View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repository of the Academy's Library

A Logical Representation of P Colonies: An Introduction

Luděk Cienciala[1], Lucie Ciencialová[1], Erzsébet Csuhaj-Varjú[2(✉)], and Petr Sosík[1]

[1] Institute of Computer Science and Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Opava, Czech Republic
{ludek.cienciala,lucie.ciencialova,petr.sosik}@fpf.slu.cz
[2] Faculty of Informatics, ELTE Eötvös Loránd University, Budapest, Hungary
csuhaj@inf.elte.hu

Abstract. We introduce a new way of representation of computation in P colonies. It is based on logical values, propositional logic and rule-based systems. A configuration of a P colony is transformed into a data structure based on a system of stacks. We present a conversion of conditions of applicability of rules, programs, multisets of programs and complete computational steps as propositional formulas in the disjunctive normal form. This representation allows, among others, to derive new results concerning the complexity of execution of computational steps of a P colony.

1 Introduction

P colonies, introduced in [5], are variants of very simple tissue-like P systems, where the cells (agents) have only one region and they interact with their joint shared environment by using programs, i.e., by finite collections of rules of special forms. The extraordinary simplicity of these constructs is demonstrated by some of their important characteristics.

At any step of their functioning, both the agents and the environment are represented by a finite number of objects, elements of an object-alphabet. (We note that the environment has an infinite number of occurrences of a special symbol, called the environmental symbol as well.) The agents in the P colony have constant capacity and each agent has the same capacity, i.e., at any computational step every agent is represented by a constant number of objects and this number is the same for any agent. Furthermore, the agents can change their contents (the objects at their disposal) and the objects in the environment by using very simple rules, namely, evolution rules (an object inside the agent is changed for some other object) or communication rules (an object inside the agent is exchanged with an object located in the environment). There exists one other type of rules as well, the so-called checking rule. A checking rule consists of either two evolution rules or of two communication rules, written as r_1/r_2. Rule r_1 has higher priority than r_2, i.e., if r_1 is applicable, then it has to be

© Springer Nature Switzerland AG 2018
C. Graciani et al. (Eds.): Pérez-Jiménez Festschrift, LNCS 11270, pp. 66–76, 2018.
https://doi.org/10.1007/978-3-030-00265-7_6

used, otherwise r_2 has to be applied. The change of the contents of the agents and the change of the current environment is performed by programs. Every agent has a finite set of programs, and each program consists of as many rules as the capacity of the agent. When a program is applied, all of its rules should be used in parallel. At every step, as many agents perform a program in parallel as possible. These synchronized actions of the agents correspond to a configuration change of the P colony. A finite sequence of configuration changes following each other and starting from the so-called initial configuration is a computation. The result of the computation is the number of copies of a distinguished object, called the final object, occurring in the environment in a final configuration of the P colony, which is usually a halting configuration.

During the years, P colonies have been studied in detail; for summaries consult [1,4]. These investigations mainly focused on studying P colonies as computing devices; a large number of results prove that even though P colonies are very simple computing devices they are computationally complete even with restricted size parameters.

In this paper we study P colonies from other aspect, namely, we provide a representation of P colonies in terms of logical values, propositional logic and rule-based systems. The startpoint of our approach is that the applicability conditions of rules, programs, multisets of programs can be given as propositional formulas in disjunctive normal form and the computational step is obtained by using a rule-based system.

Obviously, if an evolution rule of the form $a \to b$ is to be applied by an agent, then object a should be present inside the agent. Analogously, if a communication rule $c \leftrightarrow d$ is to be applied by an agent, then object c should be present inside the agent and object d should appear in the environment. To perform rules and programs, the configuration of the P colony has to satisfy such conditions.

