A Multiset-Based Model of Synchronizing Agents: Computability and Robustness

Matteo Cavaliere, Radu Mardare & Sean Sedwards
{cavaliere,mardare,sedwards}@cosbi.eu

Microsoft Research–University of Trento
Centre for Computational and Systems Biology
Trento, Italy

August 22, 2007

Abstract

We introduce a modelling framework and computational paradigm called Colonies of Synchronizing Agents (CSAs) inspired by the intracellular and intercellular mechanisms in biological tissues.

The model is based on a multiset of agents in a common environment. Each agent has a local state stored in the form of a multiset of atomic objects, which is updated by global multiset rewriting rules either independently or synchronously with another agent.

We first define the model then study its computational power, considering trade-offs between internal rewriting (intracellular mechanism) and synchronization between agents (intercellular mechanisms). We also investigate dynamic properties of CSAs, including behavioural robustness (ability to generate a core behaviour despite agent loss or rule failure) and safety of synchronization (ability of an agent to synchronize with some other agent whenever needed).

1 Motivations

Inspired by *intracellular* and *intercellular* mechanisms in biological tissues, we present and investigate an abstract distributed model of computation which we call Colonies of Synchronizing Agents (in short CSAs). Our intention is to create a framework to model, analyse and simulate biological tissues in the context of formal language and multiset rewriting.

The model is based on a population of agents (e.g., corresponding to *cells* or *molecules*) in a common environment, able to modify their contents and to synchronize with other agents in the same environment. Each agent has a contents represented by a multiset of atomic objects (e.g., corresponding to *chemical compounds* or the characteristics of individual molecules) with some of the objects classified as terminals (e.g., corresponding to chemicals or properties visible to an external observer). The agents' contents may be modified by means of multiset rewriting rules (called *evolution rules*), which may mimic chemical or

other types of *intracellular mechanisms*. Moreover, the agents can influence each other by synchronously changing their contents using pairwise *synchronization rules*. This models, in a deliberately abstract way, the various *intercellular mechanisms* present in biological tissues (e.g., signalling mechanisms that cells and biological systems use). *All rules are global*, so all agents obey the same rules: the only feature that may distinguish the agents is their contents.

Hence, a CSA is essentially a multiset of mutisets, acted upon by multiset rewriting rules.

In this paper we consider CSAs as generative computing devices and consider various trade-offs between the power of the evolution rules and the power of the synchronizing rules. We consider CSAs working in a maximally parallel way (all agents are updated synchronously), modelling the idea that if something can happen then it *must* happen. However, from both a biological and a mathematical point of view, it is also useful to investigate systems where the update of the agents is not obligatory (i.e., not synchronous). We prove that the computational power of maximal parallel and asynchronous CSAs can range from that of finite sets of vectors to that of Turing machines, by varying the power of the evolution and synchronization rules. Moreover, an intermediate class of CSAs, equivalent to partially blind counter machines (hence, not universal), is investigated.

Having investigated the computational power of CSAs, we study the *robustness* of colonies by considering their ability to generate core behaviours despite the failure (i.e., removal) of agents or of rules. We show that for an arbitrary CSA, robustness cannot be decided but that it is possible to individuate classes of (non-trivial) CSAs where this property can be efficiently decided.

In the final part of the paper we are interested in dynamic properties of CSAs concerning the applications of the rules.

For this reason, we provide a decidable temporal logic to specify and investigate *dynamic properties* of CSAs. For instance, we show that the proposed logic can be used to specify and then check whether or not in a CSA an agent has the ability to apply a synchronization whenever it needs: CSAs for which such a property is true we describe *safe on synchronization* of rules. This models, in an abstract way, the ability of a cell to use an intercellular mechanism whenever it needs.

CSAs are computational devices that have features inspired by many different models. In particular, they have similarities (and significant differences) with other models inspired by cell-tissues and investigated in the area of membrane computing (i.e., P systems). Specifically, CSAs can be considered a generalization of P colonies [15], which is also based on interacting agents but has agents with limited contents (two objects) which can only change their contents using very restricted rewriting rules (following an earlier definition of an agent in formal language theory ([14]). In our case, in order to be more general, the rewriting rules employed by an agent and the contents of an agent can be arbitrarily complex. Moreover, in P colonies objects can be introduced into the agent from an external environment (with unbounded copies of a given object) and the objects present in an agent may only be transferred to another agent by means of the common environment; no direct communication between agents is allowed (as is the case in CSAs).

CSAs also have similarities with population P systems [5], a class of tissue P systems [17], and in particular with EC tissue P systems [6], where evolution

(rewriting) is combined with communication. Cells (i.e., agents) can change their contents by means of (non-cooperative) rewriting rules and hence different types of agents can have different sets of rules. It is also possible to move objects between the agents using 'bonds' and agents may also communicate with an environment that has unbounded resources. Computation is generally implemented in two different phases: local rewriting plus bond making rules, applied in an alternate manner. The main differences with these computing devices and the model we propose here are that we do not have explicit bonds (edges) between agents (in a sense our agents are linked by a complete graph), rewriting in our case is arbitrarily complex (i.e., it can be cooperative) for both evolution and synchronization rules, and the agents we propose do not have explicit types: rules are global and only the agents' contents differentiate them. This latter characteristic makes CSAa similar to the model of self-assembly of graphs presented in [4], however in that case (*i*) a graph is constructed from an initial seed using multiset-based aggregation rules to enlarge the structure, (*ii*) there is no internal rewriting of the agent contents and (*iii*) there is no synchronization between the agents.

CSAs are also distinct from cellular automata [9], where cells exist on a regular grid, where each cell has a finite number of possible states and where cells interact with a defined neighbourhood. In our case, as a result of the multiset-based contents and because of the general rewriting rules, the possible different internal states of a cell may be infinite. Although our initial definition does not include an explicit description of space, the extensions we propose include agents located at arbitrary positions and with the potential to interact with any other agent in the system.

2 Preliminaries

2.1 Formal Language and Multisets Theory

We briefly recall the basic theoretical notions of formal languages and multiset rewriting used in this paper.

An introduction to the area of formal languages is [13]. A coverage of all the aspects of the area is the handbook [20]. Alternatively, the chapter introducing formal language in [19] contains all the notions needed in the paper.

Given the set A we denote by $|A|$ its cardinality and by $\emptyset$, the empty set. We denote by $\mathbb{N}$ the set of natural numbers. We denote by 2^A the power set of A.

An *alphabet* V is a finite set of symbols. By V^* we denote the set of all strings over V. By V^+ we denote the set of all strings over V excluding the empty string. The empty string is denoted by λ. The *length* of a string v is denoted by $|v|$. The concatenation of two strings $u, v \in V^*$ is written uv.

The number of occurrences of the symbol a in the string w is denoted by $|w|_a$.

For a language $L \subseteq V^*$, the set $length(L) = \{|x| \mid |x \in L\}\}$ is called the *length set* of L, denoted by NL.

If FL is an arbitrary family of languages then we denote by NFL the family of length sets of languages in FL (family of sets of natural numbers).

The *Parikh vector* associated with a string $x \in V^*$ with respect to the

alphabet $V = \{a_1, a_2, \ldots, a_n\}$ is $Ps_V(x) = (|x|_{a_1}, |x|_{a_2}, \ldots, |x|_{a_n})$. For $L \subseteq V^*$ we define $Ps_V(L) = \{Ps_V(x) \mid x \in L\}$. This is called the *Parikh image* of the language L.

If FL is an arbitrary family of languages then we denote by $PsFL$ the family of Parikh images of languages in FL (family of sets of vectors of natural numbers).

We denote by FIN, REG, CF, CS, and RE the families of finite, regular, context-free, context-sensitive, and recursively enumerable languages, respectively.

By $L(G)$ we denote the language generated/produced by the grammar G.

Then, for instance, the family of Parikh images of languages in RE is denoted by $PsRE$ (this is the family of all recursively enumerable sets of vectors of natural numbers). The family of all recursively enumerable sets of natural numbers is denoted by NRE.

We denote by FL_A the family of languages over the alphabet A, e.g., REG_A, the family of all regular languages over the alphabet A.

A *multiset* is a set where each element may have a multiplicity. Formally, a multiset over a set V is a map $M : V \to \mathbb{N}$, where $M(a)$ denotes the multiplicity (i.e., number of occurrences) of the symbol $a \in V$ in the multiset M. Note that the set V can be infinite.

For multisets M and M' over V, we say that M is *included in* M' ($M \subseteq M'$) if $M(a) \leq M'(a)$ for all $a \in V$. Every multiset includes the *empty multiset*, defined as M where $M(a) = 0$ for all $a \in V$.

The *sum* of multisets M and M' over V is written as the multiset $(M + M')$, defined by $(M + M')(a) = M(a) + M'(a)$ for all $a \in V$. The *difference* between M and M' is written as $(M - M')$ and defined by $(M - M')(a) = max\{0, M(a) - M'(a)\}$ for all $a \in V$. We also say that $(M + M')$ is obtained by *adding* M to M' (or viceversa) while $(M - M')$ is obtained by *removing* M' from M.

The *support* of a multiset M is defined as the set $supp(M) = \{a \in V \mid M(a) > 0\}$. A multiset with finite support is usually presented as a set of pairs $(x, M(x))$, for $x \in supp(M)$.

The *cardinality* of a multiset M is denoted by $card(M)$ and it indicates the number of objects in the multiset. It is defined in the following way. $card(M)$ is infinite if M has infinite support. If M has finite support then $card(M) = \sum_{a_i \in supp(M)} M(a_i)$ (i.e., all the occurrences of the elements in the support are counted).

We denote by $\mathbb{M}(V)$ the set of all possible multisets over V and by $\mathbb{M}_k(V)$ and $\mathbb{M}_{\leq k}(V)$, $k \in \mathbb{N}$, the set of all multisets over V having cardinality k and at most k, respectively. That is $\mathbb{M}_k(V) = \{M \mid M \in \mathbb{M}(V), card(M) = k\}$ and $\mathbb{M}_{\leq k}(V) = \{M \mid M \in \mathbb{M}(V), card(M) \leq k\}$.

Note that, since V could be infinite, $\mathbb{M}_k(V)$ and $\mathbb{M}_{\leq k}(V)$, for $k \in \mathbb{N}$ could also be infinite.

For the case that the alphabet V is finite we can use a compact string notation to denote multisets: if $M = \{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_n, M(a_n))\}$ then the string $w = a_1^{M(a_1)} a_2^{M(a_2)} \cdots a_n^{M(a_n)}$ (and all its permutations) precisely identifies the symbols in M and their multiplicities. Hence, given a string $w \in V^*$, we can say that it identifies the multiset $\{(a, |w|_a) \mid a \in V\}$. For instance, the string bab represents the multiset $\{(a, 1), (b, 2)\}$, which may also be written as $\{a, b, b\}$ and has cardinality 3. The empty multiset is represented

by the empty string, λ.

In this paper we also make use of the notion of a *matrix grammar*.

A *matrix grammar with appearance checking* (a.c.) is a construct $G = (N, T, S, M, F)$, where N and T are disjoint alphabets of non-terminal and terminal symbols, $S \in N$ is the axiom, M is a finite set of matrices which are sequences of context-free rules of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$ (with $A_i \in N, x_i \in (N \cup T)^*$ in all cases), and F is a set of instances of rules in M.

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots A_n \to x_n)$ in M and strings $w_i \in (N \cup T)^*$, $1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$ and, for all $1 \leq i \leq n$, either

(i) $w_i = w'_i A_i w''_i$, $w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$

or

(ii) $w_i = w_{i+1}$, A_i does not appear in w_i and the rule $A_i \to x_i$ appears in F.

The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied (one says that these rules are applied in *appearance checking (a.c.)* mode). The reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$. Then the language generated by G is $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$.

In other words, the language $L(G)$ is composed of all the strings of terminal symbols that can be obtained starting from S by applying iteratively the matrices in M.

The family of languages generated by matrix grammars with appearance checking is denoted by MAT_{ac}.

G is called a *matrix grammar without appearance checking* if and only if $F = \emptyset$. In this case the generated family of languages is denoted by MAT. The following results are known (see, e.g., [8], [12]).

Theorem 1

- $CF \subset MAT \subset MAT_{ac} = RE$.
- *Each language $L \in MAT$, $L \subseteq a^*$ is regular (the proof of this statement is constructive).*

The following results are known (e.g., [8]) or they can be derived from the above assertions and from the definitions given earlier.

Theorem 2

- $PsMAT_{ac} = PsRE$.
- $NMAT_{ac} = NRE$.
- $PsREG \subset PsMAT \subset PsRE$.
- $PsCF = PsREG$.
- $NMAT = NREG = NCF$.

A matrix grammar is called *pure* if there is no distinction between terminals and non-terminals. The languages generated by a pure matrix grammar is composed of all sentential forms. The family of languages generated by pure matrix grammars without appearance checking is denoted by $pMAT$. It is easy to see that

Theorem 3 *$pMAT \subset MAT$.*

Matrix grammars without appearance checking are equivalent to *partially blind counter machines* (introduced in [11]). That is, the family of Parikh images of languages generated by matrix grammars without a.c. is equal to the family of sets of vectors of natural numbers generated by partially blind register machines (a constructive proof of their equivalence can be found, for instance, in [10]).

