Theoretical Computer Science 454 (2012) 95–108

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

An excursion in reaction systems: From computer science to biology

Luca Corolli [a], Carlo Maj [a], Fabrizio Marini [a], Daniela Besozzi [b,*], Giancarlo Mauri [a]

[a] Università degli Studi di Milano-Bicocca, Dipartimento di Informatica, Sistemistica e Comunicazione, Viale Sarca 336, 20126 Milano, Italy
[b] Università degli Studi di Milano, Dipartimento di Informatica e Comunicazione, Via Comelico 39, 20135 Milano, Italy

ARTICLE INFO

Keywords:
Reaction systems
Boolean functions
Simplification methods
Tower of Hanoi
lac operon

ABSTRACT

Reaction systems are a formal model based on the regulation mechanisms of facilitation and inhibition between biochemical reactions, which underlie the functioning of living cells. The aim of this paper is to explore the expressive power of reaction systems as a modeling framework, showing how their basic assumptions and properties can be exploited to formalize computer science and biology oriented problems. In this view, we first provide a reaction-based description of an iterative algorithm to solve the Tower of Hanoi puzzle. Then, we show how the regulation of gene expression in the *lac* operon, involved in the metabolism of lactose in *Escherichia coli* cells, can be formalized in terms of reaction systems. Finally, we present a method to derive, given a reaction system with n reactions, a functionally equivalent system with $n' \leq n$ reactions using simplification methods of boolean expressions. Some final remarks and directions for future research conclude the paper.

1. Introduction

In the last ten years the interaction between computer science and biology has been increasing in both directions. On the one hand, bioinformatics and computational biology try to provide biology with appropriate tools to handle the huge mass of data generated in the laboratories and to extract information and knowledge from them. On the other hand, biological mechanisms and processes have given to computer scientists inspiration for innovative models of computation, such as neural networks [1], evolutionary programming [2], molecular computing [3], cellular automata [4], membrane systems [5], for the purpose of solving computationally difficult problems. But even more important is the increasing awareness that biological processes have a computational nature and that we have to study them in terms of information processing, in order to achieve a full comprehension of their functioning.

Biochemical reactions are the "primitive" information processing events inside living cells: they take place in huge number and their complex interactions determine the correct or the anomalous functioning of the cell itself. Reaction systems were introduced by Andrzej Ehrenfeucht and Grzegorz Rozenberg [6,7] as a formal framework based on the basic axioms underlying the functioning of biochemical reactions, in order to investigate processes driven by the interactions between such reactions in living cells; in particular, they assume as fundamental the regulation mechanisms of facilitation and inhibition between reactions. Starting from the above seminal papers, the research on reaction systems is nowadays growing in different directions, and gaining an ever increasing attention [8–13].

In reaction systems, two main assumptions are made concerning the chemistry of cells. The first consists in a "threshold supply" of molecules, that is, either an element is present and then there is enough amount of it to let reactions occur, or

* Corresponding author. Tel.: +00393289129088; fax: +003903519902810.
E-mail addresses: luca.corolli@disco.unimib.it (L. Corolli), carlo.maj@disco.unimib.it (C. Maj), fabrizio.marini@disco.unimib.it (F. Marini), besozzi@dico.unimi.it (D. Besozzi), mauri@disco.unimib.it (G. Mauri).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.04.003

an element is not present and then no reaction having this element within its set of reagents can take place. The second assumption consists in the "no permanency" of elements, i.e., if an element is not sustained by any reaction, then it ceases to exist. In other terms, these basic assumptions rely on a qualitative rather than quantitative description of the interaction between reactions. In particular, reaction systems assume the absence of conflicts between the reactions that can take place in a given state, which is determined by the set of chemical elements appearing in the system at that step. The basic scheme of reaction systems can be incrementally extended, for instance by assigning specific measurement functions to the states, in order to add quantitative parameters such as time values, molecular amounts or mass, etc. [14].

By modeling the interaction between individual reactions, this formalism allows one to investigate how the states of the system change over (discrete) time steps. Since all reactions that can take place in the system are actually applied without considering any possible concurrency on the consumption of elements, the evolution of a reaction system is regulated by a core deterministic assumption, whereby the next state is univocally determined by the current state of the system. So doing, a dynamic behavior of the reaction system is derived by simply considering the cooperation of reactions through their mutual facilitation or inhibition. In this sense, reaction systems differ from other abstract formalisms of natural computing, since they do not work in a well defined or structured environment, but rather they *create* their own environment of interactive processes.

In this work, we concentrate on the basic assumptions and properties of reaction systems and we clearly show how they can be exploited to formalize computer science and biology oriented problems, therefore proving their expressive power and flexibility for a broad range of applications. We start by briefly recalling in Section 2 the formal definitions of reactions and reaction systems. In Section 3 we use reaction systems as a programming environment in game theory, giving a reaction-based program to solve the Tower of Hanoi puzzle [15]. The no permanency property of elements and the proper use of inhibitors are essential for the correct functioning of this program; in particular, these properties will be exploited to simulate the execution of nontrivial computational steps and to ensure that only legal moves of the puzzle are carried out from time to time. In Section 4 the formalism of reaction systems is used to model the regulation of gene expression in the *lac* operon [16], a genetic regulatory network involved in the metabolism of lactose in *Escherichia coli* cells. In spite of the complex mechanisms that control this metabolic pathway, we will propose an original description of the *lac* operon with a reaction system consisting of a few reactions and elements, defined by exploiting only the basic properties of reaction systems. Then, in Section 5 we investigate the relationships between reaction systems and boolean functions, that were first studied in [6], where Ehrenfeucht and Rozenberg show how to translate boolean vector functions into reaction systems and vice versa. Here, we will take a step forward, showing how to derive, given a reaction system with n reactions, a functionally equivalent system with $n' \leq n$ reactions using simplification methods for boolean expressions [17]. This method might be useful for the minimization of large reaction systems, for instance those built up with the bottom-up modularity principle [6], a method based on set-theoretic union for the construction of complex reaction systems starting from elementary component systems. Finally, we provide a comprehensive discussion of the capabilities of reaction systems and address some topics for future research.

The work presented hereby is the result of some investigations on reaction systems that have arisen after the short course entitled "A formal framework for processes inspired by biochemistry", held by Grzegorz Rozenberg in Milano in April, 2011.

2. Basic notions on reaction systems

In this section we recall the basic notions and notation of reaction systems that will be used in the following sections. We refer to [6,7,11,14] for a complete overview on reaction systems.

A *reaction system* is an ordered pair $\mathcal{A} = (S, A)$, where S is a finite set of elements, called *entities*, and A is a finite set of *reactions* in S. The set S is called the *background (set)* of $\mathcal{A}$; its elements represent the chemical species (e.g., atoms, ions, molecules) in an environment where the current entities change over time.

A *reaction* in S is an ordered triplet $a = (R_a, I_a, P_a)$ of nonempty finite sets, such that $R_a, I_a, P_a \subseteq S$ and $R_a \cap I_a = \emptyset$. The sets R_a, I_a and P_a are called, respectively, the *reactant set*, the *inhibitor set*[1] and the *product set* of a. Given a set A of reactions, we denote $R_A = \bigcup_{a\in A} R_a$, $I_A = \bigcup_{a\in A} I_a$, $P_A = \bigcup_{a\in A} P_a$.

For a reaction a and a finite set $T \subseteq S$, we say that a is *enabled by* T if $R_a \subseteq T$ and $I_a \cap T = \emptyset$; otherwise, a is not enabled by T. In other words, a reaction a is enabled by a set T if T separates the set of reactants R_a from the set of inhibitors I_a of a. If a is enabled by T, then the *result* of a on T is a function $res_a : 2^S \to 2^S$ that maps the set T over the set of products of a, that is, $res_a(T) = P_a$. If a is not enabled by T, then $res_a(T) = \emptyset$. Stated otherwise, the map $res_a(T)$ describes how reaction a contributes to the change of the current state T of the reaction system $\mathcal{A}$: if a is enabled by T, the next state of $\mathcal{A}$ will change according to P_a, otherwise a contributes nothing to the next state. Similarly, for a set of reactions A and a set T, we say that A is enabled on T if each $a \in A$ is enabled by T, that is, if T separates R_A from I_A; if this holds, we denote $res_A(T) = \bigcup_{a\in A} res_a(T)$ the result of A on T. Finally, given a reaction system $\mathcal{A}$, the result of $\mathcal{A}$ on T, denoted $res_{\mathcal{A}}(T)$, is defined by $res_{\mathcal{A}}(T) = res_A(T)$.

