W \times \mathbf{K}$. The proof that $^{(+++)} \leq d(\eta, \overline{\eta})$ is symmetric. Indeed:

$$
\begin{array}{l}
d(\phi_1(w, (w_2, \phi_2) : \kappa), \overline{\phi}_1(w, (\overline{w}_2, \overline{\phi}_2) : \kappa)) \\
\leq max\{ \\
\qquad d(\phi_1(w, (w_2, \phi_2) : \kappa), \overline{\phi}_1(w, (w_2, \phi_2) : \kappa)),
\end{array}
$$

$$d(\overline{\phi}_1(w,(w_2,\phi_2):\kappa),\overline{\phi}_1(w,(\overline{w}_2,\overline{\phi}_2):\kappa))$$
$$\}$$

We now that $d(\phi_1,\overline{\phi}_1) \leq d(\eta,\overline{\eta})$ and $d(\phi_2,\overline{\phi}_2) \leq d(\eta,\overline{\eta})$ therefore:

$$d(\phi_1(w,(w_2,\phi_2):\kappa),\overline{\phi}_1(w,(w_2,\phi_2):\kappa))$$
$$\leq d(\phi_1,\overline{\phi}_1) \leq d(\eta,\overline{\eta})$$

and

$$d(\overline{\phi}_1(w,(w_2,\phi_2):\kappa),\overline{\phi}_1(w,(\overline{w}_2,\overline{\phi}_2):\kappa))$$
[Lemma 5.1(c)]
$$\leq d((w,(w_2,\phi_2):\kappa),(w,(\overline{w}_2,\overline{\phi}_2):\kappa))$$
$$= d((w_2,\phi_2),(\overline{w}_2,\overline{\phi}_2)) \leq d(\eta,\overline{\eta})$$

■

In 5.3 we define a semantic environment that captures the rewriting semantics of the rules of an $\mathcal{L}_{MR}$ program. The rewriting rules are applied in a maximally parallel manner. Execution terminates ($\{\epsilon\}$ is produced as observable result) when no rule is applicable. Let $\{\!|\,|\!\} = (\emptyset, \lambda\alpha.([], \lambda\gamma.\{\epsilon\})) \in \mathbf{K}$. It is convenient to use the following notations:

$$\{\!|(w_1,\phi_1),(w_2,\phi_2)\cdots,(w_n,\phi_n)|\!\} \stackrel{not.}{=}$$
$$(w_1,\phi_1):((w_2,\phi_2):\cdots((w_n,\phi_n):\{\!|\,|\!\})\cdots)$$
$$\{\!|(w_i,\phi_i) \mid i \in I = \{i_1,\cdots,i_m\}|\!\} \stackrel{not.}{=}$$
$$\{\!|(w_{i_1},\phi_{i_1}),\cdots,(w_{i_m},\phi_{i_m})|\!\}$$

The semantic environment is defined as fixed point of an appropriate higher-order mapping.

$$\Phi : R \to \mathbf{E} \to \mathbf{E}$$
$$\Phi(r)(\eta)(o) =$$
$$([o],$$
$$\lambda(w,\kappa).$$
$$\mathsf{let}\ w' = [o] \uplus w \uplus (\biguplus_{\alpha\in id(\kappa)} fst(\kappa(\alpha)))$$
$$\varrho = appRules(r,w')$$
$$\mathsf{in}\ w' \cdot (\ \mathsf{if}\ \varrho = \{(\epsilon,\overline{w})\}\ \mathsf{then}\ \{\epsilon\}$$
$$\mathsf{else}\ \bigcup_{(r,w'')\in\varrho}\ exe(r,w'',\eta)))$$

where

$$exe(w_1 \Rightarrow x_1 \,\square\, \cdots \,\square\, w_n \Rightarrow x_n, w'', \eta) =$$
$$\mathsf{let}\ I = \{1,\cdots,n\}$$
$$\mathsf{in}\ \bigcup_{i\in I}(snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\eta \mid j \in I \setminus \{i\}|\!\})$$

We use '$\cdot$' as a prefixing operator over ($\mathbf{Q}$) sequences: $w{\cdot}q = (w,q)$, for $q \in \mathbf{Q}$. Also, we use the notation

$$w \cdot p = \{w \cdot q \mid q \in p\}$$

for any $p \in \mathbf{P}$. Notice that $d(w\cdot p_1, w\cdot p_2) = \frac{1}{2}\cdot d(p_1,p_2)$ and $d(w_1\cdot p_1, w_2\cdot p_2) = 1$ when $w_1 \neq w_2$, because W is endowed with the discrete metric.

