Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 35 - 51, 2007.

Colonies of Synchronizing Agents: An Abstract Model of Intracellular and Intercellular Processes

Matteo Cavaliere, Radu Mardare, and Sean Sedwards

Microsoft Research – University of Trento
Centre for Computational & Systems Biology
{cavaliere, mardare, sedwards}@cosbi.eu

Abstract

We present a modelling framework and computational paradigm called Colonies of Synchronizing Agents (CSAs), which abstracts intracellular and intercellular mechanisms of biological tissues. The model is based on a multiset of agents (cells) in a common environment. Each agent has a local contents, stored in the form of a multiset of atomic objects, updated by multiset rewriting rules which may act on individual agents (intracellular action) or synchronize the contents of pairs of agents (intercellular action). Using tools from formal language and temporal logic we investigate dynamic properties of CSAs, including robustness and safety of synchronization. We also identify classes of CSAs where such dynamic properties can be algorithmically decided.

1 Motivations

Inspired by biological tissues and populations of cells, 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 complex biological systems in the context of formal language theory and multiset rewriting.

The model is based on a population of *agents* (e.g., corresponding to *cells* or *molecules*) in a common environment, able to modify their contents and to synchronize with other agents in the same environment. Each agent has a contents represented by a multiset of atomic objects (e.g., corresponding to chemical compounds or the characteristics of individual molecules) with some of the objects classified as terminals (e.g., corresponding to properties or chemicals visible to an external observer). An agent's contents may be modified independently of other agents by means of multiset rewriting rules (called *internal rules*)[1] which can mimic chemistry or other types of *intracellular mechanisms.* Moreover, the agents can influence each other by synchronously changing their contents using pairwise *synchronization rules.*

[1]In [5] internal rules are called evolution rules, adopting a standard terminology from the P systems area. We prefer here a more general term.

This models, in a deliberately abstract way, the various signalling mechanisms and *intercellular mechanisms* present in biological systems. The rules are global, so all agents obey the same rules: the only feature which may distinguish the agents is their contents. Evolutions of CSAs are defined as sequences of transitions obtained by applying the rules to the agents. These transitions thus mark the passage of the system from one configuration to another.

In this paper we search for classes of CSAs where relevant *dynamic properties* can be algorithmically checked. We interpret CSAs as computational devices and can thus study CSAs by applying tools from classical fields of computer science, such as formal language, automata theory and temporal logic. For this reason we define as computations of CSAs the evolutions that reach halting configurations, i.e. configurations where the contents of the agents can no longer be changed because no rules may be applied. This situation can be interpreted as a particular kind of steady state of the system. We are interested in the configuration of the colony when a halting condition is reached and we may take the precise contents of the agents as the output (the result) produced by the CSA. Alternatively, we can use the magnitude of the agents (the total amount of contents irrespective of composition) in the halting configuration as the result produced by a CSA.

We can then investigate the *robustness of CSAs* by considering the ability of a CSA to generate a particular *core result* despite the failure (i.e., removal) of some of the agents or rules. The core result can be seen as a specific configuration in which the colony must be when the system halts. We show that for an arbitrary CSA, robustness cannot be algorithmically decided when the core result is represented by specific contents of agents, while it *can* be algorithmically decided in an efficient way when the core result is represented by agents' magnitudes.

In Section 4 we are interested in dynamic properties concerning the application of the rules. To check these properties we propose a decidable temporal logic. We show that the proposed logic can be used to specify and check whether or not, during any evolution of a CSA, an agent can apply a synchronization whenever it needs (if it can we say that the agent is safe on synchronization). We conclude the present section by comparing our model with other models based on abstractions of cell tissues which use rewriting and multisets.

The introduced model of Colonies of Synchronizing Agents has similarities and significant differences with other models inspired by cell tissues investigated, for instance, in the area of membrane computing (a.k.a. P systems, [15]). Specifically, it can be considered a generalization of P colonies [11], which is also based on interacting agents but has agents with limited contents (two objects) which change by means of restricted rewriting rules. Moreover, in P colonies no direct communication between agents is allowed.

Our model also has similarities with population P systems [3], which is a class of tissue P systems [13] where links may exist between agents and these can be modified by means of a set of bond making rules.

The main differences with population P systems is that in our case agents do not have types; rules are global and only the agents' contents differentiate them. This latter characteristic makes CSAs similar to the model of self-assembly of graphs presented in [2], however in that case; (*i*) a graph is constructed from an initial seed using multiset-based aggregation rules to enlarge the structure, (*ii*) there is no internal rewriting of the agent's contents and (*iii*) there is no synchronization between the agents.

