



The Microsoft Research - University of Trento
Centre for Computational
and Systems Biology

Technical Report CoSBI 11/2007

Colonies of Synchronizing Agents

Matteo Cavaliere

*The Microsoft Research-University of Trento Centre for Computational and Systems Biology,
Trento, Italy*

cavaliere@cosbi.eu

Radu Mardare

*The Microsoft Research-University of Trento Centre for Computational and Systems Biology,
Trento, Italy*

mardare@cosbi.eu

Sean Sedwards

*The Microsoft Research-University of Trento Centre for Computational and Systems Biology,
Trento, Italy*

sedwards@cosbi.eu

*This is the preliminary version of a paper that will appear in
In Proceedings of International Workshop Automata for Cellular and Molecular Computing,
Computer and Automation Research Institute of the Hungarian Academy of Sciences, 2007.*

Colonies of Synchronizing Agents

Matteo Cavaliere, Radu Mardare, Sean Sedwards

The Microsoft Research-University of Trento, Italy

September 30, 2009

Abstract

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

The proposed model is based on a multiset of agents present 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 asynchronously or in synchrony with another agent.

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

1 Motivations

Inspired by the *intracellular* and *intercellular* mechanisms present 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 molecular characteristics) with some of the objects classified as terminals (e.g., corresponding to chemicals 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.

In this paper we consider CSAs as generative computing devices and investigate CSAs considering various trade-offs between the power of the evolution rules (intracellular mechanisms) and the power of the synchronizing rules (intercellular mechanisms). We consider CSAs working in a maximal parallel way (all agents are update synchronously) modeling the fact that “in biological systems, if something can happen then it must happen”. However from a biological (and a mathematical) point of view it is rather attractive to also 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 the one of finite sets of vectors to the one of Turing machines, by varying the power of evolution rules and synchronization rules. Moreover, an “intermediate” class of CSAs equivalent to partially blind counter machines (hence, not universal) is presented.

After investigating the computational power of CSAs, we study the *robustness* of colonies by considering the 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 any agent has the ability to apply a synchronization whenever it needs: CSAs for which such property is true are called *safe on synchronization* of rules.

The presented model has similarities and, crucially, differences with other models inspired by cell-tissues investigated in the area of membrane computing (a.k.a. P systems, [13]). Specifically, it 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 change their contents only by using very restricted rewriting rules. 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 the agents is allowed.

Our model has similarities with population P systems [4] which is a class of tissue P systems [12] where links may exist between agents and these can be modified by means of a set of bond making rules. Agents can change their contents by means of local (non-cooperative) rewriting rules and hence different types of agents can have different sets of rules. It is possible to move objects between the agents using the bonds and agents may communicate with an environment that has unbounded resources. Computation is implemented in two different phases: local rewriting plus bond making rules, applied in an alternate manner. There are also rules for dividing and merging agents. The main differences with population P systems and the model we propose are that we do not have

explicit bonds between agents (in a sense our agents are linked by a complete graph), rewriting in our case is general (i.e., it can be cooperative) for both evolution and synchronization rules and our agents do not have 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 [3], however in that case a graph is constructed from an initial seed using multiset-based aggregation rules to enlarge the structure, there is no internal rewriting of the agent contents and there is no synchronization between the agents.

Our approach is also distinct from cellular automata [7] where cells exist on a regular grid, where each cell has a finite number of possible states and where cells react to or 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 states of a cell may be infinite. Although our initial definition does not include an explicit description of space, the extension we propose includes agents located at arbitrary positions and with the potential to interact with any other agent in the system.

2 Preliminaries

We briefly recall the basic theoretical notions used in this 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.

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 $V = \{a_1, a_2, \dots, a_n\}$ is $Ps_V(x) = (|x|_{a_1}, |x|_{a_2}, \dots, |x|_{a_n})$. 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.

Then, 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 $M : V \mapsto \mathbb{N}$, 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 $\text{supp}(M) = \{a \in V \mid M(a) > 0\}$. A multiset with finite support is usually presented as set of pairs $(x, M(x))$, for $x \in \text{supp}(M)$.

The *cardinality* of a multiset M is denoted by $\text{card}(M)$ and it indicates the number of objects in a multiset. It is defined in the following way. $\text{card}(M)$ is infinite if M has infinite support. If M has finite support then $\text{card}(M) = \sum_{a_i \in \text{supp}(M)} M(a_i)$ (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 $\mathbb{M}_k(V) = \{M \mid M \in \mathbb{M}(V), \text{card}(M) = k\}$ and $\mathbb{M}_{\leq k}(V) = \{M \mid M \in \mathbb{M}(V), \text{card}(M) \leq k\}$.

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 $M = \{(a_1, M(a_1)), (a_2, M(a_2)), \dots, (a_n, M(a_n))\}$ then the string $w = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)}$ (and all its permutations) precisely identify 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 $\{(a, 1), (b, 2)\}$, that can 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 $(A_1 \mapsto x_1, \dots, A_n \mapsto x_n)$, $n \geq 1$ (with $A_i \in N, x_i \in (N \cup T)^*$ in all cases), and F is a set of occurrences of rules in M .

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \mapsto x_1, \dots, A_n \mapsto x_n)$ 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 $A_i \mapsto x_i$ 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 \implies is denoted by \implies^* . Then the language generated by G is $L(G) = \{w \in T^* \mid S \implies^* w\}$.

In other words, the language $L(G)$ is composed by 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., [6]).

Theorem 2.1

- $CF \subset MAT \subset RE$.
- $MAT \subset MAT_{ac} = RE$.
- Each language $L \in MAT, L \subseteq a^*$ is regular.

For the last assertions and from the definitions given earlier we obtain that:

Theorem 2.2

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

Matrix grammars without appearance checking are equivalent to *partially blind counter machines* (introduced in [9]). 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 [8]).

From this last assertion and using results in ([9]) we obtain the following corollaries of interest for this paper.

Corollary 2.1

(Emptiness)

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

(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., M and M' , with terminal alphabet T , it is undecidable whether or not $Ps_T(L(M)) \subseteq Ps_T(L(M'))$ or whether or not $Ps_T(L(M)) = Ps_T(L(M'))$.

