Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 209 (2008) 107–124
www.elsevier.com/locate/entcs

Expressiveness Issues in Brane Calculi: A Survey

Nadia Busi [1]

Dipartimento di Scienze del lInformazione
Università di Bologna. Italy

Abstract

Brane calculi are a family of biologically inspired process calculi proposed in [5] for modeling the interactions of dynamically nested membranes. In [5] a basic calculus for membranes interactions – called Phago/Exo/Pino (PEP) – is proposed, whose primitives are inspired by endocytosis and exocytosis. An alternative basic calculus – called Mate/Bud/Drip (MBD) and inspired by membrane fusion and fission – is also outlined and shown to be encodable in Phago/Exo/Pino in [5]. In this paper we survey some results on the comparison of the expressivity of the PEP and the MBD calculi, w.r.t. their ability to act as computational devices.

Keywords: Membrane Computing, Brane Calculi

1 Introduction

Brane calculi [5] are a family of process calculi proposed for modeling the behavior of biological membranes. The formal investigation of biological membranes has been initiated by G. Păun [14,15], in the field of automata and formal language theory, with the definition of P systems. In a process algebraic setting, the notions of membranes and compartments are explicitly represented in BioAmbients [16], a variant of Mobile Ambients [7] based on a set of biologically inspired primitives of interaction. Brane calculi represent an evolution of BioAmbients: the main difference with respect to. previous approaches consists in the fact that the active entities reside on membranes, and not inside membranes. In this paper we are interested in two basic instances of brane calculi proposed in [5]: the Phago/Exo/Pino (PEP) and the Mate/Bud/Drip (MBD) calculi.

[1] *Managing Editor's Note:* Professor Busi unexpectedly passed away on September 5, 2005. ENTCS is privileged to have one of her last publications as part of this volume, which focuses on emerging trends in the area in which she worked.

doi:10.1016/j.entcs.2008.04.006

The interaction primitives of PEP are inspired by *endocytosis* (the process of incorporating external material into a cell by engulfing it with the cell membrane) and *exocytosis* (the reverse process). A relevant feature of such primitives is bitonality, a property ensuring that there will never be a mixing of what is inside a membrane with what is outside, although external entities can be brought inside if safely wrapped by another membrane. As endocytosis can engulf an arbitrary number of membranes, it turns out to be a rather uncontrollable process. Hence, it is replaced by two simpler operations: *phagocytosis*, that is engulfing of just one external membrane, and *pinocytosis*, that is engulfing zero external membranes.

The primitives of MBD are inspired by membrane fusion (mate) and fission (mito). Because membrane fission is an uncontrollable process that can split a membrane at an arbitrary place, it is replaced by two simpler operations: *budding*, that is splitting off one internal membrane, and *dripping*, that consists in splitting off zero internal membranes. An encoding of the MBD primitives in PEP is provided in [5]. Cardelli also observed that the reverse encoding does not exist, if the encoding must preserve the nesting structure of membranes. The reason is that in MBD the maximum nesting level of membranes cannot grow during the computation, while this property does not hold for PEP.

In this paper we survey some results on the expressiveness of PEP and MBD to act as computational devices.

In [1] we showed that a fragment of PEP, namely, the calculus comprising only the phago and exo primitives, is Turing powerful. The proof was carried out by showing how to encode a Random Access Machine [17], a well known deterministic, Turing powerful formalism. Such an encoding is deterministic and enjoys the following property: the RAM terminates if and only if its encoding terminates. As a consequence, both the universal termination property (i.e., checking if the system has a divergent computation) and the existential termination property (i.e., checking if the system has a terminating computation) turn out to be undecidable on PEP.

As far as MBD is concerned, in [1] we showed that universal termination is a decidable property. Such a proof of the decidability of universal termination is based on the theory of well-structured transition systems [8]. The decidability of universal termination for MBD provides an expressiveness gap between MBD and PEP, as a deterministic encoding of Random Access Machines can be provided in the second calculus, but not in the first calculus. As a corollary, we get the impossibility to provide an encoding of PEP in MBD that preserves the universal termination property.

The decidability of universal termination in MBD, hence the impossibility to provide a deterministic encoding of RAMs, does not prevent the existence of a weaker encoding; moreover, the decidability of existential termination in MBD is left as an open problem. In [2] we answer to the above question by providing a non-deterministic encoding of RAMs in MBD, which preserves the existence of a terminating computation. The encoding is non-deterministic because it introduces additional computations which do not follow the expected behaviour of the modeled

RAM. However, all these computations are infinite. This ensures that, given a RAM, its modeling has a terminating computation if and only if the RAM terminates. A direct consequence of this result is the undecidability of existential termination for MBD.

The decidability of universal termination for MBD in [1] ensures that we cannot do better, namely, it is impossible to provide a deterministic encoding of RAMs in MBD. It is also impossible to provide a (non-deterministic) encoding of RAMs in MBD satisfying the following property: the RAM terminates iff all the computations of the encoding terminate.

The computational power of MBD is increased if we move to the maximal parallelism semantics typical of Membrane Computing [15]. According to the maximal parallelism semantics, at each computational step a maximal set of independent reductions is simultaneously executed. Hence, all the membranes that can evolve have to do it. By exploiting such maximal progress hypothesis, we provide a deterministic encoding of RAMs in MBD with maximal parallelism that preserves the existence of a terminated computation (hence also the existence of a divergent computation). Thus we obtain the undecidability of both existential and universal termination for MBD with maximal parallelism. This result confirms the intuition emerging from [9], where the interleaving (sequential) and the maximal parallelism semantics of many variants of P systems are compared: in most cases, the computational power increases when moving from interleaving to maximal parallelism.

The paper is organized as follows: in Section 2 we present the syntax and the semantics of the two calculi, and Section 3 contains the deterministic encoding of Random Access Machines in PEP. The decidability of universal termination for MBD is presented in Section 4. In Section 5 we present the non-deterministic encoding of RAMs in MBD with interleaving semantics, while the deterministic encoding of RAMs in MBD with maximal parallelism semantics is reported in Section 6. Section 7 is devoted to conclusive remarks.