Another computational formalism widely used to simulate and model biological tissues is cellular automata (CAs, e.g., see [19]). In particular, CAs have been used to model the immune systems (e.g., [14]). In CAs, 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 model, because of the multiset-based contents and because of the arbitrary multiset rewriting rules, the possible different states of a cell may be infinite. Although the initial definition of CSAs does not include an explicit description of space, the extensions we propose include agents located at arbitrary positions and with the potential to interact with any other agent in the colony.

A specific limitation of cellular automata that use synchronous update is that many such models are computational complete (i.e., equivalent to Turing machines [19]), even when employing *simple* rules (e.g., rule 110, [19]). This makes it impossible to algorithmically analyse such systems. Precisely, non-trivial problems are undecidable for Turing machines.

2 Formal Language Preliminaries

This Section is a brief introduction to some basic notions of formal language theory needed in the paper. Further information regarding formal language and automata theory is available from the many monographs in this area, starting with [10, 4] and ending with the handbook [17].

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 $|w|_a$.

Each subset of V^* is called a *language*.

The boolean operations (with languages) of union and intersection are denoted $\cup$ and $\cap$, respectively. Concatenation of the languages L_1, L_2 is $L_1L_2 = \{xy \mid x \in L_1, y \in L_2\}$.

A generative grammar is a finite device generating in a well-specified sense the strings of a language. Chomsky grammars are particular cases of rewriting systems where the operation used in processing the strings is the rewriting (replacement of a substring of the processed string by another substring). A (Chomsky) grammar is a quadruple $G = (N, T, S, P)$ where N and T are disjoint alphabets, N being a set of non-terminals and T a set of terminals, S is the axiom and P is a finite set of productions (rewriting rules). A production is usually written in the form $r : u \to v$ with $u \in (N \cup T)^*$ with u containing at least a non-terminal (so, it cannot be the empty string).

For $x, y \in (N \cup T)^*$ we write $x \Longrightarrow y$ iff $x = x_1 u x_2, y = x_1 v x_2$ for some $x_1, x_2 \in (N \cup T)^*$ and $u \to v \in P$. One says that x directly derives y. The language generated by G denoted by $L(G)$ is defined by $L(G) = \{x \in T^* \mid S \Longrightarrow^* x\}$, where $\Longrightarrow^*$ denotes the reflexive and transitive closure of $\Longrightarrow$. So the language $L(G)$ consists of all terminal strings that can be obtained starting from S by applying iteratively the productions in P.

A grammar is called *regular* if each production is of the form $a \to v$ with $a \in N$ and $v \in T \cup TN \cup \{\lambda\}$. A grammar is called *context-free* if each production is of the form $a \to v$ with $a \in N$.

Languages generated by context-free and regular grammars are called context-free and regular languages, respectively. We denote by CF and REG the families of context-free and regular languages, respectively. Regular languages are those accepted by finite state automata.

In general, when we want to specify a terminal alphabet we add a subscript to the name of the family; e.g., REG_A is the family of all regular languages over the alphabet A.

A matrix grammar without appearance checking is a devices with matrices of context-free productions and where productions are applied according to the order given in the chosen matrix (for details see [6]).

Formally, a *matrix grammar without appearance checking* (in short, without a.c.) is a construct $G = (N, T, S, M)$, 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 productions of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$ (with $A_i \in N, x_i \in (N \cup T)^*$ in all cases).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots A_n \to x_n)$ in M and strings $w_i \in (N \cup T)^*$, $1 \leq i \leq n+1$, such that $w = w_1, z = w_n + 1$ and, for all $1 \leq i \leq n$, $w_i = w_i' A_i w_i''$, $w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$. The reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$. Then the language generated by G is $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$.

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

For a language $L \subseteq V^*$, the set *length*$(L) = \{|x| \; |x \in L\}\}$ 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 (i.e., it is a family of sets of natural numbers). For instance, $NREG$ is the family of length sets of regular languages.

The *Parikh vector* associated with a string $x \in V^*$ with respect to the alphabet $V = \{a_1, a_2, \ldots, a_n\}$ is $Ps_V(x) = (|x|_{a_1}, |x|_{a_2}, \ldots, |x|_{a_n})$. For $L \subseteq V^*$ we define $Ps_V(L) = \{Ps_V(x) | x \in L\}$. This is called the *Parikh image* of the language L. The null vector is denoted by $\overline{0}$.

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

For instance, $PsREG$ is the family of Parikh images of regular languages in REG.