From Theorem 2.1 and using the fact that containment of regular languages is decidable ([11]) we obtain the following result.

Theorem 2.3 (Containment, Equivalence)

Given two arbitrary matrix grammars without a.c., M and M' it is decidable whether or not $N(L(M')) \subseteq N(L(M))$ and whether or not $N(L(M)) = N(L(M'))$.

We now recall some basic definitions and results from P systems (membrane systems). An introductory guide to P systems (with the notions needed for this paper) is [14]: a preprint can be found at the web-page [20]. Readers already familiar with the basic notions and results of the area can skip this part of the paper.

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

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

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 $1, 2, \dots, m$;
- w_i , $1 \leq i \leq m$, are strings that represent multisets over O associated with regions $1, 2, \dots, m$ of μ ;
- R_i , $1 \leq i \leq m$, are finite sets of evolution rules over O ; R_i is associated with region i of μ ; an evolution rule is of the form $u \mapsto v$, where u is a string over O and v is a string over $\{a_{\text{here}}, a_{\text{out}} \mid a \in O\} \cup \{a_{\text{in}_j} \mid a \in O, 1 \leq j \leq m\}$.
- $i_0 \in \{0, 1, 2, \dots, m\}$; if $i_0 \in \{1, \dots, m\}$ 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 $u \mapsto v$ 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 $r : u \mapsto v \in R_i$, $i \in \{1, \dots, m\}$ can be applied in region i by assigning (if possible) to γ_r the occurrences of the objects in u taken from region i .

The application of an occurrence of the evolution rule $u \mapsto v$ in a region i means to remove the multiset of objects u from region i , and to add the multiset v , to the regions specified by 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 *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 $u \mapsto v$ cannot be applied).

A *transition* between configurations is executed using the evolution rules in a *non-deterministic maximally parallel* manner at each step (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 done, there are insufficient occurrences of the objects for any further occurrence 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 non-deterministic way.

In this way, at each step all the multisets of objects associated to the regions of the system evolve and the system passes from one configuration to another; this passage is called a *transition*.

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 *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, then 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 specified before. 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 $NO P_1(\alpha)$ and $PsOP_1(\alpha)$, respectively.

The following results are known (see, e.g., [13]).

Theorem 2.4

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

We recall also the definition of evolution-communication P systems introduced in ([5]), that join two basic models of membrane systems, that with evolution rules and symbol-objects and that with symport/antiport rules (see, e.g, [13]).

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

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

where:

- O is an alphabet of objects;
- μ is a membranes structure with m membranes hierarchically arranged in a tree structure; the membranes (and hence the regions they delimit) are injectively labeled with $1, 2, \dots, m$;
- w_i are strings which represent multisets over O associated with the regions $1, 2, \dots, m$ of μ ;
- $R_i, 1 \leq i \leq m$, 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 $u \mapsto v$, 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 ‘here’ as 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 weight of the rule. For an antiport rule $(x, in; y, out)$ the weight is the $\max\{|x|, |y|\}$.
- $i_0 \in \{0, 1, 2, \dots, m\}$ is the label of a region which is designed as the output region; if $i_0 = 0$ then the output region is the environment.

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 occurrence of the *symport rule* (x, in) to membrane i consists in moving the occurrences of the objects in x , from the region (or from the environment) surrounding the region i to region i . If an occurrence of the symport rule (x, out) is applied to membrane i , then the occurrences of the objects in x are moved out 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 occurrence of the *antiport rule* $(x, in; y, out)$ is applied to membrane i , then 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. Occurrences of the rules from R_i are applied to occurrences of objects in region i while the application of the occurrences 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*): their occurrences are chosen and applied in the *non-deterministic maximally parallel manner*, whose meaning has been already explained in the P systems with symbol-objects.

The system starts from the initial configuration and continues to move from a configuration to a new one by applying the above described transitions: this sequence of transitions from a configuration to a new one is called *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 successful *computation* of Π (or simply, computation of Π) and the number of occurrences of objects contained in the output region i_0 in the halting configuration is the result produced by 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 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, [5, 1]).

Theorem 2.5

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

3 Colonies of Synchronizing Agents

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

A *Colony of Synchronizing Agents* (a CSA) of degree m is a construct $\Pi = (A, T, C, R)$.

- 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 (with $card(C) = m$) of agents.

Formally, C is a multiset over the set of all possible agents over A and 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 $u \rightarrow v$, with $u \in A^+$ and $v \in A^*$.

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

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

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

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

An occurrence γ of a synchronization rule $r : \langle u, v \rangle \rightarrow \langle u', v' \rangle$ can be applied to the pair of agents w and w' by: (i) taking from w a multiset of objects u (hence, $u \subseteq w$) and *assigning* it to γ ; (ii) taking from w' a multiset of objects v (hence, $v \subseteq w'$) and *assigning* it to γ .

The application of an occurrence of rule r to the agents w and w' consists in: removing the multiset u from w and then adding, to the obtained multiset, the multiset u' ; removing the multiset v from w' and then adding, to the obtained multiset, the multiset v' .

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 $|u'| \leq |u| \leq 1$ and $|v'| \leq |v| \leq 1$.

We assume the existence of a *global clock* which marks the time-units for all the agents present in the system.

A *configuration* of a CSA, Π , consists of the agents present in the system, at a given time-unit.

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 $\mathbb{M}(H)$ with $H = \mathbb{M}(A)$.

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

Maximal-parallel mode (in short *mp*): A single *maximal-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 *maximal parallel and non-deterministic* way. This means that, for each agent w and each pair of agents w', w'' , present in the configuration c , the occurrences of the objects present in w, w', w'' are assigned to occurrences of the rules, with the occurrences of the objects and the occurrences 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 occurrences of the rules is done there is no occurrence 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 occurrence 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 agent w and each pair of agents w', w'' , present in the configuration c , the occurrences of the objects of w, w', w'' are *either* assigned to occurrences of the rules, with the occurrences of the objects and the occurrences of the rules chosen in a non-deterministic way, *or* left unassigned. A single occurrence of an object can only be assigned to a single occurrence of a rule.