From this last assertion and using results in [11] we obtain the following corollaries of interest for this paper.

Corollary 1
(Emptiness)
Given an arbitrary alphabet T, an arbitrary matrix grammar without a.c., G, with terminal alphabet T, it is decidable whether or not $Ps_T(L(G)) = \emptyset$.

(Union, intersection, complementation)
The sets of Parikh images of languages generated by matrix grammars without a.c. are closed under union and intersection but not under complementation.

(Containment, Equivalence)
Given an arbitrary alphabet, T, two arbitrary matrix grammars without a.c., G and G', with terminal alphabet T, it is undecidable whether or not $Ps_T(L(G)) \subseteq Ps_T(L(G'))$ or whether or not $Ps_T(L(G)) = Ps_T(L(G'))$.

From Theorem 1 and using the fact that containment of regular languages is decidable ([13]) we obtain the following result.

Theorem 4 *(Containment, Equivalence)*
Given an arbitrary terminal alphabet T of cardinality one, two arbitrary matrix grammars without a.c. G and G' over T, it is decidable whether or not $NL(G') \subseteq NL(G)$ and whether or not $NL(G) = NL(G')$.

2.2 Membrane Systems

In this section we recall some definitions and results from membrane systems (often called P systems) which are used in this paper. An introductory guide to P systems is [19]; a preprint of which, together with other information about P systems (including a bibliography) can be found on the website [23].

Definition 2.1 *A P system with symbol-objects and of degree $m \geq 1$ is defined as a construct*

$$\Pi = (O, T, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0)$$

where

- *O is an alphabet and its elements are called objects; $T \subseteq O$ is a terminal alphabet;*
- *μ is a membrane structure consisting of m membranes arranged in an hierarchical tree structure; the membranes (and hence the regions that they delimit) are injectively labeled with $1, 2, \ldots, m$;*
- *w_i, $1 \leq i \leq m$, are strings that represent multisets over O associated to regions $1, 2, \ldots, m$ of μ;*

- R_i, $1 \leq i \leq m$, *are finite sets of evolution rules over* O; R_i *is associated to region* i *of* μ; *an evolution rule is of the form* $u \rightarrow v$, *where* u *is a string over* O *and* v *is a string over* $\{a_{here}, a_{out} \mid a \in O\} \cup \{a_{in_j} \mid a \in O, 1 \leq j \leq m\}$.
- $i_0 \in \{0, 1, 2, \ldots, m\}$; *if* $i_0 \in \{1, \ldots, m\}$ *then it is the label the membrane that encloses the output region; if* $i_0 = 0$ *then the output region is the environment.*

For any evolution rule $u \rightarrow v$ the length of u is called the *radius* of the rule and the symbols *here*, *out*, in_j, $1 \leq j \leq m$, are called target indications.

According to the size of the radius of the evolution rules we distinguish between *cooperative* rules (if the radius is greater than one) and *non-cooperative* rules (otherwise).

The *initial configuration* of the system Π comprises the structure μ and the multisets represented by the strings $w_i, 1 \leq i \leq m$. In general, we call a *configuration* of the system the m-tuple of multisets of objects present at any time in the m regions of the system.

An occurrence γ_r of the rule $r : u \rightarrow v \in R_i$, $i \in \{1, \cdots, m\}$ can be applied in region i by assigning to γ_r a multiset of objects u taken from the multiset of objects present in region i.

The application of an instance of the evolution rule $u \rightarrow v$ in a region i means to remove the multiset of objects u from the multiset of objects present in region i and to add the multiset v to the multisets of objects present in the adjacent regions, according to the target indications associated to each occurrence of the objects in v. In particular, if v contains an occurrence with target indication *here*, then the occurrence will be placed in the region i, where the rule has been applied. If v contains an occurrence with target indication *out*, then the occurrence will be moved to the region immediately outside the region i (this can be the environment if the region where the rule has been applied is the outermost or *skin* membrane). If v contains an occurrence with target indication in_j then the occurrence is moved from the region i and placed in region j (this can be done only if region j is directly contained by region i; otherwise the evolution rule $u \rightarrow v$ cannot be applied).

A *transition* between configurations is executed using the evolution rules in a *non-deterministic maximally parallel* manner at each step, in each region (we suppose that a global clock exists, marking the instant of each step for the whole system). This means that occurrences of the objects are assigned to occurrences of the rules in such a way that, after the assignment is made, there are insufficient occurrences of the objects for further occurrences of any of the rules to be applied. This maximal assignment is performed simultaneously in every region of the system at each step. If an occurrence of an object can be assigned to more than one occurrence of the rules then the assignment is chosen in a non-deterministic way.

A sequence of transitions between configurations of a system is called a *evolution*; an evolution is a *successful computation* (we simply say simply *computation*) if and only if it starts from the initial configuration and *halts*, i.e., it reaches a halting configuration where no occurrence of any rule can be applied in any region.

The *output* of a computation is defined as the number of occurrences of objects from T present in the output region in the halting configuration of

Π; the set of numbers computed (or generated) in this way by the system Π, considering any computation, is denoted by $N(\Pi)$.

It is possible to consider as the result of a computation the vector of numbers representing the multiplicities of the occurrences of objects from T present in the output region in the halting configuration. In this case $Ps_T(\Pi)$ denotes the set of vectors of numbers generated by Π, considering all the computations.

We denote by $NOP_m(\alpha, tar)$ and $PsOP_m(\alpha, tar)$ the family of sets of the form $N(\Pi)$ and $Ps(\Pi)$, respectively, generated by symbol-objects P systems of degree at most $m \geq 1$ (if the degree is not bounded the subscript m becomes $*$), using evolution rules of the type α.

We can have $\alpha = coo$, indicating that the systems considered use cooperative evolution rules, and $\alpha = ncoo$ indicating that the systems use only non-cooperative rules.

Moreover, the symbol tar indicates that the communication between the membranes (and hence the regions) is made using the target indication in_j in the way previously specified. If the degree of the system is 1 (only one membrane is present) then the only possible target indications that can be used are *here* and *out* and in such case the notation is $NOP_1(\alpha)$ and $PsOP_1(\alpha)$, respectively.

The following results are known (see, e.g., [18]).

Theorem 5

- $PsOP_*(ncoo, tar) = PsOP_1(ncoo) = PsCF$.
- $PsOP_*(coo, tar) = PsOP_m(coo, tar) = PsRE$ *for all* $m \geq 1$.

We recall the definition and main results of evolution-communication P systems originally introduced in ([7]), which joins two basic models of membrane systems, that with evolution rules and symbol-objects and that with symport/antiport rules (see, e.g, [18]).

Definition 2.2 *An evolution-communication P system (in short, an EC P system) of degree* $m \geq 1$*, is defined as*

$$\Pi = (O, \mu, w_1, w_2, \ldots, w_m, R_1, \ldots, R_m, R'_1, \ldots, R'_m, i_0)$$

where:

- O, μ, i_0 *and* $w_i, 1 \leq i \leq m$ *as in Definition 2.1;*
- R_i, $1 \leq i \leq m$, *are finite sets of simple evolution rules over* O*;* R_i *is associated with the region* i *of* μ*; a simple evolution rule is of the form* $u \to v$*, where* $u \in O^+$ *and* $v \in O^*$*; hence, a simple evolution rule is an evolution rule as in P systems with symbol-objects, but with no target indications (in other words it uses an implicit* 'here' *for target indications);*
- R'_i, $1 \leq i \leq m$, *are finite sets of symport rules over* O *of the form* (x, in), (y, out) *and of antiport rules* $(x, in; y, out)$ *with* $x, y \in O^+$*;* R'_i *is associated with membrane* i *of* μ*. For a symport rule* (x, in) *or* (x, out), $|x|$ *is called the* weight *of the rule. For an antiport rule* $(x, in; y, out)$ *the weight is the* $max\{|x|, |y|\}$.

In an EC P system a configuration is represented by the membrane structure μ and by the m-tuple of multisets of objects present in the m regions of the system.

In particular, the *initial configuration* comprises the system of membranes μ and the multisets represented by the strings $w_i, 1 \leq i \leq m$.

Occurrences of evolution rules are applied as in P systems with symbol-objects.

An occurrence γ of a symport rule $(x, in) \in R'_i$ ($(x, out) \in R'_i$) can be applied to membrane i by assigning to γ the occurrences of the objects in x taken from the region surrounding region i (taken from region i, respectively).

The application of an instance of the *symport rule* (x, in) to membrane i consists of moving the occurrences of the objects in x from the region (or from the environment) surrounding the region i to region i . If an instance of the symport rule (x, out) is applied to membrane i, the occurrences of the objects in x are moved from region i to the region (or to the environment) that surrounds region i.

An occurrence γ of the antiport rule $(x, in; y, out) \in R'_i$ can be applied to membrane i by assigning to γ the occurrences of the objects in x taken from region i and the occurrences of the objects in y taken from the region surrounding region i.

If an instance of the *antiport rule* $(x, in; y, out)$ is applied to membrane i, the occurrences of the objects in x pass into the region i from the region surrounding it, while, at the same time, the occurrences of the objects in y move from the surrounding region to region i.

A *transition* between configurations is governed by the mixed application of occurrences of the evolution rules and of the symport/antiport rules. Instances of the rules from R_i are applied to occurrences of objects in region i while the application of the instances of rules from R'_i govern the communication of the occurrences of objects through membrane i. There is no distinction drawn between evolution rules and communication rules (*mixed approach*): they are applied in the *non-deterministic maximally parallel manner*, described above.

The system starts from the initial configuration and passes from one configuration to another by applying the above described transitions: this sequence of transitions is called an *evolution* of the system. The system halts when it reaches a halting configuration, i.e., a configuration where no occurrence of any rule (evolution rules or symport/antiport rules) can be applied in any region of Π.

In this case the evolution is called a successful *computation* of Π (or simply a computation of Π) and the number of occurrences of objects contained in the output region i_0 in the halting configuration is the result of the computation. The set of numbers computed (or generated) in this way by the system Π, considering any possible computation of Π, is denoted by $N(\Pi)$.

It is also possible to consider as a result of the computation the vector of numbers representing the multiplicities of the occurrences of the objects contained in the output region in the halting configuration. In this case $Ps(\Pi)$ denotes the set of vectors generated by Π, considering all computations.

The notation $NECP_m(i, j, \alpha)$, $\alpha \in \{ncoo, coo\}$, and $PsECP_m(i, j, \alpha)$, $\alpha \in \{ncoo, coo\}$, is used to denote the family of sets of numbers and the family of sets of vectors of numbers, respectively, generated by EC P systems with at most m membranes (as usual, $m = *$ if such a number is unbounded), using

symport rules of weight at most i, antiport rules of weight at most j and simple evolution rules that can be cooperative (coo) or non-cooperative ($ncoo$).

The following results are known (see, e.g, [7, 2]).

Theorem 6
- $NECP_1(1, 0, ncoo) = NCF$.
- $PsECP_1(1, 0, ncoo) = PsCF$.
- $NECP_2(1, 1, ncoo) = NRE$.
- $PsECP_2(1, 1, ncoo) = PsRE$.

3 Colonies of Synchronizing Agents

In this section we formalize the notions of colonies and agents discussed in the Introduction.

A *Colony of Synchronizing Agents* (a CSA) of degree $m \geq 1$ is a construct $\Pi = (A, T, C, R)$ with the components having the following meaning:

- A is a finite alphabet of symbols (its elements are called *objects*). $T \subseteq A$ is the alphabet of *terminal objects*.
- An *agent* over A is a multiset over the alphabet A (an agent can be represented by a string $w \in A^*$, since A is finite). C is the *initial configuration of* Π and it is a multiset over the set of all possible agents over A (with $card(C) = m$) .

  Using the notation introduced in the Preliminaries, $C \in \mathbb{M}_m(H)$ with $H = \mathbb{M}(A)$.
- R is a finite set of *rules* over A.

  We have *evolution rules* of type $u \to v$, with $u \in A^+$ and $v \in A^*$.

  An instance γ of an evolution rule $r : u \to v$ can be applied to an occurrence o_w of agent w by taking a multiset of objects u from o_w (hence, it is required that $u \subseteq w$) and *assigning* it to γ (i.e., assigning the occurrences of the objects in the taken multiset to γ).

  The application of an instance of rule r to the occurrence o_w of the agent w consists of removing from o_w the multiset u and then adding v to the resulting multiset.

  We say that an evolution rule $u \to v$ is *cooperative* (in short, coo_e) if $|u| > 1$, *non-cooperative* ($ncoo_e$) if $|u| = 1$ and *unary* (un_e) if $|v| \leq |u| = 1$.

  We have *synchronization rules* of the type $\langle u, v\rangle \to \langle u', v'\rangle$ with $uv \in A^+$ and $u', v' \in A^*$.

  An instance γ of a synchronization rule $r : \langle u, v\rangle \to \langle u', v'\rangle$ can be applied to the pair of occurrences o_w and $o_{w'}$ of, respectively, agents w and w' by: (i) taking from o_w a multiset of objects u and *assigning* it to γ; (ii) taking from $o_{w'}$ a multiset of objects v and *assigning* it to γ (hence, it is required that $u \subseteq w$ and $v \subseteq w'$).