When applying a set of reactions in a given state T, there is no notion of conflict in the interaction of the reactions: we assume that either an element $s \in S$ is present in T and then there is enough "amount" of s to enable every reaction in which

[1] In what follows, when we are not interested in formally expressing the nature of specific inhibitor elements that occur in the set I_a of some reaction a, we will simply denote that reaction by the triplet $(R_a, \{\ldots\}, P_a)$, where $\{\ldots\}$ stands for some "dummy inhibitor".

s appears as a reagent, or s is not present in T and then no reaction having this element within its set of reagents is enabled by T. Stated otherwise, in the basic setup of reaction systems we rely on a "threshold supply" of elements. Moreover, the notion of counting elements is not considered in reaction systems: this theoretical framework gives a qualitative rather than a quantitative representation of chemical systems.

Another important assumption of reaction systems is that there is *no permanency* of elements: if an element is not a reactant for any enabled reaction, then it ceases to exist and will not appear in the next state of the system. Therefore, the only way to maintain an element in the system is to sustain its presence by including it in the product set of some reaction. Formally, given a reaction $a \in A$ and an element $s \in S$ such that $s \notin P_a$ and $s \in T \backslash R_a$, the element s will vanish unless it is produced by some other reaction in A that is enabled by T (that is, there exists $a' \in A$ such that $s \in P_{a'}$ and a' is enabled by T).

Given a reaction system $\mathcal{A}$, we can define its dynamic behavior through the notion of *interactive process*. Informally, this is a sequence of states, where each state is a subset of S defined as the union of two sets: the result of transforming the previous state by the reactions in A, and a context set which can be used to represent an "input" of elements of S from the environment. In other terms, the context sets allow to formalize the notion of open (dynamic) systems. Formally, an interactive process is a pair $\pi = (\gamma, \delta)$ of finite sequences such that, for some $n \geq 1$, $\gamma = C_0, \ldots, C_n$, $\delta = D_0, \ldots, D_n$, where $C_0, \ldots, C_n, D_0, \ldots, D_n \subseteq S$, $D_0 = \emptyset$, $D_i = res_{\mathcal{A}}(C_{i-1} \cup D_{i-1})$, with $i = 1, \ldots, n$. The sequences γ and δ are called the *context sequence* and the *result sequence* of π, respectively. The set $W_i = C_i \cup D_i$, $i = 0, \ldots, n$, is called a *state* of $\mathcal{A}$, and the sequence $W_0, \ldots, W_n$ is the *state sequence* of the interactive process π. If $C_i \subseteq D_i$ for each $i = 1, \ldots, n$, then the state sequence is said to be context-independent. The sequence $E_0, \ldots, E_{n-1} \subseteq S$, such that $E_i = en_{\mathcal{A}}(W_i)$, $i = 0, \ldots, n-1$, is called the *activity sequence* of π; each set E_i represents the set of reactions from A that are enabled in the state W_i.

Finally, we recall the notion of functional equivalence between reaction systems. Given a reaction system $\mathcal{A} = (S, A)$, we say that two reactions $a, b \in A$ are *functionally equivalent* in S, denoted by $a \sim_S b$, if for each $T \subseteq S$, $res_a(T) = res_b(T)$. This notion can be extended to sets of reactions and to reaction systems: given two arbitrary sets of reactions A, B over the same background S, we say that A, B are functionally equivalent in S, denoted by $A \sim_S B$, if for each $T \subseteq S$, $res_A(T) = res_B(T)$. Similarly, given two reaction systems $\mathcal{A} = (S, A)$, $\mathcal{B} = (S, B)$ over the same background, we say that $\mathcal{A}$, $\mathcal{B}$ are functionally equivalent, denoted by $\mathcal{A} \sim \mathcal{B}$, if $A \sim_S B$. In [6] it is shown that the functional equivalence problem for arbitrary sets of reactions, and hence for reaction systems, is co-NP-complete.

3. The Tower of Hanoi puzzle formalized as a reaction system

In this section we describe how to formalize the Tower of Hanoi puzzle as a reaction system. The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. As described in [15], the puzzle consists of three vertical pegs set into a board, and a number of disks graded in size. These disks are initially stacked on one of the pegs so that the largest rests at the bottom of the stack, the next largest in size atop it, and so on, ending with the smallest disk placed at the top. A player can shift the disks from one peg to another one at a time, provided that no disk rests upon any other smaller disk. The purpose of the game is to transfer the tower of disks from the peg upon which the disks initially rest to one of the other pegs.

A key to facing this puzzle is to recognize that it can be solved by a "divide et impera" strategy, breaking the problem down into a collection of smaller and smaller problems until a solution is reached. This suggests a recursive solution. Nevertheless, there are different iterative algorithms to solve the game; here, we consider the solution proposed by Franklin [18], which is described in the next section.

3.1. Franklin's algorithm

In [18], Franklin defines an iterative method for the Tower of Hanoi puzzle that can be derived from a simple set of principles. We start by assuming the following conventions:

S1 Number the disks from 1 (the smallest) to N (the largest);
S2 The three pegs are ordered so that the actions of moving a disk to the right and to the left are meaningful.

Then, the whole solution derives from these four principles:

P1 Move odd-numbered disks only to the right and even-numbered disks only to the left;
P2 Do not move the same disk twice in a row;
P3 Do not place a larger disk on top of a smaller one;
P4 Move only those disks that are at the top of one of the pegs.

There is only one legal move at any time, and this fact can be proved by induction. The basis of the induction consists in the initial state where only disk 1 can be moved. The inductive step consists in assuming that the assertion is true for state W_t, so it must also hold for state W_{t+1}, $t \geq 0$. For the latter state we can have three different cases:

- Two pegs are occupied. On top of these pegs there are disk 1 and disk j. One of them has been moved in the previous step, by **P2** we can only move the other one.

- Three pegs are occupied. On top of them there are disk 1, disk j and disk k, with $1 < j < k$. By **P3** disk k has no legal origin (1 and j are both smaller than k) and it cannot be moved in this step for the same reason. One of the other two disks (1 or j) has been moved in the previous step. By **P2** we can only move the remaining disk.
- Only one peg is occupied. We are in the final state, all the disks are stacked and only disk 1 can be moved. If we want to stop the system, we need to treat the final state with some specific rules.

3.2. The Tower of Hanoi reaction system

In our reaction system we represent disks, pegs and states as follows. Following convention **S1**, we number the *disks* from 1 to N, where 1 is the smallest disk and N is the largest one. Following convention **S2**, the three *pegs* are coded as $\alpha \in \{A, B, C\}$, and we use the couple αk to denote that disk k is placed on peg α, for some $k \in \{1, \ldots, N\}$ and $\alpha \in \{A, B, C\}$. Moreover, we use "$\alpha_{\rightarrow}$" to indicate the peg on the right of α, and "$\alpha_{\leftarrow}$" for the peg on its left. The order is circular: if $\alpha = C$ then $\alpha_{\rightarrow}$ is equal to A and if $\alpha = A$ then $\alpha_{\leftarrow}$ is equal to C. Similarly, "$\alpha_{\rightrightarrows}$" and "$\alpha_{\leftleftarrows}$" indicate the peg two positions on the right of α, and the peg two positions on the left of α, respectively. Namely, the notation $\alpha_{\rightarrow}k$ and $\alpha_{\rightrightarrows}k$ ($\alpha_{\leftarrow}k$ and $\alpha_{\leftleftarrows}k$, respectively), with $\alpha \in \{A, B, C\}$ and $k \in \{1, \ldots, N\}$, means that disk k is on the peg that is placed one position and two positions to the right (one position and two positions to the left, respectively) of peg α. Finally, we represent the *state of the game* with a set of N couples $\{\alpha_1 1, \ldots, \alpha_N N\}$, where $\alpha_1, \ldots, \alpha_N \in \{A, B, C\}$ not necessarily distinct.

We highlight here that this coding system – together with the way reactions are defined according to Franklin's algorithm (see Sections 3.2.1–3.2.3), which ensures that only one legal move is possible at every step – will not permit the formation of illegal states. For instance, considering $N = 3$, the state $\{A1, A2, A3\}$ can only be interpreted as the state where all the disks are on the first peg (A) stacked in ascending order (1, 2, 3 from top to bottom).

In the following, even though k and the couple αk describe different things (a disk, and a disk in a certain location), we will refer to both of them as "disk" where this does not produce ambiguity.