For instance, $V = \{a, b, c\}$ is an alphabet, $x = aaabbbcaa = a^3b^3ca^2$ is a string over V, $L = \{a^nb^n \mid n \geq 1\}$ is a language over V. We have $|x| = 9$, $|x|_a = 5$, $length(L) = \{2n \mid n \geq 1\}$. The Parikh vector of x with respect to V is $Ps_V(x) = (5, 3, 1)$ and for the language L we have $Ps_V(L) = \{(n, n, 0) \mid n \geq 1\}$.

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

For instance $M = \{a, b, b, b\}$, also written as $\{(a, 1), (b, 3)\}$, is a multiset with $M(a) = 1$ and $M(b) = 3$.

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 vice versa) while $(M - M')$ is obtained by *removing* M' from M.

For example, given the multisets $M = \{a, b, b, b\}$ and $M' = \{b, b\}$, we can say that M' is included in M, that $(M + M') = \{a, b, b, b, b, b\}$ and that $(M - M') = \{a, b\}$.

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

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

We denote by $\mathbb{M}(V)$ the set of all possible multisets over V and by $\mathbb{M}_k(V)$ the set of all multisets over V having cardinality k.

For the case that the alphabet V is finite we can use a compact string notation to denote multisets: if $M = \{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_n, M(a_n))\}$ then the string $w = a_1^{M(a_1)} a_2^{M(a_2)} \cdots a_n^{M(a_n)}$ (and all its permutations) precisely 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 $\{b, a, b\} = \{(a, 1), (b, 2)\}$ which has cardinality 3. The empty multiset is represented by the empty string, λ.

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 set 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 of agents, with $card(C) = m$. [2]
• R is a finite set of *rules* over A. We have *internal rules* of type $u \to v$, with $u \in A^+$ and $v \in A^*$, and *synchronization rules* of the type $\langle u, v\rangle \to \langle u', v'\rangle$ with $uv \in A^+$ and $u', v' \in A^*$.

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

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

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

A *configuration* of a CSA, Π, consists of the agents present in the colony at a given time. We denote by $\mathbb{C}(\Pi)$ the set of *all possible configurations* of Π. Therefore, using the notation introduced in Section 1, $\mathbb{C}(\Pi)$ is exactly $\mathbb{M}_m(H)$ with $H = \mathbb{M}(A)$.

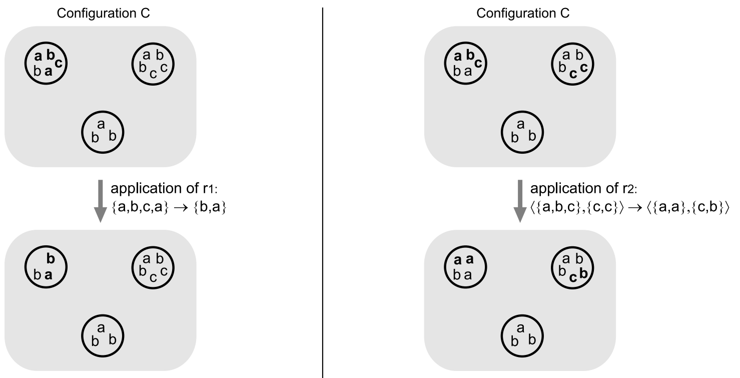
A single *asynchronous transition* (in short, *asyn*-transition) [3] of Π from an arbitrary configuration c of Π to the next one lasts exactly one time unit and is obtained by applying the rules in the set R to the agents present in c in an *asynchronous* way. This means that, for each agent w and each pair of agents w' and w'' present in c, the occurrences of the objects of w, w' and 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 may only be assigned to a single occurrence of a rule. In other words, in an *asyn*-transition any number of occurrences of rules (zero, one, or more) can be applied to the agents in the configuration c.

A sequence (possibly infinite) $\langle C_0, C_1, \cdots, C_i, C_{i+1}, \cdots\rangle$ of configurations of Π, where C_{i+1} is obtained from C_i, $i \geq 0$, by an *asyn*-transition is called an *asyn-evolution* of Π. An *asyn*-evolution of Π 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 occurrences of rules from R can be applied.

An *asyn*-evolution of Π that is halting and that *starts with the initial configuration* of Π is called an *asyn-computation* of Π. The *result/output* of an asyn-computation is the *set of vectors of natural numbers*, one vector for each agent w present in the halting configuration with the vector describing the multiplicities of terminal objects present in w. More formally, the result of an asyn-computation

[2]Formally, C is a multiset of degree m over the set of all possible agents over A. Hence, $C \in \mathbb{M}_m(\mathbb{M}(A))$.

[3]We specify *asyn*-transitions to distinguish them from the synchronous maximal parallel transitions often adopted in models coming from P systems and cellular automata.


