First Steps Towards a Wet Implementation for τ-DPP

Dario Pescini, Paolo Cazzaniga,
Claudio Ferretti, and Giancarlo Mauri

Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
{pescini,cazzaniga,ferretti,mauri}@disco.unimib.it

Abstract. In the last decade, different computing paradigms and devices inspired by biological and biochemical systems have been proposed. Here, we recall the notions of membrane systems and the variant of τ-DPP. We introduce the framework of chemical computing, in order to show how to describe computations by means of a chemical reaction system. Besides the usual encoding of primitive Boolean functions, we also present encodings for register machines instructions. Finally will discuss how this computing components can be composed in a more complex chemical computing system, with a structure based on the membrane structure of τ-DPP, to move toward a wet implementation using the micro reactors technology.

1 Introduction

In the recent years, several computational models derived from the formal abstraction of chemical reacting systems, such as the chemical abstract machine [3], and others inspired by the structure and functioning of living cells, have been proposed. One of these models, introduced in [14], is called P systems (or membrane systems). The basic definition of P systems consists of a hierarchical structure composed by several compartments, each one delimited by a membrane. Inside every compartment, a set of evolution rules and a multiset of objects are placed. The rules – precisely, multiset rewriting rules – are used to describe the modification and the communication among the membranes of the objects occurring inside the system. In particular, the current system state is represented by means of objects quantities.

Among the different variants of P systems, here we consider τ-DPP, presented in [5]. Within the framework of τ-DPP, the probabilities are associated to the rules, following the method introduced by Gillespie in [7]. In particular, τ-DPP extends the tau-leaping procedure [4] in order to quantitatively simulate the behavior of complex biological and chemical systems, embedded in membrane structures composed by different volumes.

The aim of this work is to implement the computational model based on chemical reacting systems using micro reactors. Micro reactors [9] are laboratory devices consisting of several reacting volumes (reactors) with a size at the scale of the μl, connected by channels used to transport molecules.

Hence, in this paper we will establish a correspondence between τ-DPP and chemical reacting systems occurring inside micro reactors. There is a close relation between the topological description of the two systems: they are both composed by several volumes, and among these volumes it is possible to communicate molecules. Moreover, the approach based on multiset rewriting rules, that characterizes τ-DPP, is similar to the chemical reacting process occurring within a micro reactor. Furthermore, both τ-DPP and chemical reacting systems emphasize the intrinsic stochasticity of chemical processes. The "noise", associated to the stochastic behavior, rules the system dynamics at the micro-scales. At this scale, the small volumes and the high dilutions result in a system where particles interaction should be described in a discrete fashion. Finally, the communication processes described by means of communication rules within τ-DPP, are strictly related to the channels interconnecting the reactors.

In this paper, we show how these analogies can be exploited to build a feasible wet implementation of P systems (in particular, of τ-DPP). To obtain a description of τ-DPP that can be implemented using micro reactors in a straightforward way, we consider the encoding of Boolean functions and register machine instructions through chemical reactions, following the *chemical computing principles*.

Chemical computing [6] is a technique used to process information by means of real molecules, modified by chemical reactions, or by using electronic devices that are programmed following some principles coming from chemistry. Moreover, in chemical computing, the result of a computation is represented by the emergent global behavior, obtained from the application of small systems characterized by chemical reactions.

Exploiting the chemical organization theory [12], we can define a chemical network in order to describe a system as a collection of reactions applied to a given set of molecular species. Moreover, we can identify the set of (so called) organizations, in the set of molecular species, to describe the behavior of such system. So doing, the behavior is traced by means of "movement" between organizations.

Furthermore, a different kind of problem encoding will be presented, this is based on the instructions of register machines [13]. This approach is similar to the one related to the chemical computing field. The idea is to use a set of chemical reactions to realize the instructions of the register machines. For instance, in Section 4, the formalization and the simulation of a decrement instruction by means of τ-DPP, is presented.

The paper is organized as follows: in Section 2, membrane systems and its variant of τ-DPP are explained. The chemical computing framework and chemical organization theory are presented in Section 3. In Section 4, we show the results of the simulations of small components, such as the NAND logic circuit and the decrement instruction of a register machine. We conclude with some discussion in Section 5.

2 Membrane Systems and τ-DPP

In this section we describe the framework of membrane systems [15], recalling their basic notions and definitions.

We then present τ-DPP, a computational method firstly introduced in [5], used to describe and perform stochastic simulations of complex biological or chemical systems. The "complexity" of the systems that can be handled by means of τ-DPP, is not only related to the number of reactions (rules) and species (objects) involved, but it results also from the topological structure of the system, that can be composed by many volumes.

2.1 Membrane Systems

P systems, or membrane systems, have been introduced in [14] as a class of unconventional computing devices of distributed, parallel and nondeterministic type, inspired by the compartmental structure and the functioning of living cells.

In order to define a basic P system, three main parts need to be introduced: the *membrane structure*, the *objects* and the *rules*.

The *membrane structure* defines the topological and hierarchical organization of a system consisting of distinct compartments. The definition of membrane structure is given through a set of membranes with a distinct label (usually numbers), hierarchically organized inside a unique membrane, named *skin membrane.* Among others, a representation of a membrane structure is given by using a string of square parentheses.

In particular, each membrane identifies a *region*, delimited by the membrane itself and any other adjacent membrane possibly present inside it. The number of membranes in a membrane structure is called the *degree* of the P system. The whole space outside the skin membrane is called the *environment.*

The internal state of a P system is described by the *objects* occurring inside the membranes. An object can be either a symbol or a string over a specified alphabet V. In order to denote the presence of multiple copies of objects inside the membranes, multisets are usually used.

The objects inside the membranes of a P system are transformed by means of *evolution rules.* These are multiset rewriting rules of the form $r_i : u \rightarrow v$, where u and v are multisets of objects. The meaning of the generic rule i is that the multiset u is modified into the multiset v. There exists a special symbol δ used to dissolve (namely, remove) the membrane where the rule is applied together with its set of rules. In this paper we will not make use of the dissolving operation.

Moreover, it is possible to associate a target to v, representing the membrane where the multiset v is placed when the rule is applied. There are three different types of target. If the target is *here*, then the object remains in the region where the rule is executed (usually, this target label is omitted in the systems description). If the target is *out*, then the object is sent out from the membrane containing the rule and placed to the outer region. Note that, if a rule with this target indication is applied inside the skin membrane, then the object is sent to the environment. Finally, if the target is in_j, where j is a label of a membrane, then the object is sent into the membrane labeled with j. It is possible to apply this kind of rule, only if the membrane j is placed immediately inside the membrane where the rule is executed.