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

A sequence (possibly infinite) $\langle C_0, C_1, \dots, C_i, C_{i+1}, \dots \rangle$ of configurations of Π where C_{i+1} is obtained from C_i , $i \geq 0$, by a γ -transition is called γ -*evolution* of Π , with $\gamma \in \{asyn, mp\}$. A configuration c of Π present in a γ -evolution of Π is told to be *reachable* by 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 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 successful γ -computation, or, because there is no confusion, we simply say 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 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 $\{Ps_T(w) \mid w \text{ is an agent present in } C_h\}$.

Collecting all these 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 $\{|w| \mid w \text{ is an agent present in } C_h\}$.

Again, collecting all these 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 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.

Hence, we denote by $PsCSA_m^\gamma(\alpha, \beta)$ (and $NCSA_m^\gamma(\alpha, \beta)$) where $\alpha \in \{coo_e, ncoo_e, un_e\}$, $\beta \in \{coo_s, ncoo_s, un_s\}$ and $\gamma \in \{mp, asyn\}$ the family of sets of vectors (of sets of numbers, resp.) generated by CSAs using γ -computations, evolution rules of type α , communication rules of type β and at most m agents in the initial configurations (α or β are removed if the corresponding rules are not used). If m is substituted by $*$ then the number of agents considered in the initial configurations is arbitrary.

Example 3.1 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 $R = \{r_1 : abca \mapsto ba, r_2 : \langle abc, cc \rangle \mapsto \langle aa, cb \rangle\}$.

The application of the evolution rule r_1 to the configuration C is shown raphically in Figure 1. The application of the synchronization rule r_2 to the configuration C is shown in Figure 2.

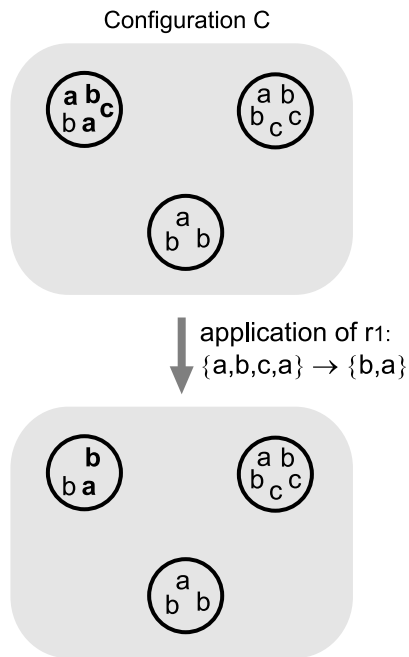


Figure 1: Application of an occurrence of the evolution rule r_1 to C from Example 3.1.

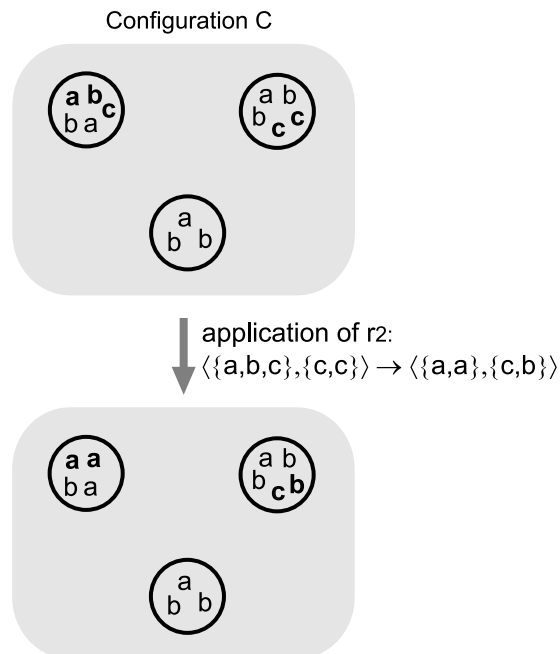


Figure 2: Application of an occurrence of the synchronization rule r_2 to C from Example 3.1.

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

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.

Also 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 or rules) 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 occurrence of rule r is used ...”.

4 Computational Power of CSAs

From the definitions of CSAs and invoking the Turing-Church thesis.

Theorem 4.1

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

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 4.2

$$PsCSA_1^{mp}(coo_e) = PsCSA_1^{mp}(coo_s) = PsRE.$$

Proof The proofs are straightforward hence we only give a short sketch. For each P system Π with symbol-objects, one membrane, cooperative evolution rules, working in maximal-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 . 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 does not use rules with the target indication ‘here’: any evolution rule used in Π 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

