Modified P Colony with Rule Application Intensity

Lucie Ciencialová[1,*,†], Luděk Cienciala[1,†]

[1]Institute of Computer Science, Faculty of Philosophy and Science in Opava, Silesian University in Opava, Opava, Czech Republic

Abstract

In this paper, we introduce a new computational model called the *modified P colony with rule application intensity (RAI)*. This model extends the traditional concept of P colonies by associating each rule with a real-valued intensity that influences how often and under what conditions the rule can be applied. The rule application intensity is dynamically adjusted during the computation based on whether a rule is used or remains unused. We formally define the structure and operational semantics of the system, including the configuration transitions and intensity updates. Several variants of the model are discussed, such as systems with fixed-size agents, λ-free rule sets, or systems with constant RAI. We also present illustrative examples, including a construction of a finite set generator and a design proposal for an idle game engine based on this model. The article concludes with an outline of current and future research directions, including the development of a simulator and the study of the computational power of these systems.

Keywords

membrane computing, P Colonies, agent-based model, nature-inspired computation model

1. Introduction

In recent years, increased interest has emerged in modeling multi-agent systems composed of simple, reactive agents interacting within a shared environment. P colonies, introduced in [1], represent one such model. Inspired by membrane computing [2] and formal grammar colonies [3], P colonies consist of agents—each with a fixed multiset of objects and a set of programs—operating in a common environment.

Each agent's object content remains constant during computation (its capacity), while the environment serves both as a communication medium and a shared object store. Among the objects, a special *environmental object* is assumed to occur in countably infinite supply. Agents act by applying simple rules that can transform or exchange objects with the environment. A computation is a sequence of transitions, and its result is determined by the presence of a designated *final object* in the environment.

Various extensions of the original model have been developed over the years (see [4] or [5] for a survey), with research focusing on computational power, descriptional complexity, and the role of the environment in influencing agent behavior. The simplicity and distributed nature of P colonies make them a suitable model for systems such as robot swarms, producer-consumer systems, or ecological simulations.

In this paper, we introduce a novel variant of P colonies that incorporates *rule application intensity* (RAI), allowing the degree to which rules are applied to evolve dynamically throughout the computation. This enhancement is motivated by the need to model systems where agents adapt their behavior based on environmental conditions. For instance, if an agent frequently encounters a surplus of specific objects it can process, its behavior may shift toward increased consumption of those objects. Conversely, if key resources required for certain processes become scarce, the related behaviors may gradually fade out or become deprioritized.

This adaptive mechanism reflects phenomena observed in natural systems: organisms often adjust their metabolic or behavioral pathways in response to resource availability, competition, or environmen-

ITAT'25: Information Technologies – Applications and Theory, September 26–30, 2025, Telgárt, Slovakia

*Corresponding author.

†These authors contributed equally.

lucie.ciencialova@fpf.slu.cz (L. Ciencialová); ludek.cienciala@fpf.slu.cz (L. Cienciala)
0000-0002-0877-7063 (L. Ciencialová); 0000-0001-9473-5945 (L. Cienciala)

tal pressures. For example, certain metabolic enzymes are upregulated when substrates are abundant, while unused or energetically costly processes are suppressed during scarcity. By enabling rule application intensity to vary over time, influenced either internally (by agent dynamics) or externally (via special input from the environment), our model captures these kinds of flexible, context-dependent behaviors, extending the expressive power and modeling depth of P colonies.

The structure of the paper is as follows. After this introduction, Section 2 presents the basic definitions and formal terminology used throughout the paper. The definition and behavior of the original P colony model form the basis for developing our modified variant incorporating rule application intensity. In Section 3, we introduce the new model of Modified P Colonies with Rule Application Intensity (RAI). Section 4 provides several illustrative examples that demonstrate the expressive power and versatility of different variants of the model. Finally, in Section 5, we summarize the contributions of this work and outline directions for future research.

2. Preliminaries and Definitions

Throughout the paper we assume the reader to be familiar with the basics of formal language theory and membrane computing [6, 4].

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

A multiset of objects M is a pair $M = (A, f)$, where A is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : A \to \mathbb{N}$; f assigns to each object in A its multiplicity in M. Any multiset of objects M with the set of objects $A = \{x_1, \dots x_n\}$ can be represented as a string w over alphabet A 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

Since this article focuses on a type of membrane system derived from the P Colony model, we provide the definitions related to P Colonies.