Starting from an initial configuration (described by a membrane structure containing a certain number of objects and a fixed set of rules), and letting the system evolve, a computation is obtained. A universal clock is assumed to exist: at each step, all rules in all regions are simultaneously applied to all objects which can be the subjects of evolution rules. So doing, the rules are applied in a maximal parallel manner, hence the membranes evolve simultaneously. If no further rule can be applied, the computation halts. The result of a computation is the multiset of objects contained into previously specified *output membrane* or sent from the skin of the system to the environment.

For a complete and extensive overview of P systems, we refer the reader to [15], and to the P Systems Web Page (`http://ppage.psystems.eu`).

2.2 τ-DPP

We now introduce a novel stochastic simulation technique called τ-DPP [5]. The aim of τ-DPP is to extend the single-volume algorithm of tau-leaping [4], in order to simulate multi-volume systems, where the distinct volumes are arranged according to a specified hierarchy. The structure of the system is required to be kept fixed during the evolution. In Section 2.1, we shown that the framework of membrane system satisfies this requirement, hence, the spatial arrangement of P system is exploited in the τ-DPP description. In particular, τ-DPP has been defined starting from a variant of P systems called dynamical probabilistic P systems (DPP). DPP were introduced in [18]: they exploit the membrane structure of P systems and they associate probabilities with the rules, such values vary (dynamically), according to a prescribed strategy, during the evolution of the system. For the formal definitions of DPP and examples of simulated systems, we refer the reader to [16,17,1,2].

There is a difference between these two membrane systems variants: DPP provides only a qualitative description of the analyzed system, that is, "time" is not associated to the evolution steps, while τ-DPP is able to give a quantitative description tracing the time-stream of the evolution.

The τ-DPP approach is designed to share a common time increment among all the membranes, used to accurately extract the rules that will be executes in each compartment (at each step). This improvement is achieved using, inside the membranes of τ-DPP, a modified tau-leaping algorithm, which gives the possibility to simulate the time evolution of every volume as well as that of the entire system.

The internal behavior of the membranes is therefore described by means of a modified tau-leaping procedure. The original method, first introduced in [8], is based on the stochastic simulation algorithm (SSA) presented in [7]. These approaches are used to describe the behavior of chemical systems, computing the probabilities of the reactions placed inside the system and the length of the step (at each iteration), according to the current system state. While SSA is proved to be equivalent to the Chemical Master Equation (CME), therefore it provides the exact behavior of the system, the tau-leaping method describes an approximated behavior with respect to the CME, but it is faster for what concerns the computational time required.

To describe the correct behavior of the whole system, all the volumes evolve in parallel, through a strategy used to compute the probabilities of the rules (and then, to select the rules that will be executed), and to choose the "common" time increment that will be used to update the system state. The method applied for the selection of the time step length is the following. Each membrane independently computes a candidate time increment (exploiting the tau-leaping procedure), based on its internal state. The smallest time increment among all membranes is then selected and used to describe the evolution of the whole system, during the current iteration. Since all volumes *locally* evolve according to the same time increment, τ-DPP is able to correctly work out the *global* dynamics of the system. Moreover, using the "common" time increment inside the membranes, it is possible to manage the communication of objects among them. This is achieved because the volumes are naturally *synchronized* at the end of each iterative step, when all the rules are executed.

The modified tau-leaping procedure of τ-DPP is also used to select the set of rules that will be executed during the current leap. This is done locally, that is, each membrane selects the kind of evolution it will follow, independently from other volumes. The membrane can evolve in three different manners (as described in [4]), executing either (1) a SSA-like step, or (2) non-critical reactions only, or (3) a set of non-critical reactions plus one critical reaction. A reaction is *critical*, if its reactants are present inside the system in very small amounts. The critical and non-critical reaction sets are identified at the beginning of every iteration. The separation of these two sets is needed in order to avoid the possibility of obtaining negative quantities after the execution of the rules (we refer the reader to [8] for more details).

After this first stage of the procedure, the membranes select the rules that will be used to update the system, exploiting the common time increment previously chosen. A detailed description of the algorithm will be given later on.

Formally, a τ-DPP Υ is defined as

$$\Upsilon = (V_0, \ldots, V_n, \mu, \mathcal{S}, M_0, \ldots, M_n, R_0, \ldots, R_n, C_0 \ldots C_n),$$

where:

- $V_0, \ldots, V_n$ are the volumes of the system, $n \in \mathbb{N}$;
- μ is a membrane structure representing the topological arrangement of the volumes;
- $\mathcal{S} = \{X_1, \ldots, X_m\}$ is the set of molecular species, $m \in \mathbb{N}$, that is, the alphabet of the system;
- $M_0, \ldots, M_n$, are the sets of multisets over $\mathcal{S}$ occurring inside the membranes $V_0, \ldots, V_n$, representing the internal state of the volumes. The multiset M_i $(0 \leq i \leq n)$ is defined over S^*;

– $R_0, \dots, R_n$ are the sets of rules defined in volumes $V_0, \dots, V_n$, respectively. A rule can be of internal or of communication type (as described below);
– $C_0, \dots, C_n$ are the sets of stochastic constants associated to the rules defined in volumes $V_0, \dots, V_n$.

Inside the volumes of a system described by means of τ-DPP, two kinds of evolution rules can be placed. These are called *internal* and *communication* rules. Internal rules describe the evolution of objects that remain in the region where the rule is executed (i.e. target *here*). Communication rules send objects from the membrane where they are applied to an adjacent volume(i.e. target in_j or *out*). Moreover, they can also modify the objects during the communication process.

The sets of stochastic constants $C_0, \dots, C_n$, associated to the sets of rules $R_0, \dots, R_n$, are needed to compute the probabilities of the rule applications (also called propensity functions), along with a combinatorial function depending on the left-hand side of the rule [7].

The general form of internal and communication rules is $\alpha_1 X_1 + \alpha_2 X_2 + \ldots + \alpha_k X_k \rightarrow (\beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k, target)$, where $X_1, \dots, X_k \in \mathcal{S}$ are the molecular species and $\alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_k \in \mathbb{N}$ represent the multiplicities of the objects involved in the rule. Note that we will usually consider the case where at most three objects appear in the left-hand side of the rule. This assumption is related to the fact that the probability of a reaction involving more than three objects is close to zero.

The target of the rules follows the same definition given for the membrane systems in Section 2.

There is a difference in the application of internal and communication rules during the computation of the time increment (τ). In the procedure use to compute τ, while for internal rules both left-hand and right-hand sides are involved, for communication rules only the left-hand side is involved. This distinction is needed because the right-hand side of internal and communication rules is differently used to update the system state. For internal rules the right-hand side modifies the membrane where the rule is applied, whereas for communication rules it affects the state of another membrane, hence it is not considered during the τ computation.