2 Brane Calculi: Syntax and Semantics

In this section we recall the syntax and the semantics of Brane Calculi [5]. A system consists of nested membranes, and a process is associated to each membrane.

Definition 2.1 The set of systems is defined by the following grammar:

$$P, Q ::= \diamond \mid P \circ Q \mid !\, P \mid \sigma(\!|P|\!)$$

The set of brane processes is defined by the following grammar:

$$\sigma, \tau ::= 0 \mid \sigma \mid \tau \mid !\,\sigma \mid a.\sigma$$

Variables a, b range over actions, that will be detailed later.

The term $\diamond$ represents the empty system; the parallel composition operator on systems is $\circ$. The replication operator "!" denotes the parallel composition of an

unbounded number of instances of a system. The term $\sigma(\!|P|\!)$ denotes the brane that performs the process σ and contains system P.

The term 0 denotes the empty process, whereas " | " is the parallel composition of processes; with $!\sigma$ we denote the parallel composition of an unbounded number of instances of the process σ. Term $a.\sigma$ is a guarded process: after performing the action a, the process behaves as σ.

We adopt the following abbreviations: with a we denote $a.0$, with $(\!|P|\!)$ we denote $0(\!|P|\!)$, and with σ we denote $\sigma(\!|\diamond|\!)$.

The structural congruence relation on systems and processes is defined as follows: [2]

Definition 2.2 The structural congruence is the least congruence relation satisfying the following axioms:

$$P \circ Q \equiv Q \circ P \qquad \sigma \mid \tau \equiv \tau \mid \sigma$$

$$P \circ (Q \circ R) \equiv (P \circ Q) \circ R \qquad \sigma \mid (\tau \mid \rho) \equiv (\sigma \mid \tau) \mid \rho$$

$$P \circ \diamond \equiv P \qquad \sigma \mid 0 \equiv \sigma$$

$$!\diamond \equiv \diamond \qquad !\,0 \equiv 0$$

$$!\,(P \circ Q) \equiv !\,P \circ !\,Q \qquad !\,(\sigma \mid \tau) \equiv !\,\sigma \mid !\,\tau$$

$$!!\,P \equiv !\,P \qquad !!\,\sigma \equiv !\,\sigma$$

$$P \circ !\,P \equiv !\,P \qquad \sigma \mid !\,\sigma \equiv !\,\sigma$$

$$0(\!|\diamond|\!) \equiv \diamond$$

2.1 Interleaving semantics of Brane Calculi

We recall the standard, interleaving semantics. At each computational step, a single reaction is chosen and executed. The next definition provides the set of generic reaction rules that are valid for all brane calculi, while the reaction axioms are specific for each brane calculus; the reaction axioms for PEP and MBD will be provided in Definitions 2.5 and 2.7, respectively.

Definition 2.3 The basic reaction rules are the following:

[2] With abuse of notation we use $\equiv$ to denote both structural congruence on systems and structural congruence on processes.

$$(\texttt{par})\ \frac{P \to Q}{P \circ R \to Q \circ R} \qquad (\texttt{brane})\ \frac{P \to Q}{\sigma(\!|P|\!) \to \sigma(\!|Q|\!)}$$

$$(\texttt{strucong})\ \frac{P \equiv P' \quad P \to Q \quad Q \equiv Q'}{P' \to Q'}$$

Rules (par) and (brane) are the contextual rules that respectively permit to a system to execute also if it is in parallel with another process or if it is inside a membrane, respectively. Rule (strucong) ensures that two structurally congruent systems have the same reactions.

With $\to^*$ we denote the reflexive and transitive closure of a relation $\to$. Given a reduction relation $\to$, we say that a system P has a *divergent computation* (or infinite computation) if there exists an infinite sequence of systems $P_0, P_1, \ldots, P_i, \ldots$ such that $P = P_0$ and $\forall i \geq 0 : P_i \to P_{i+1}$. We say that a system P *universally terminates* if it has no divergent computations. We say that P is *deterministic* iff for all P', P'': if $P \to P'$ and $P \to P''$ then $P' \equiv P''$. We say that P has a *terminating computation* (or a deadlock) if there exists Q such that $P \to^* Q$ and $Q \not\to$. A system P satisfies the universal termination property if P has no divergent computations. A system P satisfies the existential termination property if P has a deadlock. Note that the existential termination and the universal termination properties are equivalent on deterministic systems.

The system P' is a *derivative* of the system P if $P \to^* P'$; the set of derivatives of a system P is denoted by $Deriv(P)$.

We use Π (resp. $\bigcirc$) to denote the parallel composition of a set of processes (resp. systems), i.e., $\Pi_{i\in\{1..n\}}\sigma_i = \sigma_1 \mid \ldots \mid \sigma_n$ and $\bigcirc_{i\in\{1,\ldots,n\}}P_i = P_1 \circ \ldots \circ P_n$. Moreover, $\Pi_{i\in\emptyset}\sigma_i = 0$ and $\bigcirc_{i\in\emptyset}P_i = \diamond$.

2.2 The Phago/Exo/Pino Calculus (PEP)

The PEP calculus is proposed in [5], and it is inspired by endocytosis/exocytosis. Endocytosis is the process of incorporating external material into a cell by "engulfing" it with the cell membrane, while exocytosis is the reverse process. As endocytosis can engulf an arbitrary amount of material, giving rise to an uncontrollable process. In [5] two more basic operations are used: *phagocytosis*, engulfing just one external membrane, and *pinocytosis*, engulfing zero external membranes.

Definition 2.4 Let *Name* be a denumerable set of ambient names, ranged over by $n, m, \ldots$. The set of actions of PEP is defined by the following grammar:

$$a \ ::= \ \wp_n \mid \ \wp_n^{\perp}(\sigma) \mid \ \wp_n \mid \ \wp_n^{\perp} \mid \circledcirc(\sigma)$$

Action $\wp_n$ denotes phagocytosis; the co-action $\wp_n^{\perp}(\sigma)$ is meant to synchronize with $\wp_n$; names n are used to pair-up related actions and co-actions. The co-phago action is equipped with a process σ, this process will be associated to the new

membrane that engulfs the external membrane. Action $\eth_n$ denotes exocytosis, and synchronizes with the co-action $\eth_n^\perp$. Exocytosis causes an irreversible mixing of membranes. Action $\circledcirc$ denotes pinocytosis. The pino action is equipped with a process σ: this process will be associated to the new membrane, that is created inside the brane performing the pino action.