To continue the concept of logical representation of P colonies, a configuration of a P colony is transformed into a system of stacks with logical values. Elements of these stacks are then used as variables in the propositional formulas and thus a configuration defines an interpretation of the formulas. The computational step corresponds to a transition of a rule-based production system. Using this approach, we may solve several problems concerning P colonies. In this paper, we proved that the decision problem whether a configuration C is a halting configuration of a P colony Π without checking rules is in **P**, and if Π is with checking rules then it is in **NP**.

The logical representation of P colonies we provide and the approach we propose allow to apply many results known in propositional logic to resolve open problems in P colony theory. We close the paper with two open problems and a short discussion of the possible applications of this new approach.

2 Preliminaries and Basic Notions

Throughout the paper we assume the reader to be familiar with the basics of formal language theory and membrane computing, more information can be found in [6,7].

For an alphabet Σ, the set of all words over Σ (including the empty word, ε), is denoted by Σ^*. The length of a word $w \in \Sigma^*$ is denoted by $|w|$ and $|w|_a$ denotes the number of occurrences of the symbol $a \in \Sigma$ in w.

A multiset of objects M is a pair $M = (O, f)$, where O is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : O \to \mathbb{N}$; f assigns to each object in O its multiplicity in M. Any multiset of objects M with the set of objects $O = \{x_1, \ldots x_n\}$ can be represented as a string w over alphabet O with $|w|_{x_i} = f(x_i)$; $1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters can also represent the same multiset M, and ε represents the empty multiset.

2.1 P Colonies

The original concept of a P colony was introduced in [5] and presented in a developed form in [2,3].

Definition 1. *A P colony of capacity k, $k \geq 1$, is a construct $\Pi = (O, e, f, V_E, A_1, \ldots, A_n)$, where*

- *O is an alphabet, its elements are called objects;*
- *$e \in O$ is the basic (or environmental) object of the P colony;*
- *$f \in O$ is the final object of the P colony;*
- *V_E is a finite multiset over $O - \{e\}$, called the initial state (or the initial content) of the environment;*
- *A_i, $1 \leq i \leq n$, are agents, where each agent $A_i = (O_i, P_i)$ is defined as follows:*
  - *O_i is a multiset consisting of k objects over O, the initial state (or initial content) of the agent;*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite set of programs, where each program consists of k rules. Each rule is in one of the following forms:*
    - *$a \to b$, called an evolution rule;*
    - *$c \leftrightarrow d$, called a communication rule;*
    - *r_1/r_2, called a checking rule; r_1, r_2 are evolution rules or communication rules.*

We add some brief explanations to the components of the P colony. We first note that throughout the paper, we use term "object a is inside agent A" and term "$a \in w$, where w is the state of agent A" as equivalent.

The first type of rules associated to the programs of the agents, the *evolution rules*, are of the form $a \to b$. This means that object a inside the agent is rewritten to (evolved to be) object b. The second type of rules, the *communication rules*, are of the form $c \leftrightarrow d$. If such a communication rule is performed, then object c inside the agent and object d in the environment swap their location, i.e., after executing the rule, object d appears inside the agent and object c is located in the environment.

The third type of rules are the checking rules. A checking rule is formed from two rules of one of the two previous types. If a checking rule r_1/r_2 is performed,

then the rule r_1 has higher priority to be executed over the rule r_2. This means that the agent checks whether or not rule r_1 is applicable. If the rule can be executed, then the agent must use this rule. If rule r_1 cannot be applied, then the agent uses rule r_2.

The program determines the activity of the agent: the agent can change its state and/or the state of the environment.

The environment is represented by a finite number (zero included) of copies of non-environmental objects and a countably infinite copies of the environmental object e.

