Università degli studi di Udine

Undecidability of Model Checking in Brane Logic

Original

Availability:
This version is available http://hdl.handle.net/11390/853484 since

Publisher:

Published
DOI:10.1016/j.entcs.2008.10.025

Terms of use:
The institutional repository of the University of Udine (http://air.uniud.it) is provided by ARIC services. The aim is to enable open access to all the world.

Publisher copyright

(Article begins on next page)

12 October 2024

Undecidability of Model checking in Brane Logic

Giorgio Bacci[a] Marino Miculan[a,1]

[a] *Department of Mathematics and Computer Science, University of Udine, Italy*

Abstract

The Brane Calculus is a calculus intended to model the structure and the dynamics of biological membranes. In order to express properties of systems in this calculus, in previous work we have introduced a temporal-spatial logic called *Brane Logic*. A natural question of great practical importance is if model checking of this logic is decidable, that is, if it is possible to check automatically whether a given system satisfies a given formula. We have already shown that model checking is decidable for replication-free systems and guarantee-free formulas. In this paper, we show that admitting replication in systems, or any guarantee constructor in formulas (and quantifiers), leads model checking to be undecidable. Moreover, we give also a correspondence result between membranes and systems, showing that any system can be obtained by a canonical one where all information are contained on a membrane enclosing an empty compartment.

Keywords: Biological and bio-inspired computation, brane calculus, spatial logics.

1 Introduction

The *Brane Calculus* [3] is a calculus of mobile nested processes intended to model the dynamics of biological membranes. At this level of abstraction, a biological system is seen as a hierarchy of compartments, which can interact by changing their position. A process of Brane Calculus represents a system of nested membranes; the evolution of a process corresponds to membrane interactions (phagocytosis, endo/exocytosis, ...). Differently from similar spatial calculi (notably, Mobile Ambients and BioAmbients), in Brane Calculus the computational activity takes place *on* membranes, not inside them. Moreover, reactions preserve *bitonality*, that is, the even/odd parity with which components are nested inside membranes; as a consequence, fluids from inside and outside a membrane never actually mix (but can be safely "wrapped" in other membranes). This property is commonly observed in cellular-scale living systems, but not ensured in not biologically inspired calculi.

In previous work [10] we have introduced the *Brane Logic*, a modal logic designed for expressing properties about systems described using the Brane Calculus. Like Ambient Logic, our logic features *spatial* and *temporal* modalities for expressing properties about the topology and the dynamic behaviour of nested systems. However, differently from Ambient Logic, we have also a logic for expressing properties of membranes themselves.

[1] Email: miculan@dimi.uniud.it

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Membranes are more similar to CCS than to Ambients; as a consequence, the logic for membranes is similar to Hennessy-Milner logic [8], extended with spatial connectives as in [2]. These spatial connectives are useful for expressing properties about a system when it is put in a particular context, i.e., inside a membrane or close to another system. A particularly expressive form of spatial constructors are the "guarantees", e.g., "$\mathcal{A} \triangleright \mathcal{B}$" means "whenever the system comes close to another one satisfying $\mathcal{A}$, the whole system satisfies $\mathcal{B}$." Its importance in biological applications should be evident (think, e.g., of $\mathcal{A}$ as being a virus, and $\mathcal{B}$ of evolving to a virus-free system).

Now, a problem of great importance is whether model checking is decidable for this logic. In [10], we have presented a model checking algorithm for a guarantee-free fragment of the logic against replication-free systems. In this paper, first we show that model checking of guarantee-free formulas against systems with replication is undecidable. Then, we show that also admitting any guarantee constructor in formulas (and in presence of quantifiers), leads model checking to be undecidable. We give also a correspondence result between membranes and systems, showing that any system with arbitrarily nested compartments can be obtained by a canonical one composed by an empty compartment enclosed by a membrane carrying all the information.

2 Summary of Brane Calculus and Brane Logic

Brane Calculus In this paper we focus on the basic version of Brane Calculus without communication primitives and molecular complexes. For a description of the intuitive meaning of the language and the reduction rules, we refer the reader to [3].

Syntax of (Basic) Brane Calculus

Systems Π :	$P, Q ::= \diamond \mid \sigma\llparenthesis P \rrparenthesis \mid P \circ Q \mid !P$
Membranes Σ :	$\sigma, \tau ::= \mathbf{0} \mid \sigma \mid \tau \mid a.\sigma \mid !\sigma$
Actions Ξ :	$a, b ::= \mathfrak{o}_n \mid \mathfrak{o}_n^{\perp}(\sigma) \mid \mathfrak{o}_n \mid \mathfrak{o}_n^{\perp} \mid \circledcirc(\sigma)$

where n is taken from a countable set Λ of *names*. We will write a, $\llparenthesis P \rrparenthesis$ and $\sigma\llparenthesis \rrparenthesis$, instead of $a.\mathbf{0}$, $\mathbf{0}\llparenthesis P \rrparenthesis$ and $\sigma\llparenthesis \diamond \rrparenthesis$, respectively. The set of free names of a system P, of a membrane σ and of an action a, denoted by $\mathsf{FN}(P)$, $\mathsf{FN}(\sigma)$, $\mathsf{FN}(a)$ respectively, are defined as usual; notice that in this syntax there are no binders.

Systems can be rearranged according to a structural congruence relation ($\equiv$); the intended meaning is that two congruent terms actually denote the same "semantic" system.

Structural Congruence

$$P \circ Q \equiv Q \circ P \qquad P \circ (Q \circ R) \equiv (P \circ Q) \circ R \qquad P \circ \diamond \equiv P \qquad !\diamond \equiv \diamond$$

$$\mathbf{0}\llparenthesis \diamond \rrparenthesis \equiv \diamond \qquad !(P \circ Q) \equiv !P \circ !Q \qquad !!P \equiv !P \qquad !P \equiv P \circ !P$$

$$\sigma|\tau \equiv \tau|\sigma \qquad \sigma|(\tau|\rho) \equiv (\sigma|\tau)|\rho \qquad \sigma|\mathbf{0} \equiv \sigma$$

$$!\mathbf{0} \equiv \mathbf{0} \qquad !(\sigma|\tau) \equiv !\sigma|!\tau \qquad !!\sigma \equiv !\sigma \qquad !\sigma \equiv \sigma|!\sigma$$

$$\frac{P \equiv Q}{P \circ R \equiv Q \circ R} \qquad \frac{P \equiv Q}{!P \equiv !Q} \qquad \frac{\sigma \equiv \tau}{\sigma|\rho \equiv \tau|\rho} \qquad \frac{\sigma \equiv \tau}{!\sigma \equiv !\tau}$$

$$\frac{P \equiv Q \quad \sigma \equiv \tau}{\sigma\llparenthesis P \rrparenthesis \equiv \tau\llparenthesis Q \rrparenthesis} \qquad \frac{a \equiv b \quad \sigma \equiv \tau}{a.\sigma \equiv b.\tau} \qquad \frac{\sigma \equiv \tau}{\circledcirc(\sigma) \equiv \circledcirc(\tau)} \qquad \frac{\sigma \equiv \tau}{\mathfrak{o}_n^{\perp}(\sigma) \equiv \mathfrak{o}_n^{\perp}(\tau)}$$

With respect to the structural congruence of [3], we have added the possibility of rearranging the sub-membranes contained in co-phago and pino actions (last three rules of the table

above). Decidability of structural congruence can be proved as for Mobile Ambients [5] (actually our situation is simpler because we do not have restriction on names). In particular, in the structural congruence of membranes, co-phago and pino actions can be treated as "ambient-like" constructors, since they "embed" a membrane within another.