  The application of an instance of rule r to the occurrences o_w and $o_{w'}$ consists of: removing the multiset u from o_w and then adding u' to the

resulting multiset; removing the multiset v from $o_{w'}$ and then adding v' to the resulting multiset.

Synchronization rules can be considered as matrices of two rules used simultaneously.

We say that a synchronization rule $\langle u, v\rangle \rightarrow \langle u', v'\rangle$ is cooperative (coo_s) if $|u| > 1$ or $|v| > 1$, non-cooperative ($ncoo_s$) if $|u| = 1$ and $|v| = 1$, *unary* (un_s) if $|u'| \leq |u| = 1$ and $|v'| \leq |v| = 1$.

A *configuration* of a CSA, Π, consists of the occurrences of the agents present in the system at a given time (we assume the existence of a *global clock* which marks the passage of units of time).

We denote by $\mathbb{C}(\Pi)$ the set of *all possible configurations* of Π. Therefore, using the notation introduced in the Preliminaries, $\mathbb{C}(\Pi)$ is exactly $\mathbb{M}_m(H)$ with $H = \mathbb{M}(A)$.

A *transition* from an arbitrary configuration c of Π to the next lasts exactly one time unit and can be obtained in two different modes.

Maximally-parallel mode (in short *mp*): A *maximally-parallel transition* of Π (in short, an *mp-transition*) is obtained by applying the rules in the set R to the agents present in the configuration c in a *maximally parallel and non-deterministic* way. This means that for each occurrence o_w of an agent w and each pair of occurrences $o_{w'}$ and $o_{w''}$ of agents w' and w'' present in the configuration c, the occurrences of the objects present in o_w ($o_{w'}, o_{w''}$) are assigned to instances of the evolution (synchronization, resp.) rules, the occurrences of the agents, the occurrences of the objects and the instances of the rules chosen in a non-deterministic way but respecting the following condition. After the assignment of the occurrences of the objects to the instances of the rules is done there is no instance of any rule that can be applied by assigning the (still) unassigned occurrences of the objects.

A single occurrence of an object can only be assigned to a single instance of a rule.

Asynchronous mode (in short *asyn*): A single *asynchronous transition* of Π (in short, an *asyn-transition*) is obtained by applying the rules in the set R to the agents present in the configuration c in an *asynchronous* way.

This means that, for each occurrence o_w of an agent w and each pair of occurrences $o_{w'}$ and $o_{w''}$ of the agents w', w'', present in the configuration c, the occurrences of the objects of o_w ($o_{w'}, o_{w''}$) are *either* assigned to instances of the evolution (synchronization, resp.) rules *or* left unassigned. The occurrences of the agents, the occurrences of the objects and the instances of the rules are chosen in a non-deterministic way. A single occurrence of an object can only be assigned to a single instance of a rule.

In other words, in a single asynchronous transition, any number of instances of rules (zero, one or more) can be applied to the occurrences of the agents present in the configuration c.

A sequence (possibly infinite) $\langle C_0, C_1, \cdots, C_i, C_{i+1}, \cdots\rangle$ of configurations of Π, where C_{i+1} is obtained from C_i, $i \geq 0$, by a γ-transition is called a γ-*evolution* of Π, with $\gamma \in \{asyn, mp\}$. A configuration c of Π present in a γ-evolution of Π is said to be *reachable* using a γ-evolution of Π (or simply reachable if there is no confusion). Often we also say that the evolution *reaches* the configuration c.

A γ-evolution of Π, with $\gamma \in \{asyn, mp\}$, is said to be *halting* if it halts, that is if it is finite and the last configuration of the sequence is a *halting configuration*,

i.e., a configuration containing only occurrences of agents for which no rule from R is applicable.

A γ-evolution of Π that is halting and that starts with the initial configuration of Π is called a successful γ-computation or, because there is no confusion, we simply say a γ-*computation* of Π, with $\gamma \in \{asyn, mp\}$.

The *result/output* of an mp- or asyn-computation is the set of vectors of natural numbers, one vector for each agent w present in the halting configuration (i.e., with a number of occurrences greater than zero) and with the vector describing the multiplicities of terminal objects present in w.

More formally, the result of an mp- or asyn-computation which stops in the configuration C_h is the set of vectors of natural numbers $\{Ps_T(w) \mid w \in supp(C_h)\}$.

Taking the union of all the results, for all possible mp- and asyn-computations, we get the set of vectors generated by Π and denoted by $Ps_T^{mp}(\Pi)$ and $Ps_T^{asyn}(\Pi)$, respectively.

We may also consider only the total number of objects comprising the agent (the agent's *magnitude*), without considering the composition. In this case the result of an mp- or asyn-computation is the set of natural numbers, one number for each agent w present in the halting configuration and with the number being the length of w. More formally, in this case, the result of an mp- or asyn-computation that stops in the configuration C_h is then the set of numbers $\{|w| \mid w \in supp(C_h)\}$. In other words, in this case, there is no distinction between the objects composing the agents, in particular the terminals from T are ignored.

Again, taking the union of all the results, for all possible mp- and asyn-computations, we get the *set of numbers generated* by Π and denoted by $N^{mp}(\Pi)$ and $N^{asyn}(\Pi)$, respectively.

Note that in both cases, considering sets of vectors (or sets of numbers) one single computation delivers a finite family of vectors as output (or a finite set of numbers, resp.) because there could be several agents in the halting configuration. However, $Ps_T^{\gamma}(\Pi)$ ($N^{\gamma}(\Pi)$), $\gamma \in \{mp, asyn\}$, is obtained as the union of results of computations of Π, so as a union of sets of vectors (of sets of numbers, resp.).

We consider now families of CSAs and then families of sets of vectors of numbers or of sets of numbers.

We denote by $CSA_m(\alpha, \beta)$, with $\alpha \in \{coo_e, ncoo_e, un_e\}$ and $\beta \in \{coo_s, ncoo_s, un_s\}$, the class of CSAs having evolution rules of type α, synchronization rules of type β and using at most m occurrences of agents in the initial configuration (m is changed to $*$ if it is unbounded). We omit α or β if the corresponding rules are not allowed. In particular, notice that if β is omitted then there is no cooperation between the agents.

Hence, we denote by $PsCSA_m^{\gamma}(\alpha, \beta)$ (and $NCSA_m^{\gamma}(\alpha, \beta)$) with $\gamma \in \{mp, asyn\}$ the family of sets of vectors (of sets of numbers, resp.) generated by CSAs from $CSA_m(\alpha, \beta)$ using γ-computations.

Example 1 *A CSA with degree* 3 *is defined by the following.*

$\Pi = (A, T, C, R)$ *with* $A = \{a, b, c\}$, $T = \{a\}$, $C = \{(abcba, 1), (abbcc, 1), (bab, 1)\}$ *and rules* $R = \{r_1 : abca \to ba,\ r_2 : \langle abc, cc\rangle \to \langle aa, cb\rangle\}$.

The application of an instance of the evolution rule r_1 to the configuration C is shown diagrammatically in Figure 1. The application of an instance of the

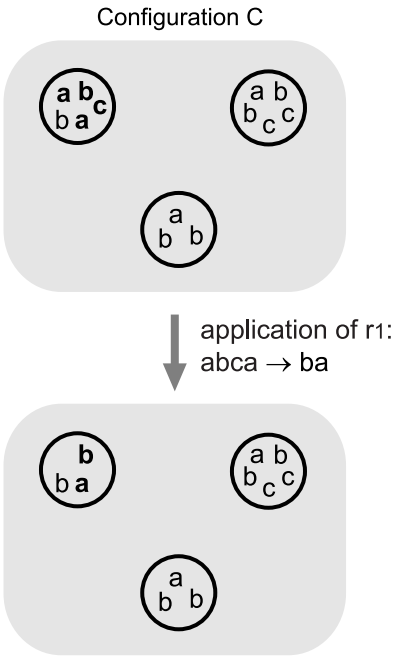

Figure 1: Application of an instance of the evolution rule r_1 to configuration C from Example 1.

synchronization rule r_2 to the configuration C is shown in Figure 2.

A more complex example is presented in Figure 3. Alternative maximally parallel and asynchronous (partial) evolutions of a CSA are shown, starting from the configuration $\{(ac, 2), (a, 1)\}$ with rules $\{ac \to aa, a \to b, \langle aa, aa\rangle \to \langle ab, ab\rangle, \langle ab, d\rangle \to \langle bb, d\rangle, b \to d\}$.

In what follows we consider the equality of families of sets of vectors modulo the null vector, i.e., if two families differ only by the null vector then we consider them to be equal. We indicate by A_Π the alphabet of the CSA Π, by T_Π the terminal alphabet of Π and by C_Π the initial configuration of Π.

Moreover, because there is no confusion, we avoid using "occurrences of ..." writing directly the entities (objects, rules or agents) involved.

For instance, when we say "an object c is used ..." we actually mean "one occurrence of object c is used ..." and when we say "the rule r is applied ..." we mean "one instance of rule r is used ...".

4 Computational Power of CSAs

From the definitions of CSAs and invoking the Turing-Church thesis we obtain:

Theorem 7

$$PsCSA_m^\gamma(\alpha) \subseteq PsCSA_m^\gamma(\alpha, \beta) \subseteq PsRE.$$

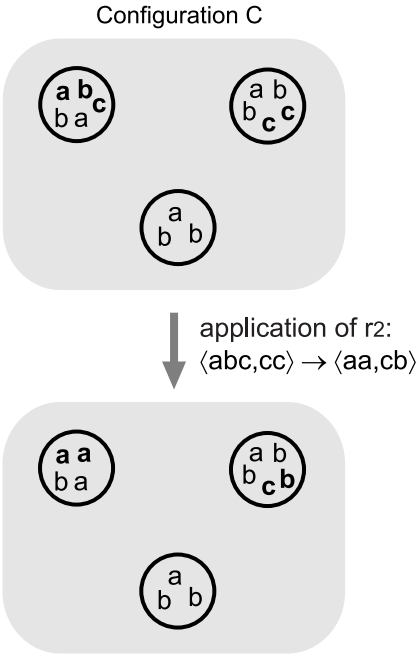

Figure 2: Application of an instance of the synchronization rule r_2 to configuration C from Example 1.

with $\alpha \in \{coo_e, ncoo_e, un_e\}$, $\beta \in \{coo_s, ncoo_s, un_s\}$, $\gamma \in \{mp, asyn\}$ *and* $m \geq 1$.

As soon as we have cooperative evolution rules and maximal-parallelism we get, as expected, maximal computational power.

Theorem 8

$$PsCSA_2^{mp}(coo_e) = PsCSA_2^{mp}(coo_s) = PsRE.$$

Proof. The proofs of the two equalities are straightforward hence we only give a short sketch. For each P system Π with symbol-objects, one membrane, cooperative evolution rules, working in maximally-parallel mode and producing as output the set of vectors of natural numbers S, there exists a CSA, Π', from $CSA_1(coo_e)$ such that $Ps_T^{mp}(\Pi') = S$ for an adequate terminal alphabet T. Just take Π' having in the initial configuration one single agent corresponding to the initial configuration of Π and with cooperative evolution rules as those defined in Π (with no loss of generality we suppose that Π uses only rules with the target indication 'here': any evolution rule with target indication 'out' that sends objects to the environment, where they are effectively lost, can be replaced by appropriate rules that delete the objects).

Also there exists a CSA Π'' from $CSA_2(coo_s)$ (i.e, using only synchronization rules) such that $Ps_T^{mp}(\Pi'') = S$, for an adequate terminal alphabet T. Again, Π'' has in the initial configuration one agent corresponding to the initial configuration of Π, while the other agent is necessary for applying synchronization rules, since a synchronization requires two different agents in order to be executed. The cooperative evolution rules of Π can easily be implemented by using cooperative synchronization rules in Π''.