We want to define a reaction system that, given a legal initial state, produces the correct sequence of states to solve the puzzle. The sketch of our approach is as follows. For each step, we identify the set F of disks that cannot be moved. We write a set of reactions to ensure that the couples $(location, f)$ with $location \in \{A, B, C\}$ and $f \in F$, will not vanish because of the "no permanency property": we call this process the *propagation* of the disks. We identify the disk d that has to be moved, and write a set of reactions to produce the new couple $(destination, d)$ and a "suspension signal" b_d, that we use to remember the last disk that has been moved. In this case, we will exploit the no permanency property of reaction systems to remove both disk d and the expired signal b_d.

3.2.1. Covered, suspended and blocked disks

In each state, the set F of disks that do not move consists in covered disks, suspended disks and blocked disks. *Covered disks* are disks that are not at the top of a peg, by **P4** these disks cannot be moved. To check if αk is a covered disk we search for all the couples with the same peg label and disk number less than k. If we find one couple of this type, the disk is covered and we propagate it. Disk 1 is never covered. The set of reactions to deal with covered disks is:

$$A_{covered} = \{(\{\alpha i, \alpha j\}, \{stop\}, \{\alpha i\}) \mid \alpha \in \{A, B, C\}, i = 2, \ldots, N, j < i\}.$$

Suspended disks are disks that have been moved in the previous step, by **P2** these disks cannot be moved. To check if αk is a suspended disk we search for his suspension signal b_k. If we find the signal we propagate the disk and let the signal vanish (by no permanency). All the disks can be suspended. The set of reactions for the suspended disks is:

$$A_{suspended} = \{(\{\alpha i, b_i\}, \{stop\}, \{\alpha i\}) \mid \alpha \in \{A, B, C\}, i = 1, \ldots, N\}.$$

Blocked disks are disks that do not have any legal destination, by **P3** these disks cannot be moved. To check if αk is a blocked disk we search for couples with a different peg label and disk number less than k. If we find one such couple for each of the two remaining pegs, the disk is blocked and we propagate αk to the next state. Disk 1 and disk 2 are never blocked. The set of reactions for the blocked disks is:

$$A_{blocked} = \{(\{\alpha i, \alpha_{\rightarrow} j, \alpha_{\rightrightarrows} m\}, \{stop\}, \{\alpha i\}) \mid \alpha \in \{A, B, C\},\ i = 2, \ldots, N, j < i, m < i, m \neq j\}.$$

The reactions belonging to the sets $A_{covered}$, $A_{suspended}$, $A_{blocked}$ are inhibited by the entity *stop*, that is produced only when the last step of the game is reached, as explained later on (Section 3.2.3). When this state is reached, these reactions are not enabled and all the disk couples will vanish, thus granting the end of the game.

3.2.2. Movement of disks

In general, the reactions for the movement of disks have the form:

$$(\{\text{disk}\}, \{\text{propagation elements}\}, \{\text{destination, suspension signal}\}),$$

meaning that given a disk as reagent, we produce a destination couple and the suspension signal for that disk, unless some propagation elements occur in the same state of the system to inhibit the application of the reaction.

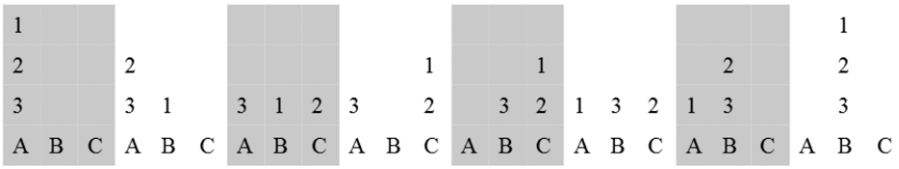

Fig. 1. A solution to the Tower of Hanoi problem with three disks.

Table 1
The Tower of Hanoi reaction system dynamics.

Index n	*Context set*	*Result set*	*State*
0	$A3, A2, A1$	$\emptyset$	$A3, A2, A1$
1	$\emptyset$	$A3, A2, B1, b_1$	$A3, A2, B1, b_1$
2	$\emptyset$	$A3, B1, C2, b_2$	$A3, B1, C2, b_2$
3	$\emptyset$	$A3, C2, C1, b_1$	$A3, C2, C1, b_1$
4	$\emptyset$	$B3, C2, C1, b_3$	$B3, C2, C1, b_3$
5	$\emptyset$	$A1, B3, C2, b_1$	$A1, B3, C2, b_1$
6	$\emptyset$	$A1, B3, B2, b_2$	$A1, B3, B2, b_2$
7	$\emptyset$	$B3, B2, B1, b_1, stop$	$B3, B2, B1, b_1, stop$
8	$\emptyset$	$\emptyset$	$\emptyset$
...	$\emptyset$	$\emptyset$	$\emptyset$

If there is nothing against the propagation of a disk, we move it. By **P1** we need two sets of reactions, one for even-indexed disks and one for odd-indexed disks, plus one reaction for disk 1:

$$A_{even} = \{(\{\alpha i\}, \{(\alpha(i-1), \ldots, \alpha 1), b_i, (\alpha_{\leftarrow}(i-1), \ldots, \alpha_{\leftarrow} 1)\}, \{\alpha_{\leftarrow} i, b_i\}) \mid \alpha \in \{A, B, C\}, i \text{ is even}\},$$
$$A_{odd} = \{(\{\alpha i\}, \{(\alpha(i-1), \ldots, \alpha 1), b_i, (\alpha_{\rightarrow}(i-1), \ldots, \alpha_{\rightarrow} 1)\}, \{\alpha_{\rightarrow} i, b_i\}) \mid \alpha \in \{A, B, C\}, i \text{ is odd}, i > 1\},$$
$$A_{disk1} = \{\{\alpha 1\}, \{b_1\}, \{\alpha_{\rightarrow} 1, b_1\}\}.$$

In each reaction belonging to the sets A_{even} and A_{odd} there are three sets of inhibitor entities, having the following meaning. The first set of inhibitors says that the disk is not covered, the second set says that the disk is not suspended, the third set says that the disk is not blocked. Whenever enabled, these reactions produce a destination couple and a suspension signal. Note that the reaction in the set A_{disk1} is just a special case of the reactions in the set A_{odd} that requires the single inhibitor b_1.

3.2.3. Identification of the final state

We need to identify the second–last state to produce a stopping signal. It is easy to show that disk 1 is the first and the last disk that moves; what distinguishes the second–last state from the second state of the system is the absence of the suspension signal b_1. Using the following two reactions we are able to block the system properly:

$$a_{stop1} = (\{\alpha 2, \ldots, \alpha N, \alpha_{\rightarrow} 1\}, \{b_1\}, \{stop\}),$$
$$a_{stop2} = (\{\alpha 2, \ldots, \alpha N, \alpha_{\rightrightarrows} 1\}, \{b_1\}, \{stop\}).$$

Reaction a_{stop1} is applied when disk 1 is on the peg placed to the right of peg α, while reaction a_{stop2} is applied when disk 1 is on the peg placed two positions to the right of α. Whenever one of these reactions is enabled, the *stop* entity is produced, and no other reaction will be enabled in the system, therefore determining the end of the game. In fact, movement reactions are not active in the final state because the only disk that can be moved (disk 1) has been moved in the previous step and so it is suspended.

3.2.4. Dynamic behavior of the Tower of Hanoi reaction system

Summing up, the reaction system $\mathcal{A}_{Hanoi} = (S, A)$ that formalizes the Franklin's algorithm for the solution of the Tower of Hanoi puzzle consists of the background set

$$S = \{\alpha i, \alpha_{\rightarrow} i, \alpha_{\rightrightarrows} i, \alpha_{\leftarrow} i, \alpha_{\lessdot} i, b_i, stop \mid \alpha \in \{A, B, C\}, i \in \{1, \ldots, N\}\},$$

and the set of reactions

$$A = A_{covered} \cup A_{suspended} \cup A_{blocked} \cup A_{even} \cup A_{odd} \cup A_{disk1} \cup \{a_{stop1}, a_{stop2}\}.$$

The dynamic behavior of the proposed reaction system is reported in Table 1 for the case of $N = 3$, starting from the initial state given by $\{A1, A2, A3\}$, where the three disks are on the first peg (A) stacked in ascending order (see Fig. 1). The interactive process considered in this case is context-independent, since we do not need any additional inflow of elements to execute the solution of the Tower of Hanoi puzzle. As it can be seen by comparing Fig. 1 with the last column of Table 1, $\mathcal{A}_{Hanoi}$ produces the correct sequence of states and movements to solve the puzzle.