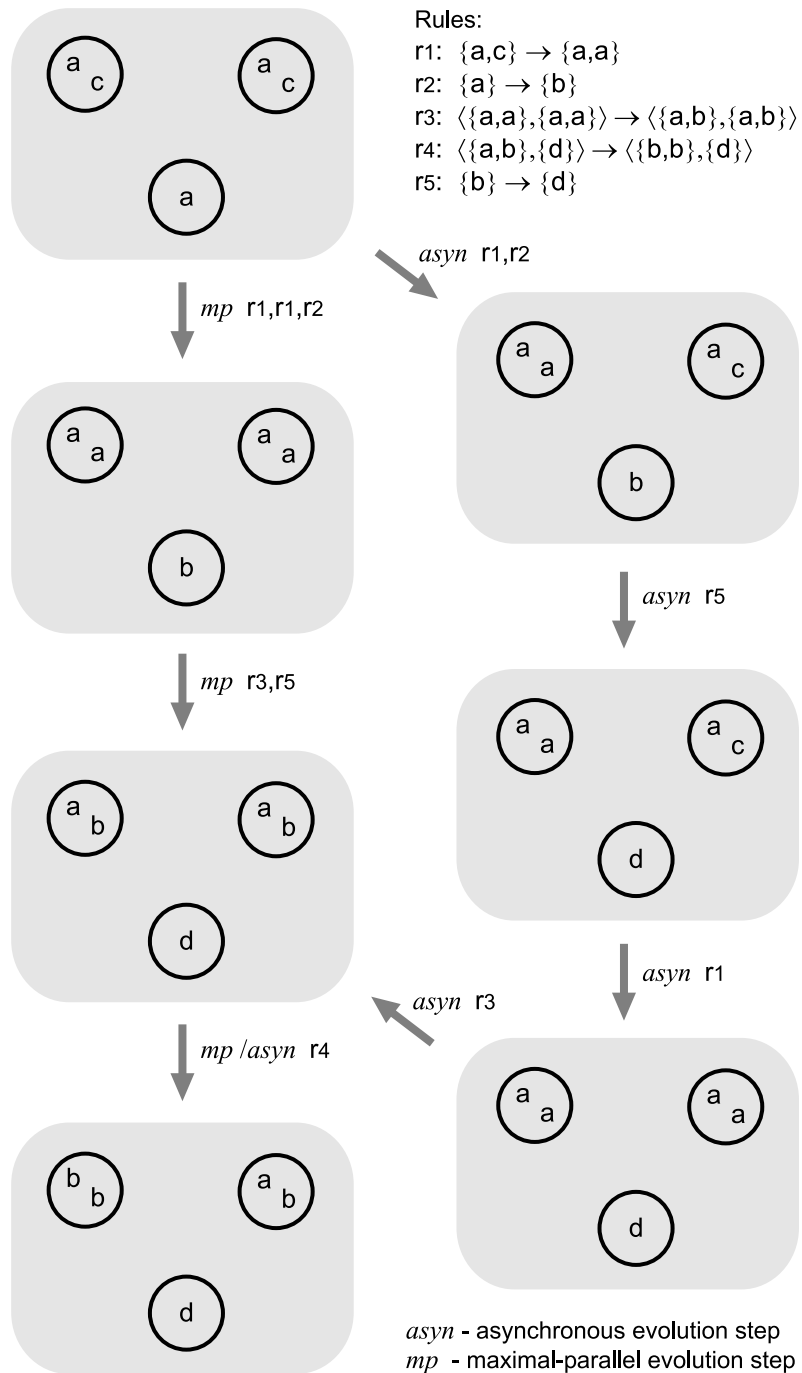


Figure 3: Alternative maximal-parallel and asynchronous evolutions of a CSA.

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'' .

The Theorems follow from the fact that P systems with symbol-objects, cooperative evolution rules, one membrane and working in the maximal-parallel mode are known to be computational complete (Theorem 2.4). \square

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

Theorem 4.3

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

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 $card(C) = m$ (i.e., C consists of m agents w_1, w_2, \dots, w_m with $w_i \in A^*$ for $i \in \{1, \dots, m\}$). We construct the sets $A_i = \{a_i \mid a \in A\}$.

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

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

We define $N = \{S\} \cup A_1 \cup A_2 \cup \dots \cup A_m$.

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 $(S \rightarrow h_1(w_1)h_2(w_2) \dots h_m(w_m))$.

Group II

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

Group III

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

The basic idea of the simulation is that the matrices in group I are used to start a derivation of G by creating the strings corresponding to the initial configuration of Π (distinguishing the objects of the different agents by using appropriate 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 $\{w'_1, w'_2, \dots, w'_m\}$ 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 occurrence of an evolution (synchronization) rule of R to one (to a pair, resp.) of the agents is simulated in G by applying the corresponding matrix from group II (or from group III , resp.). A transition of the evolution e that is obtained by applying several occurrences of rules from R to the agents is simulated in G by applying, in sequence, the corresponding matrices from groups II and III .

It is also true the reverse. If there is a string w in $L(G)$ then it must be (for the way G functions) of the type $h_1(w_1)h_2(w_2) \cdots h_m(w_m)$ with $w_1, w_2, \dots, w_m \in A^*$. And for the way G has been constructed, if there is a derivation d in G that produces the string $h_1(w_1)h_2(w_2) \cdots h_m(w_m)$, then there is an asynchronous evolution of Π starting from the initial configuration C and reaching the configuration $\{w_1, w_2, \dots, w_m\}$. In fact, Π works in the asynchronous mode, and then, in particular, can have evolutions composed by sequential transitions (i.e., with only one occurrence of one rule executed) that can directly simulate the derivation d of the grammar G .

From the language $L(G)$ we select by an opportune regular intersection the language L' of all the strings corresponding to halting configurations reached by asynchronous computations of Π . This can be clearly done by intersecting the language $L(G)$ with a regular sets 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 get then $L' = L(G) \cap R_h$. The language L' can be still generated by a matrix grammar without appearance checking since matrix grammars without a.c. are closed under regular intersection ([6]).

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

$$\begin{aligned} d_i(a_i) &= a_i, a \in T. \\ d_i(a_i) &= \lambda, a \in (A - T). \\ d_i(a_j) &= \lambda, a \in A, j \neq i. \end{aligned}$$

For each $i \in \{1, \dots, m\}$, the language $d_i(L')$ contains all the agents with objects indexed by i and that are present in the halting configurations reached by asyn-computations of Π . Moreover from each agent the objects not in T are deleted.

We are interested in collecting all the agents present in the halting configurations, considering all the computations of Π .

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

The language L'' is a language over T and can be still obtained by 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 ([6]).

For the explained construction 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 only cooperative evolution rules) can simulate a matrix grammar $G = (N, T, S, M)$ without appearance checking in a straightforward way.

Without loss of generality to make simpler we suppose that M has m matrices (labelled by $1, \dots, m$) each one with k productions (labelled by $1 \dots, k$). We also suppose, again with no loss of generality, that the only production that rewrites S is the first production of matrix 1.

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

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 $i : (a_1 \mapsto u_1, a_2 \mapsto u_2, \dots, a_k \mapsto u_k)$ in M with $a_1, a_2, \dots, a_k \in N$ and $u_1, u_2, \dots, u_k \in (N \cup T)^*$ with $i \in \{1, \dots, m\}$, we add to R the following cooperative evolution rules

$\{(m_i, m_1)a_1 \mapsto u_1(m_i, m_2)x, (m_i, m_2)a_2 \mapsto u_2(m_i, m_3), \dots, (m_i, m_{k-1})a_{k-1} \mapsto (m_i, m_k)u_{k-1}\} \cup \{x(m_i, m_k)a_k \mapsto u_k(m_j, m_1) \mid 1 \leq j \leq m\} \cup \{x \mapsto x\} \cup \{a \mapsto a \mid a \in N\}$.

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 an halting configuration $\{(m_j, m_1)w\}$ for some $1 \leq j \leq m$ is reached.