Definition 2.5 The reaction relation for PEP is the least relation containing the following axioms, and satisfying the rules in Definition 2.3:

$$\begin{array}{ll}
(\texttt{phago}) & \eth_n\,.\sigma|\sigma_0(\!|P|\!) \circ \eth_n^\perp\,(\rho).\tau|\tau_0(\!|Q|\!) \to \tau|\tau_0\,(\!|\rho(\!|\sigma|\sigma_0(\!|P|\!)|\!) \circ Q|\!) \\
(\texttt{exo}) & \eth_n^\perp\,.\tau|\tau_0\,(\!|\,\eth_n\,.\sigma|\sigma_0(\!|P|\!) \circ Q|\!) \to P \circ \sigma|\sigma_0|\tau|\tau_0(\!|Q|\!) \\
(\texttt{pino}) & \circledcirc(\rho).\sigma|\sigma_0(\!|P|\!) \to \sigma|\sigma_0(\!|\rho(\!|\,\,|\!) \circ P|\!)
\end{array}$$

2.3 The Mate/Bud/Drip Calculus (MBD)

The second calculus, also proposed in [5], is inspired by membrane fusion and splitting. To make membrane splitting more controllable, in [5] two more basic operations are used: *budding*, consisting in splitting off one internal membrane, and *dripping*, consisting in splitting off zero internal membranes. Membrane fusion, or merging, is called *mating*.

Definition 2.6 The set of actions of MBD is defined by the following grammar:

$$a \;::=\; mate_n \;\mid\; mate_n^\perp \;\mid\; bud_n \;\mid\; bud_n^\perp(\sigma) \;\mid\; drip(\sigma)$$

Actions $mate_n$ and $mate_n^\perp$ will synchronize to obtain membrane fusion. Action bud_n permits to split one internal membrane, and synchronizes with the co-action $bud_n^\perp$. Action *drip* permits to split off zero internal membranes. Actions $bud_n^\perp$ and *drip* are equipped with a process σ, that will be associated to the new membrane created by the brane performing the action.

Definition 2.7 The reaction relation for MBD is the least relation containing the following axioms, and satisfying the rules in Definition 2.3:

$$\begin{array}{ll}
(\texttt{mate}) & mate_n.\sigma|\sigma_0(\!|P|\!) \circ mate_n^\perp.\tau|\tau_0(\!|Q|\!) \to \sigma|\sigma_0|\tau|\tau_0(\!|P \circ Q|\!) \\
(\texttt{bud}) & bud_n^\perp(\rho).\tau|\tau_0(\!|bud_n.\sigma|\sigma_0(\!|P|\!) \circ Q|\!) \to \rho(\!|\sigma|\sigma_0(\!|P|\!)|\!) \circ \tau|\tau_0(\!|Q|\!) \\
(\texttt{drip}) & drip(\rho).\sigma|\sigma_0(\!|P|\!) \to \rho(\!|\,\,|\!) \circ \sigma|\sigma_0(\!|P|\!)
\end{array}$$

In [5] it is shown that the operations of mating, budding and dripping can be encoded in PEP.

2.4 Maximal parallelism semantics of MBD

In this section we recall the semantics based on maximal progress - inspired by the standard semantics of Membrane Computing [15] - proposed in [2]. The idea

is that at each computational step, a maximal set of independent reductions is simultaneously executed. Hence, all the membranes that can evolve have to do it. For example, the system

$$mate_a(\!|P|\!) \circ drip(0)(\!|Q|\!) \circ mate_a^{\perp}(\!|R|\!)$$

performs the maximal progress move

$$mate_a(\!|P|\!) \circ drip(0)(\!|Q|\!) \circ mate_a^{\perp}(\!|R|\!) \Rightarrow 0(\!|P|\!) \circ 0(\!|Q|\!) \circ 0(\!|\ |\!) \circ 0(\!|R|\!)$$

On the other hand, the following move does not involve all the membranes that can evolve, hence it is not allowed:

$$mate_a(\!|P|\!) \circ drip(0)(\!|Q|\!) \circ mate_a^{\perp}(\!|R|\!) \not\Rightarrow 0(\!|P|\!) \circ drip(0)(\!|Q|\!) \circ 0(\!|R|\!)$$

At each computational step, a membrane can be involved in at most one reduction rule. Hence, also the following move, where three membranes are simultaneously fused, is not allowed:

$$mate_a|mate_b(\!|P|\!) \circ mate_a^{\perp}(\!|Q|\!) \circ mate_b^{\perp}(\!|R|\!) \not\Rightarrow 0(\!|P \circ Q \circ R|\!)$$

In such case, one of the following computational steps can be performed:

$$mate_a|mate_b(\!|P|\!) \circ mate_a^{\perp}(\!|Q|\!) \circ mate_b^{\perp}(\!|R|\!) \Rightarrow mate_b(\!|P \circ Q|\!) \circ mate_b^{\perp}(\!|R|\!)$$

$$mate_a|mate_b(\!|P|\!) \circ mate_a^{\perp}(\!|Q|\!) \circ mate_b^{\perp}(\!|R|\!) \Rightarrow mate_a(\!|P \circ R|\!) \circ mate_a^{\perp}(\!|Q|\!)$$

A maximal parallelism computational step is obtained as a maximal sequence of independent reductions. To formalize this notion, we take a modified reduction semantics, obtained by "freezing" all the processes associated to a membrane, after that such a membrane has been involved in a reduction. After the execution of a maximal parallelism computational step, the frozen processes are "heated" and can be involved in the next computational step.