Configuration C
application of r1: {a,b,c,a} → {b,a}
Configuration C
application of r2: ⟨{a,b,c},{c,c}⟩ → ⟨{a,a},{c,b}⟩

(a) Internal rule r_1 applied to C | (b) Synchronization rule r_2 applied to C

Figure 1: Alternative application of rules r_1 and r_2 to configuration C from Example 1.

which stops in the halting configuration C_h is the set of vectors of natural numbers $\{Ps_T(w) \mid w$ is an agent present in $C_h\}$.

Because of the non-determinism in applying the rules, several possible asyn-computations of Π may exist. Taking the union of all the results for all possible asyn-computations of Π, we get the *set of vectors generated* by Π, denoted by $Ps_T^{asyn}(\Pi)$.

We may also consider the total number of objects comprising the agent (the agent's *magnitude*), without considering the internal composition. In this case the *result* of an asyn-computation is the *set of natural numbers*, one number for each agent w present in the halting configuration and each number being the length of w. More formally, in this case the result of an asyn-computation that stops in the halting configuration C_h is then the set of numbers $\{|w| \mid w$ is an agent present in $C_h\}$. Again, taking the union of all the results for all possible asyn-computations of Π, we get the *set of numbers generated* by Π, denoted by $N^{asyn}(\Pi)$.

In what follows we indicate by C_Π the initial configuration of Π.

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

$\Pi = (A, T, C, R)$ with $A = \{a, b, c\}$, $T = \{a\}$, $C = \{(abcba, 1), (abbcc, 1), (bab, 1)\}$ $= \{abcba, abbcc, bab\}$.

The rules $R = \{r_1 : abca \to ba,\ r_2 : \langle abc, cc\rangle \to \langle aa, cb\rangle\}$.

The application of an occurrence of internal rule r_1 to the agent *abcba* in the configuration C is shown diagrammatically in Figure 1(a).

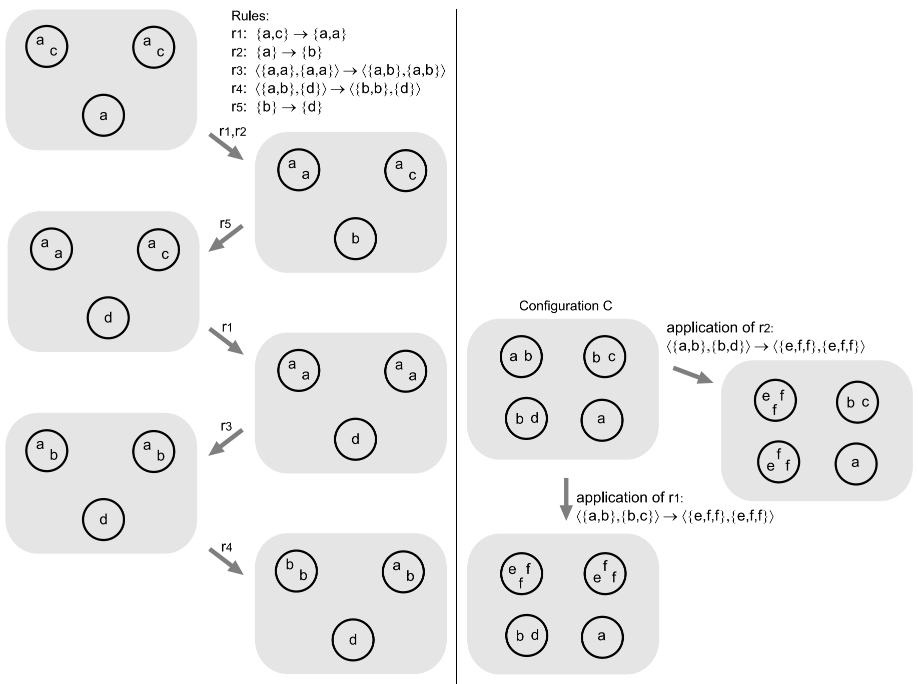
The application of an occurrence of the synchronization rule r_2 to the pair of agents *abcba* and *abbcc* in the configuration C is shown diagrammatically in Figure 1(b).

A more complex example of part of an asynchronous evolution is presented in Figure 2(a): $\Pi' = (A', T', C', R')$ with the initial configuration $C' = \{(ac, 2), (a, 1)\}$ and rules $R' = \{ac \to aa,\ a \to b,\ \langle aa, aa\rangle \to \langle ab, ab\rangle,\ \langle ab, d\rangle \to \langle bb, d\rangle,\ b \to d\}$.