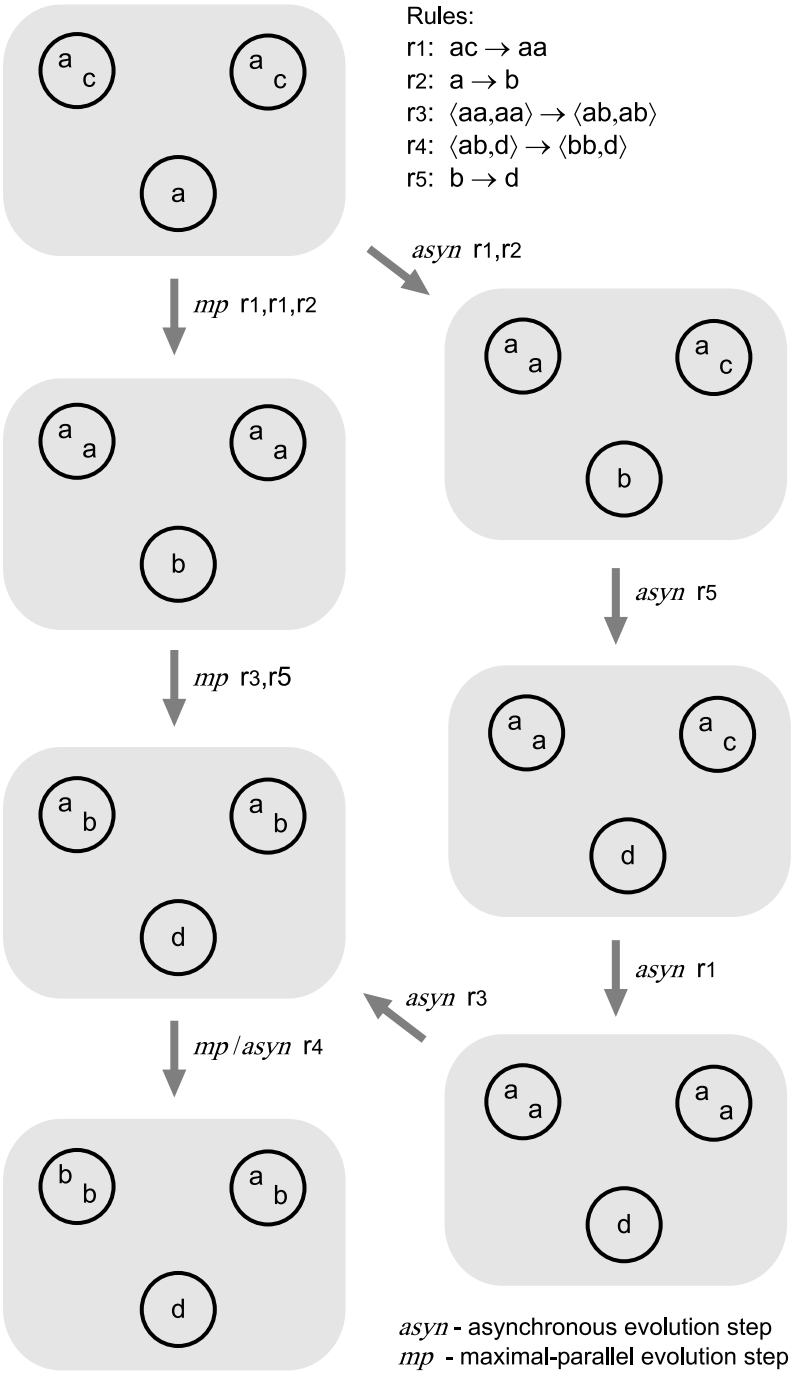

Figure 3: Alternative maximally-parallel and asynchronous evolutions of a CSA.

The equalities follow from the fact that P systems with symbol-objects, cooperative evolution rules, one membrane and working in the maximally-parallel mode are known to be computational complete (Theorem 5). □

Removing maximal parallelism decreases the computational power of the considered colonies.

Theorem 9

$$PsCSA_*^{asyn}(coo_e, coo_s) = PsCSA_*^{asyn}(coo_e) = PsMAT.$$

Proof. First we prove that for an arbitrary CSA, $\Pi = (A, T, C, R)$ from $CSA_*(coo_e, coo_s)$, there exists a matrix grammar without appearance checking, G, with terminal alphabet T, such that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$.

We suppose that $card(C) = m$. In particular, we suppose C consists of m agents $w_1, w_2, \cdots, w_m$ with $w_i \in A^*$ for $i \in \{1, \cdots, m\}$.

We construct the sets $A_i = \{a_i \mid a \in A\}$ for $i \in \{1, 2, \ldots, m\}$.

We construct the morphisms $h_i : A \to A_i$ for $i \in \{1, 2, \ldots, m\}$ defined as $h_i(a) = a_i$, $a \in A$. The inverse morphisms are denoted by h_i^{-1} for $i \in \{1, 2, \ldots, m\}$. We have $h_i^{-1}(a_i) = a$, $a \in A$.

We then construct the pure matrix grammar without appearance checking, $G = (N, N, S, M)$, in the following way.

We define $N = \{S\} \cup A_1 \cup A_2 \cup \cdots \cup A_m$ with $S \notin A_1 \cup A_2 \cdots \cup A_m$.

The matrices of M are constructed in the following manner (we group them according to their use).

Group I
We add to M the matrix $(S \to h_1(w_1)h_2(w_2)\cdots h_m(w_m))$.

Group II
For each evolution rule $u \to v$ in R, with $u = u_1u_2\cdots u_k$, $u_i \in A$ for $i \in \{1, 2, \ldots, k\}$ we add the following matrices: $\{(h_j(u_1) \to \lambda, h_j(u_2) \to \lambda, \ldots, h_j(u_{k-1}) \to \lambda, h_j(u_k) \to h_j(v)) \mid j \in \{1, 2, \ldots, m\}\}$.

Group III
For each synchronization rule $\langle u, v\rangle \to \langle u', v'\rangle$ with $u = u_1u_2\cdots u_k$, $u_r \in A$ for $r \in \{1, 2, \ldots, k\}$ and $v = v_1v_2\cdots v_p$, $v_r \in A$ for $r \in \{1, 2, \ldots, p\}$ we add the matrices: $\{(h_i(u_1) \to \lambda, h_i(u_2) \to \lambda, \ldots, h_i(u_{(k-1)}) \to \lambda, h_i(u_k) \to h_i(u'), h_j(v_1) \to \lambda, h_j(v_2) \to \lambda, \ldots, h_j(v_{p-1}) \to \lambda, h_j(v_p) \to h_j(v')) \mid i, j \in \{1, 2, \ldots, m\}, i \neq j\}$.

The basic idea of the simulation is that the matrix in group I is used to start a derivation of G by creating the string $h_1(w_1)h_2(w_2)\cdots h_m(w_m)$ corresponding to the initial configuration of Π (distinguishing the objects of the different agents by using different indexes). The matrices of group *II* are used to simulate the evolution rules present in the set R, while the matrices of group *III* are used to simulate the synchronization rules present in R.

The language $L(G)$ is the set of all the (strings representing) the configurations of Π reachable by asynchronous evolutions of Π starting with the initial configuration C.

Precisely, if there is an asynchronous evolution e of Π, starting from the initial configuration C and reaching the configuration $\{w_1', w_2', \cdots, w_m'\}$, then in $L(G)$ there is the string $h_1(w_1')h_2(w_2')\cdots h_m(w_m')$.

In particular, a transition of the evolution e obtained by applying an evolution (synchronization) rule of R to one (to a pair, resp.) of agents is simulated in G by applying the corresponding matrix from group *II* (or from group *III*, resp.). However, since Π works in an asynchronous way, we must take care of the transitions of e that are obtained by using more than one rule. Precisely, a transition of the evolution e that is obtained by applying *several* rules from R to the agents is simulated in G by applying, *sequentially*, the corresponding matrices from groups *II* and *III*.

The reverse is also true : if there is a string w in $L(G)$ then it must be (by the way G functions) of type $h_1(w'_1)h_2(w'_2)\cdots h_m(w'_m)$ with $w'_1, w'_2, \cdots, w'_m \in A^*$. And by the way G has been constructed, if there is a derivation d in G that produces the string $h_1(w'_1)h_2(w'_2)\cdots h(w'_m)$, then there is an asynchronous evolution of Π starting from the initial configuration C and reaching the configuration $\{w'_1, w'_2, \cdots, w'_m\}$. In fact, Π works in the asynchronous mode and, in particular, can have evolutions comprising sequential transitions. That is, only one rule is applied at each application of a rule that simulates the application of a matrix in the derivation d.

From the language $L(G)$ we select by an appropriate regular intersection the language L' of all the strings corresponding to halting configurations reached by asynchronous computations of Π. This can clearly be done by intersecting the language $L(G)$ with a regular set R_h of strings over N representing the halting configurations of Π (i.e., the set R_h represents the strings over N where no matrix can be applied and it is clearly a regular set).

We obtain $L' = L(G) \cap R_h$. The language L' can still be generated by a matrix grammar without appearance checking since matrix grammars without a.c. are closed under regular intersection ([8]).

We construct then the following morphisms $d_i : N \longrightarrow N \cup \{\lambda\}$ for each $i \in \{1, \cdots, m\}$, defined in the following manner.

$$d_i(a_i) = a_i, a \in T.$$
$$d_i(a_i) = \lambda, a \in (A - T).$$
$$d_i(a_j) = \lambda, a \in A, j \neq i.$$

For each $i \in \{1, \cdots, m\}$, the language $d_i(L')$ selects from each string in L' the substring corresponding to the agent with objects indexed by i.

Moreover, from each agent the objects not in T are deleted.

Now we construct the language $L'' = \bigcup_{1 \leq i \leq m}(h_i^{-1}(d_i(L')))$.

L'' is the language that collects all the agents present in the halting configurations, considering all the computations of Π.

Note that the language L'' is a language over T and can also be obtained using a matrix grammar without appearance checking (with terminal alphabet T) because matrix grammars without appearance checking are closed under arbitrary morphisms and under union ([8]).

For the construction explained above it follows that $Ps_T(L'') = Ps_T^{asyn}(\Pi)$.

Then $PsCSA_*^{asyn}(coo_e, coo_s) \subseteq PsMAT$.

On the other hand, a CSA with only one agent in the initial configuration and using only cooperative evolution rules can simulate a matrix grammar $G = (N, T, S, M)$ without appearance checking. To make matters simpler and

without loss of generality we suppose that M has p matrices (labelled by $1, \cdots, p$) each one with k productions (labelled by $1, \cdots, k$). It is always possible to add "dummy" matrices. We also suppose, again with no loss of generality, that the only production that rewrites S is the first production of matrix 1.

We construct the set $L_M = \{(m_i, m_j) \mid 1 \leq i \leq p, 1 \leq j \leq k\}$.

We then construct a CSA, $\Pi = (A = N \cup T \cup L_M \cup \{x\}, T, C, R)$, with $C = \{S(m_1, m_1)\}$ and $x \notin N \cup T \cup L_M$.

The set of rules R is obtained in the following way. For each matrix $i : (a_1 \rightarrow u_1, a_2 \rightarrow u_2, \ldots, a_k \rightarrow u_k)$ in M and with $i \in \{1, \cdots, p\}$, $a_1, a_2, \ldots, a_k \in N$, and $u_1, u_2, \cdots, u_k \in (N \cup T)^*$, we add to R the following cooperative evolution rules $\{(m_i, m_1)a_1 \rightarrow u_1(m_i, m_2)x, (m_i, m_2)a_2 \rightarrow u_2(m_i, m_3), \cdots, (m_i, m_{k-1})a_{k-1} \rightarrow (m_i, m_k)u_{k-1} \mid 1 \leq i \leq m\} \cup \{x(m_i, m_k)a_k \rightarrow u_k(m_j, m_1) \mid 1 \leq i \leq p, 1 \leq j \leq p\} \cup \{x \rightarrow x\} \cup \{a \rightarrow a \mid a \in N\}$.

It is straightforward to see that any successful derivation in G producing the string w can be simulated in Π by starting from the initial configuration C and applying the corresponding evolution rules in R until a halting configuration $\{(m_j, m_1)w\}$, for some $1 \leq j \leq p$, is reached.

Moreover, for any asynchronous computation c in Π halting in a configuration $\{(m_j, m_1)w\}$, for some $1 \leq j \leq p$, there is a derivation in G producing w.

In fact, due to the way R is defined, all computations of Π are obtained by having iterative applications of "blocks" of rules.

Each block of rules is a sequence of applications of rules, $(m_i, m_1)a_1 \rightarrow u_1(m_i, m_2)x, (m_i, m_2)a_2 \rightarrow u_2(m_i, m_3), \cdots, (m_i, m_{k-1})a_{k-1} \rightarrow (m_i, m_k)u_{k-1}$, $x(m_i, m_k)a_k \rightarrow u_k(m_j, m_1)$ for some $i \in \{1, \cdots, p\}$ and some $j \in \{1, \cdots, p\}$.

Once a block has been started (i.e., $(m_i, m_1)a_1 \rightarrow u_1(m_i, m_2)x$ is applied) it must also be completed (i.e., $x(m_i, m_k)a_k \rightarrow u_k(m_j, m_1)$ applied): in a computation, a block cannot be interrupted because this would lead to the object x being present in the configuration of the system, which would then make the evolution non-halting (because of the rule $x \rightarrow x$ in R).

It is easy to see that each element of this block of rules can be simulated in G by applying the corresponding matrix.

Moreover, there are no computations in Π halting in a configuration $\{(m_j, m_1)w\}$ for some $1 \leq j \leq p$ with w having objects from N (non terminals of G). This because of the rules $a \rightarrow a$, $a \in N$ present in R.

From the above description, we have that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$.

Thus, $PsCSA_*^{asyn}(coo_e, coo_s) \supseteq PsMAT$ and the Theorem follows.

□

Using a similar construction to that in the proof of Theorem 9 and from the last statement of Theorem 1 we obtain.

Corollary 2 *For an arbitrary CSA, Π, there exists a regular grammar G with one-letter terminal alphabet such that $N^{asyn}(\Pi) = NL(G)$.*

Proof. The proof is obtained by a slight modification of the first part of Theorem 9.

Given the CSA, $\Pi = (A, T, C, R)$ we construct a matrix grammar without a.c., $G = (N, N, S, M)$, as given in the first part of Theorem 9. Then, again following Theorem 9, we construct the language L' to contain all the strings corresponding to halting configurations reached by asynchronous computations of Π.