The original concept of a P colony was introduced in [1] and presented in a developed form in [7, 8].

Definition 1. *A P colony of capacity k, $k \geq 1$, is a construct*

$$\Pi = (A, e, f, v_E, B_1, \dots, B_n),\ \textit{where}$$

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

The first type of rules associated with the agents' programs are the *evolution rules*, which have the form $a \to b$. This indicates that object a within the agent is rewritten (or evolved) into object b.

The second type of rules are the *communication rules*, expressed as $c \leftrightarrow d$. When such a rule is applied, object c located inside the agent and object d in the environment exchange their positions. As a result, after the execution of the rule, object d is found inside the agent, and object c is placed in the environment.

The third category consists of *checking rules*. A checking rule is constructed from two rules of the types described above. Denoted as r_1/r_2, this rule expresses a priority: rule r_1 has precedence over rule r_2. When a checking rule is evaluated, the agent first verifies whether rule r_1 is applicable. If it is, then r_1 must be executed. Otherwise, rule r_2 is applied instead.

The program governs the behavior of an agent, allowing it to modify either its own internal state and/or the state of the environment.

The environment is modeled as consisting of a finite number (possibly zero) of copies of non-environmental objects, together with a countably infinite supply of a special environmental object denoted by e.

When an agent executes a program, each object it contains is affected by the operation. Depending on the program's rules, its execution may also influence the environment. This interaction between agents and the environment is a fundamental aspect of the functioning of a P Colony.

The computation of a P Colony begins in an initial configuration, which defines the starting state of the system.

The initial configuration of a P Colony is given as an $(n + 1)$-tuple of multisets, representing the distribution of objects among the agents and the environment at the start of computation. Specifically, it is defined by the multisets o_i for $1 \leq i \leq n$, corresponding to the objects within each of the n agents, and by the multiset v_E, representing the contents of the environment. Formally, a configuration of a P Colony Π is described by the tuple $(w_1, \dots, w_n, w_E)$, where each w_i is a multiset of size k (i.e., $|w_i| = k$, for $1 \leq i \leq n$), and $w_E \in (A \setminus e)^*$ denotes the multiset of objects present in the environment, excluding the infinite supply of e.

At each computation step (or transition), the states of the agents and the environment evolve according to a chosen derivation mode. In the maximally parallel mode, each agent that is able to apply one of its programs must do so (a program is selected non-deterministically). In contrast, under the sequential derivation mode, only a single agent is allowed to apply one of its programs at each step, also chosen non-deterministically. If multiple programs are applicable within an agent, one is selected non-deterministically.

A sequence of such transitions forms a computation. A computation is said to be halting if it reaches a configuration in which no further program is applicable in any agent. The result of a halting computation is defined as the number of copies of a distinguished object f present in the environment at the halting configuration.

Due to the inherent non-determinism in program selection, multiple distinct computations may arise from the same initial configuration. Accordingly, a P Colony Π computes a set of natural numbers, denoted by $N(\Pi)$, representing the outputs of all possible halting computations.

As established in [1], P Colonies with a capacity of two possess computational completeness. Additionally, their programs follow a specific structure: one rule is an evolution rule, and the other is either a communication rule or a checking rule consisting of two communication rules.

3. Modified P Colony with Rule Application Intensity

In the following section, we introduce a novel type of membrane systems that shares both the structural organization and the rule types with P Colonies.

A key structural distinction from P Colonies is found in agent capacity. While P Colonies operate under a strict and constant bound on the number of objects each agent may contain, our model removes this limitation. The number of objects inside an agent can be arbitrary and may vary during the computation due to the application of rules.

In our model, only evolution and communication rules are used.

Each rule is extended with a positive real number that represents its rule application intensity (RAI). At the beginning of the computation, the RAI of each rule is determined by the system definition; in general, the RAI is initially set to 1.