In the next Example we show how the output/result produced by a CSA is obtained.

Example 2 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 $\{r_1 : \langle ab, bc\rangle \rightarrow \langle eff, eff\rangle, r_2 : \langle ab, bd\rangle \rightarrow \langle eff, eff\rangle\}$.


Rules:
r1: {a,c} → {a,a}
r2: {a} → {b}
r3: ⟨{a,a},{a,a}⟩ → ⟨{a,b},{a,b}⟩
r4: ⟨{a,b},{d}⟩ → ⟨{b,b},{d}⟩
r5: {b} → {d}
r1,r2
r5
r1
r3
r4
Configuration C
application of r2:
⟨{a,b},{b,d}⟩ → ⟨{e,f,f},{e,f,f}⟩
application of r1:
⟨{a,b},{b,c}⟩ → ⟨{e,f,f},{e,f,f}⟩

(a) Asynchronous evolutions of Π' of Example 1

(b) The two possible asynchronous computations of Π of Example 2

Figure 2: Asynchronous evolutions and computations.

There are *only two possible asynchronous computations* of Π and these are represented diagrammatically in Figure 2(b).

We have that $Ps_T^{asyn}(\Pi) = \{(1, 2), \overline{0}\}$.

In fact, we have two possible halting configurations (for the two computations). In the first halting configuration we have the agent (in two copies) eff whose associated Parikh vector (with respect to T) is $(1, 2)$ and the agents bd and a, whose associated Parikh vectors (with respect to T) are null vectors $\overline{0}$ (these agents do not contain any terminal object from T). Then the result of this computation is the set of vectors $\{(1, 2)\} \cup \{(1, 2)\} \cup \{\overline{0}\} \cup \{\overline{0}\} = \{(1, 2), \overline{0}\}$ with each vector describing the multiplicities of the terminal objects in the agents in the halting configuration.

In the second halting configuration we have the agent (in two copies) eff whose associated Parikh vector (with respect to T) is $(1, 2)$ and the agents bc and a, whose associated Parikh vectors (with respect to T) are null vectors. Then, also in this case, the result of the computation is the set of vectors $\{(1, 2), \overline{0}\}$

Taking the union of the results for the (two) possible computations we get $Ps_T^{asyn}(\Pi) = \{(1, 2), \overline{0}\} \cup \{(1, 2), \overline{0}\} = \{(1, 2), \overline{0}\}$.

We can also collect the result in terms of magnitude (size) of the agents present in the halting configurations, thus collecting $N^{asyn}(\Pi)$. In this case we obtain

$N^{asyn}(\Pi) = \{3, 2, 1\}$. In fact, in the two halting configurations we have agents of size $3, 2$ and 1 (counting their objects). Then for both computations the result is the set of numbers $\{3, 3, 2, 1\} = \{3, 2, 1\}$ with each number being the magnitude of an agent in the halting configuration.

Taking the union of the results for the (two) possible computations we obtain $N^{asyn}(\Pi) = \{3, 2, 1\} \cup \{3, 2, 1\} = \{3, 2, 1\}$.

4 Dynamic Properties of CSAs

The goal of this Section is to investigate dynamic properties of CSAs, in particular robustness and safety on synchronization. We try to individuate classes of CSAs where such properties can be checked with algorithms and for this we employ tools from formal language theory and from temporal logic. Because of lack of space we omit the proofs, however complete proofs of all the results can be found in the technical report [5].

4.1 Robustness of CSAs

Before investigating robustness of CSAs we state the result that CSAs are equivalent (in terms of Parikh images) to matrix grammars without a.c. (hence to partially blind counter machines, [9]).

In particular, for an arbitrary CSA, $\Pi = (A, T, C, R)$, there exists a matrix grammar without a.c., G, with terminal alphabet T, such that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$, and vice-versa. Matrices can indeed simulate the application of the rules of the CSA because the rules are applied in an asynchronous manner. On the other hand, a CSA with a single agent can simulate a matrix grammar. The detailed proof of the result can be found in [5] (Theorem 8).

Theorem 3 *For an arbitrary CSA, Π, with terminal alphabet T, there exists a matrix grammar without a.c., G, with terminal alphabet T such that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$ and vice versa.*

We are now ready to define and to investigate robustness of CSAs against perturbations of some of the features of the colony. For this purpose we use a similar idea of robustness as employed in [12] in the framework of grammar systems, adapted here to the proposed CSAs. *We want to investigate situations where either some of the agents (i.e., cells) or some of the rules (i.e., intra or intercellular actions) of the colony do not function.* We would like to know the consequences to the result of the colony. We will investigate CSAs that are robust, e.g. where the produced result does not change critically if one or more agents cease to exist in the colony or if one or more rules stop working. As discussed in the Motivations, this can model the fact that CSAs always stop in a “correct steady state”, independently of agents or rules failure.