Obviously, the right-hand side of communication rules will contribute to the update of the system state, which takes place at the end of the iterative step, and will be therefore considered to determine the state of the target volume for the next iteration.

We now describe the τ-DPP algorithm needed to simulate the evolution of the entire system. Each step is executed *independently* and *in parallel* within each volume V_i $(i = 0, \dots, n)$ of the system. In the following description, the algorithm execution naturally proceeds according to the order of instructions, when not otherwise specified by means of “go to” commands.

Step 1. Initialization: load the description of volume V_i, which consists of the initial quantities of all object types, the set of rules and their respective stochastic constants.

Step 2. Compute the propensity function a_μ of each rule r_μ, $\mu = 1, \ldots, l$, and evaluate the sum of all the propensity functions in V_i, $a_0 = \sum_{\mu=1}^{l} a_\mu$. If $a_0 = 0$, then `go to` *step 3*, otherwise `go to` *step 5*.

Step 3. Set τ_i, the length of the step increment in volume V_i, to ∞.

Step 4. Wait for the communication of the smallest time increment $\tau_{min} = \min\{\tau_0, \ldots, \tau_n\}$ among those generated independently inside all volumes $V_0, \ldots, V_n$, during the current iteration, then `go to` *step 13*.

Step 5. Generate the step size τ_i according to the internal state, and select the way to proceed in the current iteration (i.e., SSA-like evolution, or tau-leaping evolution with non-critical reactions only, or tau-leaping evolution with non-critical reactions and one critical reaction), using the selection procedure defined in [4].

Step 6. Wait for the communication of the smallest time increment $\tau_{min} = \min\{\tau_0, \ldots, \tau_n\}$ among those generated independently inside all volumes, during the current iteration. Then:
- if the evolution is SSA-like and the value $\tau_i = \tau_{SSA}$ generated inside the volume is greater than τ_{min}, then `go to` *step 7*;
- if the evolution is SSA-like and $\tau_i = \tau_{SSA}$ is equal to τ_{min}, then `go to` *step 10*;
- if the evolution is tau-leaping with non-critical reactions plus one critical reaction, and $\tau_i = \tau_{nc1c}$ is equal to τ_{min}, then `go to` *step 11*;
- if the evolution is tau-leaping with non-critical reactions plus one critical reaction and $\tau_i = \tau_{nc1c}$ is greater than τ_{min}, then `go to` *step 12*;
- if the evolution is tau-leaping with non-critical reactions only ($\tau_i = \tau_{nc}$), then `go to` *step 12*.

Step 7. Compute $\tau_{SSA} = \tau_{SSA} - \tau_{min}$.

Step 8. Wait for possible communication of objects from other volumes, by means of communication rules. If some object is received, then `go to` *step 2*, otherwise `go to` *step 9*.

Step 9. Set $\tau_i = \tau_{SSA}$ for the next iteration, then `go to` *step 6*.

Step 10. Using the SSA strategy [7], extract the rule that will be applied in the current iteration, then `go to` *step 13*.

Step 11. Extract the critical rule that will be applied in the current iteration.

Step 12. Extract the set of non-critical rules that will be applied in the current iteration.

Step 13. Update the internal state by applying the extracted rules (both internal and communication) to modify the current number of objects, and then check for objects (possibly) received from the other volumes. Then `go to` *step 2*.

The algorithm begins loading the initial conditions of the membrane. The next operation is the computation of the propensity functions (and their sum a_0) in order to check if, inside the membrane, it is possible to execute some reaction. If the sum of the propensity functions is zero, then the value of τ is set to ∞ and the membrane waits for the communication of the smallest τ computed among the other membranes (τ_{min}) in order to synchronize with them; then, it checks if it is the target of some communication rule applied inside the other volumes. These operations are needed in order to properly update the internal state of the membrane.

On the other hand, if the sum of all the propensity functions is greater than zero, the membrane will compute a τ value based only on its internal state, following the first part of the original tau-leaping procedure [4]. Besides this operation, the membrane selects the kind of evolution for the current iteration (like the computation of τ, this procedure is executed independently from the other volumes).

The algorithm proceeds to *step 6*, where the membrane receives the smallest τ value computed by the volumes. This will be the common value used to update the state of the entire system. It is necessary to proceed inside every membrane using the same time increment, in order to manage the communication of objects.

At this stage, the membrane knows the length of the time step and the kind of evolution to perform. The next step consists in the extraction of the rules that will be applied in the current iteration. In order to properly extract the rules, several conditions need to be checked.

In the case the membrane is evolving using the SSA strategy: if τ_{min} is the value generated inside itself, then it is possible to extract the rule, otherwise the execution of the rule is not allowed, because the step is “too short”. In the next stage, the membrane verifies for possible incoming objects, to update its internal state according to the communication rules (possibly) executed inside other regions. Finally, if its state is changed (according to some internal or communication rule), then the membrane, in the successive iteration, will compute a new value of τ. On the contrary, the value of the time increment will be the result of the application of *step 7*.

If the evolution strategy corresponds to a tau-leaping step with the application of a set of non-critical reactions and one critical reaction, the algorithm verifies if the value of τ computed by the membrane is equal to τ_{min}. If this is true, the membrane selects the set of non-critical reactions to execute as well as the critical reaction. The execution of the critical reaction is allowed because, here τ_{min} represents the time needed to execute it. Otherwise, the application of the critical reaction is forbidden and the membrane will execute non-critical reactions only.

If the membrane is following the tau-leaping strategy with the execution of non-critical reactions only, τ_{min} is used to extract the rules (from the set of non-criticals) to apply in the current iteration.

The last step is the system update. Here every membrane executes the selected rules and updates its state according to both internal and communication rules. This step is executed in parallel inside every membrane, therefore it is possible to correctly manage the “passage” of objects and to synchronize the volumes.

3 Chemical Computing

In this section we introduce the basic notions of chemical computing, a novel computational paradigm where the information is processed by means of chemical reactions. In particular, starting from the chemical computing field, we recall the basic definitions of chemical organization theory, used to analyze chemical computing systems in order to obtain useful knowledge, such as the emergent behavior of the studied system.

Biological systems are characterized by different mechanisms, employed in their evolution, that make them able to process information. These characteristics are: robustness, self-organization, concurrency, fault-tolerance and evolvability. The global information process comes by using, inside the biological system, a large number of simple components. In particular, information is transformed by means of chemical processes, and for this reason, chemical reactions have been used to build a novel computational paradigm [6]. This new approach is called *chemical computing*, and it is related to the computation with both real molecules and electronic devices, programmed using principles taken from chemistry.