The integer part of RAI specifies how many times the rule is to be applied during a single computational step. Moreover, RAI of each rule is not fixed throughout the computation. It is explicitly updated during last phase of each computation step, reflecting how the rule has been applied in that step. This dynamic adjustment allows the rule's influence to evolve over time, depending on the system's state and rule utilization. Let A denote the finite alphabet of objects used in the system. Each *rule* in the system is defined as a quadruple $(\tau, \mathsf{lhs}, \mathsf{rhs}, \alpha)$, where τ specifies the type of the rule (either *evo* for evolution or *com* for communication), lhs and rhs represent the left-hand and right-hand sides of the rule, respectively, with $\mathsf{lhs}, \mathsf{rhs} \in A \cup \{\lambda\}$, and $\alpha \in \mathbb{R}^+$ denotes the application intensity.

According to the definition of rules, the symbol λ may occur on either the left-hand side or the right-hand side of a rule. In the case of a communication rule, the presence of λ on the left-hand side indicates that an object is transferred from the environment into the agent without any object leaving the agent. Conversely, if λ appears on the right-hand side, the agent sends an object to the environment without receiving anything in return, effectively decreasing the number of objects within the agent. For evolution rules, if λ appears on the left-hand side, a new object is generated inside the agent. If λ is on the right-hand side, the corresponding object is removed from the agent. These cases model insertion and deletion operations within the system and extend the expressive power of both rule types.

Example 1. *Let the object alphabet be $A = \{a, b, c, d, e\}$ and let the environmental object be $e \in A$. Suppose we have an agent with initial content $\omega = ad$ and the environment initially contains the multiset $w_E = c^2e^\infty$ (i.e., two copies of c and a supply of e that is always sufficient for any computation step).*

Consider the following four rules with their application intensities $\alpha = 1$, so $\lfloor \alpha \rfloor = 1$ for all rules:

- $r_1 = (\mathsf{evo}, a, \lambda, \alpha)$: *Evolution rule deletes object a inside the agent.*
- $r_2 = (\mathsf{evo}, \lambda, b, \alpha)$: *Evolution rule generating a new object b inside the agent.*
- $r_3 = (\mathsf{com}, d, \lambda, \alpha)$: *Communication rule exporting object d from the agent into the environment.*
- $r_4 = (\mathsf{com}, \lambda, c, \alpha)$: *Communication rule importing object c from the environment into the agent.*

Figure 1 illustrates a single computation step in the modified P Colony with Rule Application Intensity.

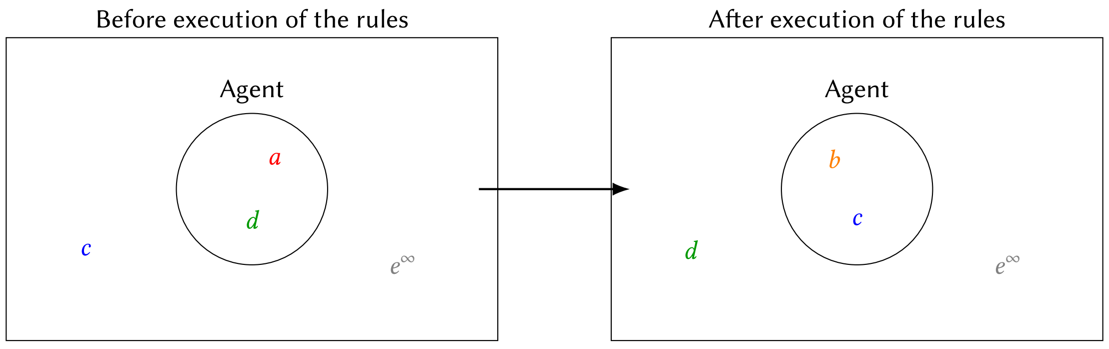

Figure 1: Application of rules r_1, r_2, r_3 and r_4

This example demonstrates all four combinations of λ usage in both types of rules and shows how they affect the contents of the agent and the environment.

The rule application intensity (RAI) of each rule is initially set to 1. As previously mentioned, the value of RAI may change during the computation, depending on the applicability and application of the rule. If the rule is applicable in the current configuration but not applied, its RAI remains unchanged. If the rule is applied, its RAI is updated according to a predefined positive adjustment. Conversely, if the rule is not applicable, its RAI is modified based on a predefined negative adjustment.