We can formalize these notions in the following way.

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 are absent from the colony.

We 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, as if, once again, they are absent from the colony.

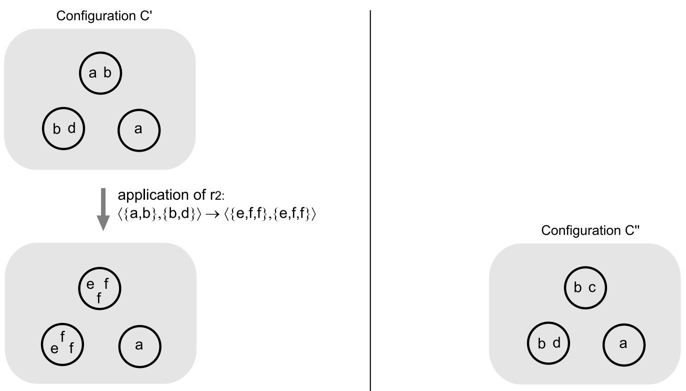
We say that a CSA, Π, is *robust* when a *core result*, i.e., the minimally acceptable result, is preserved when considering proper restrictions of it. Formally, by a core result of Π we mean *part* of the result produced by Π, hence a subset of the set of vectors generated by Π. We define these subsets by making an intersection with a regular set of vectors taken from $PsREG$. The intersection selects the regular property of the core result we are interested in. Note that the core result may be infinite.

Questions about robustness can then be formalized as follows.

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 check whether or not $Ps^{asyn}(\Pi) \cap S \subseteq Ps^{asyn}(\Pi')$, i.e., whether Π is robust against the restriction Π' in the sense that Π' will continue to generate at least the core result defined by the intersection of $Ps^{asyn}(\Pi)$ and S)?

Example 4 We produce a small example that clarifies the introduced notion of robustness in the case of agent-restriction, considering as core result specific contents of the agents (the other cases are similar).

Consider Π given in Example 2. Suppose we fix as core result the set of vectors $\{(1, 2)\}$, where it can be clearly obtained by intersection of $Ps_T^{asyn}(\Pi)$ and $\{(1, 2)\}$. 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 $Ps_T^{asyn}(\Pi') = \{(1, 2), \bar{0}\}$, which still contains the defined core result. The single computation of Π' is represented in Figure 3(a).



(a) Π': agent bc removed from C | (b) Π'': agent ab removed from C

Figure 3: The robustness and lack of robustness of (a) Π' and (b) Π'' from Example 4 when agents bc and ab, respectively, are removed from C.

On the other hand, 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 $C'' = \{(bd, 1), (bc, 1), (a, 1)\}$ we have that $Ps_T^{asyn}(\Pi'') = \{\bar{0}\}$, which does not contain the core result. The single computation of Π'', i.e., the one halting in the initial configuration (no rule can be applied), is represented in Figure 3(b).

We now analyse the case of agent-restrictions, producing a negative result.

Theorem 5 *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* $PsREG_T$*,* $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$*.*

The proof of the above Theorem is based on Theorem 3 and that, given two arbitrary matrix grammars without a.c. M and M', it is undecidable whether or not $L(M) \subseteq L(M')$ (see, e.g., [6], [8] and [9]).

Informally, Theorem 5 says that there is no algorithm to check whether or not a CSA is robust against arbitrary deletion of agents from the initial configuration. This result depends critically on the fact that the core result corresponds to a specific internal contents that the agents must have in the halting configurations. In fact, when we consider *weaker* core results we can get a positive result. For instance, suppose we take as core result a specific *magnitude* that the agents must have in the halting configurations. This means that we collect, for a CSA Π the set of numbers $N^{asyn}(\Pi)$. In this case the robustness problem can be rephrased in the following manner.

Consider an arbitrary CSA, Π, with an arbitrary agent- or rule-restriction Π' of Π and an arbitrary set S from $NREG$. Is it possible to decide whether or not $N^{asyn}(\Pi) \cap S \subseteq N^{asyn}(\Pi')$, i.e., whether Π is robust against the restriction Π' such that Π' can still generate at least the core result defined by the intersection $N^{asyn}(\Pi) \cap S$? Based on the fact that every language over a one letter alphabet produced by a matrix grammar without a.c. is regular (see [6]), on the equality of Theorem 3 we obtain the following corollary.