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:

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

where

$$aux : (R\times W) \to \mathcal{P}_{finite}((W\times X)\times W)$$
$$aux(\epsilon,w) = \emptyset$$
$$aux(w' \Rightarrow x' \,\square\, r, w) =$$
$$\mathsf{if}\ (w' \subseteq w)$$
$$\mathsf{then}\ \{((w',x'), w\setminus w')\} \cup aux(r,w)$$
$$\mathsf{else}\ aux(r,w)$$

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 w multiset, respectively by induction on the length of list r).

The definition of Φ is justified by the following Lemma.

Lemma 5.2: For all $r \in R, \eta \in \mathbf{E}$ and $o \in O$:

(a) $\Phi(r)(\eta)(o) \in \mathbf{D}$

(b) $\Phi(r) \in \mathbf{E} \stackrel{\frac{1}{2}}{\to} \mathbf{E}$

Proof:

(a) It is obvious that $fst(\Phi(r)(\eta)(o)) \in W$. In addition one has to prove that $snd(\Phi(r)(\eta)(o))$ is a nonexpansive mapping ($\in \mathbf{C} \stackrel{1}{\to} \mathbf{P}$). We leave this as an exercise to the reader.

(b) We show that for any $r \in R$, $\eta,\overline{\eta} \in \mathbf{E}, o \in O$

$$d(\Phi(r)(\eta)(o), \Phi(r)(\overline{\eta})(o)) \leq \tfrac{1}{2}\cdot d(\eta,\overline{\eta})$$

We compute as follows:

$$d(\Phi(r)(\eta)(o), \Phi(r)(\overline{\eta})(o))$$
$$= d(([o],$$
$$\lambda(w,\kappa).\ \mathsf{let}\ \overline{w} = \biguplus_{\alpha\in id(\kappa)} fst(\kappa(\alpha))$$
$$w' = [o]\uplus w \uplus \overline{w}$$
$$\varrho = appRules(r,w')$$
$$\mathsf{in}\ w' \cdot (\ \mathsf{if}\ \varrho = \{(\epsilon,w''')\}\ \mathsf{then}\ \{\epsilon\}$$
$$\mathsf{else}\ \bigcup_{(r,w'')\in\varrho}\ exe(r,w'',\eta))),$$
$$([o],$$
$$\lambda(w,\kappa).\ \mathsf{let}\ \overline{w} = \biguplus_{\alpha\in id(\kappa)} fst(\kappa(\alpha))$$
$$w' = [o]\uplus w \uplus \overline{w}$$
$$\varrho = appRules(r,w')$$
$$\mathsf{in}\ w' \cdot (\ \mathsf{if}\ \varrho = \{(\epsilon,w''')\}\ \mathsf{then}\ \{\epsilon\}$$
$$\mathsf{else}\ \bigcup_{(r,w'')\in\varrho}\ exe(r,w'',\overline{\eta})))$$
$$)$$
$$= sup_{(w,k)\in W\times\mathbf{K}}$$
$$d(\ \mathsf{let}\ \overline{w} = \biguplus_{\alpha\in id(\kappa)} fst(\kappa(\alpha))$$
$$w' = [o]\uplus w\uplus\overline{w}$$
$$\varrho = appRules(r,w')$$
$$\mathsf{in}\ w' \cdot (\ \mathsf{if}\ \varrho = \{(\epsilon,w''')\}\ \mathsf{then}\ \{\epsilon\}$$
$$\mathsf{else}\ \bigcup_{(r,w'')\in\varrho}\ exe(r,w'',\eta))),$$
$$\mathsf{let}\ \overline{w} = \biguplus_{\alpha\in id(\kappa)} fst(\kappa(\alpha))$$
$$w' = [o]\uplus w\uplus\overline{w}$$
$$\varrho = appRules(r,w')$$
$$\mathsf{in}\ w' \cdot (\ \mathsf{if}\ \varrho = \{(\epsilon,w''')\}\ \mathsf{then}\ \{\epsilon\}$$
$$\mathsf{else}\ \bigcup_{(r,w'')\in\varrho}\ exe(r,w'',\overline{\eta})))$$
$$)^{(****)}$$