For instance, if $\{w'_1, w'_2, \cdots, w'_m\}$ is a halting configuration reached by Π, then in L' there is the string $\{h_1(w_1), h_2(w_2), \cdots, h_m(w_m)\}$, where $h_1, h_2, \ldots, h_m$ are morphisms defined as in the proof of Theorem 9. As explained in the proof of Theorem 9 L' can be generated by a matrix grammar without a.c.

We construct then the following morphisms $d_i : N \longrightarrow \{z\} \cup \{\lambda\}$, for each $i \in \{1, \cdots, m\}$, defined in the following manner (z is a new symbol not in N).

$$d_i(a_i) = z, a \in A.$$
$$d_i(a_j) = \lambda, a \in A, j \neq i.$$

Then, for each $i \in \{1, \cdots, m\}$, the language $d_i(L')$ selects from each string in L' the substring corresponding to the agent with objects indexed by i and replaces all the objects of the agent by the symbol z.

Now we construct the language $L'' = \bigcup_{1 \leq i \leq m} d_i(L')$.

L'' is the language that collects all the agents present in the halting configurations, considering all the computations of Π.

¿From the construction it is clear that $N^{asyn}(\Pi) = NL''$.

Moreover, L'' can also be generated by a matrix grammar without a.c., using terminal alphabet $\{z\}$. Matrix grammars without a.c. are closed under union and arbitrary morphism, see, e.g., [8].

The result then follows from the fact that a language generated by a matrix grammar without a.c. over a one letter alphabet is regular (Theorem 1).

□

Using Theorem 8 and Theorem 9 we obtain:

Corollary 3

$$PsCSA^{asyn}_{*}(coo_e, coo_s) \subset PsCSA^{mp}_{1}(coo_e, coo_s) = PsCSA^{mp}_{1}(coo_e) = PsCSA^{mp}_{1}(coo_s).$$

When using unary rules the computational power is equivalent to that of finite sets of vectors of natural numbers, even for CSAs working in the maximally parallel mode.

Theorem 10 $PsCSA^{asyn}_{*}(un_e, un_s) = PsCSA^{mp}_{*}(un_e, un_s) = PsFIN.$

Proof. In CSAs using only unary rules the sizes of the agents present in the initial configuration cannot be increased, so, because of the finite number of possible combinations, these systems can only generate finite sets of vectors of numbers as output. On the other hand, any finite set, S, of vectors of numbers can be obtained as output of a CSA, Π, by having in the initial configuration of Π, for each vector v in S, one agent w with Parikh vector v (with respect to an adequate terminal alphabet). □

However, by combining unary synchronization rules and non-cooperative evolution rules, we obtain computational completeness for CSAs working in the maximally parallel way with two agents in the initial configuration. The proof of this result is by simulation of EC P systems.

Theorem 11 $PsCSA_2^{mp}(ncoo_e, un_s) = PsRE$.

Proof. Programmed grammars with appearance checking are grammars, known to be computationally complete (see, e.g., [8]). Without going into unnecessary details, we mention that in [2] it has been shown that for any programmed grammar with appearance checking, G, with terminal alphabet T, there exists an EC P system Π with two membranes, non-cooperative evolution rules, symport/antiport rules of weight at most one and such that $Ps_T(L(G)) = Ps(\Pi)$. This proves that $PsECP_2(1, 1, ncoo) = PsRE$.

We show that any evolution-communication P systems with two membranes, non-cooperative evolution rules and antiport rules of weight one can be simulated by using a CSA system with two agents, non-cooperative evolution rules, unary synchronization rules and working in the maximally parallel way (the two agents represent the two regions enclosed by the two membranes in the EC P system).

For an arbitrary programmed grammar with a.c., G, with terminal alphabet T, we construct (using the construction proposed in [2]) an EC P system $\Pi = (O, [\,[\,]_2\,]_1, w_1, w_2, R_1, R_2, R'_1, R'_2, i_0)$, with $T \subseteq O$, such that $Ps_T(L(G)) = Ps(\Pi)$. Π is constructed in such a way that its output at the end of a computation consists of objects corresponding to the terminals T collected in the environment. These objects are immediately sent into the environment once they are obtained in region 1 and remain there unchanged until the end of the computation (the symport rules associated to membrane 1 are used only to send to the environment these objects and no other antiport or symport rules are associated to membrane 1).

We define $O_1 = \{a_1 \mid a \in O\}$ and $O_2 = \{a_2 \mid a \in O\}$.

We define two morphisms that map the objects of O into indexed objects (the index denotes the region of Π where the object is present).

Precisely, we define $h_1 : O \to O_1$ as $h_1(a) = a_1$ for each $a \in O$, $h_2 : O \to O_2$ defined as $h_2(a) = a_2$ for each $a \in O$.

Now we construct the CSA, $\Pi' = (A, T', C, R)$, as follows.

We set $A = \{h_1(a), h_2(a) \mid a \in O\}$ and $C = \{h_1(w_1), h_2(w_2)\}$. The terminal alphabet T' is defined as $\{h_1(a) \mid a \in T\}$.

The rules in R are constructed in the following manner.

For each rule $a \to v$ in R_i, $i \in \{1, 2\}$, add to R the rule $h_i(a) \to h_i(v)$.

For each symport rule (a, in) present in R'_2, add to R the synchronization rule $\langle h_1(a), \lambda\rangle \to \langle \lambda, h_2(a)\rangle$.

For each symport rule (a, out) present in R'_2, add to R the synchronization rule $\langle h_2(a), \lambda\rangle \to \langle \lambda, h_1(a)\rangle$.

For each antiport rule $(a, in; b, out)$ present in R'_2, add to R the synchronization rule $\langle h_1(a), h_2(b)\rangle \to \langle h_1(b), h_2(a)\rangle$.

All (and only) the computations of Π are simulated by computations of Π'.

The idea is that the two agents in Π' represent the contents of the regions and of the environment of Π: the agent with objects indexed by 1 represents the contents of region 1 and the objects in the environment, while the agent with objects indexed by 2 represents the contents of region 2.

Evolution rules and symport/antiport rules in Π are simulated by the corresponding constructed evolution and synchronization rules, respectively, present in R.

The use of indexed objects for the agents guarantees that the two agents are maintained separate, such that no incorrect interaction (i.e., synchronization) can take place and every configuration of Π', reached during any computation, will always have two agents; one with all objects indexed by 1 and one with all objects indexed by 2. That is, there are no computations in Π' that reach a configuration having agents with objects with different indexes.

¿From the way Π' is constructed, it can easily be seen that for each computation in Π, producing in the environment a multiset of objects w, for $w \in T^*$ (i.e., the output of the computation of Π is the vector $v = Ps_T(w)$), there exists a computation for Π' having, in the halting configuration, the agents $h_1(ww'), h_2(w'')$ with $w'' \in O^*$, $w \in T^*$, $w' \in (O-T)^*$. That is, the output of the computation is the set composed of the vectors $v = Ps_{T'}(h_1(ww')) = Ps_{T'}(h_1(w))$ and $Ps_{T'}(h_2(w'')) = \overline{0}$. The empty vector is also present since in $h(w'')$ there are no objects from T.

On the other hand, for each computation in Π', with the agents $h_1(ww')$ and $h_2(w'')$ in the halting configuration, with $w' \in O^*$, $w \in T^*$ and $w'' \in O^*$ (i.e., the output is the set composed of the vectors $v = Ps_{T'}(h_1(ww')) = Ps_{T'}(h_1(w))$ and $Ps_{T'}(h_2(w'')) = \overline{0}$), there exists a computation in Π producing the multiset of objects w in the environment in the halting configuration (i.e., having as output the vector $v = Ps_T(w)$).

Because in the equality of sets of vectors we do not consider the null vector, the Theorem follows.

□

Note that the role of synchronization rules, even if only unary, is crucial: when this type of rule is not used, the computational power of CSAs is only regular (in terms of Parikh images).

Theorem 12 $PsCSA_*^{mp}(ncoo_e) = PsCF$.

Proof. For an arbitrary CSA, $\Pi = (A, T, C, R)$, with m agents w_1, w_2, ..., w_m (no bound on m) there exists a P system, Π', with symbol-objects and non-cooperative evolution rules working in the maximally parallel way, such that $Ps_T(\Pi') = Ps_T^{mp}(\Pi)$. The P system, $\Pi' = (A \cup \{S\}, T, [\ ]_1, S, R_1)$, needs only one region labeled 1. We add to R_1 the following rules: $\{S \to w_1, S \to w_2, \ldots, S \to w_m\}$ and all the rules present in R. Clearly, for each vector v in $Ps_T^{mp}(\Pi)$ there is a computation in Π' that halts with a multiset of objects w in region 1, such that $Ps_T(w) = v$. Equally, for each vector v obtained as the output of a computation in Π' there exists a computation in Π halting in a configuration containing the agent w with $Ps_T(w) = v$.

Vice versa, for a P system, $\Pi' = (O, T, [\ ]_1, w_1, R_1)$, with symbol-objects, non-cooperative evolution rules and working in the maximally-parallel way, it is possible to construct an equivalent CSA, $\Pi = (O, T, C, R)$, with $C = \{w_1\}$ and $R = R_1$. In a direct way we have that $Ps_T^{mp}(\Pi) = Ps_T(\Pi')$. Using Theorem 5, the result follows.

□

5 Robustness of CSAs: A (Preliminary) Formal Study

In this Section we investigate the robustness of CSAs against perturbations of some of the features of the system.

For this purpose we use a similar idea of robustness as that employed in [16], in the framework of grammar systems, adapted here to the proposed CSAs.

We want to investigate situations where either some of the agents or some of the rules of the colony do not function. What are the consequences to the behaviour of the colony?

We will try to investigate systems that are robust, e.g., where the behaviour does not change critically if one or more agents cease to exist in the system.

Let $\Pi = (A, T, C, R)$ be an arbitrary CSA.

We say that Π' is an *agent-restriction* of Π if $\Pi' = (A, T, C', R)$ with $C' \subseteq C$. Π' is a CSA where some of the agents originally present in Π no longer work, i.e., the CSA behaves as though they were absent from the system.

We also consider a *rule-restriction* of Π obtained by removing some or possibly all of the rules. Then, $\Pi' = (A, T, C, R')$ is a *rule-restriction* of Π if $R' \subseteq R$. In this case some of the rules do not work, i.e., the CSA behaves as if they were absent from the system.

We say that a CSA, Π, is *robust* when a core behaviour, i.e., the minimally accepted behaviour, is preserved when considering proper restrictions of it. A measure of the robustness of Π is the *difference* between the initial system and the *minimum* restriction preserving the core behaviour, where *difference* and *minimum* are to be defined.

By a *core behavior* of Π we mean a subset of the set of vectors of natural numbers generated by Π.

We define these subsets by making an intersection with a set of vectors from $PsREG$ that defines the regular property of the core behaviour we are interested in. Note that the core behaviour may be infinite.

Questions about robustness can then be formalized in the following manner.

Consider an arbitrary CSA, Π, an arbitrary agent- or rule-restriction Π' of Π, and an arbitrary set S from $PsREG$. Is it possible to decide whether or not $Ps(\Pi) \cap S \subseteq Ps(\Pi')$ (i.e., whether Π is robust against the restriction Π', in the sense that it will continue to generate, at least, the core behaviour)?

Example 2 *We produce a small example that clarifies the introduced notion of robustness in the case of agent-restriction and asynchronous computations. The other cases (rule-restriction, maximally-parallel computations) are conceptually similar.*

Consider a CSA $\Pi = (A, T, C, R)$ *with* $A = \{a, b, c, d, e, f\}$, $T = \{e, f\}$, $C = \{(ab, 1), (bc, 1), (bd, 1), (a, 1)\}$. *The rules in* R *are*
$\{\langle ab, bc\rangle \rightarrow \langle eff, eff\rangle, \langle ab, bd\rangle \rightarrow \langle eff, eff\rangle\}$.

There are two possible asynchronous computations of Π*, which are represented diagrammatically in Figure 4.*

As it is possible to see, collecting the results (vectors representing the multiplicities of the terminal objects in the agents in the halting configurations) we obtain that $Ps_T^{asyn}(\Pi) = \{(1, 2), \overline{0}\} \cup \{(1, 2), \overline{0}\} = \{(1, 2), \overline{0}\}$*, where* $\overline{0}$ *denotes* $(0, 0)$.