In every step, if a program is applied, then each object inside the agent is affected by its execution. Depending on the rules in the program, the program execution may affect the environment as well. *This interaction between the agents and the environment is the key factor of the functioning of the P colony.*

The functioning of the P colony starts from its initial configuration (initial state). The *initial configuration* of a P colony is an $(n + 1)$-tuple of multisets of objects present in the P colony at the beginning of the computation. It is given by the multisets O_i for $1 \leq i \leq n$ and by multiset V_E. Formally, the *configuration* of the P colony Π is given by $(w_1, \ldots, w_n, w_E)$, where $|w_i| = k$, $1 \leq i \leq n$, w_i represents all the objects present inside the i-th agent, and $w_E \in (O - \{e\})^*$ represents all the objects in the environment different from object e. A configuration $(w_1, \ldots, w_n, w_E)$ *contains* a configuration $(w'_1, \ldots, w'_n, w'_E)$ *iff* $w'_E \subseteq w_E$ and $w'_i \subseteq w_i$, $1 \leq i \leq n$.

At each *step of the computation* (at each transition), the state of the environment and that of the agents change in the following manner:

In the *maximally parallel* derivation mode a maximal number of agents performs one of its applicable (non-deterministically chosen) programs simultaneously. This means that applicable programs are added to the multiset of applied programs, one program per agent, in an arbitrary order, until no more programs can be added due to the trade-offs between them. Then the multiset of programs is applied.

The other derivation mode is the *sequential* derivation mode. In this case one agent uses one of its programs at a time. If more than one agent is able to apply at least of its programs, then the acting agent is non-deterministically chosen. If the number of applicable programs for the agent is higher than one, then the agent non-deterministically chooses one of these programs.

A transition between configurations C_1 and C_2 is denoted by $C_1 \Rightarrow C_2$. A configuration C is called *alive* if there is another configuration $C' \neq C$ such that $C \Rightarrow C'$. Otherwise, the configuration is called *dead*. Note that a dead configuration is either halting (the P colony cannot apply any rule), or else each valid multiset of applicable rules (subject to the derivation mode) leads the P colony to the same configuration.

A sequence of transitions starting in the initial configuration is called a *computation*. A computation is said to be *halting* if a configuration is reached where no program can be applied. With a halting computation, we associate a *result*

which is given as the number of copies of the objects f present in the environment in the halting configuration.

Because of the non-determinism in choosing the programs, starting from the initial configuration we obtain several computations, hence, with a P colony we can associate a set of numbers, denoted by $N(\Pi)$, computed by all possible halting computations of the given P colony.

In the original model (see [5]) the number of objects inside each agent is set to two. Therefore, the programs were formed from only two rules. Moreover, the initial configuration was defined as $(n+1)$-tuple $(ee, \ldots, ee, \varepsilon)$ so the environment of the P colony is "empty", i.e., without an input information at the beginning of the computation.

The number of agents in a given P colony is called the degree of Π; the maximal number of programs of an agent of Π is called the height of Π.

3 Logical Representation of P Colonies

In this section we introduce a new way of representation of the concept of a P colony. First, we briefly explain the idea. To represent existence (or non-existence) of objects in the P colony, we use value 1 (or 0). Let a be an object in the P colony ($a \in O$) and suppose that there are three copies of such object placed in the environment. We construct a stack called "a" and put value 0 into the bottom of stack. For every copy of object a in the environment, we push one copy of 1 to the stack. The presence of object a can be expressed as literal a interpreted as TRUE, otherwise it is FALSE.

a	1	1	1	0

An agent of capacity k is represented by an array of $|O|$ stacks. The sum of 1s in all stacks is k. For example, agent A_1 with capacity 3 working with alphabet $O = \{e, a, b, c, \}$ and with objects aae inside the agent has following representation:

A_1 :				
	a	1	1	0
	b	0		
	c	0		
	e	1	0	

The presence of object a inside the agent A_i can be expressed as literal $A_i[a]$. or more precisely, as an interpretation of this literal.

In these terms we describe how one step of the computation in the whole system is done: We divide the process into two phases – in the first phase a multiset of programs is chosen randomly from the set of all multisets containing one applicable program per each agent which can apply at least one program. In the second phase we check the actual applicability of each program in the selected multiset in relation to the number of objects needed for the execution of the programs. Now we describe the process in detail.