To this aim, we extend the grammar of systems with a new term, denoting a membrane whose process is frozen:

$$P, Q \quad := \ldots \mid \langle\sigma\rangle(\!|P|\!)$$

The reaction relation is modified as follows:

Definition 2.8 The reaction relation $\mapsto$ for MBD is the least relation containing the following axioms, and satisfying the rules in Definition 2.3 (obtained by replacing

$\rightarrow$ with $\mapsto$):

(mate) $mate_n.\sigma|\sigma_0(\!|P|\!) \circ mate_n^{\perp}.\tau|\tau_0(\!|Q|\!) \mapsto \langle \sigma|\sigma_0\ \tau|\tau_0\rangle(\!|P \circ Q|\!)$

(bud) $bud_n^{\perp}(\rho).\tau|\tau_0(\!|bud_n.\sigma|\sigma_0(\!|P|\!) \circ Q|\!) \mapsto$

$\langle \rho\rangle(\!|\langle \sigma|\sigma_0\rangle(\!|P|\!)|\!) \circ \langle \tau|\tau_0\rangle(\!|Q|\!)$

(drip) $drip(\rho).\sigma|\sigma_0(\!|P|\!) \mapsto \langle \rho\rangle(\!|\ |\!) \circ \langle \sigma|\sigma_0\rangle(\!|P|\!)$

The heating function *heated*() transforms the frozen processes of a system in active processes.

Definition 2.9 The heating function, called $heated(P)$, is defined inductively on the structure of (the extended set of) systems:

$$heated(\diamond) = \diamond$$

$$heated(P \circ Q) = heated(P) \circ heated(Q)$$

$$heated(!\,P) = !\,heated(P)$$

$$heated(\sigma(\!|P|\!)) = \sigma(\!|P|\!)$$

$$heated(\langle\sigma\rangle(\!|P|\!)) = \sigma(\!|P|\!)$$

Now we are ready to define the maximal parallelism computational step $\Rightarrow$, consisting in a maximal (not extendable) sequence of reductions $\mapsto$.

Definition 2.10 Let P, Q be MBD systems (not containing frozen processes). $P \Rightarrow Q$ iff there exists a system Q' such that $P \mapsto^{+} Q'$, $Q' \not\mapsto$ and $Q = heated(Q')$

3 A deterministic encoding of RAMs in PEP

In this section we provide a deterministic encoding of Random Access Machines (RAMs) [17] - a well known Turing powerful formalism - in PEP. A detailed description of the encoding can be found in [1]; here we only provide an intuitive description.

We start by recalling what RAMs are.

3.1 Random Access Machines

RAMs are a computational model based on finite programs acting on a finite set of registers. More precisely, a RAM R is composed of the registers $r_1, \ldots, r_n$, that can hold arbitrary large natural numbers, and by a sequence of indexed instructions $(1 : I_1), \ldots, (m : I_m)$. In [13] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : Succ(r_j))$: adds 1 to the contents of register r_j and goes to the next instruction;

- $(i : DecJump(r_j, s))$: if the contents of the register r_j is not zero, then decreases it by 1 and goes to the next instruction, otherwise jumps to the instruction s.

The computation starts from the first instruction and it continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached.

A state of a RAM is modeled by $(i, c_1, \ldots, c_n)$, where i is the program counter indicating the next instruction to be executed, and $c_1, \ldots, c_n$ are the current contents of the registers $r_1, \ldots, r_n$, respectively. We use the notation $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ to denote that the state of the RAM R changes from $(i, c_1, \ldots, c_n)$ to $(i', c'_1, \ldots, c'_n)$, as a consequence of the execution of the i-th instruction.

A state $(i, c_1, \ldots, c_n)$ is *terminated* if the program counter i is strictly greater than the number of instructions m. We say that a RAM R *terminates* if its computation reaches a terminated state.

3.2 modeling RAMs in PEP

In this section we show how to model RAMs in PEP. The modeling of RAMs is based on an encoding function, which transforms instructions and registers independently.

The basic idea for modeling the natural numbers contained in the registers is the following: the natural number n is represented by the nesting of $2n + 1$ branes. The increment is performed by producing a new membrane that performs a phago on the representation of n, while a decrement is performed by executing an exo of the membrane representing $n - 1$, that lies inside the membrane representing n.

Consider a RAM R with instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$; the encoding of an initial state $(1, c_1, \ldots, c_n)$ is defined as follows:

$$[\![(1, c_1, \ldots, c_n)]\!]_{PEP} = [\![PC = 1]\!]_{PEP} \circ ! [\![(1 : I_1)]\!]_{PEP} \circ \ldots \circ ! [\![(m : I_m)]\!]_{PEP} \circ [\![r_1 = c_1]\!]_{PEP} \circ \ldots \circ [\![r_n = c_n]\!]_{PEP} \circ GARB(\!|\!)$$

where $GARB$ is the process on the garbage collector membrane.

The encoding of an initial state of the RAM is composed by the following parts: $[\![PC = 1]\!]_{PEP}$, representing the program counter, (an unbounded number of occurrences of) the encodings of each instruction, the encodings of the initial contents of registers, and a garbage membrane used to collect no longer used membranes and to inhibit their actions.

The presence of a "program counter" system $[\![PC = i]\!]_{PEP}$ denotes the fact that the next instruction to be executed is I_i .

The encoding of the contents of register r_j is depicted in Figure 1. The encoding of the contents of register r_j with content zero is a membrane with membrane process Z_j .

If an increment operation on r_j is executed, then the system $[\![r_j = 0]\!]_{PEP}$ is engulfed in a new membrane, thus obtaining a representation of $[\![r_j = 1]\!]_{PEP}$. Then,

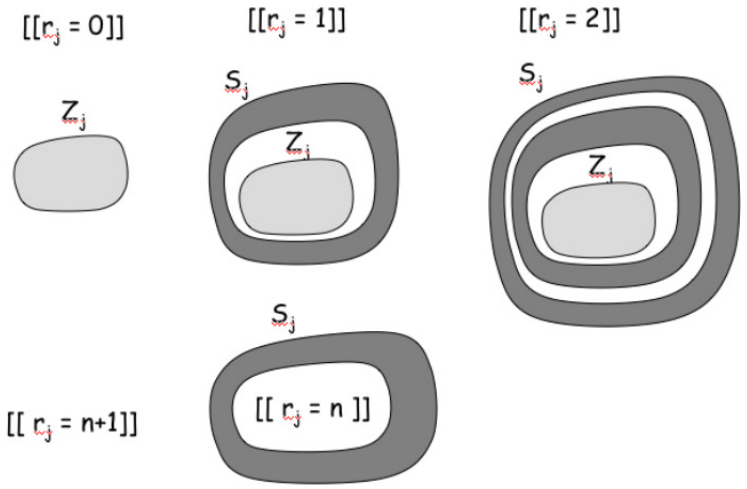

Fig. 1. The encoding of the registers of the RAM in PEP.

a membrane representing the program counter $[\![PC = i+1]\!]_{PEP}$ is produced. If the membrane of system $[\![r_j = 0]\!]_{PEP}$ is entered by a request for decrement or jump, the choice corresponding to the zero case is selected, and the program counter corresponding to the jump is expelled.