In general, the analysis of the solutions of chemical reaction processes is hard because of their nonlinearity. The same problems are related to the analysis of biological systems since the behavior of local parts can be very different from the global behavior.

In order to work out this problem, the notions of chemical organization theory can be used to obtain the emergent behavior of the system, starting from its small components, hence linking the evolution governed by every single reaction with the global dynamics of the system.

Chemical organization theory [12] is used to identify a hierarchy of self maintaining sub-networks, belonging to a chemical reaction network. These sub-networks are called organizations. In particular, a chemical organization is a set of molecular species that satisfies two properties, that is, it is algebraically closed and stoichiometrically self-maintaining. Here we report an informal definition of these concepts, and refer the reader to [12] for formal definitions and further details.

A *reaction network* is a tuple $\langle \mathcal{M}, \mathcal{R} \rangle$, where $\mathcal{M}$ is a set of molecular species and $\mathcal{R}$ is a set of reactions (also called rules). The rules in $\mathcal{R}$ are given by the relation $\mathcal{R} : P_{\mathcal{M}}(\mathcal{M}) \times P_{\mathcal{M}}(\mathcal{M})$ where $P_{\mathcal{M}}(\mathcal{M})$ denotes the set of all the multisets of the elements in $\mathcal{M}$. The general form of a reaction is $\alpha_1 m_1 + \alpha_2 m_2 + \ldots + \alpha_k m_k \rightarrow \beta_1 m_1 + \beta_2 m_2 + \ldots + \beta_k m_k$, where $m_1, \ldots, m_k \in \mathcal{M}$ are the molecular species involved in the rule and $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \mathbb{N}$ are the coefficients associated to the molecules.

A set of molecular species $\mathcal{C} \in \mathcal{R}$ is *closed*, if its elements are involved in reactions that produce only molecular species of the set $\mathcal{C}$. On the other hand, the *self-maintenance* property is satisfied when the molecules consumed by the reactions involved in the set, can also be produced by some other rule related to the self-maintaining set. Note that, in order to find the organizations of a chemical network, only stoichiometric information (set of rules) is needed.

The set of organizations of a chemical network can be exploited to describe the dynamics of the system, by means of the movement among different organizations. Namely, the dynamics is traced looking at the system state and at the organizations "represented" by the molecules occurring in the system. Therefore, this analysis consists in the study of processes where molecular species appear or disappear from the system (that is, when their amount become positive or go to zero). Note that, only the algebraic analysis of chemical organizations is sufficient in order to obtain this behavior. Furthermore, the behavior of the system can either take place spontaneously or can be induced by means of external events, such as the addition of input molecules.

If we want to use reaction networks to compute, we need to assume that a computational problem can be described as a Boolean function, which in turn can be computed as a composition of many simple functions (e.g. the binary NAND). Therefore, we will create a reaction network (called Boolean network), based on a set of Boolean functions and Boolean variables.

Consider a set of M Boolean functions $F_1, \ldots, F_M$ and a set of N (with $N \geq M$) Boolean variables $\{b_1, \ldots, b_M, \ldots, b_N\}$. The variables b_j, such that $1 \leq j \leq M$, are determined by the Boolean functions (they are also called internal variables). The remaining variables (b_j such that $M < j \leq N$) represent the input variables of the Boolean network. The values computed by the M Boolean functions, are defined as $\{b_i = F_i(b_{q(i,1)}, \ldots, b_{q(i,n_i)})$ with $i = 1, \ldots, M\}$. $b_{q(i,k)}$ is the value of the Boolean variable corresponding to the k-th argument of the i-th function. In general, the function F_i has n_i arguments, therefore, there are 2^{n_i} different input combinations.

Given a Boolean network (as described above), the associated reaction network $\langle \mathcal{M}, \mathcal{R} \rangle$, as presented in [12], is defined as follows. For each Boolean variable b_j, two different molecular species, representing the values 0 and 1 of the variable, are added to $\mathcal{M}$. In particular, lowercase letters are used for the molecular species representing the value 0 and uppercase letters for the value 1 of the variables. Therefore, the set $\mathcal{M}$ contains $2N$ molecular species. The set $\mathcal{R}$ of rules is composed by two kinds of reactions: *logical* and *destructive*. Logical reactions are related to the rows of the truth tables of the functions involved in the Boolean network; hence the left-hand side of the rule represents the input values of the Boolean function, while the right-hand side is the output value. The destructive reactions are needed to avoid the possibility to have, inside the system, two molecular species representing both states of the same variable at the same time (i.e. two molecules representing the state 0 and 1 of the same Boolean variable).

The resulting chemical network $\langle \mathcal{M}, \mathcal{R} \rangle$ implements the Boolean network without inputs specified. The input variables of the Boolean network must be externally initialized because they are not set by the Boolean functions. The initialization is encoded by means of inflow reactions. These reactions are zero-order reactions producing molecules from the empty set.

4 Definition and Simulation of *Component Reaction Networks* Using τ-DPP

To lay out our path from a model of computation to a chemical computing device, we define and simulate test case τ-DPP systems using techniques inspired by the literature on reaction systems [6,12]. Those simple systems must be powerful enough to compute, when assembled in more complex combination, any computable (Boolean) function.

Hence, in this section we describe the implementations of the NAND and XOR logic circuits, and of the decrement and increment instructions of register machines, through sets of chemical reactions. We then present some simulation results of our systems.

We recall that *register machines* [13] are universal abstract computing devices, where a finite set of uniquely labeled instructions is given, and which keep updated a finite set of registers at any time (holding integer numbers) by performing a sequence of instructions, chosen according to their labels. Every instruction can be of one of the following, here informally introduced:

– `ADD`: a specified register is increased by 1, and the label of next instruction is nondeterministically chosen between two labels specified in the last instruction applied,
– `SUB`: a specified register is checked, and if it is non-empty, then it is decreased by 1, otherwise it will not be changed; the next label will be differently chosen in the two cases,
– `HALT`: the machine stops.

Later, we will describe τ-DPP implementations of `SUB` and `ADD` instructions.