The dynamic behaviour of Brane Calculus is specified by means of a reduction relation ("reaction") between systems $P \Rightarrow Q$, whose rules are the following:

Operational Semantics

$$\mathfrak{o}^{\perp}_n(\rho).\tau|\tau_0\llparenthesis Q\rrparenthesis \circ \mathfrak{o}_n.\sigma|\sigma_0\llparenthesis P\rrparenthesis \Rightarrow \tau|\tau_0\llparenthesis \rho\llparenthesis\sigma|\sigma_0\llparenthesis P\rrparenthesis\rrparenthesis \circ Q\rrparenthesis \qquad \text{(React phago)}$$

$$\mathfrak{o}^{\perp}_n.\tau|\tau_0\llparenthesis \mathfrak{o}_n.\sigma|\sigma_0\llparenthesis P\rrparenthesis \circ Q\rrparenthesis \Rightarrow \sigma|\sigma_0|\tau|\tau_0\llparenthesis Q\rrparenthesis \circ P \qquad \text{(React exo)}$$

$$\circledcirc(\rho).\sigma|\sigma_0\llparenthesis P\rrparenthesis \Rightarrow \sigma|\sigma_0\llparenthesis \rho\llparenthesis \diamond\rrparenthesis \circ P\rrparenthesis \qquad \text{(React pino)}$$

$$\frac{P \Rightarrow Q}{\sigma\llparenthesis P\rrparenthesis \Rightarrow \sigma\llparenthesis Q\rrparenthesis} \qquad \frac{P \Rightarrow Q}{P \circ R \Rightarrow Q \circ R} \qquad \text{(React loc, React comp)}$$

$$\frac{P \equiv P' \quad P' \Rightarrow Q' \quad Q' \equiv Q}{P \Rightarrow Q} \qquad \text{(React equiv)}$$

We denote by $\Rightarrow^*$ the usual reflexive and transitive closure of $\Rightarrow$.

As in [3], the Mate-Bud-Drip calculus is easily encoded, as follows:

Derived membrane constructors and reaction

Mate : $\mathsf{mate}_n.\sigma \triangleq \mathfrak{o}_n.\mathfrak{o}_{n'}.\sigma \qquad \mathsf{mate}^{\perp}_n.\tau \triangleq \mathfrak{o}^{\perp}_n(\mathfrak{o}^{\perp}_{n'}.\mathfrak{o}_{n''}).\mathfrak{o}^{\perp}_{n''}.\tau$

$\mathsf{mate}_n.\sigma|\sigma_0\llparenthesis P\rrparenthesis \circ \mathsf{mate}^{\perp}_n.\tau|\tau_0\llparenthesis Q\rrparenthesis \Rightarrow^* \sigma|\sigma_0|\tau|\tau_0\llparenthesis P \circ Q\rrparenthesis$

Bud : $\mathsf{bud}_n.\sigma \triangleq \mathfrak{o}_n.\sigma \qquad \mathsf{bud}^{\perp}_n(\rho).\tau \triangleq \circledcirc(\mathfrak{o}^{\perp}_n(\rho).\mathfrak{o}_{n'}).\mathfrak{o}^{\perp}_{n'}.\tau$

$\mathsf{bud}^{\perp}_n(\rho).\tau|\tau_0\llparenthesis \mathsf{bud}_n.\sigma|\sigma_0\llparenthesis P\rrparenthesis \circ Q\rrparenthesis \Rightarrow^* \rho\llparenthesis\sigma|\sigma_0\llparenthesis P\rrparenthesis\rrparenthesis \circ \tau|\tau_0\llparenthesis Q\rrparenthesis$

Drip : $\mathsf{drip}_n.(\rho).\sigma \triangleq \circledcirc(\circledcirc(\rho).\mathfrak{o}_n).\mathfrak{o}^{\perp}_n.\sigma$

$\mathsf{drip}_n(\rho).\sigma|\sigma_0\llparenthesis P\rrparenthesis \Rightarrow^* \rho\llparenthesis\rrparenthesis \circ \sigma|\sigma_0\llparenthesis P\rrparenthesis$

Instead of using this encoding, in the following, we consider mate, bud and drip actions as atomic actions to be appended to the Phago-Exo-Pino calculus. Notice that, due to this choice, we will use $\Rightarrow$ instead of $\Rightarrow^*$ in mate, bud and drip reactions, and we will not explicitly deal with the freshness of auxiliary names used in the encoding.

Brane Logic has been introduced in [10] for expressing properties of systems in Brane Calculus. Notice that the actions take place *on membranes*, not only in systems. Thus, there are actually two spatial logics, interacting with each other: one for reasoning about membranes (called *membrane logic*) and one for reasoning about systems (the *system logic*).

The syntax of the Brane Logic is the following:

Syntax of Brane Logic

System formulas Φ

$\mathcal{A}, \mathcal{B} ::=$	$\mathbf{T} \mid \neg\mathcal{A} \mid \mathcal{A} \vee \mathcal{B}$	(classical propositional fragment)
	$\diamond \mid \mathcal{M}\llparenthesis\mathcal{A}\rrparenthesis \mid \mathcal{A}@\mathcal{M}$	(void system, compartment, compartment adjoint)
	$\mathcal{A} \circ \mathcal{B} \mid \mathcal{A} \rhd \mathcal{B}$	(spatial composition, composition adjoint)
	$\Diamond\mathcal{A} \mid \Diamond\!\!\!\!\diamond\mathcal{A}$	(eventually modality, somewhere modality)
	$\forall x.\mathcal{A}$	(quantification over names)

Membrane formulas Ω

$\mathcal{M}, \mathcal{N} ::=$	$\mathbf{T} \mid \neg\mathcal{M} \mid \mathcal{M} \vee \mathcal{N}$	(classical propositional fragment)
	$\mathbf{0}$	(void membrane)
	$\mathcal{M} \vert \mathcal{N} \mid \mathcal{M} \blacktriangleright \mathcal{N}$	(spatial composition, composition adjoint)
	$\langle\alpha\rangle\mathcal{M}$	(action modality)

Action formulas Θ

$\alpha, \beta ::=$	$\mathfrak{d}_\eta \mid \mathfrak{d}_\eta^\perp(\mathcal{M})$	(phago, co-phago)
	$\mathfrak{o}_\eta \mid \mathfrak{o}_\eta^\perp$	(exo, co-exo)
	$\circledcirc(\mathcal{M})$	(pino)
$\eta ::=$	$n \mid x$	(terms)

Given a formula $\mathcal{A}$, its free names $\mathsf{FN}(\mathcal{A})$ are easily defined, since there are no binders for names. Similarly, we can define the set of free variables $\mathsf{FV}(\mathcal{A})$, noticing that the only binder for variables is the universal quantifier. As usual, a formula $\mathcal{A}$ is *closed* if $\mathsf{FV}(\mathcal{A}) = \varnothing$. For sake of simplicity, we will use the shorthands $\mathcal{M}\llparenthesis\llparenthesis$ and $\langle\alpha\rangle$ in place of $\mathcal{M}\llparenthesis\diamond\rrparenthesis$ and $\langle\alpha\rangle\mathbf{0}$ respectively.

For an intuitive explanation of these logical constructors, see [10].