A rewriting rule $a \to b$ of agent A_i is applicable if there is an object a inside the agent A_i. It means that the rule is applicable if literal $A_i[a]$ is true. The communication rule $a \leftrightarrow b$ is applicable if there is an object a inside the agent and object b in the environment. In terms of logic we can write the condition as $A_1[a] \wedge b$. If $b = e$ we may omit b in the condition (there is always some copy of e in the environment). A condition of applicability of a rewriting or a communication rule will be called *elementary condition of applicability.*

We can express the condition of applicability for checking rule r_1/r_2 as $c_1 \vee c_2$ where r_1, r_2 are rewriting or communication rules with conditions of applicability c_1, c_2. Notice that we speak of applicability, and not the way of application: r_1/r_2 is applicable if at least one of r_1 and r_2 is applicable, that is, if $c_1 \vee c_2$ is TRUE. If r_1 is applicable, then r_1/r_2 can be applied. If r_2 is applicable but r_1 is not applicable, then checking rule r_1/r_2 can be applied as well. If both r_1 and r_2 are applicable, then r_1/r_2 is also applicable, in this case we apply r_1.

Lemma 1. *Given a program $p_{i,l}$, the condition of its applicability $c_{i,l}$ can be expressed in a disjunctive normal form with 2^d conjunctions, where d is the number of checking rules in the program.*

Proof. The condition of applicability of the programs is $c_{i,l} : c_1 \wedge c_2 \wedge \cdots \wedge c_k$, where c_x is the condition of applicability of the x-th rule in the program. A condition is in the form:

$A_i[a]$ (rewriting rule),
$A_i[a] \wedge b$ (communication rule),
$c_{x_1} \vee c_{x_2}$ (checking rule).

If the program contains a checking rule, then we can write the condition $c_{i,l}$ in the disjunctive normal form (DNF) $c_{i,l} : (c_1 \wedge c_2 \wedge \cdots \wedge c_{j_1} \wedge c_{j+1} \wedge \cdots \wedge c_k) \vee (c_1 \wedge c_2 \wedge \cdots \wedge c_{j_2} \wedge c_{j+1} \wedge \cdots \wedge c_k)$. (Notice that c_{j_1} and c_{j_2} are the conditions for applicability of the two subrules of the checking rule.)

If there are d checking rules in the program, then the formula contains a conjunction of d disjunctions and $k - d$ elementary conditions (i.e., literals or conjunction of literals). Its conversion to DNF results in a disjunction of 2^d conjunctions of k-tuples of elementary conditions of applicability.

Furthermore, consider a k-tuple of elementary rules corresponding to a conjunction. Generally, j rules ($1 \leq j \leq k$) may depend on the presence of the same object a inside the agent, hence the program is applicable only if the agent contains at least j objects a. To indicate that some objects should be present in several copies, we introduce a literal $A_i[a][j]$, $1 \leq j \leq k$, which is TRUE when the j-th position in the stack "a" of agent A_i exists and contains 1. Similarly, $b[j]$ is the literal which is TRUE when the j-th position in stack "b" is 1, i.e., when the environment contains at least j objects b.

Therefore, in each conjunction in the final DNF of the condition $c_{i,l}$, literals $A_i[a]$ must be substituted/indexed for $A_i[a][j]$, where j is the order of occurrence of $A_i[a]$ in the conjunction. Similarly, each literal b is substituted for $b[j]$. □

Given a DNF representing the applicability of a program $p_{i,l}$ with d checking rules, without any change in its satisfiability, we can re-order the conjunctions in DNF due to decreasing priority among rules as follows:

0. Conjunctions with elementary conditions for the first rule in all d checking rules.
1. Conjunctions with elementary condition for the second rule in one checking rule, and for the first rule in the remaining checking rules.
2. Conjunctions with elementary conditions for the second rule in two checking rules, and for the first rule in the remaining checking rules.

   ⋮

d. Conjunctions with elementary conditions for the second rule in all d checking rules.