4.1 The NAND and XOR Logic Circuits

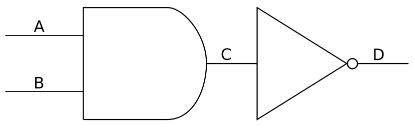
The NAND logic circuit has been implemented with the sequential composition of an AND and a NOT gate as shown in Figure 1 (left). Following the chemical computing guidelines described in Section 3, we define the logic circuit with the rules listed in Figure 1 (right). Rules $r_1, \ldots, r_4$ compute the AND function, rules $r_5, \ldots, r_6$ compute the NOT function and rules $r_7, \ldots, r_{10}$ "clean" the system when both values of a variable are present at the same time, as described in Section 3. Finally, rules $r_{11}, \ldots, r_{14}$ represent the inputs of the gate because they produce the molecules a, A, b and B, representing the inputs $A = 0$, $A = 1$, $B = 0$ and $B = 1$ of the NAND logic circuit, respectively. For instance, when the constants of the rules r_{11} and r_{13} are set to 1, the input given to the NAND gate is 0 for both the input lines because molecules a and b are produced. The rationale behind this, is that the different inputs for the system are obtained producing the molecular species used to represent that particular values. The values of the costants reported in the table have been used to perform the simulation of the NAND behavior by means of τ-DPP.

Starting from the set of rules presented above for the NAND logic circuit, it is possible to define the τ-DPP which encodes the logic circuit. Formally, the τ-DPP Υ_{NAND} is defined as

$$\Upsilon_{NAND} = (V_0, \mu, \mathcal{S}, M_0, R_0, C_0),$$

where:

– V_0 is the unique volume of the NAND logic circuit;
– μ is the membrane structure $[_0\ ]_0$;
– $\mathcal{S} = \{a, A, b, B, c, C, d, D\}$ is the set of molecular species;
– $M_0 = \{a^{m_a}, A^{m_A}, b^{m_b}, B^{m_B}, c^{m_c}, C^{m_C}, d^{m_d}, D^{m_D}\}$, is the set of multisets occurring inside the volume V_0. m_a, m_A, m_b, m_B, m_c, m_C, m_d and $m_D \in \mathbb{N}$;



Reaction	Constant
$r_1 : a + b \rightarrow c$	$c_1 = 1 \cdot 10^{-3}$
$r_2 : a + B \rightarrow c$	$c_2 = 1 \cdot 10^{-3}$
$r_3 : A + b \rightarrow c$	$c_3 = 1 \cdot 10^{-3}$
$r_4 : A + B \rightarrow C$	$c_4 = 1 \cdot 10^{-3}$
$r_5 : c \rightarrow D$	$c_5 = 1 \cdot 10^{-2}$
$r_6 : C \rightarrow d$	$c_6 = 1 \cdot 10^{-2}$
$r_7 : a + A \rightarrow \lambda$	$c_7 = 1 \cdot 10^{-1}$
$r_8 : b + B \rightarrow \lambda$	$c_8 = 1 \cdot 10^{-1}$
$r_9 : c + C \rightarrow \lambda$	$c_9 = 1 \cdot 10^{-1}$
$r_{10} : d + D \rightarrow \lambda$	$c_{10} = 1 \cdot 10^{-1}$
$r_{11} : \lambda \rightarrow a$	$c_{11} \in \{1, 0\}$
$r_{12} : \lambda \rightarrow A$	$c_{12} \in \{1, 0\}$
$r_{13} : \lambda \rightarrow b$	$c_{13} \in \{1, 0\}$
$r_{14} : \lambda \rightarrow B$	$c_{14} \in \{1, 0\}$

Fig. 1. The NAND logic circuit (left) and the set of reactions used to implement it (right)

- $R_0 = \{r_1, \ldots, r_{14}\}$ is the set of rules defined in volume V_0 and reported in Table 1. Due to the membrane structure μ, all the rules here involved are internal.
- $C_0 = \{c_1, \ldots, c_{14}\}$ is the set of stochastic constants associated to the rules defined in R_0, and reported in Table 1.

In Figure 2, the result of the simulation of the NAND gate is reported. In the initial configuration of the system, the multisets are empty, that is, the amounts of all the molecular species are set to zero. At time $t = 0$, the input of the system is a, B, corresponding to the first input line set to zero and the second line set to one. This is formulated as a τ-DPP configuration where the constants of rules r_{11} and r_{14} are set to 1, while the constants of rules r_{12} and r_{13} are set to zero. The output obtained with this configuration is 1, indeed the system, at the beginning of the simulation, produces the molecules D corresponding to the expected output value. At time $t = 400$, the input values of the system are change from a, B to A, B, setting c_{11} and c_{14} to 0 and c_{12} and c_{13} to 1. The system starts producing d molecules, but the output of the system changes only when all the D molecules have been degraded (by means of rule r_{10}) and the molecules d are then accumulated inside the membrane.

The XOR logic circuit (see Figure 3 (left)) has been implemented using the set of rules listed in Figure 3 (right). The rules $r_1, \ldots, r_4$ compute the XOR function and $r_5, \ldots, r_7$ "clean" the system when both values of a variable are present at the same time, as described in Section 3. Finally, the rules $r_8, \ldots, r_{11}$ represent the inputs of the gate. For instance, when the constants of the rules r_8 and r_{10} are set to 1, the input given to the XOR gate is 0 for both the input lines. The values of the constant reported in the table have been used to perform the simulation by means of τ-DPP.

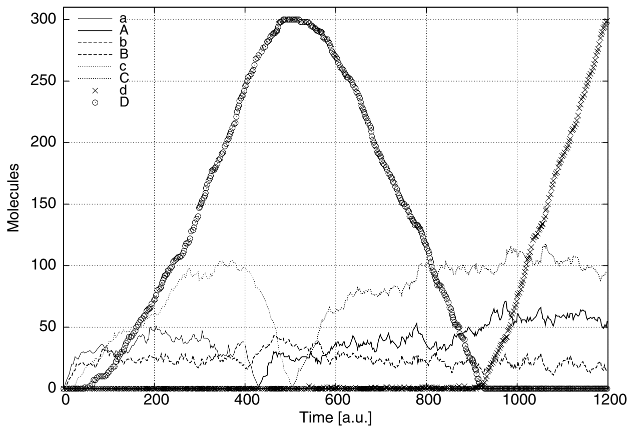
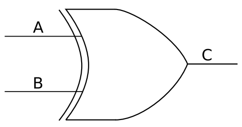


Fig. 2. Plot of the dynamics of the NAND unit with two inputs in succession. The initial multiset is 0 for all the molecular species.