The meaning of a formula is defined by means of a family of *satisfaction* relations, one for each syntactic sort of logical formulas [2]

$$\models \subseteq \Pi \times \Phi \qquad \models \subseteq \Sigma \times \Omega \qquad \models \subseteq \Xi \times \Theta$$

These relations are defined by (mutual) induction on the syntax of the formulas. Let us start with satisfaction of system formulas. First, we have to introduce the *subsystem* relation $P \downarrow Q$ (read "Q is an immediate subsystem of P"):

$$P \downarrow Q \triangleq \exists P' : \Pi, \sigma : \Sigma . P \equiv \sigma\llparenthesis Q \rrparenthesis \vert P'$$

We denote by $\downarrow^*$ the reflexive-transitive closure of $\downarrow$.

Satisfaction of System Formulas

$\forall P : \Pi$	$P \models \mathbf{T}$	
$\forall P : \Pi, \mathcal{A} : \Phi$	$P \models \neg\mathcal{A}$	$\triangleq P \not\models \mathcal{A}$
$\forall P : \Pi, \mathcal{A}, \mathcal{B} : \Phi$	$P \models \mathcal{A} \vee \mathcal{B}$	$\triangleq P \models \mathcal{A} \vee P \models \mathcal{B}$
$\forall P : \Pi$	$P \models \diamond$	$\triangleq P \equiv \diamond$
$\forall P : \Pi, \mathcal{A} : \Phi, \mathcal{M} : \Omega$	$P \models \mathcal{M}\llparenthesis\mathcal{A}\rrparenthesis$	$\triangleq \exists P' : \Pi, \sigma : \Sigma . P \equiv \sigma\llparenthesis P' \rrparenthesis \wedge P' \models \mathcal{A} \wedge \sigma \models \mathcal{M}$
$\forall P : \Pi, \mathcal{A}, \mathcal{B} : \Phi$	$P \models \mathcal{A} \circ \mathcal{B}$	$\triangleq \exists P', P'' : \Pi . P \equiv P' \circ P'' \wedge P' \models \mathcal{A} \wedge P'' \models \mathcal{B}$
$\forall P : \Pi, \mathcal{A} : \Phi, x : \vartheta$	$P \models \forall x . \mathcal{A}$	$\triangleq \forall m : \Lambda . P \models \mathcal{A}\{x \leftarrow m\}$
$\forall P : \Pi, \mathcal{A} : \Phi$	$P \models \Diamond\mathcal{A}$	$\triangleq \exists P' : \Pi . P \Rrightarrow^* P' \wedge P' \models \mathcal{A}$
$\forall P : \Pi, \mathcal{A} : \Phi$	$P \models \Diamond\!\!\!\!\!+\mathcal{A}$	$\triangleq \exists P' : \Pi . P \downarrow^* P' \wedge P' \models \mathcal{A}$
$\forall P : \Pi, \mathcal{A} : \Phi, \mathcal{M} : \Omega$	$P \models \mathcal{A}@\mathcal{M}$	$\triangleq \forall \sigma : \Sigma . \sigma \models \mathcal{M} \Rightarrow \sigma\llparenthesis P \rrparenthesis \models \mathcal{A}$
$\forall P : \Pi, \mathcal{A}, \mathcal{B} : \Phi$	$P \models \mathcal{A} \rhd \mathcal{B}$	$\triangleq \forall P' : \Pi . P' \models \mathcal{A} \Rightarrow P \circ P' \models \mathcal{B}$

This definition relies on the satisfaction of membrane formulas, which we define next.

[2] We use the same symbol $\models$ for the three relations, since it is easily distinguishable from the context.

Satisfaction of membrane formulas

$\forall \sigma : \Sigma$	$\sigma \vDash \mathbf{T}$	
$\forall \sigma : \Sigma, \mathcal{M} : \Omega$	$\sigma \vDash \neg \mathcal{M}$	$\triangleq \sigma \nvDash \mathcal{M}$
$\forall \sigma : \Sigma, \mathcal{M}, \mathcal{N} : \Omega$	$\sigma \vDash \mathcal{M} \vee \mathcal{N}$	$\triangleq \sigma \vDash \mathcal{M} \vee \sigma \vDash \mathcal{M}$
$\forall \sigma : \Sigma$	$\sigma \vDash \mathbf{0}$	$\triangleq \sigma \equiv \mathbf{0}$
$\forall \sigma : \Sigma, \mathcal{N}, \mathcal{M} : \Omega$	$\sigma \vDash \mathcal{M} \vert \mathcal{N}$	$\triangleq \exists \sigma', \sigma'' : \Sigma . \sigma \equiv \sigma' \vert \sigma'' \wedge \sigma' \vDash \mathcal{M} \wedge \sigma'' \vDash \mathcal{N}$
$\forall \sigma : \Sigma, \alpha : \Theta$	$\sigma \vDash \langle \alpha \rangle \mathcal{M}$	$\triangleq \exists \sigma', \sigma'' : \Sigma . \exists a : \Gamma . \sigma \equiv (a.\sigma') \vert \sigma'' \wedge a \vDash \alpha \wedge \sigma' \vert \sigma'' \vDash \mathcal{M}$
$\forall \sigma : \Sigma, \mathcal{M}, \mathcal{N} : \Omega$	$\sigma \vDash \mathcal{M} \blacktriangleright \mathcal{N}$	$\triangleq \forall \sigma' : \Sigma . \sigma' \vDash \mathcal{M} \Rightarrow \sigma \vert \sigma' \vDash \mathcal{N}$

In particular, the truth of the action modality $\langle \alpha \rangle \mathcal{M}$ is defined using satisfaction of action formulas $a \vDash \mathcal{M}$, which we have to define next.

Satisfaction of action formulas

$\forall a : \Gamma, n : \Lambda$	$a \vDash \mathfrak{O}_n$	$\triangleq a = \mathfrak{O}_n$
$\forall a : \Gamma, n : \Lambda, \mathcal{M} : \Omega$	$a \vDash \mathfrak{O}_n^{\perp}(\mathcal{M})$	$\triangleq \exists \sigma : \Sigma . a = \mathfrak{O}_n^{\perp}(\sigma) \wedge \sigma \vDash \mathcal{M}$
$\forall a : \Gamma, n : \Lambda$	$a \vDash \mathfrak{O}_n$	$\triangleq a = \mathfrak{O}_n$
$\forall a : \Gamma, n : \Lambda$	$a \vDash \mathfrak{O}_n^{\perp}$	$\triangleq a = \mathfrak{O}_n^{\perp}$
$\forall a : \Gamma, \mathcal{M} : \Omega$	$a \vDash \circledcirc(\mathcal{M})$	$\triangleq \exists \sigma : \Sigma . a = \circledcirc(\sigma) \wedge \sigma \vDash \mathcal{M}$

Notice that the satisfaction of action formulas is defined in terms of the satisfaction of membrane formulas, therefore these are two mutually defined judgments.

In a given membrane or system, even if infinite, only a finite number of names can appear, because Brane Calculus processes cannot create fresh names (differently from, e.g., π-calculus). As a consequence, for instance, some formulas quantifying over all possible names cannot be satisfied (e.g., $\forall x . \langle \mathfrak{O}_x \rangle \mathfrak{C} \mathbf{T} \mathfrak{D}$ is satisfied by systems which can perform a phago on all possible names, which is clearly impossible). However, this kind of formulas may become satisfiable in future extensions of the Brane Calculus, incorporating the possibility of generating fresh names (as it is already possible in Beta-binders [15]).

3 Undecidability of satisfaction in presence of replication

Using the approach of [4] we show that if the processes have unbound replication either on the membranes or on the systems, model checking for the Brane Calculus against the Brane Logic is undecidable. In fact, the fragment of the logic needed for this result is very restricted: it contains only propositional connectives, temporal and spatial modalities and the compartment connective. There is no need of quantifiers or adjoint connectives.