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

Precisely, a transition of c obtained by applying one occurrence of one rule is simulated by a single (opportune) matrix application in G ; a transition of c obtained by applying several occurrences of the rules is simulated by using an (opportune) sequences of matrices in G .

In any evolution of Π , the object x is present in the configuration of Π from the moment the simulation of a matrix is started until the moment the simulation is terminated. This object, together with rules $x \mapsto x$ in R guarantees that when an evolution of Π halts, then the simulation of a matrix of G has been completed (i.e., no ‘‘partial’’ simulation of a matrix can be been done).

The rules $\{a \mapsto a \mid a \in N\}$ in R guarantees that when an evolution of Π halts in the halting configuration $\{(m_j, m_1)w\}$ for some $1 \leq j \leq m$ then no objects from N is present in w .

From the above description, it is easy to see that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$.

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

From Theorem 4.3 and from the last statement of Theorem 2.2 we obtain.

Corollary 4.1

$$NCSA_*^{asyn}(coo_e, coo_s) = NREG.$$

Using Theorem 4.2 we obtain:

Corollary 4.2

$$\begin{aligned}
NCSA_*^{asy}(coo_e, coo_s) \subset NCSA_1^{mp}(coo_e, coo_s) &= NCSA_1^{mp}(coo_e) \\
&= NCSA_1^{mp}(coo_s). \\
PsCSA_*^{asy}(coo_e, coo_s) \subset PsCSA_1^{mp}(coo_e, coo_s) &= PsCSA_1^{mp}(coo_e) \\
&= PsCSA_1^{mp}(coo_s).
\end{aligned}$$

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 in maximal parallel mode.

Theorem 4.4 $PsCSA_*^{asy}(un_e, un_s) = PsCSA_*^{mp}(un_e, un_s) = PsFIN$.

Proof In CSAs using only unary rules the lengths 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). \square

However, by only combining unary synchronization rules and non-cooperative evolution rules we obtain computational completeness for CSAs working in the maximal parallel way and two agents. The proof of this is by simulation of EC P systems, described in the Preliminaries.

Theorem 4.5 $PsCSA_2^{mp}(ncoo_e, un_s) = PsRE$.

Proof In [1] 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 $Ps_T(L(G)) = Ps(\Pi)$. Then, the fact that programmed grammars with appearance checking are computationally complete (see e.g., [6]), 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 maximal 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 [1]) an EC P system $\Pi = (O, [[]_1, w_1, w_2, R_1, R_2, R'_1, R'_2,]]_2, T)$, with $T \subseteq O$, such that $Ps_T(L(G)) = Ps(\Pi)$. The EC P system Π is constructed in such a way that an output produced by Π at the end of a computation consists of objects corresponding to the terminals T collected in the environment at the end of the computation. 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 (with the symport rules associated to membrane 1 used only to send to the environment these objects and no other antiport or symport rules associated to membrane 1).

We then construct a CSA $\Pi' = (A, T', C, R)$ as follows.

We first 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, where the index denotes the region of Π where the object is present.

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

We set $A = \{h_1(a), h_2(a) \mid a \in O\}$, $C = \{h_1(w_1), h_2(w_2)\}$. 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 $a \rightarrow v$ in R_i , $i \in \{1, 2\}$, add to R the rule $h_i(a) \rightarrow h_i(v)$.

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

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

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

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.

For the way Π' is constructed it can be easily 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 $v = P_{S_T}(w)$), then there exists a computation for Π' having, in the halting configuration, the agents $h_1(ww')$, $h_2(w'')$ with $w'' \in O^*$, $w \in T^*$, $w' \in (O - T)^*$ (i.e., the output of the computation are then the two vectors $v = P_{S_{T'}}(h_1(ww')) = P_{S_{T'}}(h_1(w))$ and $P_{S_{T'}}(h_2(w'')) = \bar{0}$). 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, in the halting configuration, the agents $h_1(ww')$ and $h_2(w'')$ with $w' \in O^*$, $w \in T^*$ and $w'' \in O^*$ (i.e., the output are the vectors $v = P_{S_{T'}}(h_1(ww')) = P_{S_{T'}}(h_1(w))$ and $P_{S_{T'}}(h_2(w'')) = \bar{0}$) then 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 $v = P_{S_T}(w)$).

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 these rules are not used the computational power of CSAs is only regular (in terms of Parikh images).

Theorem 4.6 $PsCSA_*^{mp}(ncoo_e) = PsCF$.

Proof For an arbitrary CSA, $\Pi = (A, T, C, R)$, with m agents w_1, w_2, \dots, w_m (no bound on m) there exists a P system with symbol-objects Π' and non-cooperative evolution rules working in maximal 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 by 1. We add to R_1 the following rules: $\{S \rightarrow w_1, S \rightarrow w_2, \dots, S \rightarrow w_m\}$ 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 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 maximal-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 2.4, the result follows. \square

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 employed in [16] in the framework of grammar systems, but 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., 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., as if they are 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 the core behaviour)?

Example 5.1 *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, maximal-parallel computations) are 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 $\{\langle ab, bc \rangle \mapsto \langle eff, eff \rangle, \langle ab, bd \rangle \mapsto \langle eff, eff \rangle\}$.