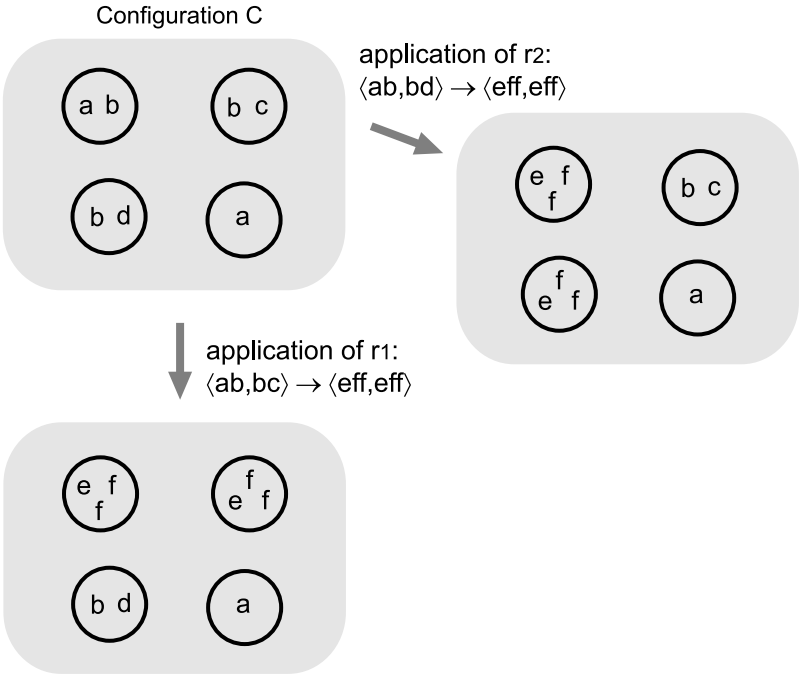

Figure 4: The two possible asynchronous computations of Π of Example 2

In fact, we have in the two halting configurations (for the two computations), two agents eff, whose associated Parikh vector (with respect to T) is $(1,2)$ and the other agents, bd, bc and a, whose associated Parikh vectors, with respect to T, are the null vector (the agents do not contain any terminal object).

Now, suppose we fix a core behaviour to be the set of vectors $\{(1,2)\}$ (it can be clearly obtained by an intersection of $Ps_T^{asyn}(\Pi)$ with $\{(1,2)\}$, which is in $PsREG$).

The system Π is robust when the agent bc is deleted from its initial configuration. In fact, if we consider $\Pi' = (A, T, C', R)$, with $C' = \{(ab,1), (bd,1), (a,1)\}$, we have that $Ps_T^{asyn}(\Pi') = \{(1,2), \overline{0}\}$, which still contains the defined core behaviour. The only possible computation of Π' is represented in Figure 5.

On the other hand, the system Π is not robust when the agent ab is deleted from its initial configuration. If we consider $\Pi'' = (A, T, C'', R)$, with $C'' = \{(bd,1), (a,1)\}$, we have that $Ps_T^{asyn}(\Pi'') = \{\overline{0}\}$, which does not contain the core behaviour. The system Π'' is represented in Figure 6. The only possible computation of Π'' is the one that halts in the initial configuration C''.

We move now to analyse the case of rule-restrictions with asynchronous evolution and demonstrate a negative result.

In what follows we suppose that an arbitrary set S from $NREG$ (from $PsREG$) is given by having the corresponding grammar G from REG such that $NL(G) = S$ ($Ps(L(G)) = S$, resp.).

Theorem 13 *It is undecidable whether or not for an arbitrary CSA, Π, with terminal alphabet T, arbitrary rule restriction Π' of Π and arbitrary set S from $PsREG_T$, $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$.*

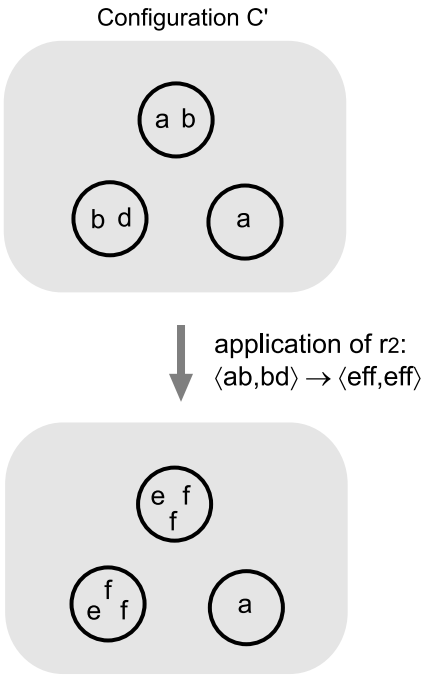

Figure 5: Robust behaviour of Π' of Example 2 when agent bc is removed from C.

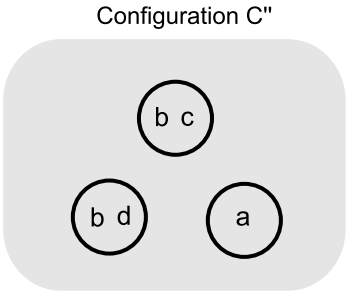

Figure 6: No robustness displayed by Π'' of Example 2 when agent ab is removed from C.

Proof. We start by having two arbitrary matrix grammars without a.c., $G = (N, T, S, M)$ and $G' = (N', T, S, M')$ with $N \cap N' = \{S\}$. It is undecidable whether or not $Ps_T(L(G)) \subseteq Ps_T(L(G'))$ (see Corollary 1).

To make matters simpler and without loss of generality we suppose that M has p matrices (labelled by $1, 2, \cdots, p$) of k productions (labelled by $1, 2, \cdots, k$) and M' has m' matrices (labelled by $1, 2, \cdots, m'$) of k' productions (labelled by $1, 2, \cdots, k'$). Again with no loss of generality we also suppose that the only production in M (and in M') that rewrite the axiom S is the production 1 of matrix 1.

We construct the sets $L_M = \{(m_i, m_j) \mid 1 \leq i \leq p, 1 \leq j \leq k\}$ and $L_M = \{(m'_i, m'_j) \mid 1 \leq i \leq m', 1 \leq j \leq k'\}$.

As in the second part of Theorem 9, we construct a CSA, Π, equivalent to G in the following way.

$\Pi = (A = N \cup N' \cup T \cup L_M \cup L'_M \cup \{x\}, T, C, R)$ with $C = \{S(m_1, m_1)(m'_1, m'_1)\}$ and with the set of rules R obtained in the following way.

For each matrix $i : (a_1 \to u_1, a_2 \to u_2, \ldots, a_k \to u_k)$ in M with $a_1, a_2, \ldots, a_k \in N$ and $u_1, u_2, \cdots, u_k \in (N \cup T)^*$, with $i \in \{1, \cdots, p\}$, we add to R the following cooperative evolution rules
$\{(m_i, m_1)a_1 \to u_1(m_i, m_2)x, (m_i, m_2)a_2 \to u_2(m_i, m_3), \cdots, (m_i, m_{k-1})a_{k-1}$
$\to (m_i, m_k)u_{k-1}\} \cup \{x(m_i, m_k)a_k \to u_k(m_j, m_1) \mid 1 \leq j \leq p\} \cup \{a \to a \mid a \in N\} \cup \{x \to x\}$.

Using the same arguments as in the proof of Theorem 9 it is possible to see that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$.

In a similar way we construct a CSA, $\Pi' = (A, T, C, R')$, which is the same as Π except that has rules R', constructed as follows.

For each matrix $i : (a_1 \to u_1, a_2 \to u_2, \cdots, a_{k'} \to u_{k'})$ in M' with $a_1, a_2, \cdots, a_k \in N'$ and $u_1, u_2, \cdots, u_k \in (N' \cup T)^*$, with $i \in \{1, \cdots, m'\}$, we add to R' the following cooperative evolution rules
$\{(m'_i, m'_1)a_1 \to u_1(m'_i, m'_2)x, (m'_i, m'_2)a_2 \to u_2(m'_i, m'_3), \cdots, (m'_i, m'_{k'-1})a_{k'-1}$
$\to (m'_i, m'_{k'})u_{k'-1}\} \cup \{x(m'_i, m'_{k'})a_{k'} \to u_{k'}(m'_j, m'_1) \mid 1 \leq j \leq m'\} \cup \{a \to a \mid a \in N'\} \cup \{x \to x\}$.

Again, using the same arguments as in the proof of Theorem 9, we have that $Ps_T^{asyn}(\Pi') = Ps_T(L(G'))$.

We then construct the CSA $\Pi'' = (A, T, C, R' \cup R)$. It can be seen that $Ps_T^{asyn}(\Pi'') = Ps_T^{asyn}(\Pi) \cup Ps_T^{asyn}(\Pi') = Ps_T(L(G)) \cup Ps_T(L(G'))$. In fact, applying rules from R one gets $Ps_T^{asyn}(\Pi)$, while applying rules from R' one gets $Ps_T^{asyn}(\Pi')$. The application of the rules cannot be "mixed" since $N \cap N' = \{S\}$.

Now suppose there exists an algorithm to decide whether or not, for arbitrary CSA, Π, arbitrary rule restriction Π' of Π and arbitrary set S from $PsREG_T$, $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$.

We could then apply this algorithm to decide whether or not $Ps_T^{asyn}(\Pi'') \cap Ps_T(T^*) \subseteq Ps_T^{asyn}(\Pi')$. Notice that Π' is a rule restriction of Π''.

If the answer is true then $Ps_T^{asyn}(\Pi) \subseteq Ps_T^{asyn}(\Pi')$, otherwise (answer false) $Ps_T^{asyn}(\Pi) \not\subseteq Ps_T^{asyn}(\Pi')$.

So we could also decide whether or not $Ps_T(L(G)) \subseteq Ps_T(L(G'))$, which is not possible. Hence, by contradiction, the Theorem follows. □

Note, however, that the result is different when the considered core behaviour is finite.

Theorem 14 *It is decidable whether or not, for an arbitrary CSA,* Π*, with terminal alphabet* T*, arbitrary rule restriction* Π' *of* Π *and arbitrary finite set* S *from* $PsREG_T$*,* $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$*.*

Proof. To check whether or not $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$ we only need to construct $S' = Ps_T^{asyn}(\Pi) \cap S$ and then to check whether or not each vector in S' is in $Ps_T^{asyn}(\Pi')$. This can be done because:

- S is finite.
- For any arbitrary CSA, Π, with terminal alphabet T, we can construct a matrix grammar without a.c., G, with terminal alphabet T, such that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$ (Theorem 9) and the membership problem for matrix grammars without a.c. is decidable (see, e.g., [8]).
- Given a vector v, there is only a finite set of strings (over T) whose Parikh vector with respect to T is exactly v.

□

Suppose that we are only interested in the *size* of the agents and not in their internal structure. This means that we collect, for a colony Π, the set of numbers $N(\Pi)$. In this case the robustness problem can be rephrased in the following manner.

Consider an arbitrary CSA, Π, with an arbitrary agent- / rule-restriction Π' of Π and an arbitrary set S from $NREG$.

Is it possible to decide whether or not $N(\Pi) \cap S \subseteq N(\Pi')$ (i.e., whether Π is robust against the restriction Π')? Note that in this case the core behaviour is defined by specific sizes of the agents.

In this case we get the following positive results, even when considering infinite core behaviour.

Theorem 15 *It is decidable whether or not, for an arbitrary CSA,* Π*, arbitrary rule restriction* Π' *of* Π *and arbitrary set* S *from* $NREG$*,* $N^{asyn}(\Pi) \cap S \subseteq N^{asyn}(\Pi')$*.*

Proof.

We know that for an arbitrary CSA Π' we can construct a regular grammar G' with a one-letter terminal alphabet such that $N^{asyn}(\Pi') = NL(G')$ (Corollary 2).

Moreover, we can also construct a regular grammar G over a one-letter alphabet such that $N(L(G)) = N^{asyn}(\Pi') \cap S$.

The result then follows from the fact that, given two arbitrary regular grammars G_1 and G_2 it is decidable whether or not $L(G_1) \subseteq L(G_2)$ (see, e.g., [13]). In particular, this is true when the terminal alphabet of the two regular grammars is of cardinality one and the decidability result can be easily extended to the length sets of the languages. □

The same positive result holds when, considering vectors of numbers, the CSAs work in maximally-parallel mode but use only non-cooperative evolution rules.

Theorem 16 *It is decidable whether or not, for an arbitrary CSA,* Π *from* $CSA_*(ncoo_e)$*, with terminal alphabet* T*, arbitrary rule restriction* Π' *of* Π *and arbitrary set* S *from* $PsREG_T$*,* $Ps_T^{mp}(\Pi) \cap S \subseteq Ps_T^{mp}(\Pi')$*.*

Proof. For any CSA Π from $CSA_*(ncoo_e)$ with terminal alphabet T it is possible to construct a regular grammar G with terminal alphabet T' such that $Ps_T^{mp}(\Pi) = Ps_{T'}(L(G))$ (because of Theorem 12 and Theorem 2). Then it is also possible to construct a regular grammar G' over T' such that $Ps_T^{mp}(\Pi) \cap S = Ps_{T'}(L(G'))$. The result then follows from the fact that containment is decidable for regular languages (see, e.g., [13]) and this result can easily be extended to the Parikh images of regular languages because there is only a finite number of strings over an alphabet T' having a given Parikh vector with respect to T'. □

Note, however, that even if robustness against rule absence is in many cases undecidable when the core behaviour is infinite, it is still possible to decide whether a rule (evolution or synchronization) is used or not by a CSA. So, if a rule is not used we can remove it and the system will be robust against such deletion.