The undecidability proof is done by a reduction of the Post Correspondence Problem (PCP). In the following we use α, β, γ for words in $\{a, b\}^*$, σ for letters in $\{a, b\}$ and ϵ for the empty word. An instance of PCP is a set of pairs of words $\{(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)\}$ over the two-letter alphabet $\{a, b\}$ (that is, $\alpha_i, \beta_i \in \{a, b\}^*$). The question is whether there exists a sequence $i_0, i_1, \ldots, i_k$ ($1 \leq i_j \leq n$ for all $0 \leq j \leq k$) such that $\alpha_{i_0} \cdot \ldots \cdot \alpha_{i_k} = \beta_{i_0} \cdot \ldots \cdot \beta_{i_k}$, where $\cdot$ denotes word concatenation. It is well known that PCP is undecidable [13].

The idea of the reduction is to construct for a given instance of PCP a system $\mathbf{PCP}$ whose reductions simulates all possible concatenations of pairs of words in the instance. Then we have only to check if a system representing two equal words is reachable. This approach is used both for showing the undecidability in the case of replication on sys-

tems, and in the case of replication on membranes. To discriminate the two distinct cases we define $\mathbf{PCP}_S$ for systems (where we admit replication only on systems); $\mathbf{PCP}_m$ for membranes (where we admit replication only on membranes).

3.1 Replication on systems

The system $\mathbf{PCP}_S$ is defined as the composition

$$\mathbf{PCP}_S \triangleq \mathsf{mate}^{\perp}_{start}\llparenthesis \mathbf{Word}_1(\epsilon) \circ \mathbf{Word}_2(\epsilon) \circ \mathbf{End} \rrparenthesis \circ \mathbf{Concatenate} \circ \mathbf{Compare}$$

where $\mathbf{Word}_i(\gamma)$ is a system representing the word γ.

Before giving the definition of the entire system, we briefly describe the leading idea. $\mathbf{Concatenate}$ is the system responsible for concatenating pairs of words from the given instance of PCP: a pair (α_i, β_i) is nondeterministically choosen and $\mathbf{Word}_1(\alpha) \circ \mathbf{Word}_2(\beta)$ is rewritten to $\mathbf{Word}_1(\alpha_i \cdot \alpha) \circ \mathbf{Word}_2(\beta_i \cdot \beta)$; this is done again and again. $\mathbf{Compare}$ is the system deputed to check if the two words represented by $\mathbf{Word}_1$ and $\mathbf{Word}_2$ are equal. This is done by nondeterministically choosing the letter a or b and trying to delete it simultaneously from both words; this is repeated until both words are empty or they starts with a different letter. In this way, an instance of PCP has a solution if and only if there exists a (nonempty) execution of $\mathbf{PCP}_S$ that ends with the representation of two empty words.

$\mathbf{Word}_1$ and $\mathbf{Word}_2$ are enveloped in a "protective" membrane designed to permit only a kind of manipulation per time, i.e. $\mathbf{Compare}$ could enter and work on the words if and only if $\mathbf{Concatenate}$ has terminated is job and viceversa. This membrane is also used for synchronization during the concatenation process. $\mathbf{End}$ is the system responsible for the final step of all jobs: it waits for a signal (in the form of a small membrane) from the manipulator system, then recreates the protective membrane giving it the capability to accept the new manipulation.

Given an instance $\{(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)\}$, then $\mathbf{Concatenate}$ is defined as follows

$$\mathbf{Concatenate} \triangleq !\mathbf{Concatenate}(\alpha_1, \beta_1) \circ \ldots \circ !\mathbf{Concatenate}(\alpha_n, \beta_n)$$

where $\mathbf{Concatenate}(\alpha_i, \beta_i)$ is one of the infinite replications which performs the concatenation: one (and only one) $\mathbf{Concatenate}(\alpha_i, \beta_i)$ enters the protective membrane and then do the requested concatenation. Actually, the following property holds

$$\mathsf{mate}^{\perp}_{start}\llparenthesis \mathbf{Word}_1(\alpha) \circ \mathbf{Word}_2(\beta) \circ \mathbf{End} \rrparenthesis \circ \mathbf{Concatenate}(\alpha_i, \beta_i) \quad \Rightarrow^* \quad \mathsf{mate}^{\perp}_{start}\llparenthesis \mathbf{Word}_1(\alpha_i \cdot \alpha) \circ \mathbf{Word}_2(\beta_i \cdot \beta) \circ \mathbf{End} \rrparenthesis$$

Intuitively, a string $\gamma = \sigma_1 \ldots \sigma_k$ in $\{a, b\}^*$ is represented by an ordered nesting of membranes such that each membrane shows an action labelled with the character σ_i. In order to distinguish $\mathbf{Word}_1$ from $\mathbf{Word}_2$ we envelope the strings-encoding in a membrane "labelled" with a fresh such as w_i.

Definition of $\mathbf{Word}_i(\gamma)$

$$\begin{aligned}
\mathbf{Word}_i(\gamma) &\triangleq \mathsf{mate}^{\perp}_{w_i}\llparenthesis \mathbf{String}(\gamma) \rrparenthesis \\
\mathbf{String}(\epsilon) &\triangleq \mathsf{mate}^{\perp}_{op}\llparenthesis \diamond \rrparenthesis \\
\mathbf{String}(\sigma\alpha) &\triangleq \mathsf{mate}^{\perp}_{op}\llparenthesis \mathsf{mate}_{\sigma}\llparenthesis \mathbf{String}(\alpha) \rrparenthesis \rrparenthesis
\end{aligned}$$

Notice that, since the strings are represented as a nested structure, if we want to concatenate two words we need to insert the fist string into the other. For this reason we include in the encoding the ability to attach actions on strings, therefore, a string could be programmed by a process to move into another string: this is the task of the membrane $\mathsf{mate}^{\perp}_{op}\mathfrak{C}\ldots\mathfrak{D}$.

Definition of $\mathbf{Concatenate}(\alpha,\beta)$

$$
\begin{array}{lcl}
\mathbf{Concatenate}(\alpha,\beta) & \triangleq & \mathsf{mate}_{start}.\mathfrak{S}^{\perp}_{start}\mathfrak{C} \\
 & & \mathsf{mate}_{w_1}.\mathfrak{S}^{\perp}_{w_1}\mathfrak{C}\mathbf{Patch}(\alpha)\circ\mathbf{String}'(\alpha,\mathbf{Signal}_1(\alpha^R))\mathfrak{D}\circ \\
 & & \mathsf{mate}_{w_2}.\mathfrak{S}^{\perp}_{w_2}\mathfrak{C}\mathbf{Patch}(\beta)\circ\mathbf{String}'(\beta,\mathbf{Signal}_2(\beta^R))\mathfrak{D}\mathfrak{D} \\
\mathbf{Patch}(\gamma) & \triangleq & \mathsf{mate}_{op}.MoveIn(\gamma)\mathfrak{C}\diamond\mathfrak{D} \\
MoveIn(\epsilon) & \triangleq & \mathsf{mate}^{\perp}_{op}.\mathbf{0} \\
MoveIn(\sigma\alpha) & \triangleq & \mathfrak{S}_{\sigma}.MoveIn(\alpha) \\
\mathbf{String}'(\epsilon,\mathbf{P}) & \triangleq & \mathbf{P} \\
\mathbf{String}'(\sigma\alpha,\mathbf{P}) & \triangleq & \mathfrak{S}^{\perp}_{\sigma}(\mathsf{mate}_{make}).\mathfrak{S}^{\perp}_{op}\mathfrak{C}\mathsf{mate}^{\perp}_{make}.\mathfrak{S}^{\perp}_{\sigma}\mathfrak{C}\mathbf{String}'(\alpha,\mathbf{P})\mathfrak{D}\mathfrak{D} \\
\mathbf{Signal}_i(\epsilon) & \triangleq & \mathfrak{S}_{w_i}.\mathsf{mate}^{\perp}_{w_i}\mathfrak{C}\mathsf{mate}_{s_i}\mathfrak{C}\diamond\mathfrak{D}\mathfrak{D} \\
\mathbf{Signal}_i(\sigma\alpha) & \triangleq & \mathfrak{S}_{\sigma}.\mathsf{mate}_{\sigma}\mathfrak{C}\mathfrak{S}_{op}.\mathsf{mate}^{\perp}_{op}\mathfrak{C}\mathbf{Signal}_i(\alpha)\mathfrak{D}\mathfrak{D}
\end{array}
$$