An important aspect of our approach is that the rule application intensity (RAI) is defined as a real number rather than an integer. This choice is motivated by the intended applications of Modified

P Colonies, where a more fine-grained mechanism for both promoting and inhibiting rules is desirable. Real values of RAI allow us to capture gradual adjustments in the activity of rules, reflecting subtle changes in the dynamics of the system. In contrast, integer values would provide only coarse control, limiting the ability to model nuanced behaviors that may arise in practical scenarios.

Definition 2. *A* modified P colony with rule application intensity *with n agents, $n \in \mathbb{N}$, is defined as a tuple*

$$\Pi = (A, e, B_1, \dots, B_n, \omega_E, \oplus\delta^+, \ominus\delta^-, F),$$

where:

- *A is a finite alphabet of objects,*
- *$e \in A$ is an* environmental object*, assumed to be present in arbitrary quantity in the environment throughout the computation,*
- *for each $1 \le i \le n$, $B_i = (\omega_i, P_i)$ is the i-th* agent*, where:*
  - *$\omega_i \in A^*$ is the initial multiset of objects in the agent,*
  - *$P_i = [r^i_1, \dots, r^i_{k_i}]$ is a finite sequence of* rules*, each of which is a quadruple*

    $$r = (\tau, \mathsf{lhs}, \mathsf{rhs}, \alpha),$$

    where:
    * *$\tau \in \{\mathsf{evo}, \mathsf{com}\}$ indicates the rule type (evolution or communication),*
    * *$\mathsf{lhs}, \mathsf{rhs} \in A \cup \{\lambda\}$ are the left-hand and right-hand sides of the rule,*
    * *$\alpha \in \mathbb{R}^+$ is the initial application intensity of the rule.*
- *$\omega_E \in (A \setminus \{e\})^*$ is the initial multiset of non-environmental objects in the environment,*
- *$\delta^+, \delta^- \in \mathbb{R}^+$ are the global positive and negative* adjustment parameters,
- *$\oplus$ and $\ominus$ are binary operations on $\mathbb{R}^+$, used to update the application intensity based on rule usage (e.g., addition/multiplication for $\oplus$, subtraction/division for $\ominus$).*
- *F is the set of properties of the result. Members of the set are:*
  - *$f \in A$ - final object - the result is composed only from the objects f, if no final object is specified, then the result of the computation is determined by the occurrences of all objects from the alphabet in the output region.*
  - *o_i where $i \in \{1, \dots, n, E\}$ - output membrane (or the environment)*
  - *result mode: hr - result is taken from output membrane after halting computation only; emr - result is taken after every step of computation; epr - in each step, the system produces a part of the result.*

To capture the evolving state of the system during a computation, we introduce the notion of a *configuration*, which reflects both the current distribution of objects in agents and the environment, as well as the current values of rule application intensities for each agent. Each configuration thus complements the static definition of the system with a dynamic snapshot of its runtime state, allowing us to formally describe transitions and the computational behavior of the model over time.

Definition 3. *A* configuration *of a modified P colony with RAI $\Pi = (A, e, B_1, \dots, B_n, \omega_E, \oplus\delta^+, \ominus\delta^-)$ is a tuple*

$$C = (w_E, (w_1, \boldsymbol{\rho}_1), \dots, (w_n, \boldsymbol{\rho}_n)),$$

where:

- *$w_E \in (A \setminus \{e\})^*$ is the multiset of all non-environmental objects currently present in the environment (the environment is assumed to contain a sufficient number of copies of $e \in A$),*
- *for each $1 \le i \le n$:*
  - *$w_i \in A^*$ is the current multiset of objects inside the i-th agent,*
  - *$\boldsymbol{\rho}_i = (\alpha^i_1, \dots, \alpha^i_{k_i})$ is the vector of rule application intensities corresponding to the rules in the rule sequence $P_i = (r^i_1, \dots, r^i_{k_i})$, where $\alpha^i_j \in \mathbb{R}^+$ for all j.*

Transition Between Configurations

A single transition between configurations in the modified P Colony with RAI proceeds in three phases:

1. **Identification of Applicable Rules:**
   In each agent B_i, we identify the subset of *applicable rules* based on the current multiset w_i of objects and the vector of rule application intensities $\boldsymbol{\rho}_i$.
   A rule $r_j^i = (\tau, \mathsf{lhs}, \mathsf{rhs}, \alpha_j^i)$ is considered applicable if the following conditions hold:
   - The agent contains at least $\lfloor \alpha_j^i \rfloor$ copies of the object lhs, i.e., $\mathsf{lhs}^{\lfloor \alpha_j^i \rfloor} \subseteq w_i$.
   - If the rule is of type com, then the environment w_E contains at least $\lfloor \alpha_j^i \rfloor$ copies of the object rhs, i.e., $\mathsf{rhs}^{\lfloor \alpha_j^i \rfloor} \subseteq w_E$.
2. **Construction of the Maximal Applicable Set:**
   From the union of applicable rules across all agents, a multiset of *selected rules* is non-deterministically constructed. The selection must satisfy the following constraints:
   - Each object in the system (either in agents or in the environment) may be used by at most one rule during this step.
   - The set of selected rules is *maximal* under the above constraint, i.e., no further applicable rule can be added without violating the exclusivity of object usage.
3. **Execution and Updates:**
   All selected rules are executed in parallel. The execution results in the following updates:
   - The contents of the agents and the environment $(w_1, \dots, w_n, w_E)$ are updated based on the type of each applied rule:
     - For an *evolution rule* $(\text{evo}, \mathsf{lhs}, \mathsf{rhs}, \alpha)$, the agent rewrites lhs into rhs inside itself, applied $\lfloor \alpha \rfloor$ times. That is, $\mathsf{lhs}^{\lfloor \alpha \rfloor}$ is replaced by $\mathsf{rhs}^{\lfloor \alpha \rfloor}$.
     - For a *communication rule* $(\text{com}, \mathsf{lhs}, \mathsf{rhs}, \alpha)$, an exchange of objects takes place:
       * The agent removes $\mathsf{lhs}^{\lfloor \alpha \rfloor}$ and receives $\mathsf{rhs}^{\lfloor \alpha \rfloor}$ from the environment,
       * Simultaneously, the environment removes $\mathsf{rhs}^{\lfloor \alpha \rfloor}$ and receives $\mathsf{lhs}^{\lfloor \alpha \rfloor}$ from the agent.
   - The RAI vectors $\boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_n$ are updated as follows:
     - If a rule was selected and applied, its RAI is updated via the *positive adjustment*: $\alpha \mapsto \alpha \oplus \delta^+$.
     - If a rule was applicable but not selected, its intensity remains unchanged.
     - If a rule was not applicable, its intensity is updated via the *negative adjustment*: $\alpha \mapsto \alpha \ominus \delta^-$.

This three-phase process defines a single computation step of the system, where both the multiset contents and the intensities of rule applications evolve according to the agent-environment interaction and usage history of rules.

We now proceed with a formal definition of the update mechanism for the rule application intensity.

Definition 4 (Rule Application Intensity Update). *Let $r = (\tau, \mathsf{lhs}, \mathsf{rhs}, \alpha)$ be a rule in the system, where $\alpha \in \mathbb{R}^+$ is the current application intensity of r. The system is parametrized by two fixed positive real numbers $\delta^+, \delta^- \in \mathbb{R}^+$, and by two binary operations $\oplus, \ominus$ defined over $\mathbb{R}^+$.*

Given a configuration C_t at time step t, the updated application intensity α' of rule r at time $t+1$ is defined as:

$$\alpha' = \begin{cases} \alpha \oplus \delta^+ & \text{if the rule } r \text{ is applied in } C_t, \\ \alpha & \text{if the rule } r \text{ is applicable but not applied in } C_t, \\ \max(1, \alpha \ominus \delta^-) & \text{if the rule } r \text{ is not applicable in } C_t. \end{cases}$$

Result of computation

To associate a result with a computation (i.e., a sequence of configurations), we consider a set of output-related features F that determine how the result is extracted. The result is always derived from the contents of a designated output region—either a specific agent or the environment.

If no terminal object $f \in A$ is specified, the result includes all objects present in the output region at the relevant point of the computation, excluding the occurences of the environmental object e. If a terminal object f is defined, only the multiplicity of f in the output region contributes to the result.

The features in F further specify the timing and structure of the result. In the case of *epr*, the system produces parts of the result incrementally during the computation. In the *emr* type of result, each configuration contributes one element to the final set. In the *halting result* (*hr*), the result is determined solely by the final configuration.