Clearly, for any $(w,\kappa) \in W \times \mathbf{K}$ such that $\varrho = \{(\epsilon, w''')\}$ (for some $w''' \in W$) $^{(****)} = 0$. When for $(w,\kappa) \in W \times \mathbf{K}$, $\varrho \neq \{(\epsilon, w''')\}$ (for any $w''' \in W$) we have (ϱ is as in $^{(****)}$, i.e. ϱ is a function of o, w and κ):

$$
\begin{array}{l}
^{(****)} = \\
\quad \frac{1}{2} \cdot sup_{(w,\kappa)\in W\times\mathbf{K}}\ d(\bigcup_{(r,w'')\in\varrho}\ exe(r,w'',\eta),\\
\qquad\qquad\qquad\qquad \bigcup_{(r,w'')\in\varrho}\ exe(r,w'',\overline{\eta}))\\
\quad [\cup \text{ is nonexpansive}]\\
\quad = \frac{1}{2} \cdot sup_{(w,\kappa)\in W\times\mathbf{K}}\\
\qquad\qquad max\{d(exe(r,w'',\eta), exe(r,w'',\overline{\eta}))\\
\qquad\qquad\qquad \mid (r,w'') \in \varrho\}
\end{array}
$$

To obtain the desired result it suffices to show that:

$$d(exe(r,w'',\eta), exe(r,w'',\overline{\eta})) \leq d(\eta,\overline{\eta})$$

for any $(r,w'') \in \varrho$. Now assume that $I = \{1,\cdots,n\}$ and $r = w_1 \Rightarrow x_1 \,\square\, \cdots \,\square\, w_n \Rightarrow x_n$. Then

$$
\begin{array}{l}
d(exe(r,w'',\eta), exe(r,w'',\overline{\eta}))\\
= d(\bigcup_{i\in I}(snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\eta\\
\qquad\qquad\qquad\qquad \mid j \in I\setminus\{i\}|\!\}),\\
\qquad \bigcup_{i\in I}(snd([\![x_i]\!]\overline{\eta}))(w'', \{\!|[\![x_j]\!]\overline{\eta}\\
\qquad\qquad\qquad\qquad \mid j \in I\setminus\{i\}|\!\}))\\
= max_{i\in I}\\
\qquad d((snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\eta \mid j \in I\setminus\{i\}|\!\}),\\
\qquad\quad (snd([\![x_i]\!]\overline{\eta}))(w'', \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\}))
\end{array}
$$

Again, it suffices to show that

$$
\begin{array}{l}
d((snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\eta \mid j \in I\setminus\{i\}|\!\}),\\
\quad (snd([\![x_i]\!]\overline{\eta}))(w'', \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\}))\\
\leq d(\eta,\overline{\eta})
\end{array}
$$

for any $i \in I$.

$$
\begin{array}{l}
d((snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\eta \mid j \in I\setminus\{i\}|\!\}),\\
\quad (snd([\![x_i]\!]\overline{\eta}))(w'', \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\}))\\
\leq max\{\\
\qquad d((snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\eta \mid j \in I\setminus\{i\}|\!\}),\\
\qquad\quad (snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\})),\\
\qquad d((snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\}),\\
\qquad\quad (snd([\![x_i]\!]\overline{\eta}))(w'', \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\}))\\
\qquad \}
\end{array}
$$

Finally,

$$
\begin{array}{l}
d((snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\eta \mid j \in I\setminus\{i\}|\!\}),\\
\quad (snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\}))\\
\quad [\text{Lemma 5.1(c)}]\\
\leq d(\{\!|[\![x_j]\!]\eta \mid j \in I\setminus\{i\}|\!\}, \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\})\\
= \frac{1}{2} \cdot max_{j\in I\setminus\{i\}}\ d([\![x_j]\!]\eta, [\![x_j]\!]\overline{\eta})\\
\quad [\text{Lemma 5.1(d)}]\\
\leq \frac{1}{2} \cdot d(\eta,\overline{\eta})
\end{array}
$$

and

$$
\begin{array}{l}
d((snd([\![x_i]\!]\eta))(w'', \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\}),\\
\quad (snd([\![x_i]\!]\overline{\eta}))(w'', \{\!|[\![x_j]\!]\overline{\eta} \mid j \in I\setminus\{i\}|\!\}))\\
\leq d(snd([\![x_i]\!]\eta), snd([\![x_i]\!]\overline{\eta}))\\
\leq d([\![x_i]\!]\eta, [\![x_i]\!]\overline{\eta}) \qquad [\text{Lemma 5.1(d)}]\\
\leq d(\eta,\overline{\eta})
\end{array}
$$

■

As for any $r \in R$, $\Phi(r)$ is a contraction, we can define: $\eta_0 = fix(\Phi(r))$. The mapping $\mathcal{D}[\![\cdot]\!]$ can be used to compute the collection of sequences of multisets representing the execution of an $\mathcal{L}_{MR}$ program.