$\mathbf{Concatenate}(\alpha_i,\beta_i)$ enters the protective membrane that envelops the two words, and leads the strings in $\mathbf{Word}_1$ and $\mathbf{Word}_2$ respectively inside α_i and β_i, which are the instances of PCP it carries. Each concatenation takes place in two distinct membranes (the two membranes that cover and give the name to the words), so that each activity is disjunct from the other. $\mathbf{Patch}$ "programs" the string in $\mathbf{Word}$ attaching a list of actions ($MoveIn$) that forces it to enter $\mathbf{String}'$. $\mathbf{String}'$ differs from the previously given encoding of a string: it is defined such that it will become a string (in the sense of our encoding) only after the string concatenation process is done. To make possible this transformation *only after* the concatenation process is finished, we have defined $\mathbf{String}'$ with an auxiliary process $\mathbf{Signal}$ in it, that reconstructs the right encoding and releases a signal to $\mathbf{End}$.

Definition of $\mathbf{End}$

$$\mathbf{End} \triangleq\ !(\mathsf{mate}^{\perp}_{s_1}.\mathsf{mate}^{\perp}_{s_2}.\mathfrak{S}_{start}.\mathsf{mate}^{\perp}_{start}\mathfrak{C}\diamond\mathfrak{D})$$

$\mathbf{End}$ waits for the signal from the two concatenation processes and, only if both correctly terminate, it recomposes the original state of the protective external membrane, in order to recreate the initial conditions that permits a new concatenation process.

The comparison is carried out by the process $\mathbf{Compare}$

$$\mathbf{Compare} \triangleq !\mathbf{Consume}(a)\circ!\mathbf{Consume}(b)$$

$\mathbf{Consume}(\sigma)$ is the system which deletes the first character both in $\mathbf{Word}_1(\alpha)$ and $\mathbf{Word}_2(\beta)$ if $\alpha = \sigma\alpha'$ and $\beta = \sigma\beta'$, otherwise terminates. $\mathbf{Consume}(\sigma)$ is defined such that the following property holds

$$\mathsf{mate}^{\perp}_{start}\mathfrak{C}\mathbf{Word}_1(\sigma\alpha)\circ\mathbf{Word}_2(\sigma\beta)\circ\mathbf{End}\mathfrak{D}\circ\mathbf{Consume}(\sigma) \quad \Rightarrow^{*} \quad \mathsf{mate}^{\perp}_{start}\mathfrak{C}\mathbf{Word}_1(\alpha)\circ\mathbf{Word}_2(\beta)\circ\mathbf{End}\mathfrak{D}$$

Definition of $\mathbf{Consume}(\sigma)$

$$
\begin{array}{lcl}
\mathbf{Consume}(\sigma) & \triangleq & \mathsf{mate}_{start}.\mathfrak{S}^{\perp}_{start}\llparenthesis \\
 & & \quad \mathsf{mate}_{w_1}.\mathfrak{S}^{\perp}_{w_1}\llparenthesis\mathbf{DelChar}(\sigma,\mathbf{Signal}_1)\rrparenthesis\circ \\
 & & \quad \mathsf{mate}_{w_2}.\mathfrak{S}^{\perp}_{w_2}\llparenthesis\mathbf{DelChar}(\sigma,\mathbf{Signal}_2)\rrparenthesis\rrparenthesis \\
\mathbf{DelChar}(\sigma,\mathbf{P}) & \triangleq & \mathsf{mate}_{op}.\mathfrak{S}^{\perp}_{op}.\mathfrak{S}^{\perp}_{exit}\llparenthesis\mathsf{mate}^{\perp}_{\sigma}.\mathfrak{S}^{\perp}_{\sigma}.\mathfrak{S}^{\perp}_{exit}\llparenthesis\mathbf{Patch}\circ\mathbf{Del}\circ\mathbf{P}\rrparenthesis\rrparenthesis \\
\mathbf{Patch} & \triangleq & \mathsf{mate}_{op}.\mathfrak{S}_{del}.\mathsf{mate}^{\perp}_{op}\llparenthesis\diamond\rrparenthesis \\
\mathbf{Del}(\sigma) & \triangleq & \mathfrak{S}^{\perp}_{del}(\mathfrak{S}_{op}).\mathfrak{S}_{\sigma}\llparenthesis\diamond\rrparenthesis \\
\mathbf{Signal}_i & \triangleq & \mathfrak{S}_{exit}\llparenthesis\mathfrak{S}_{exit}\llparenthesis\mathfrak{S}_{w_i}.\mathsf{mate}^{\perp}_{w_i}\llparenthesis\mathsf{mate}_{s_i}\llparenthesis\diamond\rrparenthesis\rrparenthesis\rrparenthesis\rrparenthesis
\end{array}
$$

$\mathbf{Consume}$ enters the external protective membrane, and works separately on the two words. $\mathbf{DelChar}$ attacks the encoding of the string, enters the first two membranes (i.e. the two membrane that represent the first character) releasing three subprocess: $\mathbf{Patch}$, $\mathbf{Del}$ and $\mathbf{Signal}$. $\mathbf{Patch}$ attaches on the surface of the substring (i.e., the string without the first character) the actions that force it to enter $\mathbf{Del}$, which moves the substring out of the membrane representing the first character. Now the substring and the first character are separated. Finally $\mathbf{Signal}$ exits from what remains of the double compartment (the first character) and from the membrane that covers the whole word; then, it releases to $\mathbf{End}$ the signal that the character has been completely cancelled. It is interesting to note that $\mathbf{Signal}$ also dissolves the double compartment, thus deleting what remains of the first character. If $\mathbf{End}$ receives both signals from the two deletion processes (i.e. the comparison is successful), it restores the original state of the protective membrane.