Variants of Modified P Colonies with Rule Application Intensity

Several subclasses of the modified P colonies with rule application intensity (RAI) can be considered by introducing restrictions on the structure or dynamics of the system.

One possible variant restricts the number of objects that can be stored inside each agent. This bound may be either an upper bound ($\leq k$) or an exact constraint ($= k$), similar to the classical P Colony model, where the capacity of each agent is fixed.

Another subclass includes systems with constant RAI, where the adjustment factors δ^+ and δ^- are both set to $\times 1$. In this case, the intensity does not evolve throughout the computation and each rule can be applied at most once per step, similar to traditional rule-based systems.

Further constraints may involve limiting the number of agents in the system or placing a bound on the number of rules associated with each agent. These restrictions are useful for studying the computational power of more resource-constrained models and for exploring trade-offs between system complexity and expressiveness.

A special variant is the *λ-free* system, where the empty string symbol λ is not allowed on either side of any rule.

4. Illustrative Examples of Modified P Colonies with RAI

In this section, we present a collection of illustrative examples of modified P colonies with RAI to demonstrate the expressive power and behavioral diversity of different system variants introduced in the previous section. Each example is designed to highlight a specific constraint or feature such as fixed or unfixed RAI or λ-free design.

We present a simple modified P colony with rule application intensity (RAI) that generates the set $\{2^n - 1 \mid n \geq 0\}$. The system uses a single agent and a single evolution rule. The rule application intensity increases exponentially, doubling after each successful application. The system outputs the number of terminal objects a in the agent after each computation step.

Example 2. *We define a modified P colony with rule application intensity*

$$\Pi_1 = (A, e, B_1, \omega_E, \oplus\delta^+, \ominus\delta^-, F),$$

that generates the set $\{2^n - 1 \mid n \geq 0\}$*, where:*

- $A = \{a, e\}$ *is the alphabet of objects, with* a *being the terminal object and* e *the environmental object,*
- e *is the environmental object, available in sufficient quantity in the environment,*
- $B_1 = (\omega_1, P_1)$ *is the only agent in the system, where:*
  - $\omega_1 = \lambda$ *(the agent starts empty),*

Step	w_E (Environment)	w_1 (Agent Content)	ρ_1 (RAI)
0	λ	λ	1
1	λ	a	2
2	λ	aaa	4
3	λ	$aaaaaaa$	8
4	λ	$aaaaaaaaaaaaaaa$	16
5	λ	$aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$	32

Table 1
Configuration table of Π_1

  – *$P_1 = (r)$ is a single-rule sequence, with:*

$$r = (\text{evo}, \lambda, a, 1)$$

  which creates object a inside the agent,

- *$\omega_E = \lambda$ (the environment is initially empty of non-environmental objects),*
- *$\delta^+ = 2$ and $\delta^- = 2$ are the adjustment parameters,*
- *$\oplus = \times$ and $\ominus = /$ (application intensity is multiplied by 2 when the rule is applied, and divided by 2 when it is not),*
- *$F = \{o_1, f = a, emr\}$ — the result is produced in the agent B_1, terminal object is a, and one element of the result set is produced in each step (emr).*

This system works as follows: in each configuration, the agent applies the rule r as many times as set by the integer part of its application intensity (RAI). Since the left-hand side of the rule is λ, the agent creates new objects a without consuming any. Initially, the RAI is 1, and it is multiplied by 2 after each application. Therefore, the number of a objects generated in the n-th step is 2^{n-1}, and the total accumulated number is $2^n - 1$. The output is read from the agent's content in each step, as specified in F.
The first six configurations of the system are shown in the Table 1.

We now construct a modified P colony with rule application intensity that generates a finite set of natural numbers. The system is *lambda-free* and operates with a *constant RAI* — that is, the application intensity of each rule remains unchanged during the computation.