Definition 5.3: We define $\mathcal{D}[\![\cdot]\!] : \mathcal{L}_{MR} \to \mathbf{P}$ by

$$
\begin{array}{l}
\mathcal{D}[\![(r,x)]\!] = \ \mathsf{let}\ \eta_0 = fix(\Phi(r))\\
\qquad\qquad\qquad\quad \gamma_0 = ([], \{\!|\,|\!\})\\
\qquad\qquad\qquad\quad \mathsf{in}\ (snd([\![x]\!]\eta_0))(\gamma_0)
\end{array}
$$

Remark 5.4: Let $mset : X \to W$ be given by: $mset(o) = [o]$, $mset(x_1 \parallel x_2) = mset(x_1) \uplus mset(x_2)$. It is easy to check that $fst([\![x]\!]\eta_0) = mset(x)$, for all $x \in X$.

Examples 5.5:

(a) Let $r_2 = [o_1] \Rightarrow o_2 \,\square\, [o_1] \Rightarrow o_3$, $\rho_2 = (r_2, o_1 \parallel o_2)$. Let also $\eta_0 = fix(\Phi(r_2))$. We compute:

$$
\begin{array}{l}
\mathcal{D}[\![\rho_2]\!] = \mathcal{D}[\![(r_2, o_1 \parallel o_2)]\!]\\
= (snd([\![o_1 \parallel o_2]\!]\eta_0))([], \{\!|\,|\!\})\\
= (snd([\![o_1]\!]\eta_0))([], \{\!|[\![o_2]\!]\eta_0|\!\}) \cup\\
\quad (snd([\![o_2]\!]\eta_0))([], \{\!|[\![o_1]\!]\eta_0|\!\})\\
= (snd(\eta_0(o_1)))([], \{\!|[\![o_2]\!]\eta_0|\!\})^{(*)} \cup\\
\quad (snd(\eta_0(o_2)))([], \{\!|[\![o_1]\!]\eta_0|\!\})^{(**)}
\end{array}
$$

We compute $^{(*)}$. Let $w' = [o_1] \uplus [] \uplus (fst([\![o_2]\!]\eta_0)) = [o_1, o_2]$, by Remark 5.4. As $appRules(r_2, w') = \{([o_1] \Rightarrow o_2, [o_2]), ([o_1] \Rightarrow o_3, [o_2])\}$, we obtain:

$$
\begin{array}{l}
^{(*)} = [o_1, o_2] \cdot ((snd([\![o_2]\!]\eta_0))([o_2], \{\!|\,|\!\}) \cup\\
\qquad\qquad (snd([\![o_3]\!]\eta_0))([o_2], \{\!|\,|\!\}))
\end{array}
$$

Next, we compute:

$$
\begin{array}{l}
(snd([\![o_3]\!]\eta_0))([o_2], \{\!|\,|\!\})^{(***)} =\\
(snd(\eta_0(o_3)))([o_2], \{\!|\,|\!\})
\end{array}
$$

Let $w'' = [o_3] \uplus [o_2] \uplus [] = [o_2, o_3]$. As $appRules(r_2, [o_2, o_3]) = \{(\epsilon, [o_2, o_3])\}$ we obtain:

$$^{(***)} = [o_2, o_3] \cdot \{\epsilon\} = \{[o_2, o_3]\}$$

Similarly

$$(snd([\![o_2]\!]\eta_0))([o_2], \{\!|\,|\!\}) = \{[o_2, o_2]\}$$

Therefore,

$$^{(*)} = \{[o_1, o_2][o_2, o_2], [o_1, o_2][o_2, o_3]\}$$

It turns out that $^{(**)} = ^{(*)}$. Therefore:

$$
\begin{array}{l}
\mathcal{D}[\![([o_1] \Rightarrow o_2 \,\square\, [o_1] \Rightarrow o_3, o_1 \parallel o_2)]\!] =\\
\quad \{[o_1, o_2][o_2, o_2], [o_1, o_2][o_2, o_3]\}
\end{array}
$$