We have the following theorem:

Theorem 3.1 *The model checking problem for Brane Calculi with replication on systems against the Brane Logic is undecidable.*

Proof. Let $\mathbf{PCP}_S$ the system defined above (note that the definition of $\mathbf{PCP}_S$ depends on the instance of PCP). We have already seen that the instance has a solution if and only if there exists an execution of $\mathbf{PCP}_S$ starting with the concatenation of at least one pair and ending in a configuration representing the pair of empty words. This can be represented by the formula

$$\mathcal{A} \triangleq \Diamond(nonempty(w_1) \wedge \Diamond(empty(w_1) \wedge empty(w_2)))$$

where

$$
\begin{array}{rcl}
nonempty(w_i) & \triangleq & \Diamond\langle\mathsf{mate}^{\perp}_{w_i}\rangle\llparenthesis\langle\mathsf{mate}^{\perp}_{op}\rangle\llparenthesis\langle\mathsf{mate}_a\rangle\llparenthesis\mathbf{T}\rrparenthesis \vee \langle\mathsf{mate}_b\rangle\llparenthesis\mathbf{T}\rrparenthesis\rrparenthesis\rrparenthesis \\
empty(w_i) & \triangleq & \Diamond\langle\mathsf{mate}^{\perp}_{w_i}\rangle\llparenthesis\langle\mathsf{mate}^{\perp}_{op}\rangle\llparenthesis\diamond\rrparenthesis\rrparenthesis
\end{array}
$$

Here w_i is a name used in the encoding of the process $\mathbf{Word}_i(\gamma)$, and the formula $\langle\mathsf{mate}^{\perp}_{op}\rangle\llparenthesis\langle\mathsf{mate}_a\rangle\llparenthesis\mathbf{T}\rrparenthesis \vee \langle\mathsf{mate}_b\rangle\llparenthesis\mathbf{T}\rrparenthesis\rrparenthesis$ is matched by (the encoding of) the first letter in the word γ. It is easy to check that, for systems obtained from the translation $\mathbf{PCP}_S$, it is $nonempty(w_i) \iff \neg empty(w_i)$.

Then, $\mathbf{PCP}_S \models \mathcal{A}$ if and only if the instance of PCP has a solution. $\square$

3.2 Replication on membranes

For the case of replication on membranes, we do not directly define the system $\mathbf{PCP}_m$ as done for the case of replication on systems, but we will reduce its definition to the definition of $\mathbf{PCP}_S$. In fact, we show that given any system P, there exists a single empty "bubble" $\sigma\llparenthesis\diamond\rrparenthesis$, with the correct membrane, which reduces exactly to P. This allows to reduce the model checking for membranes to that for systems.

This special membrane, which we denote by $\mathbf{Generate}_\phi(P)$ where $\phi : \Pi \to \Lambda$ is an *injective* labeling function from systems to names, can be generated by induction on the structure of P, as follows:

Definition of $\mathbf{Generate}_\phi(P)$

$$
\begin{array}{lcl}
\mathbf{Generate}_\phi(\diamond) & \triangleq & \mathbf{0} \\
\mathbf{Generate}_\phi(\sigma\llparenthesis P\rrparenthesis) & \triangleq & \mathsf{drip}(\mathbf{Endo}^\perp_\phi(P,\sigma))|\mathbf{Endo}_\phi(P) \\
\mathbf{Generate}_\phi(P \circ Q) & \triangleq & \mathsf{drip}(\mathbf{Generate}_\phi(P))|\mathsf{drip}(\mathbf{Generate}_\phi(Q)) \\
\mathbf{Generate}_\phi(!P) & \triangleq & !\mathbf{Generate}_\phi(P)
\end{array}
$$

$$
\begin{array}{lcl}
\mathbf{Endo}_\phi(\diamond) & \triangleq & \mathbf{0} \\
\mathbf{Endo}_\phi(\tau\llparenthesis Q\rrparenthesis) & \triangleq & \begin{cases} \mathbf{0} & \text{if } Q \equiv \diamond \\ \mathsf{drip}(\mathfrak{Ｏ}_{\phi(\tau\llparenthesis Q\rrparenthesis)}.\mathbf{Generate}_\phi(Q)) & \text{otherwise} \end{cases} \\
\mathbf{Endo}_\phi(P \circ Q) & \triangleq & \mathbf{Endo}_\phi(P)|\mathbf{Endo}_\phi(Q) \\
\mathbf{Endo}^\perp_\phi(\diamond,\sigma) & \triangleq & \sigma \\
\mathbf{Endo}^\perp_\phi(\tau\llparenthesis Q\rrparenthesis,\sigma) & \triangleq & \begin{cases} \circledcirc(\tau).\sigma & \text{if } Q \equiv \diamond \\ \mathfrak{Ｏ}^\perp_{\phi(\tau\llparenthesis Q\rrparenthesis)}(\tau).\sigma & \text{otherwise} \end{cases} \\
\mathbf{Endo}^\perp_\phi(P \circ Q,\sigma) & \triangleq & \mathbf{Endo}^\perp_\phi(P,\mathbf{Endo}^\perp_\phi(Q,\sigma))
\end{array}
$$

The special capabilities of this membrane is due to the following property:

Proposition 3.2 *Let P a replication-free system and $\phi : \Pi \to (\Lambda \setminus \mathsf{FN}(P))$ an injective labeling function, then:*

(i) $\mathbf{Generate}_\phi(P)\llparenthesis\diamond\rrparenthesis \Rightarrow^* P$;

(ii) $\mathbf{Generate}_\phi(!P)\llparenthesis\diamond\rrparenthesis \Rightarrow^* \mathbf{Generate}_\phi(!P)\llparenthesis\diamond\rrparenthesis \circ P$.

The first part of this proposition can be easily proved by induction on the structure of P. Note that the labeling function ϕ uses only names not in $\mathsf{FN}(P)$ so that the generation process does not influence the evolution of the system P.

The second part of Proposition 3.2 can be seen as a variant of the equivalence $!P \equiv !P \circ P$. Indeed, using this result we can replace replication on systems with replication on membrane, since $\mathbf{Generate}_\phi(!P) = !\mathbf{Generate}_\phi(P)$ (the substitution must be done only using labeling functions ϕ that does not use names in $\mathsf{FN}(P)$).

The fact that any system of the Brane Calculus can be generated from a single membrane enclosing an empty compartment, is due to the expressive power of the *endo*-actions: phago and pino actions carry a "membrane patch" which will become a nested membrane in the reaction, modifying at the same time the tree structure of the system. See Fig. 1 for an example of generation.