Corollary 1 *For an arbitrary CSA,* Π*, there exists a regular language* L *such that* $N^{asyn}(\Pi) = NL$ *and vice versa.*

Because containment of regular languages is algorithmically decidable (see, e.g., [10]), we obtain the following result.

Theorem 6 *It is decidable whether or not, for an arbitrary CSA,* Π*, arbitrary agent restriction* Π' *of* Π *and arbitrary set* S *from* $NREG$*,* $N^{asyn}(\Pi) \cap S \subseteq N^{asyn}(\Pi')$*.*

Informally, the above result says that it is possible to check in an efficient way whether or not a CSA is robust against arbitrary deletion of agents, subject to the core result being defined in terms of magnitudes of agents.

We can also investigate the case when rule-restrictions are considered and we obtain similar results. With a similar idea to that of Theorem 5, we obtain the following negative result.

Theorem 7 *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 $PsREG_T$, $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$.*

However, using the same ideas as those in Theorem 6 we get a positive result.

Theorem 8 *It is decidable whether or not, for an arbitrary CSA, Π, arbitrary rule restriction Π' of Π and arbitrary set S from $NREG$, $N(\Pi) \cap S \subseteq N(\Pi')$.*

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

Theorem 9 *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.*

The proof is based on the result stated by Theorem 3 and on the fact that membership and emptiness for matrix grammars without a.c. can be algorithmically decided ([6]). The idea of the proof is to reduce the problem to decide if the language produced by a matrix grammar without a.c. is the empty one.

4.2 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 [1, 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 [20]. They are devised with operators for expressing and quantifying on possible evolutions or configurations of systems. For instance, for an arbitrary system it is possible to specify properties such as *'for any possible evolution, ϕ is fulfilled'*, *'there exists an evolution such that ϕ is not true'*, *'in the next state ϕ will be satisfied'*, *'eventually ϕ will be satisfied'* and *'ϕ happens until ψ is satisfied'*, with ϕ and ψ properties of the system. We show how to use these operators to formally specify and verify complex properties of CSAs, such as *'the agent will always eventually reach a certain configuration'*, or *'rule r is not applicable until rule r' is used'*, etc.

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.

Definition 4.1 (Preconditions) *Let A be an arbitrary alphabet and R an arbitrary set of rules over A. We define the mapping $prec : R \to 2^{\mathbb{M}(A)}$ by*

- *if $r \in R$ is the evolution rule $u \to v$ then $prec(r) = \{u\}$.*

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

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

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

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

Definition 4.2 (*asyn*-complete evolutions) *A sequence of CSAs $\langle \Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_i, \ldots\rangle$ with $\Pi_i = (A, T, C_i, R) \in \mathcal{C}$, $i \geq 0$, is called* asyn-complete evolution *in $\mathcal{C}$ starting in Π_0 if $\langle C_0, C_1, C_2, \ldots C_i, \ldots\rangle$, $i \geq 0$, is a halting or an infinite asyn-evolution of Π_0.*

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

Let $e = \langle \Pi_0, \Pi_1, \ldots, \Pi_i, \Pi_{i+1} \ldots\rangle$ be an arbitrary asyn-complete evolution in $\mathcal{C}$ starting in Π_0. We call $\langle \Pi_i, \Pi_{i+1}, \ldots\rangle$, $i \geq 0$, an i-suffix evolution[4] *of e and we denote it by e_i.*

Definition 4.3 (Syntax of $\mathcal{L}_{\mathcal{C}}$) *The set $AP(\mathcal{C})$ is defined by:*

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

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

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

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

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

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

Definition 4.4 (Temporal structures) *We define the structure $\mathcal{T}_{\mathcal{C}}^{asyn} = (\mathcal{S}, \mathfrak{R})$ as follows:*