Example 3. *We define a modified P colony with rule application intensity that generates a finite set of natural numbers $\{r_1, r_2, \dots, r_s\}$ such that $r_1 > r_2 > \cdots > r_s$. Let $M = r_1$ and $m = r_s$. The system*

$$\Pi_2 = (A, e, B_1, \dots, B_{M-m}, \omega_E, \oplus\delta^+, \ominus\delta^-, F),$$

is constructed as follows:

- *Alphabet: $A = \{a, h_1, \dots, h_s, e\}$,*
- *The environmental object is e,*
- *The environment initially contains M objects a, i.e., $\omega_E = a^M$,*
- *Number of agents: $M - m$,*
- *Each agent is associated with a number from S and with a rule type depending on the index i, $1 \leq i \leq (M - m)$ of the agent. There are two types of rules:*
  1. $(\text{com}, h_i, a, 1)$,
  2. $(\text{evo}, h_i, h_{i+1}, 1)$.
- *Initially, each agent has object h_1: $\omega_i = h_1$, $1 \leq i \leq (M - m)$.*
- *Adjustments: $\delta^+ = \delta^- = 1$, i.e., RAI is constant,*

- *Result configuration: $F = \{o_E, f = a, emr\}$, i.e., objects a in the environment (which is the output region) form the result, observed in each step.*

At step i, exactly $r_i - r_{i+1}$ agents use their communication rule to move object a from the environment into the agent (which removes it from the output), while the remaining agents evolve from h_i to h_{i+1}. After step i, the environment contains r_{i+1} copies of object a. After $s-1$ steps, all agents hold object h_s, and no further rule is applicable — the computation halts.

Let the set to generate be $S = \{6, 5, 2, 0\}$ with

$$r_1 = 6, \quad r_2 = 5, \quad r_3 = 2, \quad r_4 = 0.$$

Modified P Colony with RAI

$$\Pi_{2S} = (A, e, B_1, \dots, B_6, \omega_E, \oplus\delta^+, \ominus\delta^-, F),$$

- Alphabet: $A = \{a, h_1, h_2, h_3, h_4, e\}$,
- Environmental object: e,
- Initial environment content: $\omega_E = a^6$,
- Number of agents: $6 - 0 = 6$,
- Agents' initial contents: $\omega_j = h_1$ for all $j = 1, \dots, 6$,
- Rules assigned to agents:
  - Agent B_1 has only one rule: $(\text{com}, h_1, a, 1)$.
  - Agents B_2, B_3, B_4 have equivalent sets of rules: $(\text{evo}, h_1, h_2, 1)$; $(\text{com}, h_2, a, 1)$.
  - Agents B_5, B_6 have equivalent sets of rules: $(\text{evo}, h_1, h_2, 1)$, $(\text{evo}, h_2, h_3, 1), (\text{evo}, h_3, h_4, 1)$, $(\text{com}, h_4, a, 1)$.
- Adjustment parameters: $\delta^+ = \delta^- = 1$ (constant RAI),
- operators are $\times$ and $/$.
- Result is the count of a in the environment,
- The result property $F = \{o_E, f = a, emr\}$.

At each step i, the specified number of agents apply the communication rule to transfer a from the environment to their internal content (decreasing the count of a in the environment), while the others evolve their internal object h_i to h_{i+1}. After step i, the environment contains r_{i+1} copies of a. After $s - 1 = 3$ steps, no further rules are applicable, and the computation halts.
The first four configurations of the system are shown in the Table 2.

Step	**Environment w_E**	**Contents of Agents**
0	$aaaaaa$	$[h_1, h_1, h_1, h_1, h_1, h_1]$
1	$aaaaa$	$[a, h_2, h_2, h_2, h_2, h_2]$
2	aa	$[a, a, a, a, h_3, h_3]$
3		$[a, a, a, a, a, a]$

Table 2
Configuration table of Π_{2S}

Note that at step 3 the environment has $r_4 = 0$ objects a. The system successfully generates the set $\{6, 5, 2, 0\}$ in decreasing order as the number of a objects in the environment after each step.

Example 4. *The next example outlines a design of a system that can serve as a game engine for a class of games known as* Idle Games. *In such games, active entities (e.g., production units) gradually increase their performance over time, often by paying for upgrades using accumulated in-game currency.*

In this modified P colony with RAI, the rule application intensity is not increased automatically but is instead driven by a special *input symbol* inserted into the environment. This input represents an upgrade and can be added only if the system has accumulated a sufficient amount of money objects. The insertion of an upgrade object may be modeled by a simple rewriting rule such as:

$$\texttt{money}^{\text{value}} \cdot \texttt{upgrade}_i \to \texttt{upgrade}_{i+1}.$$

Once the symbol $\texttt{upgrade}_{i+1}$ is present in the environment, it can trigger an update of the RAI for the relevant agents via a special rule. The RAI is modified using the adjustment function δ^+, which is not constant but defined as a function of the number of times an upgrade has occurred. For example:

$$\delta^+(x) = \delta^+(x-1) + k \quad \text{or} \quad \delta^+(x) = \delta^+(x-1) \cdot \left(1 + \frac{1}{x}\right),$$

where $x - 1$ is the number of completed upgrades.