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

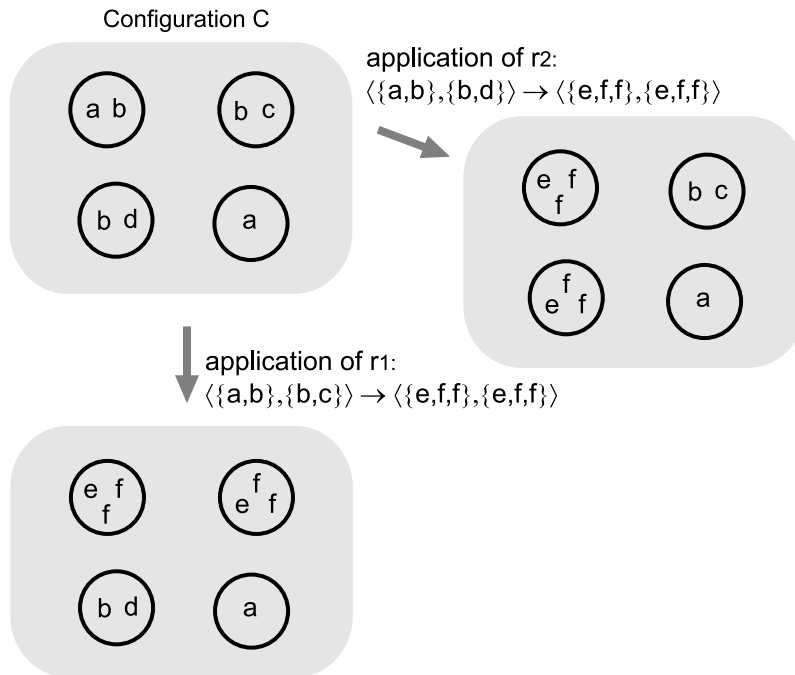


Figure 4: The two possible asynchronous computations of Π of Example 5.1

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 $Ps_T^{asyn}(\Pi) = \{(1, 2)\} \cup \{(1, 2)\} \cup \bar{0} = \{(1, 2)\} \cup \bar{0}$. In fact, we have in the two halting configurations, the 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 with $PsREG$).

The system Π is robust when an occurrence of 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 $Ps_T^{asyn}(\Pi') = \{(1, 2)\} \cup \bar{0}$ so still contains the defined core behavior. The unique computation of Π' is represented in Figure 5.

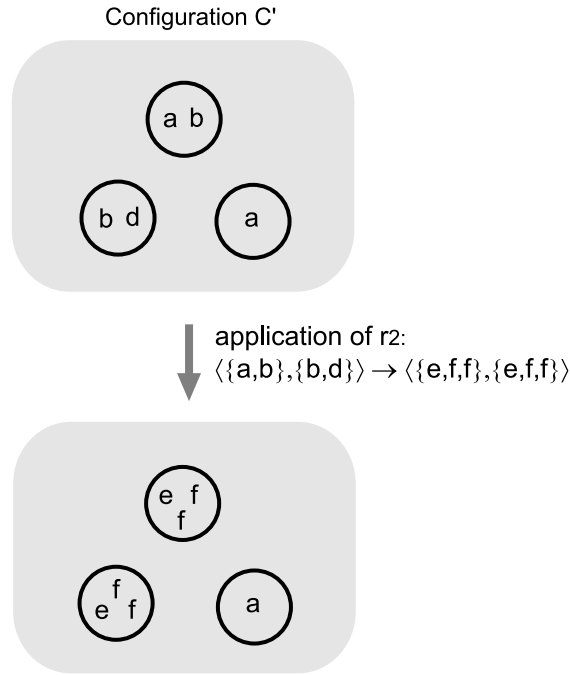


Figure 5: Robust behaviour of Π' of Example 5.1 when agent bc is removed from C .

On the other hand, the system Π is not robust when an occurrence of ab is deleted from its initial configuration. In fact, if we consider $\Pi'' = (A, T, C'', R)$ with $C'' = \{(bd, 1), (a, 1)\}$ we have that $Ps_T^{asyn}(\Pi'') = \bar{0}$ so it does not contain the core behavior. The system Π'' is represented in Figure 6. The unique computation of Π'' is the one that halts in the initial configuration C'' .

We first analyse the case of rule-restrictions with asynchronous evolution and get the following negative result.

Theorem 5.1 *It is undecidable whether or not for an arbitrary CSA, Π , with arbitrary terminal alphabet T , arbitrary rule restriction Π' of Π and arbitrary set S from $PsREG_T$, $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$.*

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 2.1).

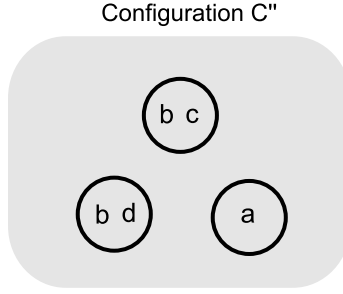


Figure 6: No robustness displayed by Π'' of Example 5.1 when agent ab is removed from C .

Without loss of generality to make it simpler we suppose that M has m matrices (labelled by $1, 2, \dots, m$) of k productions (labelled by $1, 2, \dots, k$) and M' has m' matrices (labelled by $1, 2, \dots, m'$) of k' productions (labelled by $1, 2, \dots, k'$). 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.

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

As done in the second part of Theorem 4.3 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 $i : (a_1 \mapsto u_1, a_2 \mapsto u_2, \dots, a_k \mapsto u_k)$ in M with $a_1, a_2, \dots, a_k \in N$ and $u_1, u_2, \dots, u_k \in (N \cup T)^*$ with $i \in \{1, \dots, m\}$, we add to R the following cooperative evolution rules.

$$\{(m_i, m_1)a_1 \mapsto u_1(m_i, m_2)x, (m_i, m_2)a_2 \mapsto u_2(m_i, m_3), \dots, (m_i, m_{k-1})a_{k-1} \mapsto (m_i, m_k)u_{k-1}\} \cup \{x(m_i, m_k)a_k \mapsto u_k(m_j, m_1) \mid 1 \leq j \leq m\} \cup \{a \mapsto a \mid a \in N\} \cup \{x \mapsto x\}.$$

Using the same arguments as in Theorem 4.3 it is possible to see that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$.

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

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

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

Again, using the same arguments as in Theorem 4.3, 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 $N \cap N' = \{S\} \cup T$.

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. Therefore by contradiction, the Theorem follows. \square

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

Theorem 5.2 *It is decidable whether or not, for an arbitrary CSA, Π , with arbitrary 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., M , with terminal alphabet T' , such that $Ps_T^{asyn}(\Pi) = Ps_{T'}(L(M))$ (Theorem 4.3) and the membership problem for matrix grammars without a.c. is decidable (see, e.g., [6]).
- Given a vector v , there is only a finite set of strings (over T) whose Parikh vector with respect to T is exactly v .