(b) Program ρ_1 given in section III is a bit too complex to give here the details of how the (mathematical) denotational mapping computes its meaning. In order to test more complex $\mathcal{L}_{MR}$ programs one can use the semantic interpreter available at [14], which is a Haskell implementation of the denotational mapping. Running (the Haskell implementation of) ρ_1 with the semantic interpreter one obtains the following result:[1]

```
[[["o1","o2"],["o2","o4","o3"],["o3","o4","o3"]]
 [["o1","o2"],["o2"],["o3"]]]
```

[1]In the implementation available at [14] sets and multisets are modeled as Haskell lists. For example, the list `["o3","o4","o3"]` is the Haskell implementation of the multiset $[o_3, o_3, o_4]$ $(= [[o_3, o_4, o_3] = [o_4, o_3, o_3])$.

VI. Concluding remarks and future research

We presented a denotational semantics for a simple abstract concurrent language $\mathcal{L}_{MR}$. $\mathcal{L}_{MR}$ combines two important features encountered also in membrane computing [10], namely the parallel composition operator is based on the concept of *maximal parallelism* and computations are specified by means of *multiset rewriting rules*.

The biologically-inspired model of computation called membrane systems (or P systems) is represented by complex hierarchical structures with a flow of materials and information which supports their functioning. Essentially, the membrane systems are composed of various compartments with different tasks, all of them working simultaneously to accomplish a more general task. However it is possible to construct a "flat" membrane system by replacing the objects in membranes of a hierarchical structure with pairs of objects and labels of membranes. Each rule of the hierarchical system is translated into sets of rules in the "flat" system, and an evolution step of the hierarchical system is translated into a single evolution step of the flat corresponding system. Informally, a *membrane system* consists of a hierarchy of nested membranes, placed inside a distinguishable membrane called *skin*. Each membrane can contain multisets of *objects*, *evolution rules* and other *membranes*. The structure of a membrane system is represented by a tree structure (with the skin as its root), or equivalently, by a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane. The membranes are labelled in a one-to-one manner. A membrane without any other membrane inside is said to be elementary. Each membrane contains a finite set of rules having the form $u \Rightarrow v$, where u and v are usually nonempty multisets of objects. The rules are applied in a maximally parallel manner. This model has the same computational power as a Turing machine, even a rather small number of objects, rules and membranes are involved [10], [6].

The model presented in this paper is both parallel and nondeterministic. All rules that can be applied are applied in parallel, and nondeterminism occurs when several combinations of rules are applicable. The denotational semantics is designed within the mathematical framework of metric semantics [3] by using continuation semantics for concurrency (CSC) [13].

In the future we intend to continue the investigation of the behaviour of membrane systems by using methods in the tradition of programming languages semantics. In previous work we have defined the operational semantics for membrane systems [2], [5], and proved certain semantic properties. We have also considered some alternative modes of evolution, based on either minimal parallelism, maximizing the quantities of consumed resources or maximizing the number of applied rules.

We plan to further develop the semantic model given in this paper in order to obtain a metric denotational semantics of the full model of membrane computing [10]. We believe the CSC technique provides sufficient flexibility for expressing in a denotational manner the nested structure and the complex interactions of a membrane computing system. Next, we will study the formal relationship between the denotational and the operational semantics of membrane systems also by using techniques from metric semantics.

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] S. Banach. Sur les Operation dans les Ensembles Abstrait et leurs Application aux Equation Integrales, *Fundamenta Mathematicae*, vol. 3, pp. 133-181, 1922.

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

[6] G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez. *Applications of Membrane Computing*, Natural Computing Series, Springer, 2006.

[7] G. Ciobanu, E.N. Todoran. Continuation Semantics for Concurrency Applied to Parallel Rewriting of Multisets. In *Proceedings SYNASC 2010*, pp. 387–395, IEEE Computer Press, 2010.

[8] C. Fournet, G. Gonthier. The Join Calculus: a Language for Distributed Mobile Programming, *Lecture Notes in Computer Science* vol. 25, pp. 68-332, 2002.

[9] G. Giertz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, D.S. Scott. *Continuos lattices and domains.* Cambridge University Press, 2003.

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

[11] 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/`.

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

[13] 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.

[14] `ftp://ftp.utcluj.ro/pub/users/gc/synasc2011`.