This design offers a flexible and controlled mechanism for performance progression:

- The rate of production can be tuned based on the game's economic model,
- The system allows differentiated upgrades with increasing cost and benefit,
- RAI is no longer a fixed parameter but a dynamic value, modifiable through environmental interaction.

Overall, this type of system combines the computational model of modified P colonies with an interactive upgrade mechanic, bringing it closer to the logic and dynamics of *Idle Games*. Figure 2 depicts a production chain utilizing agents in the Modified P Colony.

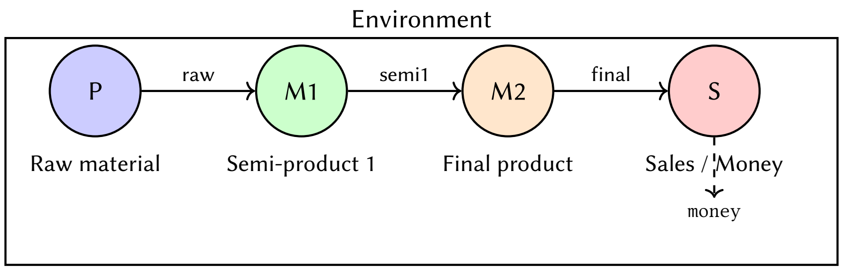

Figure 2: Idle Game Production Chain Using Agents in a Modified P Colony

5. Conclusion

In this paper, we introduced a novel computational model — the *Modified P colony with Rule Application Intensity (RAI)*. We presented its formal definition, described the operational semantics in terms of configurations and rule execution phases, and illustrated the behavior of the system through a series of examples. We also introduced several variants of this model, such as systems with a fixed number of objects per agent, constant RAI updates, λ-free systems, or systems with restrictions on agent count or rule set size.

Currently, our ongoing work focuses on three main directions. First, we are investigating the *computational power* of the various subclasses of modified P colonies with RAI, aiming to classify their capabilities in relation to classical models of computation. Second, we are developing a *software simulator* capable of executing computations in these systems, which will support experimentation and visualization. Finally, we are designing a prototype of *idle-style game* whose core engine will be based on the principles of P colony computations, offering a practical and interactive application of the model.

Acknowledgments

The research was supported by the Silesian University in Opava under the Student Funding Plan, project SGS/9/2024.

Declaration on Generative AI

During the preparation of this work, the author(s) used Chat-GPT-4 in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] J. Kelemen, A. Kelemenová, Gh. Păun, 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), Boston, Massachusetts, USA, 2004, pp. 82–86.

[2] Gh. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (2000) 108–143. doi:10.1006/jcss.1999.1693.

[3] J. Kelemen, A. Kelemenová, A grammar-theoretic treatment of multiagent systems, Cybern. Syst. 23 (1992) 621–633. doi:10.1080/01969729208927485.

[4] Gh. Păun, G. Rozenberg, A. Salomaa, The Oxford Handbook of Membrane Computing, Oxford University Press, Inc., New York, NY, USA, 2010.

[5] L. Ciencialová, E. Csuhaj-Varjú, L. Cienciala, P. Sosík, P colonies, Journal of Membrane Computing 1 (2019) 178–197. URL: https://doi.org/10.1007/s41965-019-00019-w. doi:10.1007/s41965-019-00019-w.

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

[7] J. Kelemen, A. Kelemenová, On P colonies, a biochemically inspired model of computation, in: Proc. of the 6^{th} International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH, Hungary, 2005, pp. 40–56. URL: http://conf.uni-obuda.hu/mtn2005/Kelemen.pdf.

[8] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun, Gy. Vaszil, Computing with cells in environment: P colonies, Journal of Multiple-Valued Logic and Soft Computing 12 (2006) 201–215.