Reaction	**Constant**
$r_1: \ a + b \rightarrow c$	$c_1 = 1 \cdot 10^{-3}$
$r_2: \ a + B \rightarrow C$	$c_2 = 1 \cdot 10^{-3}$
$r_3: \ A + b \rightarrow C$	$c_3 = 1 \cdot 10^{-3}$
$r_4: \ A + B \rightarrow c$	$c_4 = 1 \cdot 10^{-3}$
$r_5: \ a + A \rightarrow \lambda$	$c_5 = 1 \cdot 10^{-1}$
$r_6: \ b + B \rightarrow \lambda$	$c_6 = 1 \cdot 10^{-1}$
$r_7: \ c + C \rightarrow \lambda$	$c_7 = 1 \cdot 10^{-1}$
$r_8: \ \lambda \rightarrow a$	$c_8 \in \{1, 0\}$
$r_9: \ \lambda \rightarrow A$	$c_9 \in \{1, 0\}$
$r_{10}: \lambda \rightarrow b$	$c_{10} \in \{1, 0\}$
$r_{11}: \lambda \rightarrow B$	$c_{11} \in \{1, 0\}$

Fig. 3. The XOR logic circuit (left), and set of reactions used to implement it (right)

Formally, the τ-DPP Υ_{XOR}, corresponding to the XOR logic circuit, is defined as

$$\Upsilon_{XOR} = (V_0, \mu, \mathcal{S}, M_0, R_0, C_0),$$

where:

- V_0 is the unique volume of the XOR logic circuit;
- μ is the membrane structure $[_0 \ ]_0$;
- $\mathcal{S} = \{a, A, b, B, c, C, d, D\}$ is the set of molecular species;
- $M_0 = \{a^{m_a}, A^{m_A}, b^{m_b}, B^{m_B}, c^{m_c}, C^{m_C}\}$, is the set of multisets occurring inside the membrane V_0. m_a, m_A, m_b, m_B, m_c, and $m_C \in \mathbb{N}$;
- $R_0 = \{r_1, \ldots, r_{11}\}$ is the set of rules defined in volumes V_0 and reported in Table 3. Due to the membrane structure μ, all the rules here involved are internal.
- $C_0 = \{c_1, \ldots, c_{11}\}$ is the set of stochastic constants associated to the rules defined in V_0 and reported in Table 3.

In Figure 4, the result of the simulation of the XOR gate is reported. In the initial configuration of the system, the multisets are empty, that is, the amounts of all the molecular species are set to zero. At time $t = 0$, the input of the system is a, B, corresponding to the first input line set to zero and the second one set to one. This is formulated as a τ-DPP configuration where the constants of rules r_8 and r_{11} are set to 1, while the constants of rules r_9 and r_{10} are set to zero. The output obtained with this configuration is 1, indeed the system, at the beginning of the simulation, produces the molecules C corresponding to the expected output value. At time $t = 200$ the input values of the system change from a, B to A, B, setting c_8 to 0 and c_9 to 1. The system starts producing c molecules, but the output of the system changes only when all the C molecules have been degraded (by means of rule r_7) and the molecules c are then accumulated inside the membrane.

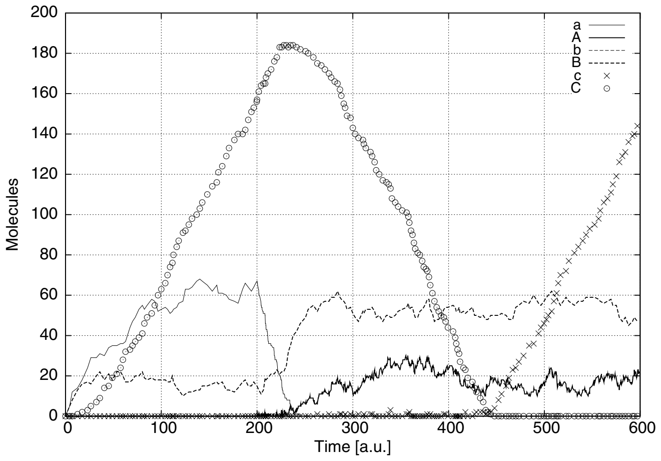


Fig. 4. Plot of the dynamics of the XOR unit with two inputs in succession. The initial multiset is 0 for all the molecular species.

4.2 The SUB Instruction

We now describe and simulate a τ-DPP composed by 2 volumes reproducing, by means of chemical reactions operating on a set of molecular species, the behavior of a `SUB` instruction of a register machine. This type of instruction is important because it hides a conditional behavior, checking whether a register is zero or not, respectively choosing a different label for the next instruction. The availability of conditional instructions is a key issue in computing devices.

We implement a system where the quantity stored inside the register is represented by the amount of objects u occurring in volume V_1. The label for the next instruction in related to the production of objects p or z in volume V_0. Finally, to start the system, objects s are produced inside V_0. s' and z' are additional molecular species used to implement the instruction.

In order to correctly execute the SUB instruction, when molecules s are sent inside the volume V_1, the system first checks if the register value is zero, that is, if any u molecule occurs inside V_1, then, the label for the next instruction is produced.

Formally, a τ-DPP Υ_{SUB} is defined as

$$\Upsilon_{SUB} = (V_0, V_1, \mu, \mathcal{S}, M_0, M_1, R_0, R_1, C_0, C_1),$$

where:

– V_0, V_1 are the volumes of the SUB unit;
– μ is the nested membrane structure $[_0\ [_1\ \ ]_1\ ]_0$;
– $\mathcal{S}_0 = \{p, s, s', z\}$ and $\mathcal{S}_1 = \{p, s, u, z, z'\}$ are the sets of molecular species of volumes V_0 and V_1, respectively;
– $M_0 = \{p^{m_p}, s^{m_s}, s'^{m_{s'}}, z^{m_z}\}$, $M_1 = \{p^{m_p}, s^{m_s}, u^{m_u}, z^{m_z}, z'^{m_{z'}}\}$, are the multisets occurring inside the membranes V_0 and V_1, respectively. m_p, m_s, $m_{s'}$, m_z, m_u and $m_{z'} \in \mathbb{N}$;
– $R_0 = \{r_{0_1}, \ldots, r_{0_5}\}$, $R_1 = \{r_{1_1}, \ldots, r_{1_3}\}$ are the sets of rules defined in volumes V_0, V_1, respectively, and reported in Table 1;
– $C_0 = \{c_{0_1}, \ldots, c_{0_5}\}$, $C_1 = \{c_{1_1}, \ldots, c_{1_3}\}$ is the sets of stochastic constants associated to the rules defined in V_0 and V_1, respectively, and reported in Table 1.

Table 1. Reactions for the SUB unit (R_0 on the left and R_1 on the right). The initial multisets are $\{s'^{40}\}$ in V_0, and $\{u^{20}, z^5\}$ in V_1.