This reordering induced by checking rules is crucial for the process of correct execution of one computational step of the P colony, as it is described after Lemma 2. In this process, conjunctions in DNF are evaluated in the left-to-right order, which ensures that the first elementary rule in checking rules has always priority over the second elementary rule.

Clearly, the logical condition whether an agent A_i can be active (i.e., is able to apply some of its programs) can be expressed as a disjunction of conditions for all programs of that agent: $c_i = c_{i,1} \vee c_{i,2} \vee \cdots \vee c_{i,k_i}$, where k_i is the number of programs of the agent A_i.

Lemma 2. *Given a P colony Π as in Definition 1, the condition whether Π can perform a computational step can be expressed in a DNF with $\sum_{i=1}^{n} \sum_{j=1}^{k_i} 2^{d_{i,j}}$ conjunctions, where $d_{i,j}$ is the number of checking rules in the program $p_{i,j}$.*

Proof. A P colony can perform a computational step (regardless of the sequential or parallel mode) if at least one of its agents can apply some program. Hence, the condition of applicability of a computational step of the colony Π is the disjunction of conditions $c_{i,l}$ for all programs of all agents of Π. By Lemma 1, these conditions are already in the DNF, so their disjunction leads to a greater DNF. The total number of conjunctions in the resulting DNF is just the sum of individual DNF's, and the statement follows again by Lemma 1. □

The process of execution of one computational step of the P colony in logical representation under maximally parallel mode can be now completed as follows.

1st phase:

(a) For each program $p_{i,j}$ in the colony construct the formula $c_{i,j}$ of its applicability as described in Lemma 1.
(b) For each agent A_i choose one the formulas $c_{i,j}$, $1 \leq j \leq k_i$, which is TRUE in the actual configuration (the configuration interprets all literals). If an agent has no such formula, then it cannot apply any program.

(c) In each chosen $c_{i,j}$ in the DNF (ordered by their priorities induced by checking rules) find the first conjunction which is TRUE and add it to a resulting multiset M of formulas (corresponding to the multiset of applicable programs).
(d) If the multiset M is empty, then the configuration is halting. Otherwise, construct a disjunction c_M of all conjunctions in M (the order of conjunctions is random).

2^{nd} phase:

(a) Re-index literals a for all $a \in O$ (corresponding to objects in the environment) to $a[j]$ using the total order of occurrence of a in the whole formula c_M. This is necessary as the limited amount of environmental objects may cause trade-offs among chosen programs.
(b) Re-interpret all conjunctions in the formula c_M. Some of them may be now FALSE, while the whole formula remains TRUE.
(c) Apply (in parallel) all sequences of elementary rules corresponding to those conjunctions which are still TRUE.

The execution of a multiset of rules can be understood as an action of a rule-based production system: as sensory precondition we use condition of applicability and an action can be constructed from functions `push` and `pop` as it is usual for stacks. Function $\texttt{push}(x)$ means put 1 to the top of stack x. Function $\texttt{pop}(x)$ means remove 1 from the top of stack x.

Let us show one step of computation for simple P colony $\Pi = (O, e, f, V_E, A_1)$ with capacity two and one agent and with $O = \{a, b, c, d, e, f\}$, $w_E = \varepsilon$, $A_1 = (ee, \{\langle a \leftrightarrow c/c \leftrightarrow d;\ c \leftrightarrow f/a \leftrightarrow e\rangle;\ \langle a \to b;\ e \leftrightarrow b\rangle\})$.

Let us construct a condition of applicability of the program $\langle a \to b;\ e \leftrightarrow b\rangle$: It is formed from one rewriting and one communication rule.

rule	elementary condition of applicability
$a \to b$	$A_1[a]$
$e \leftrightarrow b$	$A_1[e] \wedge b$

The condition of applicability of the program after the substitution (indexing) of literals is $A_1[a][1] \wedge A_1[e][1] \wedge b[1]$.