\square

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 5.3 *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 matrix grammar without a.c., M , such that $N^{asyn}(\Pi) = N(L(M))$ (Theorem 4.3). So there is also a matrix grammar without a.c., M' , such that $N^{asyn}(\Pi) \cap S = N(L(M'))$ (Corollary 2.1). The result then follows from the fact that containment for length sets of languages generated by matrix grammars without a.c. is decidable (Theorem 2.3). \square

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

Theorem 5.4 *It is decidable whether or not, for an arbitrary CSA, Π from $CSA_*(ncoo_e)$, with arbitrary 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 $CSA_*(ncoo_e)$ and 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 4.6 and Theorem 2.2). 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 follows from the fact that containment is decidable for regular languages (see, e.g., [11]) 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' . \square

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 the rule and the system will be robust against such deletion.

Theorem 5.5 *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 occurrence 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 proof of Theorem 4.3, 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 Theorem 4.3 as follows. It is added a matrix 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 not the empty set (Corollary 1). \square

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

Theorem 5.6 *It is undecidable whether or not, for an arbitrary CSA, Π , with arbitrary 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 2.1).

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

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

As done in the second part of Theorem 4.3 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 $i : (a_1 \mapsto u_1, a_2 \mapsto u_2, \dots, a_k \mapsto u_k)$ in M with $a_1, a_2, \dots, a_k \in N$ and $u_1, u_2, \dots, u_k \in (N \cup T)^*$ with $i \in \{1, \dots, m\}$, we add to R the following cooperative evolution rules

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

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

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

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

Using the same arguments as in Theorem 4.3 we have that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$.

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

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

Then we construct the CSA $\Pi'' = (A, T, C + C' - \{(S, 1)\}, R)$.

We have that $Ps_T^{asyn}(\Pi'') = Ps_T^{asyn}(\Pi) \cup Ps_T^{asyn}(\Pi') = Ps_T(L(G)) \cup Ps_T(L(G'))$.

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 answer is yes, then $Ps_T^{asyn}(\Pi) \subseteq Ps_T^{asyn}(\Pi')$, while if the answer is not, then $Ps_T^{asyn}(\Pi) \not\subseteq Ps_T^{asyn}(\Pi')$. So we can decide whether or not $Ps_T(L(G)) \subseteq Ps_T(L(G'))$ and this is not possible (Corollary 2.1). From this, by contradiction, the Theorem follows. \square

Using the same ideas as in Theorem 5.2 we get the following result.

Theorem 5.7 *It is decidable whether or not, for an arbitrary CSA, Π , with arbitrary terminal alphabet T , arbitrary agent restriction Π' of Π and arbitrary finite set S from $PsREG_T$, $Ps_T^{asy^n}(\Pi) \cap S \subseteq Ps_T^{asy^n}(\Pi')$.*

Using the same ideas as in Theorem 5.3 we get the following result.

Theorem 5.8 *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., [11]). So this is already shown to be true for CSAs with non-cooperative evolution rules, unary synchronization rules and working in the maximal parallel mode.

Therefore, from Theorem 4.5, we have the following result.

Theorem 5.9 *It is undecidable whether or not, for an arbitrary CSA, Π , from $CSA_*(coo_e, un_s)$ with arbitrary 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 5.10 *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 5.11 *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 [2, 18].

Temporal logics are the most used logics in model-checking analysis: efficient algorithms and tools having already been developed for them, e.g. NuSMV [19]. 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, 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.

In what follows we denote by $CSA_m^{A,T,R}$ the class of all CSAs having the alphabet A , terminal alphabet T , set of rules R over A and degree m . When we say an arbitrary class $CSA_m^{A,T,R}$ we mean the class of all CSAs having alphabet A , terminal alphabet T , set of rules R over A , degree m , with A , T , R and m arbitrarily chosen.

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

- if $r \in R$ is the evolution rule $u \rightarrow v$ then $prec(r) = \{(u, 1)\}$.
- if $r \in R$ is a synchronization rule $\langle u, v \rangle \rightarrow \langle u', v' \rangle$ then $prec(r) = \{(u, 1)\} + \{(v, 1)\}$.

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

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.

Definition 6.2 (γ -complete evolutions) *Given an arbitrary class $\mathcal{C} = CSA_m^{A,T,R}$, a sequence of CSAs $\langle \Pi_0, \Pi_1, \Pi_2, \dots, \Pi_i, \dots \rangle$ with $\Pi_i = (A, T, C_i, R) \in CSA_m^{A,T,R}$, $i \geq 0$, is called γ -complete evolution in \mathcal{C} starting in Π_0 if $\langle C_0, C_1, C_2, \dots, C_i, \dots \rangle$, $i \geq 0$, 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 $e = \langle \Pi_0, \Pi_1, \dots, \Pi_i, \Pi_{i+1}, \dots \rangle$ be an arbitrary γ -complete evolution in \mathcal{C} starting in Π_0 . We call $\langle \Pi_i, \Pi_{i+1}, \dots \rangle$, $i \geq 0$, an i -suffix evolution¹ of e and we denote it by e_i .

Definition 6.3 (Syntax of $\mathcal{L}_{\mathcal{C}}$) *Let $\mathcal{C} = CSA_m^{A,T,R}$ be an arbitrary class. The set $AP(\mathcal{C})$ is defined by:*

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

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 $\mathcal{L}_{\mathcal{C}}$.

¹Observe 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} .

- 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 union of configuration formulas and evolution formulas of $\mathcal{L}_{\mathcal{C}}$ is called the *language of $\mathcal{L}_{\mathcal{C}}$* .

Definition 6.4 (Temporal structures) *Given an arbitrary class $\mathcal{C} = CSA_m^{A,T,R}$, 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 $\{\Pi_1, \Pi_2, \dots \mid \langle \Pi_0, \Pi_1, \Pi_2, \dots \rangle \in E_{\mathcal{C}}^{\gamma}(\Pi_0)\} \subseteq \mathcal{S}$.
- $\mathfrak{R} \subseteq \mathcal{S} \times \mathcal{S}$ such that $(\Pi_1, \Pi_2) \in \mathfrak{R}$ iff there exists $\langle \Pi_1, \Pi_2, \dots \rangle \in E_{\mathcal{C}}^{\gamma}(\Pi_1)$.