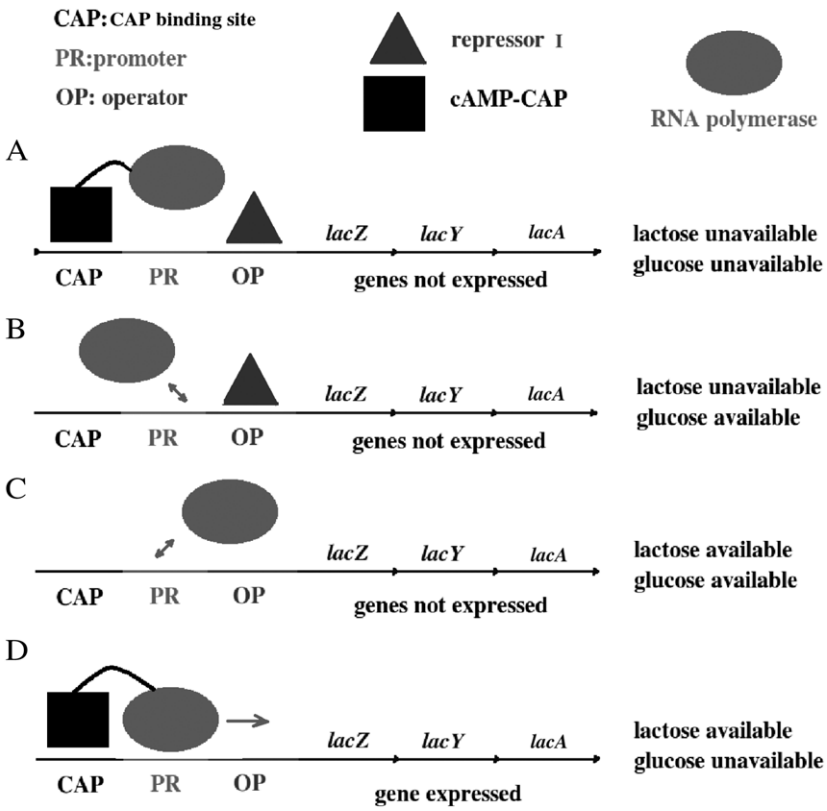

Fig. 2. Regulation of the *lac* operon expression when the environment provides: (A) neither glucose nor lactose; (B) only glucose; (C) both lactose and glucose; (D) only lactose.

4. The *lac* operon expression formalized as a reaction system

In this section we give a formalization of the *lac* operon expression in terms of reaction systems. In genetics, an *operon* is a functional unit of genomic DNA containing a cluster of genes, which are all under the control of a specific regulatory signal. The genes in an operon are usually involved in the same biological process and their coordinate expression is required in order to accomplish a specific biochemical task. The *lac* operon is involved in the metabolism of lactose in *Escherichia coli* cells, where it allows the effective digestion of lactose only when this process is useful for the bacterium [19]. For this reason, the *lac* operon is under the control of an integrated signaling system which leads to the activation or inhibition of gene expression in dependence on the nutrients available in the environment [16].

4.1. The lac operon

The *lac* operon is constituted by three adjacent structural genes and some regulatory components (Fig. 2). The three genes are called *lac*Z, *lac*Y and *lac*A: they encode for two enzymes (hereby denoted Z and A) and a transporter (Y) involved in the digestion of lactose. The main regulatory components are:

- A gene, called *lac*I, that encodes for a repressor protein (I).
- A specific DNA sequence, called *promoter*, that is recognized by RNA polymerases to initiate the transcription of the structural genes.
- A segment of DNA, called *operator* (OP), whose role is to obstruct the RNA polymerase functionality whenever the repressor protein I is tightly bound to it (I–OP), therefore blocking gene expression.
- A short DNA sequence, called CAP-*binding site*, that, when bound to a specific molecular complex (formed by a protein called CAP and a signal molecule named cAMP), has the role of enhancing the interaction between the RNA polymerase and the promoter, therefore promoting gene expression. The genes encoding for the production of cAMP and CAP protein are called, respectively, *cya* and *crp*, and they are indirectly involved in the regulation of the *lac* operon expression.

The signal which regulates the *lac* operon functionality depends on the integration of two different control mechanisms, one mediated by lactose and the other by glucose, which can lead either to the promotion or to the inhibition of *lac* expression, according to the (combination of) carbon sources that are available in the environment [20]. In fact, since gene

expression demands energy consumption, it is deleterious for the bacterium to synthesize the proteins involved in the metabolism of lactose when this nutrient is not available in the environment, or when the environment provides a more readily available source of energy, such as glucose.

The first control mechanism in the metabolism of lactose exploits the repressor protein I: in absence of *allolactose* (a metabolite that is produced only when lactose is available), I is able to bind the operator sequence preventing the *lac* operon expression. On the contrary, if lactose is available, allolactose makes protein I unable to bind the operator by altering its shape, therefore the *lac* operon can be potentially expressed [21]. The second control mechanism exploits the signal molecule cAMP and protein CAP to greatly increase the *lac* operon expression in absence of glucose [22]. In fact, the intracellular amount of cAMP is inversely proportional to glucose concentration: when glucose is not present, the cAMP concentration is high and the binding between the molecular complex cAMP–CAP and the CAP-binding site increases, thus enhancing gene expression.

Considering the biochemical environments in which *E. coli* cells could be in a specific time, four different conditions are able to regulate the *lac* operon expression:

1. When neither lactose nor glucose are available, even if the cAMP–CAP complex binds the CAP binding site, the presence of the repressor protein I prevents a stable interaction between the RNA polymerase and the promoter, thus blocking the operon gene expression (Fig. 2A).
2. When lactose is unavailable while glucose is available, the presence of the repressor protein I prevents the binding of the RNA polymerase to the promoter, thus inhibiting the operon gene expression (Fig. 2B).
3. When both glucose and lactose are available, even if the repressor protein I is not present, the absence of the cAMP–CAP complex prevents the formation of a stable interaction between RNA polymerase and the promoter, thus markedly reducing the operon gene expression (Fig. 2C).
4. When lactose is available while glucose is unavailable, the repressor protein I is absent and the contemporary presence of the cAMP–CAP complex assists the binding between RNA polymerase and the promoter, thus promoting the operon gene expression (Fig. 2D).

4.2. The lac operon reaction system

We provide here a description of the *lac* operon as a reaction system $\mathcal{A}_{lac} = (S, A)$, where the background set S represents the main biochemical components involved in this genetic system, while the reaction set A contains the biochemical reactions involved in the regulation of the *lac* operon expression. Formally, the *lac* operon reaction system is defined as follows:

$$S = \{lac, Z, Y, A, lacI, I, I\text{–}OP, cya, cAMP, crp, CAP, cAMP\text{–}CAP, lactose, glucose\},$$

and A consists of the following 10 reactions:

$$\begin{aligned}
a_1 &= (\{lac\}, \{\ldots\}, \{lac\}),\\
a_2 &= (\{lacI\}, \{\ldots\}, \{lacI\}),\\
a_3 &= (\{lacI\}, \{\ldots\}, \{I\}),\\
a_4 &= (\{I\}, \{lactose\}, \{I\text{–}OP\}),\\
a_5 &= (\{cya\}, \{\ldots\}, \{cya\}),\\
a_6 &= (\{cya\}, \{\ldots\}, \{cAMP\}),\\
a_7 &= (\{crp\}, \{\ldots\}, \{crp\}),\\
a_8 &= (\{crp\}, \{\ldots\}, \{CAP\}),\\
a_9 &= (\{cAMP, CAP\}, \{glucose\}, \{cAMP\text{–}CAP\}),\\
a_{10} &= (\{lac, cAMP\text{–}CAP\}, \{I\text{–}OP\}, \{Z, Y, A\}).
\end{aligned}$$

In Table 2 we provide a short description of the biological meaning of each reaction. Note that reactions a_1, a_2, a_5, a_7 are needed to grant the permanency of the genes in the system, while reactions a_4, a_9, a_{10} can only be enabled if the current state of the system does not include the inhibitor elements specified in each reaction. Namely, reaction a_4 can be applied only in the absence of lactose, reaction a_9 in the absence of glucose, and reaction a_{10} when repressor I is not bound to the operator OP.

4.3. Dynamic behavior of the lac operon reaction system

As stated in Section 4.1, the *lac* operon expression depends on which substrates the environment provides. In order to translate this situation in the *lac* operon reaction system, we need to evaluate what happens to the system when the context provides both glucose and lactose, only glucose, only lactose, or none of them. To this aim, we define a "default condition" (DC) that mimics the real biological system in which the genomic elements plus their encoded proteins are normally present.

Table 2
Biological description of the reactions of the *lac* operon reaction system.