Theorem 17 *It is decidable whether or not, for an arbitrary CSA* $\Pi = (A, C, T, R)$ *and an arbitrary rule* r *from* R*, there exists at least one asynchronous computation for* Π *containing at least one configuration obtained by applying at least one instance of rule* r*.*

Proof. Given an arbitrary CSA, $\Pi = (A, C, T, R)$, and an arbitrary rule r from R we can construct, by modifying the construction given in the first part of the proof of Theorem 9, a matrix grammar without a.c., G, with terminal alphabet T, such that $Ps_T(L(G))$ is not the empty set if and only if there exists at least one asynchronous computation for Π having at least one transition where r is applied. This can be done, for instance, by modifying the matrix grammar G given in the proof of Theorem 9 as follows. A matrix is added that is applied at the beginning of each derivation of G and that introduces a non-terminal, X, which is removed only when the matrix that simulates the rule r is used. In this case we will have that $L(G)$ (also its Parikh image) is not the empty set if and only if there is a derivation in G where the matrix that simulates rule r is used. The Theorem follows from the fact that it is possible to decide whether or not $Ps_T(L(G))$ is the empty set (Corollary 1). □

Now we analyse the case when agent-restrictions are considered. In this case the problems remain undecidable when the core behaviour is infinite.

Theorem 18 *It is undecidable whether or not, for an arbitrary CSA,* Π*, with terminal alphabet* T*, arbitrary agent restriction* Π' *of* Π *and arbitrary set* S *from* $PsREG_T$*,* $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$*.*

Proof.

We start by having two arbitrary matrix grammars without a.c., $G = (N, T, S, M)$ and $G' = (N', T, S, M')$ with $N \cap N' = \{S\} \cup T$. It is undecidable whether or not $Ps_T(L(G)) \subseteq Ps_T(L(G'))$ (see Corollary 1).

To make matters simpler and without loss of generality we suppose that M has p matrices (labelled by $1 \cdots, p$) of k productions (labelled by $1, \cdots, k$) and M' has m' matrices (labelled by $1, \cdots, m'$) of k' productions (labelled by $1, \cdots, k'$). Again with no loss of generality we suppose that in M and M' only the production 1 of matrix 1 can rewrite S.

We construct the sets $L_M = \{(m_i, m_j) \mid 1 \leq i \leq p, 1 \leq j \leq k\}$ and $L'_M = \{(m'_i, m'_j) \mid 1 \leq i \leq m', 1 \leq j \leq k'\}$.

As in the second part of the proof of Theorem 9 we construct a CSA Π equivalent to G in the following way.

$\Pi = (A = N \cup N' \cup T \cup L_M \cup L'_M \cup \{x\}, T, C, R)$ with $C = \{S(m_1, m_1)\}$ and with the set of rules R obtained in the following way.

For each matrix $i : (a_1 \to u_1, a_2 \to u_2, \ldots, a_k \to u_k)$ in M with $a_1, a_2, \ldots, a_k \in N$ and $u_1, u_2, \cdots, u_k \in (N \cup T)^*$ with $i \in \{1, \cdots, p\}$, we add to R the following cooperative evolution rules

$\{(m_i, m_1)a_1 \to u_1(m_i, m_2)x, (m_i, m_2)a_2 \to u_2(m_i, m_3), \cdots, (m_i, m_{k-1})a_{k-1}$
$\to (m_i, m_k)u_{k-1}\} \cup \{x(m_i, m_k)a_k \to u_k(m_j, m_1) \mid 1 \leq j \leq p\}$.

For each matrix $i : (a_1 \to u_1, a_2 \to u_2, \ldots, a_{k'} \to u_{k'})$ in M' with $a_1, a_2, \ldots, a_k \in N'$ and $u_1, u_2, \cdots, u_k \in (N' \cup T)^*$ with $i \in \{1, \cdots, m'\}$, we add to R the following cooperative evolution rules

$\{(m'_i, m'_1)a_1 \to u_1(m'_i, m'_2)x, (m'_i, m'_2)a_2 \to u_2(m'_i, m'_3), \cdots, (m'_i, m'_{k'-1})a_{k'-1}$
$\to (m'_i, m'_{k'})u_{k'-1}\} \cup \{x(m'_i, m'_{k'})a_{k'} \to u_{k'}(m'_j, m'_1) \mid 1 \leq j \leq m'\}$.

We also add to R the set of rules $\{a \to a \mid a \in N\} \cup \{x \to x\}$.

Using the same arguments as in the proof of Theorem 9 we have that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$.

In a similar way we construct a CSA, $\Pi' = (A, T, C', R)$, with the only difference being the initial configuration $C' = \{S(m'_1, m'_1)\}$.

In this case, $Ps_T^{asyn}(\Pi') = Ps_T(L(G'))$.

We then construct the CSA $\Pi'' = (A, T, C + C', R)$.

We have that $Ps_T^{asyn}(\Pi'') = Ps_T^{asyn}(\Pi) \cup Ps_T^{asyn}(\Pi') = Ps_T(L(G)) \cup Ps_T(L(G'))$. Suppose that $Ps_T^{asyn}(\Pi)$ and $Ps_T^{asyn}(\Pi')$ are not the empty set.

Now suppose that there is an algorithm to decide, for an arbitrary CSA, Π, arbitrary agent restriction Π' of Π and arbitrary set S from $PsREG_T$, whether or not $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$.

We may use this algorithm to decide whether or not $Ps_T^{asyn}(\Pi'') \cap Ps_T(T^*)$ $\subseteq Ps_T^{asyn}(\Pi')$.

In fact, Π' is an agent restriction of Π''.

If the proposition is true then $Ps_T^{asyn}(\Pi) \subseteq Ps_T^{asyn}(\Pi')$, while if the proposition is false, $Ps_T^{asyn}(\Pi) \not\subseteq Ps_T^{asyn}(\Pi')$ (the case when $Ps_T^{asyn}(\Pi)$ or/and $Ps_T^{asyn}(\Pi')$ is the empty set is trivial, emptiness for Parikh images of languages generated by matrix grammars without a.c. is decidable, Corollary 1).

So we can decide whether or not $Ps_T(L(G)) \subseteq Ps_T(L(G'))$ and this is not possible (Corollary 1). From this, by contradiction, the Theorem follows. □

Using the same ideas as in the proof of Theorem 14 we get the following result.

Theorem 19 *It is decidable whether or not, for an arbitrary CSA,* Π*, with terminal alphabet* T*, arbitrary agent restriction* Π' *of* Π *and arbitrary finite set* S *from* $PsREG_T$*,* $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$.

Using the same ideas as in the proof of Theorem 15 we get the following result.

Theorem 20 *It is decidable whether or not, for an arbitrary CSA,* Π*, arbitrary agent restriction* Π' *of* Π *and arbitrary set* S *from* $NREG$*,* $N(\Pi) \cap S \subseteq N(\Pi')$*.*

Obviously, for CSAs that are computationally complete (in a constructive way), every non-trivial property is undecidable (Rice's Theorem, see, e.g., [13]). So this is already shown to be true for CSAs with non-cooperative evolution rules, unary synchronization rules and working in the maximally parallel mode.

Therefore, from Theorem 11, we have the following result.

Theorem 21 *It is undecidable whether or not, for an arbitrary CSA,* Π *from* $CSA_*(coo_e, un_s)$ *with terminal alphabet* T*, arbitrary agent or rule restriction* Π' *of* Π *and arbitrary set* S *from* $PsREG_T$*,* $Ps_T^{mp}(\Pi) \cap S \subseteq Ps_T^{mp}(\Pi')$*.*

Note that, invoking Rice's Theorem once again, we get the same negative results even when considering finite core behaviour and length sets.

Theorem 22 *It is undecidable whether or not for an arbitrary CSA,* Π *from* $CSA_*(coo_e, un_s)$*, with terminal alphabet* T*, arbitrary agent or rule restriction* Π' *of* Π *and arbitrary finite set* S *from* $PsREG_T$*,* $Ps_T^{mp}(\Pi) \cap S \subseteq Ps_T^{mp}(\Pi')$*.*

Theorem 23 *It is undecidable whether or not, for an arbitrary CSA,* Π *from* $CSA_*(coo_e, un_s)$*, arbitrary agent or rule restriction* Π' *of* Π *and arbitrary set* S *from* $NREG$*,* $N^{mp}(\Pi) \cap S \subseteq N^{mp}(\Pi')$*.*

6 A Computational Tree Logic for CSAs

In this section we continue the investigation of the dynamic properties of CSAs and for this purpose we introduce a *computational tree logic (CTL temporal logic)* to formally specify, verify and model-check properties of CSAs. An introduction to the basic notions and results of temporal logics can be found in [3, 21].

Temporal logics are the most used logics in model-checking analysis: efficient algorithms and tools having already been developed for them, e.g. NuSMV [22]. They are devised with operators for expressing and quantifying on possible evolutions or configurations of systems. For instance, for an arbitrary system it is possible to specify properties such as *'for any possible evolution,* ϕ *is fulfilled'*, *'there exists an evolution such that* ϕ *is not true'*, *'in the next state* ϕ *will be satisfied'*, *'eventually* ϕ *will be satisfied'* and *'*ϕ *happens until* ψ *is satisfied'*, with ϕ and ψ properties of the system. We show how to use these operators to formally specify and verify complex properties of CSAs, such as *'the agent will always eventually reach a certain configuration'*, or *'rule* r *is not applicable until rule* r' *is used'*, etc.

Definition 6.1 (Preconditions) *Let* A *be an arbitrary alphabet and* R *an arbitrary set of rules over* A*. We define the mapping* $prec : R \to 2^{\mathbb{M}(A)}$ *by*

- *if* $r \in R$ *is the evolution rule* $u \to v$ *then* $prec(r) = \{u\}$*;*
- *if* $r \in R$ *is a synchronization rule* $\langle u, v\rangle \to \langle u', v'\rangle$ *then* $prec(r) = \{u\} \cup \{v\}$*.*

We define $prec(R) = \bigcup_{r \in R} prec(r)$.

We now extend the definition of γ-evolutions for a given CSA by introducing the notion of γ-*complete evolution* defined for arbitrary classes of CSAs.

In what follows, let $\mathcal{C} = CSA_m^{A,T,R}$ be a class of all the CSAs having alphabet A, terminal alphabet T, set of rules R over A, degree m, with A, T, R and m arbitrarily chosen.

Definition 6.2 (γ-complete evolutions) *A sequence of CSAs* $\langle \Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_i, \ldots \rangle$ *with* $\Pi_i = (A, T, C_i, R) \in \mathcal{C}$, $i \geq 0$, *is called* γ-complete evolution *in* $\mathcal{C}$ *starting in* Π_0 *if* $\langle C_0, C_1, C_2, \ldots C_i, \ldots \rangle$, $i \geq 0$, *is a halting or an infinite* γ*-evolution of* Π_0, *with* $\gamma \in \{asyn, mp\}$.

We denote by $E_{\mathcal{C}}^{\gamma}(\Pi_0)$ *the set of all* γ*-complete evolutions in* $\mathcal{C}$ *starting at* Π_0.

Let $e = \langle \Pi_0, \Pi_1, \ldots, \Pi_i, \Pi_{i+1} \ldots \rangle$ *be an arbitrary* γ*-complete evolution in* $\mathcal{C}$ *starting in* Π_0. *We call* $\langle \Pi_i, \Pi_{i+1}, \ldots \rangle$, $i \geq 0$, *an i-suffix evolution*[1] *of* e *and we denote it by* e_i.

Definition 6.3 (Syntax of $\mathcal{L}_{\mathcal{C}}$) *The set* $AP(\mathcal{C})$ *is defined by:*

- $\top \in AP(\mathcal{C})$.
- $prec(R) \subseteq AP(\mathcal{C})$.
- *if* $w_1, w_2, \ldots, w_i \in prec(R) \cup \{\top\}$, $i \leq m$, *then* $w_1 \oplus \ldots \oplus w_i \in AP(\mathcal{C})$.

We call the elements of $AP(\mathcal{C})$ atomic formulas *of the logic* $\mathcal{L}_{\mathcal{C}}$.
We define the configuration formulas *of* $\mathcal{L}_{\mathcal{C}}$ *and the* evolution formulas *of* $\mathcal{L}_{\mathcal{C}}$ *in the following way.*

- *any atomic formula of* $\mathcal{L}_{\mathcal{C}}$ *is a configuration formula of* $\mathcal{L}_{\mathcal{C}}$.
- *if* ϕ, ψ *are configuration formulas of* $\mathcal{L}_{\mathcal{C}}$ *then* $\neg\phi$ *and* $\phi \wedge \psi$ *are configuration formulas of* $\mathcal{L}_{\mathcal{C}}$.
- *if* ϕ *is an evolution formula of* $\mathcal{L}_{\mathcal{C}}$ *then* $E\phi$ *is a configuration formula of* $\mathcal{L}_{\mathcal{C}}$.
- *if* ϕ, ψ *are configuration formulas of* $\mathcal{L}_{\mathcal{C}}$ *then* $X\phi$ *and* $\phi U \psi$ *are evolution formulas of* $\mathcal{L}_{\mathcal{C}}$.

The configuration formulas and evolution formulas of $\mathcal{L}_{\mathcal{C}}$ *form the* language of $\mathcal{L}_{\mathcal{C}}$.