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

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

Definition 4.5 (CSA-Semantics) *Let $\mathcal{T}_{\mathcal{C}}^{asyn} = (\mathcal{S}, \mathfrak{R})$ be a temporal structure in $\mathcal{C}$. For an arbitrary $\Pi \in \mathcal{S}$, an arbitrary $e \in E_{\mathcal{C}}^{asyn}(\Pi)$ and an arbitrary formula ϕ from the language of $\mathcal{L}_{\mathcal{C}}$, we define coinductively the satisfiability relations $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi$ and $\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi$ by:*
$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \top$ always.
$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models w$ for $w \in prec(R)$ iff $C_\Pi = \{(w', 1)\}$ and $w \subseteq w'$.
$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models w_1 \oplus w_2 \oplus \ldots \oplus w_i$ for $w_j \in prec(R) \cup \{\top\}, 1 \leq j \leq i$ iff $C_\Pi = C_1 + C_2 + \ldots + C_i$ s.t. for any $w_j \neq \top$, $1 \leq j \leq i$, $C_j = \{(w_j + u_j, 1)\}$ for some $u_j \in \mathbb{M}(A)$.
$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi \wedge \psi$ iff $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi$ and $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \psi$.
$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \neg\phi$ iff $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \not\models \phi$.
$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models E\phi$ iff there exists $e \in E_{\mathcal{C}}^{asyn}(\Pi)$ such that $\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi$.
$\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi U \psi$ iff there exists $i \geq 0$ such that $\mathcal{T}_{\mathcal{C}}^{asyn}, e_i \models \psi$ and for all $j \leq i$ $\mathcal{T}_{\mathcal{C}}^{asyn}, e_j \models \phi$.
$\mathcal{T}_{\mathcal{C}}^{asyn}, e \models X\phi$ iff $\mathcal{T}_{\mathcal{C}}^{asyn}, e_1 \models \phi$.

Definition 4.6 (Validity and satisfiability) *A configuration formula ϕ (evolution formula ϕ) from $\mathcal{L}_{\mathcal{C}}$ is* valid *iff for every temporal structure $\mathcal{T}_{\mathcal{C}}^{asyn} = (\mathcal{S}, \mathfrak{R})$ in $\mathcal{C}$ and any $\Pi \in \mathcal{S}$ (any $e \in E_{\mathcal{C}}^{asyn}(\Pi)$, resp.) we have $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi$ ($\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi$, resp.). A configuration formula ϕ (evolution formula ϕ) is satisfiable iff there exists a temporal structure $\mathcal{T}_{\mathcal{C}}^{asyn} = (\mathcal{S}, \mathfrak{R})$ and a $\Pi \in \mathcal{S}$ (an $e \in E_{\mathcal{C}}^{asyn}(\Pi)$, resp.) such that $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi$ ($\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi$, resp.).*

Definition 4.7 (Derived formulas) *We define the following derived formulas for $\mathcal{L}_{\mathcal{C}}$.*
$A\phi = \neg E \neg \phi$.
$F\phi = \top U \phi$.
$G\phi = \neg F \neg \phi$.

The semantics of the derived formulas are the following.

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

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

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

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

Theorem 10 (Decidability) *The satisfiability, validity and model-checking problems for $\mathcal{L}_\mathcal{C}$ against the CSA-semantics are decidable.*

Proof. The result derives from the fact that CTL logic is decidable (see, e.g., [18, 1]) and from the fact that $AP(\mathcal{C})$, the set of atomic formulas, is a finite set. □

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

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

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

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

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

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

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

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

Similarly, $\mathcal{T}_\mathcal{C}^{asyn}, \Pi_i \models u \oplus v \oplus \top$ iff $C_{\Pi_i} = C'_1 + C'_2 + C'_3$, $C'_1, C'_2, C'_3 \in \mathbb{M}(\mathbb{M}(A))$ and $C'_1 = \{(u + u'', 1)\}$, $C'_2 = \{(v + v', 1)\}$, i.e., the configuration of Π_i contains two agents w_1 and w_2 such that $u \subseteq w_1$ and $v \subseteq w_2$, which precisely indicates that Π_0 is safe on synchronization of rule $r : \langle u, v\rangle \to \langle u', v'\rangle$.

5 Prospects

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

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

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

With the addition of the features just mentioned, it will also be interesting to extend the investigation and proofs given above to identify further classes of CSAs demonstrating robustness and having decidable properties. It is hoped that this approach will then provide insight in challenging areas of systems biology.

References

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

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

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

[4] C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, editors. *Multiset Processing: Mathematical, Computer Science, and Molecular Computing Point of View*, LNCS 2235, Springer-Verlag, 2001.

[5] M. Cavaliere, R. Mardare, S. Sedwards. Colonies of Synchronizing Agents. Technical Report CoSBi 11/2007. Available at `www.cosbi.eu/Rpty_Tech.php`

[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. In *Proc. Third Brainstorming on Membrane Computing.* RGCN Report 01/2005, Sevilla, 2005. Available at `www.gcn.us.es`

[9] S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3), 1978.

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

[11] 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.

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

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

[14] J. Mata, M. Cohn. Cellular Automata-Based Modelling Program: Synthetic Immune Systems. *Immunol Rev*, 207, 2007.

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

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

[17] G. Rozenberg, A. Salomaa, editors. *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] S. Wolfram. *A New Kind of Science.* Wolfram Media, 2002.

[20] `http://nusmv.irst.itc.it/`

[21] `http://psystems.disco.unimib.it`