The condition of applicability of the program $\langle a \leftrightarrow c/c \leftrightarrow d;\ c \leftrightarrow f/a \leftrightarrow e\rangle$ is formed from two checking rules, each formed from two communication rules.

rule	elementary condition of applicability
$a \leftrightarrow c$	$c_{11} : A_1[a] \wedge c$
$c \leftrightarrow d$	$c_{12} : A_1[c] \wedge d$
$c \leftrightarrow f$	$c_{21} : A_1[c] \wedge f$
$a \leftrightarrow e$	$c_{22} : A_1[a]$
$a \leftrightarrow c/c \leftrightarrow d$	$(A_1[a] \wedge c) \vee (A_1[c] \wedge d)$
$c \leftrightarrow f/a \leftrightarrow e$	$(A_1[c] \wedge f) \vee A_1[a]$

The condition of applicability of the program is formed from four conjunctions: $c_{11} \wedge c_{21}$ with highest priority, $c_{12} \wedge c_{21}$ and $c_{11} \wedge c_{22}$, and $c_{12} \wedge c_{22}$ with lowest priority. After indexing of literals we obtain

$$\begin{aligned}
&(A_1[a][1] \wedge c[1] \wedge A_1[c][1] \wedge f[1]) \vee \\
\vee\ &(A_1[c][1] \wedge d[1] \wedge A_1[c][2] \wedge f[1]) \vee \\
\vee\ &(A_1[a][1] \wedge c[1] \wedge A_1[a][2]) \vee \\
\vee\ &(A_1[c][1] \wedge d[1] \wedge A_1[a][1])
\end{aligned}$$

Rules for execution of programs are:

- $\texttt{if}\ A_1[a][1] \wedge c[1] \wedge A_1[c][1] \wedge f[1]\ \texttt{then}\ (\texttt{pop}(A_1[a]) \wedge \texttt{push}(A_1[c]) \wedge \texttt{pop}(c) \wedge \texttt{push}(a) \wedge \texttt{pop}(A_1[c]) \wedge \texttt{push}(A_1[f]) \wedge \texttt{pop}(f) \wedge \texttt{push}(c))$
- $\texttt{if}\ A_1[c][1] \wedge d[1] \wedge A_1[c][2] \wedge f[1]\ \texttt{then}\ (\texttt{pop}(A_1[c]) \wedge \texttt{push}(A_1[d]) \wedge \texttt{pop}(d) \wedge \texttt{push}(c) \wedge \texttt{pop}(A_1[c]) \wedge \texttt{push}(A_1[f]) \wedge \texttt{pop}(f) \wedge \texttt{push}(c))$
- $\texttt{if}\ A_1[a][1] \wedge c[1] \wedge A_1[a][2]\ \texttt{then}\ (\texttt{pop}(A_1[a]) \wedge \texttt{push}(A_1[c]) \wedge \texttt{pop}(c) \wedge \texttt{push}(a) \wedge \texttt{pop}(A_1[a]) \wedge \texttt{push}(A_1[e]) \wedge \texttt{push}(a))$
- $\texttt{if}\ A_1[c][1] \wedge d[1] \wedge A_1[a][1]\ \texttt{then}\ (\texttt{pop}(A_1[c]) \wedge \texttt{push}(A_1[d]) \wedge \texttt{pop}(d) \wedge \texttt{push}(c) \wedge \texttt{pop}(A_1[a]) \wedge \texttt{push}(A_1[e]) \wedge \texttt{push}(a))$

The logical representation of P colonies, particularly the conditions of applicability of (multisets of) rules allows for a clearer view of complexity of the process of execution of a P colony. Here we focus on the problem whether a given configuration is halting. This problems amounts to checking whether there exists an applicable multiset of programs.

Theorem 1. *Consider a P colony Π without checking rules, and a configuration C of Π. The problem whether C is a halting configuration is in* **P**.