Reaction	*Description of the biochemical process*
a_1	*lac* operon duplication
a_2	Repressor gene duplication
a_3	Repressor gene expression
a_4	Regulation mediated by lactose
a_5	*cya* duplication
a_6	*cya* expression
a_7	*crp* duplication
a_8	*crp* expression
a_9	Regulation mediated by glucose
a_{10}	*lac* operon expression

Table 3
The *lac* operon reaction system dynamics.

Index n	*Context set* C_n	*Result set* D_n	*lac expression*
0	DC	$\emptyset$	No
1	{*glucose*}	DC ∪ {*I–OP*, *cAMP–CAP*}	No
2	{*glucose*}	DC ∪ {*I–OP*}	No
3	{*glucose*}	DC ∪ {*I–OP*}	No
4	{*glucose*}	DC ∪ {*I–OP*}	No
5	{*glucose*}	DC ∪ {*I–OP*}	No
6	{*glucose*, *lactose*}	DC ∪ {*I–OP*}	No
7	{*glucose*, *lactose*}	DC	No
8	{*glucose*, *lactose*}	DC	No
9	{*glucose*, *lactose*}	DC	No
10	{*glucose*, *lactose*}	DC	No
11	{*lactose*}	DC	No
12	{*lactose*}	DC ∪ {*cAMP–CAP*}	No
13	{*lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
14	{*lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
15	{*lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
16	$\emptyset$	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
17	$\emptyset$	DC ∪ {*I–OP*, *cAMP–CAP*, *Z*, *Y*, *A*}	Yes
18	$\emptyset$	DC ∪ {*I–OP*, *cAMP–CAP*}	No
19	$\emptyset$	DC ∪ {*I–OP*, *cAMP–CAP*}	No
20	$\emptyset$	DC ∪ {*I–OP*, *cAMP–CAP*}	No
21	{*lactose*}	DC ∪ {*I–OP*, *cAMP–CAP*}	No
22	{*lactose*}	DC ∪ {*cAMP–CAP*}	No
23	{*lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
24	{*lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
25	{*lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
26	{*glucose*, *lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
27	{*glucose*, *lactose*}	DC ∪ {*Z*, *Y*, *A*}	Yes
28	{*glucose*, *lactose*}	DC	No
39	{*glucose*, *lactose*}	DC	No
30	{*glucose*, *lactose*}	DC	No
31	{*lactose*}	DC	No
32	{*lactose*}	DC ∪ {*cAMP–CAP*}	No
33	{*lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
34	{*lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
35	{*lactose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
36	{*glucose*}	DC ∪ {*cAMP–CAP*, *Z*, *Y*, *A*}	Yes
37	{*glucose*}	DC ∪ {*Z*, *Y*, *A*, *I–OP*}	Yes
38	{*glucose*}	DC ∪ {*I–OP*}	No
39	{*glucose*}	DC ∪ {*I–OP*}	No
40	{*glucose*}	DC ∪ {*I–OP*}	No

The genomic elements are the *lac* operon, the repressor gene *lac*I, the *cya* and the *crp* genes, while the respective genetic products are the repressor I, cAMP and CAP. Therefore, starting from the state $DC = \{lac, lacI, I, cya, cAMP, crp, CAP\}$ we can analyze how the reaction system responds when the context provides different nutrients. We evaluate the behavior of the system considering a dynamic situation in which the context sequence provides an ever-changing state, hence simulating a real natural environment where the bacterial cells could live. Starting with an initial context $C_0 = DC$, we consider an interactive process where, for each $i \geq 1$, C_i can be equal to one of the following sets: $\emptyset$, {*glucose*}, {*lactose*}, {*glucose*, *lactose*}.

The example of context sequence given in Table 3 (second column) shows that the elements Z, Y, A from the background S appear in a result state D_j for some $j \geq 2$ (third column), only when $C_{j-2} = \{lactose\}$, independently from which context

Table 4
Truth table for the boolean function g.

x_1	x_2	x_3	$g(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

is provided at step $j-1$. In other words, the *lac* operon is expressed in a certain result state when the context set given two steps before that result state contains *lactose* as the only input element. The same property holds if only *lactose* is provided continuously to the system: the elements Z, Y, A will appear in each set of consecutive result states $D_j, D_{j+1}, \ldots, D_{j+k}$, with $j \geq 2, k \geq 1$, if and only if there exists a sequence of consecutive context states such that $C_{j-2} = \cdots = C_{j+k-2} = \{lactose\}$, for any $C_{j+k-1} \in \{\emptyset, \{glucose\}, \{glucose, lactose\}\}$.

The rationale behind this behavior is the following: reaction a_{10}, which is the only one that allows the expression of the *lac* operon, is involved in a negative feedback mechanism with reaction a_4 and in a positive feedback mechanism with reaction a_9. When lactose appears in the context state, reaction a_4 is not enabled and the repressor element I cannot bind to the operator site; in this condition, the element *I–OP*, which is the only inhibitor of reaction a_{10}, is not produced. Instead, reaction a_9, whose product is *cAMP–CAP* (which is also a reactant of a_{10}), is enabled only when glucose is *not* present in the context. In other terms, reaction a_{10} is under the regulation of lactose through the interaction with reaction a_4 and under the regulation of glucose through the interaction with reaction a_9. Since the enabling of reaction a_{10} requires the simultaneous inactivation of a_4 and activation of a_9, reaction a_{10} will be enabled only one step ahead of the context state that provided only lactose, leading to the production of the Z, Y, A elements two steps ahead of that state.

This behavior is consistent with the real natural situation in which a cell reacts to the modified nutritional conditions by regulating the expression of its genes, so that only when the proper intracellular *adaptation* has been carried out (which requires a step-by-step activation of downstream reactions), the system will be able to respond correctly to the new environmental condition. As a consequence of adaptation, another noteworthy dynamic property of the *lac* operon can emerge in this reaction system, namely, the *hysteresis* behavior. By hysteresis we mean the dependence of the biological system not only on the current environment where it is living (in terms of reaction systems, the context set given at the current step), but also on the past environmental states; as a whole, hysteresis is the result of the fact that the system can be in different internal states. In the case of the *lac* operon reaction system, this behavior arises because the current result state is determined, in a context-dependent way, by an orchestrated cooperation of reactions which induce the aforementioned delayed adaption to the given environmental states.

5. Simplification methods for boolean functions applied to reaction systems

In this section we show how simplification methods for boolean functions can be exploited to potentially reduce the number of reactions in a reaction system. We start by providing the necessary notation for boolean functions and we recall the relation between reaction systems and boolean functions [6]. Simplification methods for boolean functions are then discussed and applied to reaction systems, with the following goal: given a reaction system with n reactions, derive an *equivalent* reaction system with $n' \leq n$ reactions.

5.1. Boolean functions and reaction systems

Let $f : \{0, 1\}^n \to \{0, 1\}$ be a boolean function and let us denote $x_1, \ldots, x_n$ its boolean variables. The function f can be univocally represented by writing all the possible input combinations of the values of $x_1, \ldots, x_n$ and their corresponding output value $f(x_1, \ldots, x_n)$ in a table, called *truth table*, which is formed by 2^n rows, or, in more compact form, using *boolean expressions*. It is well known that a boolean function can be represented by many equivalent boolean expressions. In what follows, we write boolean expressions in the disjunctive normal form (DNF), that is, as the sum of products of literals (called *monomials*). Given two boolean variables x_1, x_2, the sum $x_1 + x_2$ represents the logic "OR", while the product x_1x_2 represents the logic "AND". The negation of a boolean variable x is denoted $\bar{x}$.

Describing boolean functions with reaction systems is straightforward, since it simply requires to translate the input variables and output values into entities of a reaction system $\mathcal{A}$, and to appropriately express the boolean function in terms of a set of reactions that satisfy some conditions [6].

We explain this process with an example. Table 4 shows the truth table of a boolean function $g : \{0, 1\}^3 \to \{0, 1\}$. It is possible to directly convert this representation into a reaction system. The output of the function $g(x_1, x_2, x_3)$ can be interpreted as the presence (output equal to 1) or the absence (output equal to 0) of the product of a reaction. Similarly, the values of the input variables x_1, x_2, x_3 denote the required presence or absence of some compounds in a reaction. Analyzing

Table 5
Karnaugh map for the function g.

$x_3 \backslash x_1x_2$	00	01	11	10
0	0	1	0	0
1	0	0	1	1

the truth table row by row, we only need to take into consideration the rows associated to an output equal to 1, as they are the only ones that generate a product. Therefore, every such line defines a reaction, where elements corresponding to inputs equal to 0 are inhibitors, as they must be absent in order for the reaction to be enabled, while elements corresponding to inputs equal to 1 are reactants, as they must be present in order for the reaction to be enabled. In what follows, for the sake of simplicity, we use a notation where reactant and inhibitor elements directly correspond to input boolean variables, i.e., x_1, x_2, x_3, while product elements correspond to the output value, i.e., $g(x_1, x_2, x_3)$.