The encoding of register r_j with content $n+1$ is the system $[\![r_j = n+1]\!]_{PEP} = S_j(\!|\sigma(\!|[\![r_j = n]\!]_{PEP}|\!)\!)$.

The case of an increment operation is treated in the same way as in the encoding of $r_j = 0$.

If the membrane of system $r_j = n+1$ is entered by a request for decrement or jump, then the choice corresponding to the non-zero case is selected, and the membrane representing $r_j = n$ is expelled. At this point, the no longer used external membrane of the system $[\![r_j = n+1]\!]_{PEP}$ is engulfed by the garbage collector membrane. Finally, the program counter corresponding to the next instruction is expelled.

We need to engulf the external membrane of the system $[\![r_j = n+1]\!]_{PEP}$ in the garbage collector, to inhibit the possibility for the process decorating such a membrane to capture a subsequent request for increment of register r_j . In [3] we showed that the above encoding is deterministic and enjoys the following property: the RAM R with program $(1 : I_1), \ldots, (m : I_m)$ and initial state $(1, c_1, \ldots, c_n)$ terminates if and only if the (unique) computation of the system $[\![(i, c_1, \ldots, c_n)]\!]_{PEP}$ terminates.

An immediate consequence of the two facts above is the undecidability of both existential and universal termination for PEP systems.

4 Decidability of universal termination for MBD

In this section we recall the basic ideas of the proof of the decidability of the existence of a divergent computation for MBD presented in [1]. Such a proof is based on the theory of well-structured transition systems [8]: the existence of an infinite

computation starting from a given state is decidable for finitely branching transition systems, provided that the set of states can be equipped with a well-quasi-ordering, i.e., a quasi-ordering relation which is compatible with the transition relation and such that each infinite sequence of states admits an increasing subsequence.

We start by providing some basic definitions and results on well-structured transition systems and on well-quasi-ordering on sequences of elements belonging to a well-quasi-ordered set. Then, we show the decidability of termination for MBD; to this aim, we first show how to provide an alternative semantics that is equivalent w.r.t. termination to the one presented in Section 2, but which is based on a finitely branching transition system and permits to define a well-quasi-ordering on the derivatives of a given system (i.e., the set of systems reachable from a given initial system). Then, by exploiting the theory developed in [8], we show that termination is decidable for MBD systems.

4.1 Well-Structured Transition System

We start by recalling some basic definitions and results from [8], concerning well-structured transition systems. A *quasi-ordering* (qo) is a reflexive and transitive relation.

Definition 4.1 A *well-quasi-ordering* (wqo) is a quasi-ordering $\leq$ over a set X such that, for any infinite sequence $x_0, x_1, x_2, \ldots$ in X , there exist indexes $i < j$ such that $x_i \leq x_j$.

Note that, if $\leq$ is a wqo, then any infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \ldots$ (with $i_0 < i_1 < i_2 < \ldots$).

Transition systems can be formally defined as follows.

Definition 4.2 A *transition system* is a structure $TS = (S, \rightarrow)$, where S is a set of states and $\rightarrow \subseteq S \times S$ is a set of *transitions*.

We write $Succ(s)$ to denote the set $\{s' \in S | s \rightarrow s'\}$ of immediate successors of $s \in S$.

TS is *finitely branching* if $\forall_s \in S : Succ(s)$ is finite. We restrict to finitely branching transition systems.

Well-structured transition systems, defined as follows, provide the key tool to decide properties of computations.

Definition 4.3 A *well-structured transition system* (with *strong compatibility*) is a transition system $TS = (S, \rightarrow)$, equipped with a quasi-ordering $\leq$ on S , also written $TS = (S, \rightarrow, \leq)$, such that the two following conditions hold:

(i) **well-quasi-ordering**: $\leq$ is a well-quasi-ordering, and

(i) **strong compatibility**: $\leq$ is (upward) compatible with $\rightarrow$, i.e., for all $s_1 \leq t_1$ and all transitions $s_1 \rightarrow s_2$, there exists a state t_2 such that $t_1 \rightarrow t_2$ and $s_2 \leq t_2$.

The following theorem (a special case of a result in [8]) is used to obtain our decidability result.

Theorem 4.4 *Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with decidable $\leq$ and computable Succ. The existence of an infinite computation starting from a state $s \in S$ is decidable.*

4.2 Higman's Lemma

To show that the quasi-ordering relation we will define on MBD systems is a well-quasi-ordering we need the following result, due to Higman [10] and stating that the set of the finite sequences over a set equipped with a wqo is well-quasi-ordered.

Given a set S , with S^* we denote the set of finite sequences of elements in S.

Definition 4.5 Let S be a set and $\leq$ a wqo over S . The relation $\leq_*$ over S^* is defined as follows. Let $t, u \in S^*$, with $t = t_1 t_2 \ldots t_m$ and $u = u_1 u_2 \ldots u_n$. We have that $t \leq_* u$ iff there exists an injection f from $\{1, 2, \ldots, m\}$ to $\{1, 2, \ldots, n\}$ such that $t_i \leq u_{f(i)}$ and $i \leq f(i)$ for $i = 1, \ldots, m$.

Note that relation $\leq_*$ is a quasi-ordering over S^*.