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

Definition 6.5 (CSA-Semantics) *Let $\mathcal{C} = CSA_m^{A,T,R}$ be an arbitrary class and $\mathcal{T}_{\mathcal{C}}^{\gamma} = (\mathcal{S}, \mathfrak{R})$ 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 $w \subseteq w'$.
- $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models w_1 \oplus w_2 \oplus \dots \oplus w_i$ for $w_j \in prec(R) \cup \{\top\}$, $1 \leq j \leq i$ iff $C_{\Pi} = C_1 + C_2 + \dots + C_i$ and $w_j \subseteq C_j$ for $w_j \neq \top$, $1 \leq j \leq i$.
- $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \phi \wedge \psi$ 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 $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi$.
- $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \top$ always.
- $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi$ iff $\mathcal{T}_{\mathcal{C}}^{\gamma}, \Pi \models \phi$, with $e \in E_{\mathcal{C}}^{\gamma}(\Pi)$ and ϕ a configuration formula
- $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi \wedge \psi$ iff $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi$ and $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \psi$.
- $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \neg\phi$ iff $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \not\models \phi$.
- $\mathcal{T}_{\mathcal{C}}^{\gamma}, e \models \phi U\psi$ 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 ones.

- $\mathcal{T}_C^\gamma, \Pi \models A\phi$ iff for any $e \in E_C^\gamma(\Pi)$ we have $\mathcal{T}_C^\gamma, e \models \phi$.
- $\mathcal{T}_C^\gamma, e \models F\phi$ iff there exists $i \geq 0$ such that $\mathcal{T}_C^\gamma, e_i \models \phi$.
- $\mathcal{T}_C^\gamma, e \models G\phi$ iff for any $i \geq 0$ we have $\mathcal{T}_C^\gamma, e_i \models \phi$.

Theorem 6.1 (Decidability) *Given an arbitrary class $\mathcal{C} = CSA_m^{A,T,R}$, the satisfiability, validity and model-checking problems for \mathcal{L}_C against the CSA-semantics are decidable.*

Proof The result derives from the fact that CTL logic is decidable (see, e.g., [18, 2]) and from the fact that $AP(\mathcal{C})$, the set of atomic formulas, is a finite set. \square

To show the potentiality of the introduced logic we give a small example of properties that can be specified. We pose the natural 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 $r : \langle u, v \rangle \rightarrow \langle u', v' \rangle$ 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 with $u \subseteq w_1$ 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.

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

Indeed, consider a CSA Π_0 taken from an arbitrary class of CSAs \mathcal{C} . If we consider the introduced CSA-semantics we have that:

- $\mathcal{T}_C^\gamma, \Pi_0 \models AG((u \oplus \top) \rightarrow (u \oplus v \oplus \top))$
- iff for any $e \in E_C^\gamma(\Pi_0)$ we have $\mathcal{T}_C^\gamma, e \models G((u \oplus \top) \rightarrow (u \oplus v \oplus \top))$
- iff for any $e = \langle \Pi_0, \Pi_1, \dots, \Pi_i, \dots \rangle \in E_C^\gamma(\Pi_0)$ and any $i \geq 0$ we have
- $\mathcal{T}_C^\gamma, \Pi_i \models (u \oplus \top) \rightarrow (u \oplus v \oplus \top)$.

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, $\mathcal{T}_C^\gamma, \Pi_i \models u \oplus \top$ iff $C_{\Pi_i} = C_1 + C_2$ and $u \subseteq C_1$, i.e., the configuration of Π_i contains an agent w_1 such that $u \subseteq w_1$.

And, similarly, $\mathcal{T}_C^\gamma, \Pi_i \models u \oplus v \oplus \top$ iff $C_{\Pi_i} = C'_1 + C'_2 + C'_3$ and $u \subseteq C'_1$, $v \subseteq C'_2$, i.e., the configuration of Π_i contains two agents w'_1, w'_2 such that $u \subseteq w'_1$ and $v \subseteq w'_2$, that exactly indicates that the CSA Π_0 is safe on synchronization of rule $r : \langle u, v \rangle \rightarrow \langle u', v' \rangle$.

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 agents 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 and movement.

Some additional biologically-inspired primitives are also planned, such as agent *division* (one agent becomes two) and agent *death* (deletion from the system). These primitives can simulate, for example, the effects of mitosis and apoptosis in biological cells and morphogenesis in general.

With the addition of the features just mentioned, it will be interesting to extend the investigation and proofs given above to individuate classes of CSA demonstrating robustness and having decidable properties.

References

- [1] A. Alhazov, Minimizing Evolution-Communication P Systems and EC P Automata, *New Generation Computing*, 22, 4, 2004.
- [2] M. Ben-Ari, A. Pnueli, and Z. Manna, The Temporal Logic of Branching Time, *Acta Informatica*, 20, 1983.
- [3] F. Bernardini, R. Brijder, G. Rozenberg, C. Zandron, Multiset-Based Self-Assembly of Graphs, *Fundamenta Informaticae*, 75, 2007.
- [4] F. Bernardini, M. Gheorghe, Population P Systems, *Journal of Universal Computer Science*, 10, 5, 2004.
- [5] M. Cavaliere, Evolution-Communication P Systems, *Proceedings of Workshop on Membrane Computing 2002*, LNCS 2597, 2003.
- [6] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
- [7] A. Ilachinski, *Cellular Automata - A Discrete Universe*, World Scientific Publishing, 2001.
- [8] 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.
- [9] S. Greibach, Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7, 3, 1978.
- [10] D. Hauschildt, M. Jantzen, Petri Net Algorithms in the Theory of Matrix Grammars, *Acta Informatica*, 31, 1994.

- [11] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
- [12] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón, Tissue P Systems, *Theoretical Computer Science*, 296, 2, 2003.
- [13] Gh. Păun, *Membrane Computing - An Introduction*, Springer-Verlag, Berlin, 2002.
- [14] Gh. Păun, Introduction to Membrane Computing, in *Applications of Membrane Computing*, G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds., Springer-Verlag, Berlin, 2006.
- [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] G. Rozenberg, A. Salomaa, eds., in *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.
- [18] J. Van Benthem, Temporal logic, in *Handbook of Logic in Artificial Intelligence and Logic Programming: Epistemic and Temporal reasoning*, Oxford University Press, 1995.
- [19] <http://nusmv.irst.itc.it/>
- [20] <http://psystems.disco.unimib.it>