Proof. If no checking rules are present then, by Lemma 1, each condition $c_{i,l}$ corresponding to an applicability of a program $p_{i,l}$, $1 \leq i \leq n$, $1 \leq l \leq k_i$, consists of a single conjunction. By Lemma 2, the formula – condition of applicability of a computational step of Π is a DNF consisting of $\sum_{i=1}^{n} k_i$ conjunctions, each containing at most $2k$ literals. Therefore, the size of the formula is polynomial in the size of description of the P colony Π, and so is the algorithm for its construction and interpretation. □

Theorem 2. *Consider a P colony Π with checking rules, and a configuration C of Π. The problem whether C is a halting configuration is in* **NP**.

Proof. In presence of checking rules, each condition $c_{i,l}$ corresponding to an applicability of a program $p_{i,l}$, $1 \leq i \leq n$, $1 \leq l \leq k_i$, is expressed in DNF consisting of up to 2^k conjunctions (Lemma 1). One could argue that the size of the formula is smaller before its conversion to the DNF. However, the conversion seems necessary since all literals in the formula must be indexed (see the proof of Lemma 1) to interpret the formula correctly, and this is done only after the conversion to the DNF.

By Lemma 2, the formula – condition of applicability of a computational step of Π is a DNF consisting of $\sum_{i=1}^{n} \sum_{j=1}^{k_i} 2^{d_{i,j}} = \mathcal{O}(n2^k)$ conjunctions, each conjunction containing at most $2k$ literals, where $1 \leq d_{i,j} \leq k$ is the number of checking rules in the program $p_{i,j}$. Therefore, the size of the formula (in number of literals) is $\mathcal{O}(nk2^k)$. □

Open Problem 1. Consider the problem "is a given configuration of a P colony with checking rules halting?" Is the problem NP-complete? We conjecture yes but no proof is known yet.

Open Problem 2. How complex is the problem to decide whether a given configuration of a P colony is dead or alive? (Consult Sect. 2.1 for the definition of a dead configuration.)

4 Conclusions

In this paper we introduced a concept of logical representation of P colonies, particularly the transformation of applicability of its rules and computational steps into propositional formulas. A configuration of a P colony is transformed into a system of stacks with logical values. Elements of these stacks are then used as variables in the mentioned propositional formulas and thus a configuration defines an interpretation of the formulas. The application of a computational step then can be viewed as a transition of a rule-based production system.

The logical representation of P colonies allows to apply many results known in propositional logic to resolve open problems concerning P colonies. Particularly, it allows to convert the conditions of applicability of programs and multisets of programs into the form of logical formulas in DNF. This transformation, in turn, results in a straightforward characterization of computational complexity of execution of computational steps of a P colony. Some of them may characterize the borderline between P and NP, although the proof is not known yet. Many related problems remain open, two of which are mentioned in the previous section.

Acknowledgments. This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602, by SGS/13/2016 and by Grant No. 120558 of the National Research, Development, and Innovation Office - NKFIH, Hungary.

References

1. Ciencialová, L., Csuhaj-Varjú, E., Cienciala, L., Sosík, P.: P colonies. Bull. Int. Membr. Comput. Soc. **1**(2), 119–156 (2016)
2. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, Gy.: Computing with cells in environment: P colonies. J. Mult.-Valued Log. Soft Comput. **12**(3–4 Spec. Iss.), 201–215 (2006)
3. Kelemen, J., Kelemenová, A.: On P colonies, a biochemically inspired model of computation. In: Proceedings of the 6th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH, Hungary, pp. 40–56 (2005)
4. Kelemenová, A.: P colonies. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, Chap. 23.1, pp. 584–593. Oxford University Press, Oxford (2010)

5. Kelemen, J., Kelemenová, A., Păun, Gh.: Preview of P colonies: a biochemically inspired computing model. In: Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX), Mass, Boston, pp. 82–86 (2004)
6. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press Inc., New York (2010)
7. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages I-III. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59126-6