Available online at www.sciencedirect.com
ScienceDirect
BioSystems 91 (2008) 458–472
www.elsevier.com/locate/biosystems

Stochastic P systems and the simulation of biochemical processes with dynamic compartments

Antoine Spicher[a,*], Olivier Michel[a,1], Mikolaj Cieslak[b], Jean-Louis Giavitto[a], Przemyslaw Prusinkiewicz[b]

[a] IBISC-FRE 2873 CNRS & Université d'Évry, Genopole Tour Évry 2, 523 place des terrasses de l'Agora, 91000 Évry, France
[b] Department of Computer Science, University of Calgary, 2500 University Dr. NW, Calgary, Alberta T2N 1N4, Canada

Received 12 May 2006; received in revised form 19 October 2006; accepted 23 December 2006

Abstract

We introduce a sequential rewriting strategy for P systems based on Gillespie's stochastic simulation algorithm, and show that the resulting formalism of stochastic P systems makes it possible to simulate biochemical processes in dynamically changing, nested compartments. Stochastic P systems have been implemented using the spatially explicit programming language `MGS`. Implementation examples include models of the Lotka–Volterra auto-catalytic system, and the life cycle of the Semliki Forest virus.
© 2007 Elsevier Ireland Ltd. All rights reserved.

Keywords: Stochastic simulation algorithm (SSA); Dynamic compartments; Biochemical processes; P systems; SP systems

1. Introduction

Numerous natural processes have been proposed as unconventional paradigms of computation. Biology has been a particularly rich source of ideas, inspiring such notions as neural networks (McCulloch and Pitts, 1943), genetic algorithms (Holland, 1973), cellular automata (Ulam, 1962; Von Neumann, 1966), L-systems (Lindenmayer, 1968; Prusinkiewicz and Lindenmayer, 1990), and membrane computing (Păun, 2001; Cardelli, 2004).

The synergy between biology and computer science is well illustrated by the formalism of Lindenmayer systems. Introduced as a mathematical model of the development of multicellular organisms (Lindenmayer, 1968), L-systems gave rise to a branch of formal language theory (Herman and Rozenberg, 1975; Rozenberg and Salomaa, 1980), before being reapplied to biology and computer graphics as a method for simulating and visualizing plant development (Prusinkiewicz and Lindenmayer, 1990; Prusinkiewicz, 1999). Further applications of L-systems include the generation of space-filling curves (Prusinkiewicz et al., 1991), and geometric modeling (Prusinkiewicz et al., 2003).

In this paper, we present a formalism for stochastic simulation of biochemical processes taking place in compartmentalized structures. Examples of such structures include living cells enclosing the nucleus, the mitochondria, the Golgi complex, and other organelles, or – at a larger scale – tissues and organs comprising individual cells. The formalism combines:

- *Gillespie's stochastic simulation algorithm* (SSA) (Gillespie, 1977), which makes it possible to simulate

* Corresponding author.
E-mail addresses: aspicher@ibisc.fr (A. Spicher), michel@ibisc.fr (O. Michel), cieslak@cpsc.ucalgary.ca (M. Cieslak), giavitto@ibisc.fr (J.-L. Giavitto), pwp@cpsc.ucalgary.ca (P. Prusinkiewicz).
[1] On sabbatical leave at the Department of Computer Science, University of Calgary, 2500 University Dr. NW, Calgary, Alberta T2N 1N4, Canada.

0303-2647/$ – see front matter © 2007 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.biosystems.2006.12.009

reactions in well-mixed chemical systems using the discrete-event simulation paradigm (Kreutzer, 1986) and

- *Păun systems* (P systems) (Păun, 2001), which make it possible to represent processes that take place in nested, dynamic (changing over time) compartments.

Stochastic P systems preserve the definition of atomic operations (application rules) previously defined for P systems, but the commonly used *maximum parallel application strategy* is replaced with a *stochastic sequential strategy*. According to this strategy, atomic operations are chosen at random and applied one at a time. A related strategy was introduced by Obtułowicz (2003), who assumed that the application rules are assigned fixed probabilities. In contrast, we assume that the probabilities may change in the course of simulation. This feature is essential to the implementation of Gillespie's algorithm.

The idea of incorporating stochastic strategies into rewriting systems has a relatively long history. Stochastic and probabilistic L-systems were introduced to the theory of formal languages by Jürgensen (1976); Eichhorst and Savitch (1980), and Yokomori (1980). Related notions were applied by Nishida (1980); Prusinkiewicz (1987), and Prusinkiewicz and Hanan (1989) to simulate variations in the development of modeled plants. The concept of dynamically computing the probabilities of rule application in L-systems was introduced in Prusinkiewicz (1987). A recent extension of L-systems incorporates a stochastic application strategy based explicitly on Gillespie's algorithm (Cieslak, 2006).