Reaction	**Constant**
$r_{0_1} : 2p \rightarrow (p, here)$	$c_{0_1} = 1$
$r_{0_2} : z + p \rightarrow (z, here)$	$c_{0_2} = 1$
$r_{0_3} : 2z \rightarrow (z, here)$	$c_{0_3} = 1$
$r_{0_4} : s \rightarrow (s, in_1)$	$c_{0_4} = 1$
$r_{0_5} : s' \rightarrow (s, here)$	$c_{0_5} = 6 \cdot 10^{-2}$

Reaction	**Constant**
$r_{1_1} : s + u \rightarrow (p, out)$	$c_{1_1} = 1 \cdot 10^3$
$r_{1_2} : s + z \rightarrow (z + z', here)$	$c_{1_2} = 1$
$r_{1_3} : z' \rightarrow (z, out)$	$c_{1_3} = 1$

The simulation starts with a positive register value within V_1, represented by the u molecules; the system receives a sequence of SUB requests, due to the presence of s' molecules in V_0, transformed in s by the application of rule r_{0_5} and then sent to V_1 by rule r_{0_4}. Figure 5 shows the two execution phases: in the first phase the counter is decremented, as long as there are s' molecules available in V_0, and objects p are produced in V_0. Afterwards, when the register counter reaches zero (all u molecules are consumed), only objects z are produced in V_0.

This system is initialized with small quantities for molecular species, and this makes it fragile with respect to the inherent stochasticity, but our goal is to qualitatively show the required sharp change of behavior occurring when the simulated register goes to zero.

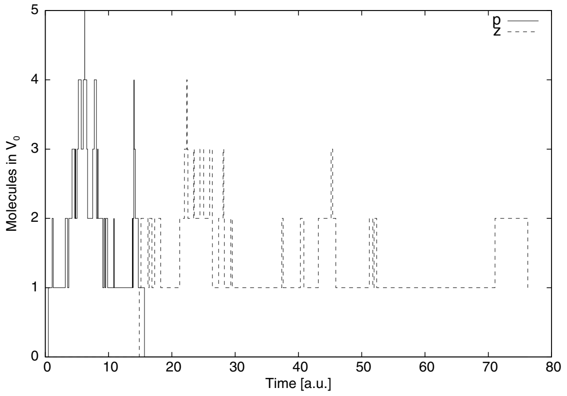


Fig. 5. Plot of the dynamics of the SUB unit

4.3 The SUBADD Module

The SUB unit can be extended to perform both a `SUB` and an `ADD` instructions, according to the object received from the environment: s or a, respectively. The register value is stored inside volume V_2 and it is represented by the amount of molecules u occurring inside of it. The rules shown in Table 2 are defined to perform both operations, and the choice of molecular species avoid mixing them. In particular, rules $r_{0_1}, \ldots, r_{0_4}$, $r_{1_1}, \ldots, r_{1_6}$ and $r_{2_1}, \ldots, r_{2_3}$ define the SUB instruction (checking whether the register value is zero or not). The other rules are used to perform the ADD instruction.

The τ-DPP Υ_{SUBADD} implementing the SUBADD module is defined as

$$\Upsilon_{SUBADD} = (V_0, V_1, V_2, \mu, \mathcal{S}, M_0, M_1, M_2, R_0, R_1, R_2, C_0, C_1, C_2),$$

where:

- V_0, V_1, V_2 are the volumes of the SUBADD module;
- μ is the nested membrane structure $[_0\ [_1\ [_2\ ]_2\ ]_1\ ]_0$;
- $\mathcal{S}_0 = \{l, l', s, z, m, n, p, k, k', a, A, o, q\}$, $\mathcal{S}_1 = \{s, p, z, a, A\}$ and $\mathcal{S}_2 = \{s, u, z, z', a, a'\}$ are the sets of molecular species;
- $M_0 = \{l^{m_l}, l'^{m_{l'}}, s^{m_s}, z^{m_z}, m^{m_m}, n^{m_n}, p^{m_p}, k^{m_k}, k'^{m_{k'}}, a^{m_a}, A^{m_A}, o^{m_o}, q^{m_q}\}$, $M_1 = \{s^{m_s}, p^{m_p}, z^{m_z}, a^{m_a}, A^{m_A}\}$ and $M_2 = \{s^{m_s}, u^{m_u}, p^{m_p}, z^{m_z}, z'^{m_{z'}}, a'^{m_{a'}}\}$, are the multisets occurring inside the membranes V_0, V_1 and V_2, respectively. m_l, $m_{l'}$, m_m, m_k, $m_{k'}$, m_o, m_q, m_s, m_p, m_z, m_a, m_A, m_u, $m_{z'}$ and $m_{a'} \in \mathbb{N}$;
- $R_0 = \{r_{0_1}, \ldots, r_{0_8}\}$, $R_1 = \{r_{1_1}, \ldots, r_{1_8}\}$, $R_2 = \{r_{2_1}, \ldots, r_{2_5}\}$ are the sets of rules defined in volumes V_0, V_1 and V_2 respectively, and reported in Table 2;
- $C_0 = \{c_{0_1}, \ldots, c_{0_8}\}$, $C_1 = \{c_{1_1}, \ldots, c_{1_8}\}$, $C_2 = \{c_{2_1}, \ldots, c_{2_5}\}$ are the sets of stochastic constants associated to the rules defined in V_0, V_1 and V_2, respectively, and reported in Table 2.

The results of the simulations are shown in Figure 6.

Table 2. Reactions for the SUBADD module (R_0 on the left, R_1 on the right and R_2 on the bottom). The initial multisets M_0 and M_1 are empty, while M_2 is $\{u^{30}\}$.

Reaction	Constant
$r_{0_1} : l \rightarrow (l' + s, here)$	$c_{0_1} = 1$
$r_{0_2} : s \rightarrow (s, in_1)$	$c_{0_2} = 1$
$r_{0_3} : l' + z \rightarrow (n, here)$	$c_{0_3} = 1$
$r_{0_4} : l' + p \rightarrow (m, here)$	$c_{0_4} = 1$
$r_{0_5} : k \rightarrow (k' + a, in_1)$	$c_{0_5} = 1$
$r_{0_6} : a \rightarrow (a, in_1)$	$c_{0_6} = 1$
$r_{0_7} : k' + A \rightarrow (o, here)$	$c_{0_7} = 1$
$r_{0_8} : k' + A \rightarrow (q, here)$	$c_{0_8} = 1$