Lemma 4.6 (Higman) *Let S be a set and $\leq$ a wqo over S . Then, the relation $\leq_*$ is a wqo over S^*.*

Also the following propositions will be used to prove that the relation on systems is a well-quasi-ordering:

Proposition 4.7 *Let S be a finite set. Then the equality is a wqo over S.*

Proposition 4.8 *Let S, T be sets and $\leq_S$,$\leq_T$ be wqo over S and T , respectively. The relation $\leq$ over $S \times T$ is defined as follows: $(s_1, t_1) \leq (s_2, t_2)$ iff ($s_1 \leq_S s_2$ and $t_1 \leq_T t_2$). The relation $\leq$ is a wqo over $S \times T$.*

4.3 A finitely branching semantics for MBD systems

Because of the structural congruence rules, the reaction transition system for MBD is not finitely branching. To obtain a finitely branching transition system (with the same behavior w.r.t. termination), we need to take the transition system whose states are the equivalence classes of structural congruence.

Technically, it is possible to define a normal form for systems, up to the commutative and associative laws for the $\circ$ and $|$ operators.

In a system in normal form, the presence of a replicated version of a sequential process $!\,a.\sigma$ (resp. system $!\,(\sigma(\!|P|\!))$) forbids the presence of any occurrence of the non-replicated version of the same process (resp. system), as well as of other occurrences of the replicated version of the process (resp. system). Moreover, replication is distributed over the components of parallel composition operators, and redundant replications and empty systems and terms are removed.

4.4 Decidability of termination for MBD systems

Given a system P in normal form, it is possible to define a quasi-order on the derivatives of P (and a quasi-order on brane processes) that turns out to be a wqo compatible with the reduction relation on normal forms. Hence, by exploiting the results in Section 4.2, we obtain decidability of termination.

We note that each system (resp. process) in normal form is essentially a finite sequence of objects of kind $\sigma(\!|Q|\!)$ or $!\,(\sigma(\!|Q|\!))$ (resp. of objects of kind $a.\sigma$ or $!\,a.\sigma$). If we consider the nesting level of membranes, we note that each subsystem Q contained in a subterm $\sigma(\!|Q|\!)$ or $!\,(\sigma(\!|Q|\!))$ of a system R is simpler than R. More precisely, the maximum nesting level of membranes in Q is strictly smaller than the maximum nesting level of membranes in R. As already observed in [6], the reactions in MBD preserve the nesting level of membranes; hence, the nesting level of membranes in a system P provides an upper bound to the nesting level of membranes in the set of the (normal forms of the) derivatives of P.

The quasi-order $\preceq_{proc}$ on processes in normal form is roughly defined in the following way: $\sigma \preceq_{proc} \tau$ if

- for each occurrence of a replicated guarded process at top-level in σ there is a corresponding occurrence of the same process at top-level in τ ;
- for each occurrence of a guarded process at top-level in σ there is either a corresponding occurrence of the same process or an occurrence of the replicated version of the process at top-level in τ.

The quasi-order on systems is roughly defined as $R \preceq_{sys} S$ if

- for each replicated membrane $!\,(\rho(\!|R_1|\!))$ at top-level in R there is a corresponding replicated membrane $!\,(\sigma(\!|S_1|\!))$ at top-level in S such that ρ is smaller than σ and R_1 is smaller than S_1 ;
- for each occurrence of a membrane $\rho(\!|R_1|\!)$ at top-level in R there is
  - either a corresponding occurrence of a membrane $\sigma(\!|S_1|\!)$ at top-level in S such that ρ is smaller than σ and R_1 is smaller than S_1.
  - or an occurrence of a replicated membrane $!\,(\rho(\!|R_1|\!))$ at top-level in S.

The proof that $\preceq_{sys}$ is a wqo essentially proceeds by induction on the nesting level of membranes, and makes use of repeated applications Higman's Lemma and of Proposition 4.8.

5 A non-deterministic encoding of RAMs in MBD

In [2] we provide a non-deterministic encoding of RAMs in MBD (with interleaving semantics), which preserves the existence of a terminating computation. The encoding is non-deterministic because it introduces additional computations which do not follow the expected behaviour of the modeled RAM. However, all these computations are infinite. This ensures that, given a RAM, its modeling has a terminating computation if and only if the RAM terminates. A direct consequence of this result is the undecidability of existential termination for MBD.

The encoding satisfies the following property. If the RAM terminates, then the encoding has at least one terminating computation; otherwise, no computation of the encoding terminates. Hence, even if the RAM terminates, it may happen that a run of the encoding diverges. This is due to the fact that it is not possible to perform a test for zero on the (representation of the) contents of registers. When a *DecJump* instruction is performed, one of the two branches (decrement or jump) is chosen non-deterministically. If the right branch is taken, then the encoding behaves correctly. On the other hand, if the wrong branch is taken, then a system is reached such that any computation starting from such a system will diverge.

The modeling of RAMs is based on an encoding function, which transforms instructions and registers independently.

The basic idea for modeling the natural numbers contained in the registers is the following: the natural number n contained in register r_j is represented by n copies of a system R_j collected inside a register membrane. The increment is performed by fusing the register membrane with a membrane containing one copy of R_j , thus obtaining $n+1$ copies of R_j inside the register membrane. The decrement is performed by mating the register membrane with a membrane whose process permits to perform a budding of one of the systems R_j contained inside the register membrane, thus leaving $n-1$ copies of R_j inside the register membrane.

Consider a RAM R with instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$; the encoding of an initial state $(1, c_1, \ldots, c_n)$ is defined as follows:

$$[\![(1, c_1, \ldots, c_n)]\!]_{MBD} = [\![PC = 1]\!]_{MBD} \circ ! [\![(1 : I_1)]\!]_{MBD} \circ \ldots \circ ! [\![(m : I_m)]\!]_{MBD} \circ [\![r_1 = c_1]\!]_{MBD} \circ \ldots \circ [\![r_n = c_n]\!]_{MBD} \circ LOOP(\!|\,|\!)$$

where the loop membrane $LOOP(\!|\,|\!)$ ensures that the system will diverge if the wrong branch of the encoding of a *DecJump* instruction is taken. If a membrane $mate_{loop}(\!|\ldots|\!)$ is produced, then such a membrane may fuse with the loop membrane, and another similar membrane is dripped, that may fuse with the loop membrane, and so on, thus preventing the system to terminate.