The meanings of $\top, \neg, \wedge$ are those from classical logic. In addition, we have the temporal operators: $E\phi$ that expresses an existential quantification on evolutions, $X\phi$ which means "at the next configuration ϕ is satisfied" and $\phi U \psi$ which means "ϕ is satisfied until ψ is satisfied". In what follows, the properties we can express by using these operators are checked for some models called temporal structures.

[1] Observe that for an arbitrary γ-complete evolution e in $\mathcal{C}$, for each $i \geq 0$, e_i is also a γ-complete evolution in $\mathcal{C}$.

Definition 6.4 (Temporal structures) *We define the structure* $\mathcal{T}_{\mathcal{C}}^{\gamma} = (\mathcal{S}, \mathfrak{R})$, $\gamma \in \{asyn, mp\}$, *as follows:*

- $\mathcal{S} \subseteq \mathcal{C}$, *such that if* $\Pi_0 \in \mathcal{S}$ *then* $\{\Pi_1, \Pi_2, \ldots \mid \langle \Pi_0, \Pi_1, \Pi_2, \ldots \rangle \in E_{\mathcal{C}}^{\gamma}(\Pi_0)\} \subseteq \mathcal{S}$.
- $\mathfrak{R} \subseteq \mathcal{S} \times \mathcal{S}$, *such that* $(\Pi_1, \Pi_2) \in \mathfrak{R}$ *iff there exists* $\langle \Pi_1, \Pi_2, \ldots \rangle \in E_{\mathcal{C}}^{\gamma}(\Pi_1)$.

We call $\mathcal{T}_{\mathcal{C}}^{\gamma}$ *a temporal structure in* $\mathcal{C}$.

Definition 6.5 (CSA-Semantics) *Let* $\mathcal{T}_{\mathcal{C}}^{\gamma} = (\mathcal{S}, \mathfrak{R})$ *be a temporal structure in* $\mathcal{C}$. *For an arbitrary* $\Pi \in \mathcal{S}$, *an arbitrary* $e \in E_{\mathcal{C}}^{\gamma}(\Pi)$ *and an arbitrary formula* ϕ *from the language of* $\mathcal{L}_{\mathcal{C}}$, *we define coinductively the satisfiability relations* $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \phi$ *and* $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi$ *by:*

$\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \top$ *always.*

$\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models w$ *for* $w \in prec(R)$ *iff* $C_{\Pi} = \{(w', 1)\}$ *and* $w \subseteq w'$.

$\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models w_1 \oplus w_2 \oplus \ldots \oplus w_i$ *for* $w_j \in prec(R) \cup \{\top\}, 1 \leq j \leq i$ *iff* $C_{\Pi} = C_1 + C_2 + \ldots + C_i$ *s.t. for any* $w_j \neq \top$, $1 \leq j \leq i$, $C_j = \{(w_j + u_j, 1)\}$ *for some* $u_j \in \mathbb{M}(A)$.

$\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \phi \wedge \psi$ *iff* $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \phi$ *and* $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \psi$.

$\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \neg\phi$ *iff* $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \not\models \phi$.

$\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models E\phi$ *iff there exists* $e \in E_{\mathcal{C}}^{\gamma}(\Pi)$ *such that* $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi$.

$\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi U \psi$ *iff there exists* $i \geq 0$ *such that* $\mathcal{T}_{\mathcal{C}}^{\gamma}, e_i \models \psi$ *and for all* $j \leq i$ $\mathcal{T}_{\mathcal{C}}^{\gamma}, e_j \models \phi$.

$\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models X\phi$ *iff* $\mathcal{T}_{\mathcal{C}}^{\gamma}, e_1 \models \phi$.

Definition 6.6 (Validity and satisfiability) *A configuration formula* ϕ *(evolution formula* ϕ*) from* $\mathcal{L}_{\mathcal{C}}$ *is* valid *iff for every temporal structure* $\mathcal{T}_{\mathcal{C}}^{\gamma} = (\mathcal{S}, \mathfrak{R})$ *in* $\mathcal{C}$ *and any* $\Pi \in \mathcal{S}$ *(any* $e \in E_{\mathcal{C}}^{\gamma}(\Pi)$, *resp.) we have* $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \phi$ *(*$\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi$, *resp.). A configuration formula* ϕ *(evolution formula* ϕ*) is satisfiable iff there exists a temporal structure* $\mathcal{T}_{\mathcal{C}}^{\gamma} = (\mathcal{S}, \mathfrak{R})$ *and a* $\Pi \in \mathcal{S}$ *(an* $e \in E_{\mathcal{C}}^{\gamma}(\Pi)$, *resp.) such that* $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \phi$ *(*$\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi$, *resp.).*

Definition 6.7 (Derived formulas) *We define the following derived formulas for* $\mathcal{L}_{\mathcal{C}}$.

$A\phi = \neg E \neg \phi$.

$F\phi = \top U \phi$.

$G\phi = \neg F \neg \phi$.

The semantics of the derived formulas are the following.

$\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models A\phi$ iff for any $e \in E_{\mathcal{C}}^{\gamma}(\Pi)$ we have $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi$.

$\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models F\phi$ iff there exists $i \geq 0$ such that $\mathcal{T}_{\mathcal{C}}^{\gamma}, e_i \models \phi$.

$\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models G\phi$ iff for any $i \geq 0$ we have $\mathcal{T}_{\mathcal{C}}^{\gamma}, e_i \models \phi$.

$A\phi$ is a universal quantification on evolutions. $F\phi$ means "eventually ϕ is satisfied" (i.e., $F\phi$ is satisfied by an evolution that contains at least one configuration that has the property ϕ). $G\phi$ means "globally ϕ is satisfied" (i.e., $G\phi$ is satisfied by an evolution that contains only configurations satisfying ϕ).

Theorem 24 (Decidability) *The satisfiability, validity and model-checking problems for* $\mathcal{L}_{\mathcal{C}}$ *against the CSA-semantics are decidable.*

Proof. The result derives from the fact that CTL logic is decidable (see, e.g., [21, 3]) and from the fact that $AP(\mathcal{C})$, the set of atomic formulas, is a finite set. □

To show the potential of the introduced logic we give a small example of properties that can be specified. We pose the question whether or not during any evolution the agents can always synchronize when they are *ready* to do so.

In other words, given an arbitrary CSA, Π, and an arbitrary rule $r : \langle u, v\rangle \to \langle u', v'\rangle$, we would like to check whether or not it is true that, whenever during an evolution of Π, a configuration with an agent w_1, where $u \subseteq w_1$, is reached, then in the same configuration there is also an agent w_2 with $v \subseteq w_2$ (so rule r can actually be applied). If this is true we say that Π is *safe on synchronization* of rule r.

This property can be expressed in the proposed temporal logic by the following formula.

$$AG((u \oplus \top) \to (u \oplus v \oplus \top)).$$

Taking a CSA, Π_0, from $\mathcal{C}$. If we consider the introduced CSA-semantics we have that:

$\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi_0 \models AG((u \oplus \top) \to (u \oplus v \oplus \top))$
iff for any $e \in E_{\mathcal{C}}^{\gamma}(\Pi_0)$ we have $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models G((u \oplus \top) \to (u \oplus v \oplus \top))$
iff for any $e = \langle \Pi_0, \Pi_1, \ldots, \Pi_i, \ldots\rangle \in E_{\mathcal{C}}^{\gamma}(\Pi_0)$ and any $i \geq 0$ we have
$\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi_i \models (u \oplus \top) \to (u \oplus v \oplus \top)$.

This means that if any configuration present in a γ-evolution of Π_0 satisfies $u \oplus \top$ then it will also satisfy $u \oplus v \oplus \top$.

In fact, we know that $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi_i \models u \oplus \top$ iff $C_{\Pi_i} = C_1 + C_2$, $C_1, C_2 \in \mathbb{M}(\mathbb{M}(A))$ and $C_1 = \{(u + u', 1)\}$, i.e., the configuration of Π_i contains an agent w that contains u.

Similarly, $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi_i \models u \oplus v \oplus \top$ iff $C_{\Pi_i} = C_1' + C_2' + C_3'$, $C_1', C_2', C_3' \in \mathbb{M}(\mathbb{M}(A))$ and $C_1' = \{(u + u'', 1)\}$, $C_2' = \{(v + v', 1)\}$, i.e., the configuration of Π_i contains two agents w_1 and w_2 such that $u \subseteq w_1$ and $v \subseteq w_2$, which precisely indicates that Π_0 is safe on synchronization of rule $r : \langle u, v\rangle \to \langle u', v'\rangle$.

7 Prospects

In this paper we have defined a basic model of Colonies of Synchronizing Agents, however several enhancements to this are already in prospect. Primary among these is the addition of *space* to the colony. Precisely, each agent will have a triple of co-ordinates corresponding to its position in Euclidean space and the rules will be similarly endowed with the ability to modify an agent's position. A further extension of this idea is to give each agent an *orientation*, i.e. a rotation relative to the spatial axes, which may also be modified by the application of rules.

The idea is to make the application of a rule dependent on either an absolute position (thus directly simulating a chemical gradient) or on the relative distance between agents in the case of synchronization. Moreover, in the case of the application of a synchronization rule, the ensuing translation and rotation of the two agents may be defined *relative to each other*. In this way it will be possible to simulate reaction-diffusion effects, movement and local environments.

Some additional biologically-inspired primitives are also planned, such as agent *division* (one agent becomes two) and agent *death* (deletion from the colony) (as, for instance, done in [5]). These primitives can simulate, for example, the effects of mitosis, apoptosis and morphogenesis. In combination with the existing primitives, it will be possible (and is planned) to model, for example, many aspects of the complex multi-scale behaviour of the immune system.

With the addition of the features just mentioned, it will also be interesting to extend the investigation and proofs given above to identify further classes of CSAs demonstrating robustness and having decidable properties. Moreover, we plan to investigate classes of CSAs where model-checking based on the presented temporal logic can be efficiently implemented, e.g., having an associated temporal structure that can be algorithmically constructed in efficient time and space. We would also like to extend the investigation to design efficient algorithms that implement distributed computations, as used, for instance, in the area of amorphous computing [1].

References

[1] H. Abelson, D. Allen, D, Coore, C. Hanson, G. Homsy, T.F. Knight, R. Nagpal, E. Rauch, G.J. Sussman, R. Weiss, Amorphous Computing, *Communications of the ACM*, 43, 5, 2000.

[2] A. Alhazov, Minimizing Evolution-Communication P Systems and EC P Automata, *New Generation Computing*, 22, 4, 2004.

[3] M. Ben-Ari, A. Pnueli, and Z. Manna, The Temporal Logic of Branching Time, *Acta Informatica*, 20, 1983.

[4] F. Bernardini, R. Brijder, G. Rozenberg, C. Zandron, Multiset-Based Self-Assembly of Graphs, *Fundamenta Informaticae*, 75, 2007.

[5] F. Bernardini, M. Gheorghe, Population P Systems, *Journal of Universal Computer Science*, 10, 5, 2004.

[6] F. Bernardini, M. Gheorghe, Cell Communication in Tissue P systems: Universality Results, *Soft Computing*, 9, 9, 2005.

[7] M. Cavaliere, Evolution-Communication P Systems, *Proceedings of Workshop on Membrane Computing 2002*, LNCS 2597, 2003.

[8] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[9] A. Ilachinski, *Cellular Automata - A Discrete Universe*, World Scientific Publishing, 2001.

[10] R. Freund, Gh. Păun, O.H. Ibarra, H.-C.Yen, Matrix Languages, Register Machines, Vector Addition Systems, *Proc. Third Brainstorming on Membrane Computing*, Sevilla, 2005, RGCN Report 01/2005.

[11] S. Greibach, Remarks on Blind and Partially Blind One-Way Multicounter Machines. *Theoretical Computer Science*, 7, 3, 1978.

[12] D. Hauschildt, M. Jantzen, Petri Net Algorithms in the Theory of Matrix Grammars, *Acta Informatica*, 31, 1994.

[13] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[14] J. Kelemen, A. Kelemenová, A Grammar-Theoretic Treatment of Multiagent Systems, *Cybernetics and Systems*, 23,6, 1992.

[15] J. Kelemen, A. Kelemenová, Gh. Păun, Preview of P Colonies - A Biochemically Inspired Computing Model, *Proceedings of Workshop on Artificial Chemistry*, ALIFE9, Boston, USA, 2004.

[16] J. Kelemen, Gh. Păun, Robustness of Decentralized Knowledge Systems: A Grammar-Theoretic Point of View, *Journal Expt. Theor. Artificial Intelligence*, 12, 2000.

[17] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodriguez-Patón, Tissue P Systems, *Theoretical Computer Science*, 296, 2, 2003.

[18] Gh. Păun, *Membrane Computing - An Introduction*, Springer-Verlag, Berlin, 2002.

[19] Gh. Păun, Introduction to Membrane Computing, in *Applications of Membrane Computing*, G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds., Springer-Verlag, Berlin, 2006.

[20] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.

[21] J. Van Benthem, Temporal logic, in *Handbook of Logic in Artificial Intelligence and Logic Programming: Epistemic and Temporal reasoning*, Oxford University Press, 1995.

[22] http://nusmv.irst.itc.it/

[23] http://psystems.disco.unimib.it