Reaction	Constant
$r_{1_1} : s \rightarrow (s, in_2)$	$c_{1_1} = 1$
$r_{1_2} : 2p \rightarrow (p, here)$	$c_{1_2} = 1$
$r_{1_3} : 2z \rightarrow (z, here)$	$c_{1_3} = 1$
$r_{1_4} : p + z \rightarrow (p, here)$	$c_{1_4} = 1$
$r_{1_5} : z \rightarrow (z, out)$	$c_{1_5} = 1$
$r_{1_6} : p \rightarrow (p, out)$	$c_{1_6} = 1$
$r_{1_7} : a \rightarrow (a, in_2)$	$c_{1_7} = 1$
$r_{1_8} : A \rightarrow (A, out)$	$c_{1_8} = 1$

Reaction	Constant
$r_{2_1} : s + u \rightarrow (p, out)$	$c_{2_1} = 1 \cdot 10^3$
$r_{2_2} : s + z \rightarrow (z + z', here)$	$c_{2_2} = 1$
$r_{2_3} : z' \rightarrow (z, out)$	$c_{2_3} = 1$
$r_{2_4} : a \rightarrow (u + a', here)$	$c_{2_4} = 1$
$r_{2_5} : a' \rightarrow (A, out)$	$c_{2_5} = 1$

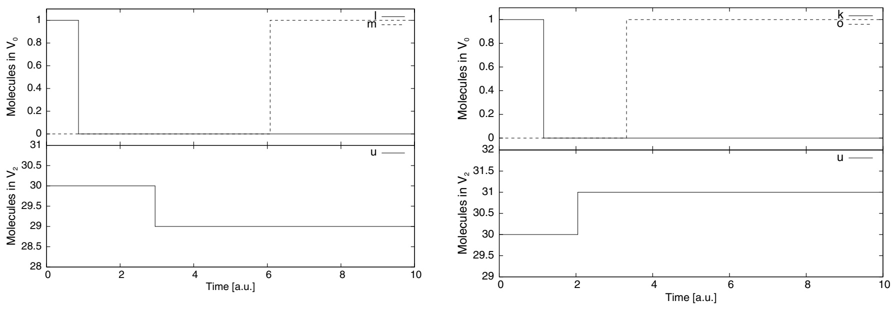


Fig. 6. Plot of the dynamics of the SUBADD module performing a decrement instruction (left) and an increment instruction (right) on a register

5 Complete Systems, Discussion, and Open Problems

Our results are a starting point, since they only tackle the building of basic elements of a computing device. A more complex problem is related to the connectivity among these components.

The general instance of Boolean network, as well as the general reaction network considered in literature, often require a complex grid of channels communicating variables/objects to the required destination gates/volumes.

For usual P systems, such a grid of channels can only be reproduced with a tree-like structure of nested membranes, communicating between adjacent ones. To avoid this limitation, we could move our studies to other variants of P systems, which allow more free adjacency relations between membranes, such as "Tissue P

Systems" [11]. In particular, this P systems variant can be exploited to represent general micro reactors grids.

Within our approach, by using SUBADD modules, we can outline the structure of a τ-DPP system simulating a complete register machine with just three levels of nested membranes: the skin membrane, enclosing a number of "register" membranes structured like volume V_1 in SUBADD module. The key idea to be developed, is to simulate the steps of the register machine by having inside the skin membrane the molecules representing the current instruction label. For instance, if instruction l increments register r, then the rules would be defined to produce objects a_r and send them to an internal membrane representing register r. That internal membrane will then produce objects A_r, and a rule in skin membrane would transform pairs of objects $l + A_r$ into (non-deterministically chosen) objects m, where m is one of the outcome labels specified by the `ADD` instruction being simulated. Additional details related to the halting of the computation need to be specified.

This approach to the implementation of complex systems leads to some open problems worth being studied. How does the passage from single simple components to complete universal devices, with the required connectivity, scale? It is well known that small universal register machines can be built, as shown in [10], but their τ-DPP implementation, and eventually their chemical system implementation have to be evaluated.

Moreover, the computational efficiency of these systems can be studied, for instance with respect to NP-complete problems such as SAT. Anyway, the usual trade-off between space and time in structural complexity perhaps has to be applied with negative results to τ-DPP, since objects could exponentially grow in polynomial time (by using rules like $p \to 2p$), but the space structure of volumes is fixed. Note that, the stochasticity of τ-DPP has to be considered in the computational complexity study.

References

1. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: Modelling metapopulations with stochastic membrane systems. Biosystems 91, 499–514 (2008)
2. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: Seasonal variance in P system models for metapopulations. Progress in Natural Science 17, 392–400 (2007)
3. Berry, G., Boudol, G.: The chemical abstract machine. Theoretical Computer Sci. 96, 217–248 (1992)
4. Cao, Y., Gillespie, D.T., Petzold, L.R.: Efficient step size selection for the tau-leaping simulation method. Journal Chemical Physics 124, 44109 (2006)
5. Cazzaniga, P., Pescini, D., Besozzi, D., Mauri, G.: Tau leaping stochastic simulation method in P systems. In: Hoogeboom, H.J., et al. (eds.) WMC 2006. LNCS, vol. 4361, pp. 298–313. Springer, Heidelberg (2006)
6. Dittrich, P.: Chemical computing. In: Banâtre, J.-P., Fradet, P., Giavitto, J.-L., Michel, O. (eds.) UPP 2004. LNCS, vol. 3566, pp. 19–32. Springer, Heidelberg (2005)
7. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal Physical Chemistry 81, 2340–2361 (1977)

8. Gillespie, D.T., Petzold, L.R.: Approximate accelerated stochastic simulation of chemically reacting systems. Journal Chemical Physics 115, 1716–1733 (2001)
9. Haswell, S.J., Middleton, R.J., O'Sullivan, B., Skelton, V., Watts, P., Styring, P.: The application of microreactors to synthetic chemistry. Chemical Communication, 391–398 (2001)
10. Korec, I.: Small universal register machines. Theoretical Computer Sci. 168, 267–301 (1996)
11. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Sci. 296, 295–326 (2003)
12. Matsumaru, N., Centler, F., Speroni di Fenizio, P., Dittrich, P.: Chemical organization theory as a theoretical base for chemical computing. Intern. J. Unconventional Computing 3, 285–309 (2005)
13. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
14. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
15. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
16. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic P systems. Intern. J. Foundations of Computer Sci. 17, 183–204 (2006)
17. Pescini, D., Besozzi, D., Mauri, G.: Investigating local evolutions in dynamical probabilistic P systems. In: Proc. Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), pp. 440–447. IEEE Computer Society Press, Los Alamitos (2005)
18. Pescini, D., Besozzi, D., Zandron, C., Mauri, G.: Analysis and simulation of dynamics in probabilistic P systems. In: Carbone, A., Pierce, N.A. (eds.) DNA 2005. LNCS, vol. 3892, pp. 236–247. Springer, Heidelberg (2006)