The encoding of an initial state of the RAM is composed by the following parts: the program counter, (an unbounded number of occurrences of) the encodings of each instruction, the encodings of the initial contents of registers, and the loop membrane.

The presence of the program counter membrane $[\![PC = i]\!]_{MBD}$ denotes the fact that the next instruction to be executed is I_i . The encoding of the program counter membrane $[\![PC = i]\!]_{MBD}$ will fuse with the encoding of the i-th instruction to activate the execution of such instruction.

The encoding of the contents of register r_j is depicted in Figure 2. If an increment operation on r_j is executed, then a membrane, containing one copy of R_j , is fused with $[\![r_j = c_j]\!]_{MBD}$, thus obtaining a representation of $[\![r_j = c_j + 1]\!]_{MBD}$.

Suppose that the i-th instruction is a decrement of register r_j , or jump to instruction s if the contents of r_j is zero. Independently of the actual contents of register r_j , the program counter membrane is fused with either the decrement part

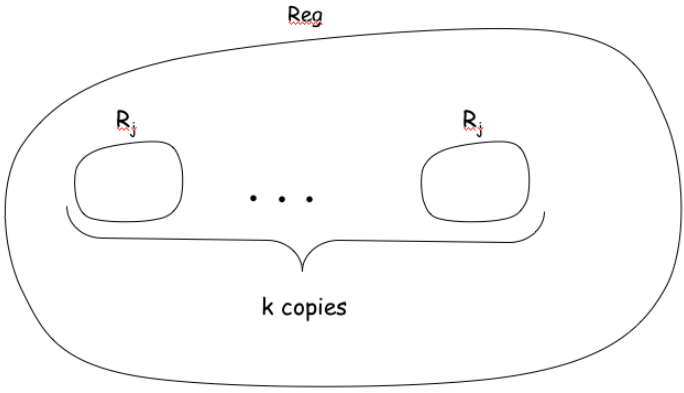

Fig. 2. The encoding of the registers of the RAM in MBD with interleaving semantics.

or the zero part of the instruction, thus selecting non-deterministically one of the two branches of the *DecJump* instruction.

If the decrement part is selected, then a membrane $mate_{loop}(\!| \dots |\!)$ is produced. Then, a membrane - decorated with a budding instruction - is fused with $[\![r_j = c_j]\!]_{MBD}$. At this point, the only operation that can be performed by the register membrane is such a budding. If $c_j > 0$, then at least one copy of R_j is present in the register membrane; by performing a budding action, one copy of R_j is "expelled" from the register membrane. Such an expelled copy is surrounded by a membrane with an empty program, hence becoming an innocuous garbage that can neither perform reductions nor interact with the other membranes. At this point, the membrane $mate_{loop}(\!| \dots |\!)$ is removed - by performing a fusion - and the membrane corresponding to the program counter $[\![PC = i + 1]\!]_{MBD}$ is produced.

If $c_j = 0$, then the register membrane contains no membranes and no further operation can be performed by the register membrane. Hence, the loop membrane $LOOP(\!|\,|\!)$ is activated and a divergent computation is taken.

If the zero branch is selected, then a membrane - decorated with a budding instruction - is fused with $[\![r_j = c_j]\!]_{MBD}$, and a new system $[\![r_j = 0]\!]_{MBD}$ is produced. If $c_j = 0$, then the old register membrane contains no membranes inside; as the only instruction that the old register membrane can perform is a budding, it becomes innocuous garbage. If $c_j > 0$, then the old register membrane contains at least one copy of R_j ; such R_j can be expelled, and surrounded by a membrane that can activate the loop membrane, thus starting a divergent computation.

In [2] we showed that the above encoding enjoys the following property: the RAM R with program $(1 : I_1), \dots, (m : I_m)$ and initial state $(1, c_1, \dots, c_n)$ terminates if and only if the system $[\![(1, c_1, \dots, c_n)]\!]_{MBD}$ has a terminating computation.

As a consequence, existential termination turns out to be undecidable for MBD systems (with interleaving semantics).

6 A deterministic encoding of RAMs in MBD with maximal parallelism semantics

In this section we show how to obtain an encoding that behaves deterministically under the maximal parallelism hypothesis.

The modeling of the RAM is quite similar to the one of the previous section. The key idea is to use the maximal progress hypothesis to ensure that the right branch of a *DecJump* instruction is taken. Both the decrement and the zero branches of the instruction are activated in parallel, but the execution of the relevant part of the zero branch is delayed by innocuous $drip(0)$ operations, so that the zero branch will be executed only if the decrement branch fails.

The modeling of the contents of registers and of the increment instruction is the same as for the previous encoding, but in the present encoding all the components are surrounded by an external membrane. Such an external membrane permits to bud the garbage membranes that are not innocuous but could interfere with the correct components.

For completeness, here we report the whole encoding, and we highlight the differences.

Consider a RAM R with instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$; the encoding of an initial state $(1, c_1, \ldots, c_n)$ is defined as follows:

$$\begin{aligned}[\![(1, c_1, \ldots, c_n)]\!]^{mp}_{MBD} = EXT(\!| &[\![PC = 1]\!]^{mp}_{MBD} \circ \\ &! [\![(1 : I_1)]\!]^{mp}_{MBD} \circ \ldots \circ ! [\![(m : I_m)]\!]^{mp}_{MBD} \circ \\ &[\![r_1 = c_1]\!]^{mp}_{MBD} \circ \ldots \circ [\![r_n = c_n]\!]^{mp}_{MBD} |\!)\end{aligned}$$

where EXT is the process surrounding the external membrane, permitting to expell the garbage membranes.

The encoding of the program counter is the same as in the previous section, and the encoding of the contents of registers is slightly simpler (as it is no longer necessary to start a loop in the case the wrong branch is taken).

The main difference w.r.t. the previous section is represented by the encoding of the *DecJump* instruction, whereas the encoding of the *Succ* instruction is unchanged.

As in the previous section, instruction $(i : I_i)$ is activated by fusing it with the program counter membrane.

Suppose that the i-th instruction is a decrement of register r_j , or jump to instruction s if the contents of r_j is zero.