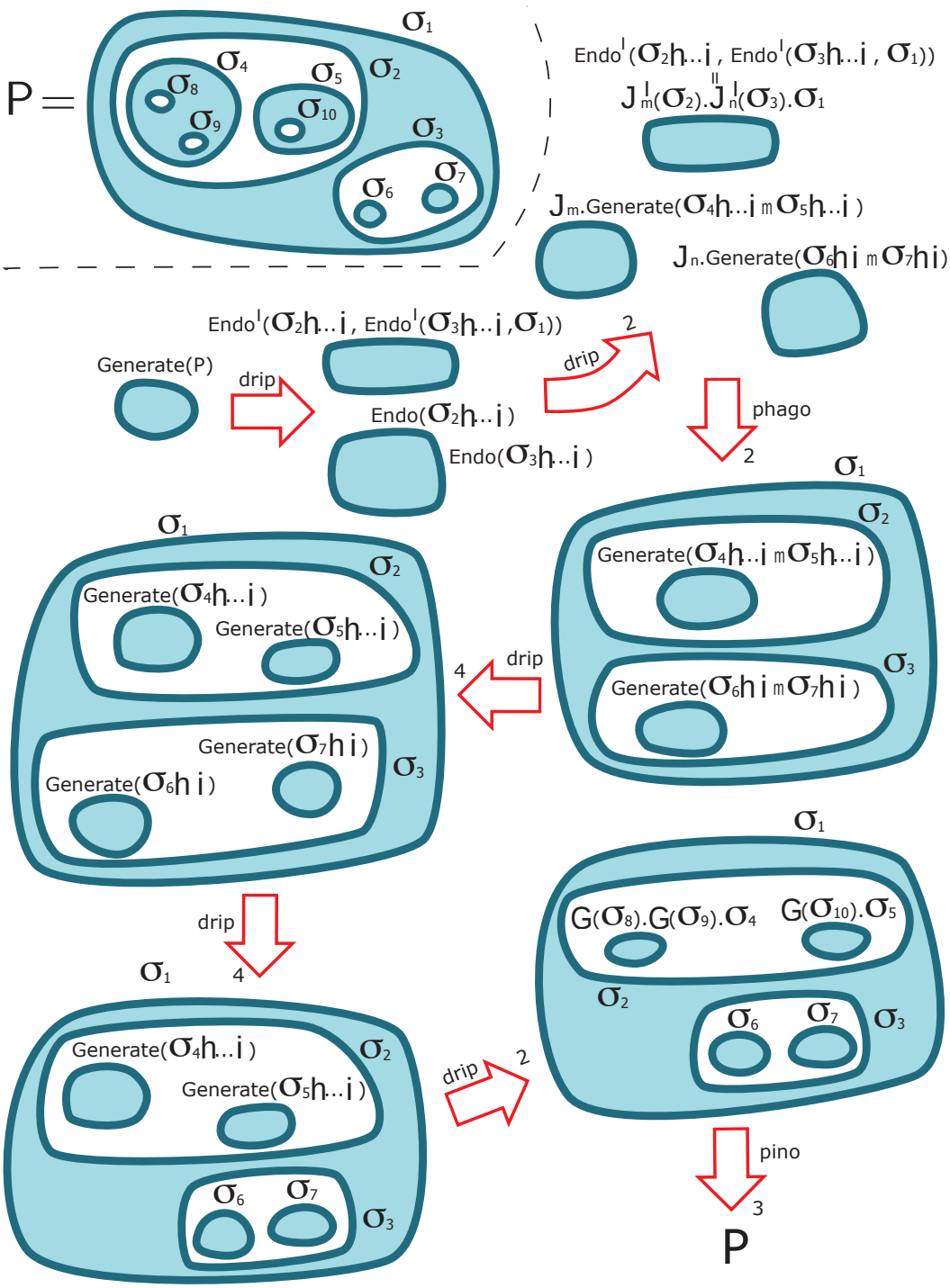

Fig. 1. **Generate**(P)⦇⋄⦈ reduces to P (m, n are names not in FN(P))

Note that a **Generate**-like membrane could also be defined using only (well-)nested pino actions and drip actions, without using phago (so without use a labeling function ϕ). However this is not easy to define by induction on systems, so we prefer the previous one.

Using Proposition 3.2(ii), in the definition of system $\mathbf{PCP}_m$ we can replace on systems with replication on membrane. So the following result holds, in virtue of Theorem 3.1.

Theorem 3.3 *The model checking problem for Brane Calculi with replication on membranes against the Brane Logic is undecidable.*

4 Undecidability in presence of guarantee and quantifiers

In this section we consider the problem of model checking the finite state Brane Calculus (i.e., without replication) against formulas that may contain guarantee and quantifiers. As observed in [10] the logic with guarantee can express the satisfiability of a formula via the $\mathcal{A}^{\mathbf{F}}$ operator, defined as $\mathcal{A}^{\mathbf{F}} \triangleq \mathcal{A} \rhd \mathbf{F}$; it is easy to check that $P \models \neg(\mathcal{A}^{\mathbf{F}})$ if and only if $\mathcal{A}$ is satisfiable. Therefore, the model checking problem subsumes the satisfiability problem of formulas: if we can decide $P \models \neg(\mathcal{A}^{\mathbf{F}})$ then we can decide whether $\mathcal{A}$ is satisfiable. We will show now that the satisfiability problem for brane formulas (even without guarantee) is an undecidable problem; this is obtained by reducing to it the finite-model problem for first-order logic. As a consequence, also model checking is undecidable.

Let us consider the set **FO** of first-order formulas defined on a countable set of variables $x, y, z, \ldots$ and some relational symbols $\{R_1, R_2, \ldots, R_k\}$, each having strictly positive arity. Formulas from **FO** are interpreted over structures; a structure $\mathcal{S}$ over some domain $\mathcal{D}$ is simply a set containing objects of the form $R_i(a_1, \ldots, a_n)$ where R_i is an n-ary relational symbol and $a_1, \ldots, a_n$ are elements of $\mathcal{D}$. We say that a structure $\mathcal{S}$ is *finite* whenever its domain $\mathcal{D}$ is finite. For first-order formula φ and a structure $\mathcal{S}$ with domain $\mathcal{D}$, a valuation σ is a mapping from the free variables of φ to $\mathcal{D}$. A structure $\mathcal{S}$ is a model for a formula φ under a valuation σ if $\mathcal{S}, \sigma \models \varphi$ (defined as usual). Then, it is undecidable to know whether a given first-order formula φ has a finite model [17].

We use this result to prove that satisfiability on Brane Logic with guarantee and quantifiers is undecidable. We give a translation $[\![\cdot]\!]$ of **FOL** in the logic for Brane Calculus; thus, we prove that a first-order formula φ admits a finite model if and only if there exists a Brane Calculus system P (without replication) that satisfies $[\![\varphi]\!]$ in the Brane Logic.

$$
\begin{array}{lcl}
[\![R_i(x_1, \ldots, x_k)]\!] & \triangleq & \langle \mathfrak{o}_{r_i} \rangle \mathfrak{C} \langle \mathfrak{o}_{x_1} \rangle \mathfrak{C} \langle \mathfrak{o}_{x_2} \rangle \mathfrak{C} \ldots \langle \mathfrak{o}_{x_k} \rangle \mathfrak{C} \diamond \mathfrak{D} \ldots \mathfrak{D} \mathfrak{D} \mathfrak{D} \circ \mathbf{T} \\
[\![\varphi \wedge \psi]\!] & \triangleq & [\![\varphi]\!] \wedge [\![\psi]\!] \\
[\![\neg\varphi]\!] & \triangleq & \neg [\![\varphi]\!] \\
[\![\exists x.\varphi]\!] & \triangleq & \exists x.((\langle \mathfrak{o}_d \rangle \mathfrak{C} \langle \mathfrak{o}_x \rangle \mathfrak{C} \diamond \mathfrak{D} \mathfrak{D} \circ \mathbf{T}) \wedge [\![\varphi]\!])
\end{array}
$$

This encoding identifies first-order variables with Brane Logic variables. Let us consider the semantics: if $\mathcal{S}$ is a model for $\exists x.\varphi$, the first-order quantificator ranges over the domain $\mathcal{D}$ of the structure but, in the encoding $\exists x.[\![\varphi]\!]$, the Brane Logic quantificator ranges over a *countable* set of names. This imply that the encoding is correct only if we consider first-order semantical structure with countable domains. In the following we consider only finite structure, so the encoding is correct.

The key idea of this encoding is to think the composition of Brane Calculus as a (multi-)set constructor. Then, the finite domain $\mathcal{D}$, as well as the structure $\mathcal{S}$ are encoded in

a straightforward way using simply action names d for elements of $\mathcal{D}$ and action names r_i for relational symbols R_i in $\mathcal{S}$. We now define the relation $\rightsquigarrow_{Sys}$ between finite structures and brane processes without replication. For a system P and structure $\mathcal{S}$ whose domain is $\mathcal{D}$, $P \rightsquigarrow_{Sys} \mathcal{S}$ if:

- $\exists P'.P \equiv \mathfrak{o}_d \mathfrak{(} \mathfrak{o}_a \mathfrak{(} \diamond \mathfrak{)} \mathfrak{)} \circ P'$ iff $a \in \mathcal{D}$
- if $a_1, \ldots, a_k \in \mathcal{D}$ then $\exists P''.P \equiv \mathfrak{o}_{r_i} \mathfrak{(} \mathfrak{o}_{a_1} \mathfrak{(} \ldots \mathfrak{o}_{a_k} \mathfrak{(} \diamond \mathfrak{)} \ldots \mathfrak{)} \mathfrak{)} \circ P''$ iff $R_i(a_1, \ldots, a_k) \in \mathcal{S}$

We denote with $\rightsquigarrow_{Struct}$ the symmetric relation of $\rightsquigarrow_{Sys}$.