According to these rules, it is possible to translate the boolean function g into the reaction system $\mathcal{A}_g = (S, A)$, where:

$$S = \{x_1, x_2, x_3, g(x_1, x_2, x_3)\},$$

$$A = \{a_1, a_2, a_3\},$$

and

$$a_1 = (\{x_2\}, \{x_1, x_3\}, \{g(x_1, x_2, x_3)\}),$$

$$a_2 = (\{x_1, x_3\}, \{x_2\}, \{g(x_1, x_2, x_3)\}),$$

$$a_3 = (\{x_1, x_2, x_3\}, \{\dots\}, \{g(x_1, x_2, x_3)\}).$$

5.2. Simplification methods for boolean functions

Given the truth table of a boolean function, it is possible to derive the DNF of the corresponding boolean expression. The easiest way to do this is by writing the sum of the input combinations (as products of literals) that lead to an output equal to 1. Considering the example from Table 4, we obtain the following boolean formula for the function g:

$$g(x_1, x_2, x_3) = \overline{x_1}x_2\overline{x_3} + x_1\overline{x_2}x_3 + x_1x_2x_3.$$

This expression can be reduced to a simpler equivalent expression (i.e., defining the same function g) using one of the simplification methods available for boolean expressions.

One of the most used simplification methods is the *Karnaugh map method* [17]. A Karnaugh map for a function with n variables is a matrix containing 2^n cells, one for each possible configuration of the n input values. Coordinates are assigned to each cell so that when moving horizontally or vertically from one cell to an adjacent one, the value of only one input variable changes. Terminal cells are also considered to be adjacent, forming horizontal and vertical "rings". Every cell contains the value of the output that corresponds to the input identified by its coordinates. The Karnaugh map for the function g of Table 4 is depicted in Table 5.

A function can be simplified using a Karnaugh map observing that two adjacent cells differ only on one input variable, which has value 0 in one cell and value 1 in the other. Therefore, the product of the other $n-1$ variables implies both cells, as the value of the variable that changes is irrelevant for determining the output value. This observation can be applied to any rectangular group of 2^i adjacent cells, with $i \geq 1$. Given these facts, Karnaugh maps allow us to determine the minimum set of 2^i adjacent cells that cover all the cells with an output value equal to 1. The union (logic sum) of the elements of this set (each of which corresponds to a product of input variables) forms the simplest expression representing the desired function.

Going back to the example, the Karnaugh map of Table 5 leads to the following simplest expression for the function g represented by Table 4:

$$g(x_1, x_2, x_3) = \overline{x_1}x_2\overline{x_3} + x_1x_3.$$

The Karnaugh map method is inefficient for functions of more than 6 variables, as it is based on a graphic representation. For larger functions, other methods can be used, such as the Quine–McCluskey method [23,24]. This method is implemented in tabular form, which makes it more efficient to use than the Karnaugh map method. However, the problem it solves is NP-hard, so its runtime grows exponentially with the number of input variables. To simplify functions with a large number of input variables it is possible to use non-optimal heuristic methods. One of these methods is implemented by the Espresso algorithm, developed as a logic minimization algorithm for Very Large Scale Integration (VLSI). For more information on the functioning of this algorithm, we refer the interested reader to [25]. One of the most well known implementations of the Espresso algorithm is the Espresso logic minimizer, a computer program designed to reduce the complexity of digital electronic gate circuits.

5.3. Simplification of reaction systems

In Section 5.1 we have shown how to model boolean functions in terms of reaction systems. The inverse procedure is also possible, and this allows us to obtain the truth table associated to a reaction system. As we have seen, we can apply a simplification method in order to obtain the simplest boolean expression for a given boolean function. This expression can be directly translated into a reaction system, making two fundamental considerations:

- Every monomial in the boolean expression corresponds to a reaction of the simplified reaction system. This is true because if any of the monomials has value 1, then the result of the function is 1. In reaction terms, this means that if one of the reactions is enabled, then we get a product.
- Every reaction has as reactants the non-negated variables and as inhibitors the negated variables of the corresponding monomial. This definition enables the reaction only when all the non-negated variables are equal to 1, which corresponds to the presence of the reactants, and when the negated variables have value 0, which corresponds to the absence of the inhibitors.

Let us consider a reaction system $\mathcal{A}$ that we wish to simplify, having a reaction set A constituted by n reactions $a_i = (R_i, I_i, P_i)$, $i = 1, \ldots, n$. The purpose of the simplification is to obtain another reaction system that is functionally equivalent to $\mathcal{A}$ but has a lower number of reactions. The first step of the simplification procedure consists in splitting A into a set of disjoint subsets of reactions, such that the product set of each reaction belonging to a given subset is identical to the product set of all the other reactions belonging to the same subset, but it is different from the product set of every other reaction belonging to any other subset. Formally, we have

$$A = A_1 \cup A_2 \cup \cdots \cup A_L$$

where $A_l = \{a_{l1}, \ldots, a_{lm_l}\}$ for some $m_l \geq 1$, with $\sum_{l=1}^{L} m_l = n$, $a_{lj} \in A$ for each $l = 1, \ldots, L$, $j = 1, \ldots, m_l$, and $A_1, \ldots, A_L$ are such that (i) $P_{A_1} \cap \cdots \cap P_{A_L} = \emptyset$ and (ii) $P_{lj_1} = P_{lj_2}$ for each $a_{lj_1}, a_{lj_2} \in A_l, j_1, j_2 \in \{1, \ldots, m_l\}$.

It is necessary to separate the reactions into these disjoints subsets because reduction methods for boolean functions only work on a single output variable that, in terms of reaction systems, might correspond to a set of products. Hence, we need to separate the reactions that produce different elements, and apply the simplification method separately on each subset of reactions.

Having defined these sets of reactions, it is now possible to apply the reduction method on every single set A_l (with $1 \leq l \leq L$). First of all, each set of reactions has to be transformed into its corresponding boolean function, as follows:

$$f_l(A_l) = f_l\left(\bigcup_{j=1}^{m_l}(R_{lj} \cup I_{lj})\right) = \sum_{j=1}^{m_l}\left(\prod_{o=1}^{O} r_{ljo}\right)\left(\prod_{q=1}^{Q} \overline{in}_{ljq}\right)$$

where r_{ljo} represents the presence of a reagent element in the reagent set R_{lj} of a_{lj}, with $|R_{lj}| = O$, and $\overline{in}_{ljo}$ represents the absence of an inhibitor element in the inhibitor set I_{lj} of a_{lj}, with $|I_{lj}| = Q$.

A simplification method for boolean functions can now be applied to all the boolean functions $f_l(A_l)$, $l = 1, \ldots, L$. The result, for each function, will have the form

$$simpl(f_l) = \sum_{u=1}^{U_l}\left(\prod_{o'=1}^{O'} r_{luo'}\right)\left(\prod_{q'=1}^{Q'} \overline{in}_{luq'}\right),$$

where U_l is the number of addends of the simplified boolean expression obtained from each $f_l(A_l)$, and O' and Q' are the number of non-negated and negated boolean variables left after the reduction, which correspond to a subset of the reagents and of the inhibitors of the reaction a_{lu}, respectively.

The last step consists in transforming each function $simpl(f_l)$ back into a set of U_l reactions of the form

$$a_{lu} = (\{r_{lu1}, \ldots, r_{luo'}\}, \{in_{lu1}, \ldots, in_{luq'}\}, \{P_{A_l}\}),$$

for each $u = 1, \ldots, U_l$. The system is based on the fact that $U_l \leq m_l$, for each $l = 1, \ldots, L$, which guarantees that the number of reactions in the reduced reaction system will not increase with respect to the initial reaction system. In the worst case, this number may not decrease. In the best case, the number of reactions is reduced to 1. In general, the degree of simplification of a reaction system depends on its redundancy – informally characterized by the number of different reactions that are used to produce the same elements of the background set – though an *a priori* average estimate is difficult to formulate for arbitrary reaction systems.