The programs *DECR* and *JUMP* , respectively corresponding to the decrement and to the zero branches, are executed simultaneously, and the innocuous $drip(0)$ instructions are used to synchronize the two programs.

First, the *DECR* branch is tried (while the *JUMP* branch performs a sequence of $drip(0)$ instructions): a mutual exclusion membrane is produced, then the decrement is tried and - if the decrement is successful - the mutual exclusion membrane

is removed. After a delay sufficient for the *DECR* branch to terminate, the *ZERO* branch is started. If the *DECR* branch is failed, then the mutual exclusion membrane has not been removed, and such a membrane activates the production of a new membrane for $[\![r_j = 0]\!]^{mp}_{MBD}$. Then, in any case, the non-innocuous garbage is expelled outside the membrane by a budding operation.

In [2] we showed that the above encoding is deterministic and enjoys the following property: the RAM R with program $(1 : I_1), \ldots, (m : I_m)$ and initial state $(1, c_1, \ldots, c_n)$ terminates if and only if the (unique) computation of the system $[\![(i, c_1, \ldots, c_n)]\!]^{mp}_{MBD}$ terminates.

As a consequence, both existential and universal termination are undecidable for MBD with maximal parallelism semantics.

7 Conclusion

In this paper we survey some expressiveness results - presented in [1] and [2] - on the two basic Brane Calculi PEP and MBD w.r.t. their ability to encode computable functions.

Regarding the ability to encode RAMs, we obtained the following results:

- there exists a deterministic encoding of RAMs in PEP with interleaving semantics, preserving both existential and universal termination;
- there exist no deterministic encoding of RAMs in MBD with interleaving semantics, preserving either existential or universal termination;
- there exists a non-deterministic encoding of RAMs in MBD with interleaving semantics, preserving both existential and universal termination;
- there exists a deterministic encoding of RAMs in MBD with maximal parallelism semantics, preserving both existential and universal termination.

Regarding the decidability of termination properties, we obtained the following results:

- Both existential and universal termination are undecidable for PEP with interleaving semantics;
- Universal termination is decidable for MBD with interleaving semantics;
- Existential termination is undecidable for MBD with interleaving semantics;
- Both existential and universal termination are undecidable for MBD with maximal parallelism semantics.

The comparison of a (non-deterministic) model with its deterministic fragment is an interesting topic in automata theory, that has recently attracted the interest of the research community working on membrane computing (see, e.g., [11] and the references therein). From the results recalled in this paper, we deduce the following:

- The deterministic fragment of PEP is as powerful as the full (non-deterministic) PEP calculus w.r.t. the ability to encode RAMs;
- There exists a gap between the deterministic fragment and the full MBD calculus,

as there exists a weak, existential termination preserving encoding of RAMs in MBD, but there exist no such encoding in the deterministic fragment of MBD (this is a consequence of the decidability of universal termination on MBD, and of the equivalence of universal and existential termination on deterministic systems).

In the present paper we showed that universal termination is a decidable property for MBD. The technique employed to prove the decidability of universal termination is based on the theory of well-structured transition systems: besides universal termination, such a theory permits to analyse other interesting properties, such as, e.g., control state maintainabiliy, inevitability and boundedness [8]. In [3] we provide the decidabiliy of some of these properties (i.e., the properties whose analysis relies on the so-called tree saturation methods) for a relevant fragment of the full Brane Calculus, i.e., the calculus comprising all the membrane interaction primitives but phago, and extended with molecules, molecule-to-molecule and membrane-to-molecule interaction primitives. In [4] we discuss the use of such decidability results for the analysis of the LDL Cholesterol Degradation Pathway [12].

References

[1] N. Busi and R. Gorrieri. On the computational power of Brane Calculi. *Transactions on Computational Systems Biology VI*, 16-43, LNCS 4220, Springer. 2006.

[2] N. Busi. On the computational power of the Mate/Bud/Drip Brane Calculus: interleaving vs. maximal parallelism. In *Proc 6th International Workshop on Membrane Computing (WMC6)*, LNCS 3850, Springer, 2006.

[3] N. Busi. Deciding Behavioural Properties in Brane Calculi. In *Proc. International Conference on Computational Methods in Systems Biology, CMSB 2006*, LNCS 4210, Springer, 2006.

[4] N. Busi and C. Zandron. Modeling and analysis of biological processes by mem(brane) calculi and systems. In *Proceedings of the Winter Simulation Conference (WSC 2006)*, ACM, 2006.

[5] L. Cardelli. Brane Calculi - Interactions of biological membranes. In *Proc. Computational Methods in System Biology 2004 (CMSB 2004)*, LNCS 3082, Springer, 2005.

[6] L. Cardelli. Abstract Machines for System Biology. Draft, 2005.

[7] L. Cardelli and A.D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177-213, 2000.

[8] A. Finkel and Ph. Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science*, 256:63-92, Elsevier, 2001.

[9] R. Freund. Asynchronous P Systems and P Systems Working in the Sequential Mode In *Proc. 5th International Workshop on Membrane Computing (WMC5)*, LNCS 3365, Springer, 2005.

[10] G. Higman. Ordering by divisibility in abstract algebras. In *Proc. London Math.* Soc., vol. 2, pages 236-366, 1952.

[11] O. H. Ibarra. Some Recent Results Concerning Deterministic P Systems. In *Proc 6th International Workshop on Membrane Computing (WMC6)*, LNCS 3850, Springer, 2006.

[12] H. Lodish, A. Berk, P. Matsudaira, C.A. Kaiser, M. Krieger, M. P. Scott, S. L. Zipursky and J. Darnell. Molecular cell biology. W.H. Freeman and Company, 4th edition, 1999.

[13] M.L. Minsky. Computation: finite and infinite machines. Prentice-Hall, 1967.

[14] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108-143, 2000.

[15] G. Păun. Membrane Computing. An Introduction. Springer, Berlin, 2002.

[16] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, E. Shapiro. BioAmbients: An Abstraction for Biological Compartments. *Theoretical Computer Science*, 325(1):141-167, Elsevier, 2004.

[17] J.C. Shepherdson and J.E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217-255, 1963.