Proposition 4.1 *Let φ a first-order formula of* **FO***, then the following holds:*

(i) *let $\mathcal{S}$ a finite structure over a domain $\mathcal{D}$ and σ a valuation for the free variables of φ. If $\mathcal{S}, \sigma \vDash \varphi$ and $\mathcal{S} \rightsquigarrow_{Sys} P$, then $P \vDash [\![\varphi]\!]\sigma$;*

(ii) *let P a system of finite state Brane Calculus and σ a mapping from free variables of φ to action names. If $P \vDash [\![\varphi]\!]\sigma$ and $P \rightsquigarrow_{Struct} \mathcal{S}$, then $\mathcal{S}, \sigma \vDash \varphi$.*

The proof is a trivial induction on the structure of φ.

As a result of the last proposition, the following lemma holds.

Lemma 4.2 *A closed first-order formula φ of* **FO** *admits a finite model if and only if there exists a finite state Brane Calculus system P such that $P \vDash [\![\varphi]\!]$.*

It is easy to see that Lemma 4.2 and undecidability of the existence of finite models for first-order logic yield the undecidability of the satisfiability problem of the logic without guarantee over finite state brane systems, hence holds the following theorem:

Theorem 4.3 *The model checking problem of finite states Brane Calculus against formulas with guarantee is undecidable.*

Note that the encoding uses only a little fragment of the logic, and the only use of guarantee and quantification it is enough to yield undecidability.

5 Conclusions

In this paper, we have shown the undecidability of model checking of the fragment of Brane Logic without quantifiers and adjoints, in presence of the replication (either on systems or on membranes). This result has been obtained by reducing the Post Correspondence Problem to the model checking problem, following [4]. Also, we have shown the undecidability of model checking of the fragment of Brane Logic with quantifiers and adjoints, in absence of replication. This result has been obtained by reducing the existence of finite models for first-order formulas, to the model checking problem. Beside this, we have given a correspondence between membranes and systems, showing that any system can be obtained by a canonical one where all information are contained on a membrane enclosing an empty compartment. We think that these results can be readily transposed to other variants of the Brane Calculus (with enough expressive power), such as the Projective Branes [6].

At this point, we plan to look for some weaker logical connectives to be used in place of adjoints, but yielding a decidable logic. In this direction, the work about epistemic logics [9] seems very promising. Also the comparison with customary first order temporal logics can be fruitful. Actually, a quite natural attempt is to consider a first order

temporal logic extended with indexed modalities for describing nesting properties (e.g., $P \models \langle\!|\mathcal{M}|\!\rangle\mathcal{A} \overset{\triangle}{\iff} \exists\sigma, Q, R.P \equiv \sigma(\!|Q|\!) \circ R \wedge \sigma \models \mathcal{M} \wedge Q \models \mathcal{A}$); however, it seems that the fragments of Brane Logics we have considered, are strictly more expressive than a first order temporal logic with nesting modalities, due to the presence of spatial connectives.

Another possibility is to look for subsets of the calculus for which the satisfaction problem is dedidable. In particular, we can consider restricted forms of replication or movement, like the Mate-Bud-Drip calculus; it is already known that these subcalculi have nice decidability properties [1], although they are not as expressive as the full one.

We can also consider different notions of quantifier, in place of the classical "forall" quantifier. Recently, several logics for process algebras use some "freshness" quantifier, like Miller-Tiu's "nabla" [11] or Gabbay-Pitts' "new" [12], to represent freshness of local names. For instance, $\nabla x.\mathcal{A}(x) \circ \mathcal{B}(x)$ could represent processes which can be split in two parts, sharing a private name (which can be seen as a bound site connecting the two parts). In these cases, we want to express that the bound is private, i.e., the name used in the connection does not clash with those in the context; this is impossible to achieve with the forall/exists quantifier, but would be possible with nabla and new operators.

One may consider the logical equivalence induced by Brane Logic over membranes and systems. As for Ambient Logic [16], we could give a coinductive characterization of logical equivalence; however, our feeling is that logical equivalence is very close, if not equal, to structural congruence.

References

[1] Busi, N., *Deciding behavioural properties in brane calculi.*, in: Priami [14], pp. 17–31.

[2] Caires, L., *Behavioral and spatial observations in a logic for the pi-calculus.*, in: I. Walukiewicz, editor, *FoSSaCS*, Lecture Notes in Computer Science **2987** (2004), pp. 72–89.

[3] Cardelli, L., *Brane calculi.*, in: Danos and Schachter [7], pp. 257–278.

[4] Charatonik, W., S. Dal-Zilio, A. D. Gordon, S. Mukhopadhyay and J.-M. Talbot, *Model checking mobile ambients*, Theor. Comput. Sci. **308** (2003), pp. 277–331.

[5] Dal-Zilio, S., *Spatial congruence for ambients is decidable*, in: *ASIAN '00: Proceedings of the 6th Asian Computing Science Conference on Advances in Computing Science* (2000), pp. 88–103.

[6] Danos, V. and S. Pradalier, *Projective brane calculus.*, in: Danos and Schachter [7], pp. 134–148.

[7] Danos, V. and V. Schachter, editors, "Computational Methods in Systems Biology, International Conference CMSB 2004," Lecture Notes in Computer Science **3082**, Springer, 2005.

[8] Hennessy, M. and R. Milner, *Algebraic laws for nondeterminism and concurrency*, J. ACM **32** (1985), pp. 137–161.

[9] Mardare, R. and C. Priami, *Decidable extensions of Hennessy-Milner logic*, in: E. Najm, J.-F. Pradat-Peyre and V. Donzeau-Gouge, editors, *Proc. FORTE*, Lecture Notes in Computer Science **4229** (2006), pp. 196–211.

[10] Miculan, M. and G. Bacci, *Modal logics for brane calculus*, in: Priami [14], pp. 1–16.

[11] Miller, D. and A. Tiu, *A proof theory for generic judgments*, ACM Trans. Comput. Log. **6** (2005), pp. 749–783.

[12] Pitts, A. M., *Nominal logic, a first order theory of names and binding*, Information and Computation **186** (2003), pp. 165–193.

[13] Post, E. L., *Recursively enumerable sets of positive integers and their decision problems*, Bulletin of the American Mathematical Society **50** (1944), pp. 284–316.

[14] Priami, C., editor, "Computational Methods in Systems Biology, International Conference, CMSB 2006, Trento, Italy, October 18-19, 2006, Proceedings," Lecture Notes in Computer Science **4210**, Springer, 2006.

[15] Priami, C. and P. Quaglia, *Beta binders for biological interactions*, in: V. Danos and V. Schächter, editors, *Proc. CMSB*, Lecture Notes in Computer Science **3082** (2004), pp. 20–33.

[16] Sangiorgi, D., *Extensionality and intensionality of the ambient logics*, in: *Proc. POPL*, 2001, pp. 4–13.

[17] Trakhtenbrot, B. A., *The impossibility of an algorithm for the decision problem for finite models*, Doklady Akademii Nauk SSR **70** (1950), pp. 569–572.