Finally, we provide an example of the application of this simplification method. We intentionally apply the simplification on a small reaction system, in order to allow the reader to follow the fundamental steps of the method easily. Therefore, let us consider the reaction system $\mathcal{A} = (S, A)$ defined by the background set $S = \{c_1, c_2, c_3\}$ and a set of reactions

$A = \{a_1, a_2, a_3, a_4, a_5\}$, where

$$a_1 = (\{c_1\}, \{c_2\}, \{c_3\}),$$
$$a_2 = (\{c_1, c_2\}, \{\ldots\}, \{c_3\}),$$
$$a_3 = (\{c_2\}, \{c_3\}, \{c_1, c_2\}),$$
$$a_4 = (\{c_3\}, \{c_1\}, \{c_1, c_2\}),$$
$$a_5 = (\{c_1, c_2, c_3\}, \{\ldots\}, \{c_1, c_2\}).$$

Following the above procedure, we identify two subsets of reactions: the first subset $A_1 = \{a_1, a_2\}$ is formed by the reactions whose product is c_3, and the second subset $A_1 = \{a_3, a_4, a_5\}$ is formed by the reactions whose products are c_1, c_2. We can now determine the two boolean functions assigned to these sets:

$$f_1(A_1) = f_1(c_1, c_2) = c_1\overline{c_2} + c_1c_2,$$
$$f_2(A_2) = f_2(c_1, c_2, c_3) = c_2\overline{c_3} + c_3\overline{c_1} + c_1c_2c_3.$$

Applying a simplification method to these boolean functions, we obtain:

$$simpl(f_1) = c_1,$$
$$simpl(f_1) = c_2 + c_3\overline{c_1}.$$

These two boolean functions correspond to a reaction system with the following reactions:

$$b_1 = (\{c_1\}, \{\ldots\}, \{c_3\}),$$
$$b_2 = (\{c_2\}, \{\ldots\}, \{c_1, c_2\}),$$
$$b_3 = (\{c_3\}, \{c_1\}, \{c_1, c_2\}).$$

This simplified reaction system $\mathcal{B} = (S, \{b_1, b_2, b_3\})$ is formed by 3 reactions, which is less than the 5 reactions that composed the original system $\mathcal{A}$. It is straightforward to verify that $\mathcal{B}$ is functionally equivalent to $\mathcal{A}$ in S.

6. Discussion

The concept of interaction between chemical reactions is at the heart of the framework of reaction systems. These interactions are based on mutual facilitation and inhibition, which rely on the fact that the result of one reaction can be a reactant of another reaction, and the inhibitor of a reaction is usually an element of some other reaction. In other terms, the chemical reactions occurring in a living cell constitute a *regulative network*. The use of chemical reactions as a computing paradigm has been previously considered in computer science, both in qualitative and quantitative terms, from a theoretical point of view [26,27], as well as a possible natural device for the implementation of logic gates and circuits (see, e.g., [28,29]). The common view among these paradigms is that they represent formal models of biochemical reactions that allow one to study how the state of a system changes in time.

In analogous terms, though from a different perspective, the dynamical evolution of networks of biochemical reactions, such as cellular processes, can be investigated with the purpose of analyzing their *emergent behaviors*. With this concept we refer to the possible dynamics that the system can present in time, according both to its internal state (the type and amounts of molecular species, such as enzymes, substrates, ions, etc.) and to the environmental conditions where the system is living (temperature, light, availability of nutrients, extracellular conditions, etc.). The classical approach to the analysis of these properties relies on the definition of a mathematical model, developed according to the specific open problems that one wishes to solve about the biochemical system of interest. For instance, traditional models based on systems of ordinary differential equations (ODEs) [30] assume an underlying deterministic interpretation of molecular interactions – based on ensemble properties holding for large amounts of molecules – which is however unable to take into account the effects of biological noise, that is inherent to most cellular systems [31]. On the contrary, stochastic modeling approaches [32] are able to detect the random behaviors due to the stochastic molecular interaction between individual molecules, which can trigger the emergence of complex phenomena such as bistability (that is, the switching between two different stable states).

A plethora of different modeling approaches, based either on the deterministic or the stochastic assumption, and considering distinct levels of abstraction for the formalization of the real system, have been developed in the last years to achieve new results and insights into the functioning of biological systems [30,33]. In this context, also some formalisms that were originally developed in computer science either to model systems of interacting components (Petri nets, process algebras, etc.), or to develop biologically-inspired computational models (DNA computing, membrane systems, etc.), have been successfully applied in the field of computational biology and proved their intrinsic suitability for reproducing or analyzing the dynamics of many cellular processes (see, e.g., [34–37], just to mention a few). In relation to these well-established frameworks, reaction systems indeed represent an innovative approach, whose global modeling power seems anyway still difficult to debate thoroughly in comparison to the wide spectrum of aspects, potentialities, consistent observations and theoretical results that the other approaches have already demonstrated. Nevertheless, some strengths of this formal approach can already be explicitly stated as the direct follow-on from the topics discussed in this work, where

we intentionally investigated the expressive power and the flexibility of reaction systems as a modeling tool for computer science and biology oriented applications. Therefore, we discuss hereby the potentialities of reaction systems that we were able to highlight with our work.

Concerning the modeling power in biology oriented applications, in Section 4 we exploited reaction systems to describe a complex biological phenomenon that relies on a fine mechanism to control the uptake and consumption of environmental nutrients. Namely, we formalized the regulation of gene expression in the *lac* operon, a genetic regulatory network involved in the metabolism of carbon sources in *E. coli* bacterial cells. Notwithstanding the complexity of the control mechanisms that are present in this metabolic pathway, we stress the fact that we were able to propose an original description of this biological system by using a reaction system that consists of a few reactions and elements, by just relying on the basic assumptions of this formalism and on a proper use of inhibitors. The dynamic behavior of this reaction system, in a context-dependent formalization, shows a qualitative correspondence with the response of cells to natural environments, where they are exposed to ever-changing conditions and are able to react properly by regulating the expression of their genes. The further addition of a quantitative description of the *lac* operon, where the amounts of chemicals are also specified, would require the definition of ad hoc measurement functions [14], and will be the scope of future research. Anyway, even the simple formulation of the *lac* operon expression in terms of reaction system given in Section 4 is able to show two interesting biological properties, such as adaptation and hysteresis. The first is due to a step-by-step enabling of downstream reactions, which in turn depends on the interaction of positive and negative feedback mechanisms between enabled and inhibited reactions. The second arises as a consequence of adaptation, and shows the ability of the reaction system to retain memory of the past environmental states. This behavior is simply explained by the capability of the reaction system to react to the (context-dependent) stimulus with an initial adaptive mechanism, and consequently to carry out the proper internal response to the external signal. In situations of this type, where a biological system can show hysteresis, it is therefore not enough to know the current context and result sets to be able to predict its evolution, but we also need to know its past. In the *lac* operon reaction system that we defined, this property seems to emerge as a natural consequence of the facilitation and inhibition assumptions, that drive the evolution of the system in a deterministic way, according to the incoming context set. Such properties might not emerge as much easily with other modeling frameworks; for instance, by using models based on ODEs – which assume as well an underlying deterministic approach – the hysteresis response would possibly require larger computational costs, due to the fact that the analysis of the emergent dynamics demands the numerical integration of the whole set of equations at each time step, while in reaction systems it only requires the identification and application of the enabled reactions. On the other way, by using models based on, e.g., membrane systems or Petri nets, the usual non-deterministic application of rules/transitions might not lead the system directly to the expected hysteresis behavior. Even if we cannot debate here or compare the effective capability of reaction systems (with respect to all these well-known frameworks) for the investigation of *any* biological system, we believe that their ability to easily simulate common natural phenomena as adaptation and hysteresis suggests that they possess good potentiality for the modeling of biochemical processes.

Concerning the expressive power of reaction systems in computer science oriented problems, in Section 3 we presented an application in game theory, namely we used this formalism as a common programming language. As an example, we showed an implementation of an iterative version of the Tower of Hanoi algorithm, as developed by Franklin in [18]. The tricky solution in this application of reaction systems consists in the exploitation of the no permanency property of elements and in the proper use of inhibitors to move disks from one peg to another, ensuring that no illegal move can be executed. The idea behind this solution is that at each step of the game only one disk can be moved, while all the other disks are either blocked (they have no legal destination), suspended (they were moved at the previous step) or covered (they cannot be moved because lying below some other disk). These disks are translated into elements of the background set and act as inhibitors for the movement of disks. Moreover, specific elements called ''suspension signals'' are produced to remember which disk was moved in the previous step of the game, but then they are not kept sustained any further by other reactions: so doing, the no permanency assumption ensures that the game can go on correctly step by step, by facilitating or inhibiting the various moves of the puzzle. Once more, we demonstrated that the basic assumptions of reaction systems can be exploited to simulate the execution of nontrivial computational steps, in a similar way as we were writing a ''program'' to implement an algorithmic procedure.

Then, in Section 5 we provided a formulation of boolean functions in terms of reaction systems. This is a straightforward transformation that allows to define a reaction for every boolean input configuration that returns an output equal to 1: for each of these configurations, the reagents correspond to the non-negated boolean variables, the inhibitors correspond to the negated boolean variables, while a specific product element from the background set represents the boolean output. Next, we have introduced simplification methods for boolean functions, such as Karnaugh maps, that allow to reduce the number of monomials in a boolean formula expressed in DNF. The converse process is also possible: given a reaction system with an arbitrary number of reactions, it is possible to derive a corresponding representation in terms of boolean expressions in DNF. We can exploit this trick to reduce the number of reactions in a given reaction system: first, we need to separate into disjoints subsets all the reactions that produce different product elements, and then we can apply the simplification method independently on each subset of reactions. So doing, we derive an equivalent simplified reaction system, i.e., a reaction system with a possibly smaller reaction set with respect to the reaction set of the initial system. A simple example of simplification of a reaction system was presented to show the easiness of use of the proposed method. Though for large reaction systems the procedure might be cumbersome to execute by hand due to the presence of many entities, it indeed

represents a simple approach for the simplification of arbitrary reaction systems, which might be executed by exploiting existing algorithms [25], or by developing novel software specifically dedicated to reaction systems. We leave this issue as a further topic for future research. Indeed, an interesting and useful application of this simplification method would consist in the reduction of the number of reactions for the description of large biological systems, for instance those that might be defined by exploiting the modular, bottom-up approach based on set-theoretic union presented for reaction systems in [6].

In conclusion, it is our intention to deepen the investigation of reaction systems from both a theoretical and applicative point of view, since the examples proposed in this paper have provided a positive confirmation of their feasibility and their expressive power as a modeling framework in multidisciplinary applications, that range from computer science to biology [6,7,11,12,14].

References

[1] M.A. Arbib (Ed.), Handbook of Brain Theory and Neural Networks, The MIT Press, 1995.
[2] K.D. Jong, Evolutionary Computation: A Unified Approach, The MIT Press, 2006.
[3] C. Calude, G. Păun, Computing with Cells and Atoms: An Introduction to Quantum, DNA and Membrane Computing, CRC Press, 2000.
[4] S. Wolfram, A New Kind of Science, Wolfram Media, 2002.
[5] G. Păun, G. Rozenberg, A. Salomaa (Eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, 2009.
[6] A. Ehrenfeucht, G. Rozenberg, Reaction systems, Fundamenta Informaticae 75 (2007) 263–280.
[7] A. Ehrenfeucht, G. Rozenberg, Events and modules in reaction systems, Theoretical Computer Science 376 (2007) 3–16.
[8] R. Brijder, A. Ehrenfeucht, G. Rozenberg, Reaction systems with duration, in: A. Kelemen, J. Kelemenová (Eds.), Computation, Cooperation, and Life, in: Lecture Notes in Computer Science, vol. 6610, Springer-Verlag, 2011, pp. 191–202.
[9] R. Brijder, A. Ehrenfeucht, M. Main, G. Rozenberg, A tour of reaction systems, International Journal of Foundations of Computer Science 22 (7) (2011) 1499–1517.
[10] J. Kleijn, M. Koutny, G. Rozenberg, Modelling reaction systems with Petri nets, in: M. Heiner, H. Matsuno (Eds.), Proceedings of the International Workshop on Biological Processes & Petri Nets, BioPPN-2011, vol. 724, CEUR Workshop Proceedings, 2011, pp. 36–52.
[11] A. Ehrenfeucht, M. Main, G. Rozenberg, Functions defined by reaction systems, International Journal of Foundations of Computer Science 22 (1) (2011) 167–178.
[12] A. Ehrenfeucht, M. Main, G. Rozenberg, Combinatorics of life and death for reaction systems, International Journal of Foundations of Computer Science 21 (3) (2010) 345–356.
[13] R. Brijder, A. Ehrenfeucht, G. Rozenberg, A note on causalities in reaction systems, in: C. Ermel, H. Ehrig, F. Orejas, G. Taentzer (Eds.), International Colloquium on Graph and Model Transformation On the occasion of the 65th birthday of Hartmut Ehrig, GraMoT 2010, vol. 30, ECEASST, 2010, pp. 1–9.
[14] A. Ehrenfeucht, G. Rozenberg, Introducing time in reaction systems, Theoretical Computer Science 410 (2009) 310–322.
[15] M. Petković, Famous Puzzles of Great Mathematicians, American Mathematical Society, 2009.
[16] M. Savageau, Design of gene circuitry by natural selection: analysis of the lactose catabolic system in Escherichia coli, Biochemical Society Transactions 27 (2) (1999) 264–270.
[17] M. Karnaugh, The map method for synthesis of combinational logic circuits, Transactions of the American Institute of Electrical Engineers Part I 72 (1953) 593–599.
[18] W.R. Franklin, A simpler iterative solution to the Towers of Hanoi problem, ACM SIGPLAN Notices 19 (8) (1984) 87–88.
[19] J. Vilar, C. Guet, S. Leibler, Modeling network dynamics: the lac operon, a case study, Journal of Cell Biology 161 (3) (2003) 471–476.
[20] C. Wilson, H. Zhan, L. Swint-Kruse, K. Matthews, The lactose repressor system: paradigms for regulation, allosteric behavior and protein folding, Cellular and Molecular Life Sciences 64 (1) (2007) 3–16.
[21] J. Davies, F. Jacob, Genetic mapping of the regulator and operator genes of the lac operon, Journal of Molecular Biology 36 (3) (1968) 413–417.
[22] P. Wong, S. Gladney, J. Keasling, Mathematical model of the lac operon: inducer exclusion, catabolite repression, and diauxic growth on glucose and lactose, Biotechnology Progress 13 (2) (1997) 132–143.
[23] W. Quine, The problem of simplifying truth tables, American Mathematical Monthly 59 (8) (1952) 521–531.
[24] E. McCluskey, Minimization of boolean functions, Bell System Technical Journal 35 (5) (1956) 1417–1444.
[25] R.K. Brayton, A.L. Sangiovanni-Vincentelli, C.T. McMullen, G.D. Hachtel, Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Norwell, MA, USA, 1984.
[26] G. Berry, G. Boudol, The chemical abstract machine, Theoretical Computer Science 96 (1992) 217–248.
[27] J.-P. Banâtre, P. Fradet, D.L. Métayer, Gamma and the chemical reaction model: fifteen years later, in: C. Calude, G. Păun, G. Rozenberg, A. Salomaa (Eds.), Multiset Processing, in: Lecture Notes in Computer Science, vol. 2235, Springer-Verlag, 2001, pp. 17–44.
[28] P. Dittrich, Chemical computing, in: J.-P. Banâtre, J.-L. Giavitto, P. Fradet, O. Michel (Eds.), Unconventional Programming Paradigms, UPP 2004, in: Lecture Notes in Computer Science, vol. 3566, 2005, pp. 19–32.
[29] A. Leporati, D. Besozzi, P. Cazzaniga, D. Pescini, C. Ferretti, Computing with energy and chemical reactions, Natural Computing 9 (2) (2009) 493–512.
[30] Z. Szallasi, J. Stelling, V. Periwal, Systems Modeling in Cellular Biology, The MIT Press, 2006.
[31] M. Elowitz, A. Levine, E. Siggia, P. Swain, Stochastic gene expression in a single cell, Science 297 (5584) (2002) 1183–1186.
[32] D.J. Wilkinson, Stochastic Modelling for Systems Biology, Chapman & Hall/CRC, 2006.
[33] J.M. Bower, H. Bolouri (Eds.), Computational Modeling of Genetic and Biochemical Networks, The MIT Press, 2004.
[34] G. Ciobanu, G. Păun, M.J. Pérez-Jiménez (Eds.), Applications of Membrane Computing, Springer-Verlag, Berlin, 2005.
[35] A. Regev, W. Silverman, E. Shapiro, Representation and simulation of biochemical processes using the pi-calculus process algebra, Pacific Symposium on Biocomputing (2001) 459–470.
[36] V.N. Reddy, M.L. Mavrovouniotis, M.N. Liebman, Petri net representations in metabolic pathways, in: Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology, AAAI Press, 1993, pp. 328–336.
[37] M. Muskulus, D. Besozzi, R. Brijder, P. Cazzaniga, S. Houweling, D. Pescini, G. Rozenberg, Cycles and communicating classes in membrane systems and molecular dynamics, Theoretical Computer Science 372 (2007) 242–266.