In addition to Obtułowicz (2003), stochastic extensions of P systems were proposed by Madhu (2003); Ardelean and Cavaliere (2003), and Pescini et al. (2006). That work was primarily devoted to a theoretical analysis of variants of P systems, expressed in terms of formal language theory. One exception is the paper by Pescini et al. (2006), which was devoted to the modeling and simulation of biochemical processes. We compare their approach to our own in the conclusion. Furthermore, the PhD thesis by Bernardini (2005) has led to recently published results that largely parallel ours (Bernardini et al., 2005; Cazzaniga et al., 2006a, b).

Mechanisms for the probabilistic application of rules were also introduced into general rewriting system environments. In Maude (Koushik et al., 2003), the authors define the notion of *probabilistic rewriting theories*. A probabilistic rewriting strategy was also proposed for the Elan rewriting system (Bournez and Kirchner, 2002; Bournez and Hoyrup, 2003). In both cases, rewriting strategies can be specified by the user. Gillespie's algorithm could thus presumably be coded using these systems, although no example has been given so far.

Gillespie-based stochastic simulation of biochemical systems with static compartments has previously been supported by selected systems biology packages, such as E-cell (Tomita et al., 1999) and StochSim (Novère and Shimizu, 2001). Dynamic compartments have been supported less frequently; a notable exception is the process algebra of BioAmbients (Regev et al., 2004). In contrast to that work, we are able to eliminate a compartment and all its contents in one primitive operation, dissolve a compartment and merge its contents with the parent compartment, create several identical sibling compartments from a single one, and split the contents of a compartment into several siblings.

Our paper is organized as follows. In Sections 2 and 3 we review the two foundations of our work: P systems and Gillespie's stochastic simulation algorithm. These notions are combined into the definition of stochastic P systems (SP systems) in Section 4. In Section 5 we outline an implementation of SP systems in the `MGS` programming language. A systematic translation of SP system rules into `MGS` is described. The resulting implementation makes it possible to simulate biochemical processes that take place in a well-mixed solution or are dynamically compartmentalized. Two examples are given in Section 6. The first example, the Lotka–Volterra auto-catalytic system, only requires a single static compartment. Dynamic compartments are used in the second example, a model of a viral infection. In Section 7 we present conclusions and directions for future work.

2. P Systems

Păun systems, also called *P systems* or *membrane systems*, are a biologically motivated formalism describing parallel distributed computation (Păun, 2000, 2001). P systems are inspired by the organization and functioning of a biological cell.

A cell is considered in an abstract way as a hierarchy of *compartments* enclosed by *membranes*. Each compartment may include elementary objects (molecules) as well as other compartments. Processes in a cell are viewed as sequences of discrete events. Examples of events are: a chemical reaction between molecules within a compartment, transport of molecules outside of, or into a compartment, and creation and dissolution of compartments.

In the following sections, we give a formal definition of the P system formalism. In contrast to the standard approach, we do not represent membranes explicitly, but consider them as a consequence of the nesting of multisets.

2.1. Compartments and Multisets

Let $\mathcal{O} = \{a, b, c, \ldots\}$ be the set of *elementary objects* on which a P system will operate. These objects can be contained in compartments, which are represented as *multisets*: sets in which repetitions of the same element are allowed. By analogy to set notation, the brackets {|and|} are used to enclose the elements of a multiset m. The empty multiset is written as $\{||\}$.

An *elementary compartment* contains only elementary objects. To represent the content of several nested compartments, we consider multisets with elements that are either elementary objects or, recursively, multisets. For example,

$$m = \{|\{|a|\}, b, b, c, \{|a, b, \{|c|\}|\}|\}$$

is a multiset that contains three elementary objects (two elements b and one element c), and two multisets: $m_1 = \{|a|\}$ and m_2. The multiset m_2 contains one element a, one element b and a singleton multiset containing one element c. Several representations of this multiset are shown in Fig. 1. When required, we assign types to compartments and indicate these types using labels (Fig. 1, III).

We use the *cons* operator :: to add an element to a multiset. For example, if $m_1 = \{|1, \{|1|\}, 2|\}$ and $m_2 = \{|2, 2, 3|\}$, then $2 :: m_1$ is equal to $\{|2, 1, \{|1|\}, 2|\}$, and $m1 :: m2$ is equal to $\{|\{|1, \{|1|\}, 2|\}, 2, 2, 3|\}$. Furthermore, we use the *comma* operator to merge the content of two (possibly nested) multisets. For example, $m_1, m_2 = \{|1, \{|1|\}, 2, 2, 2, 3|\}$. Finally, we overload the comma operator to allow one or both of its arguments to be elementary objects. For example, if a and b are elementary objects and m is a multiset, then $a, m = a :: m$ and $a, b = \{|a, b|\}$. With this notation, the expressions $(a :: (b :: (c :: \{||\})))$ and a, b, c denote the same multiset $\{|a, b, c|\}$.

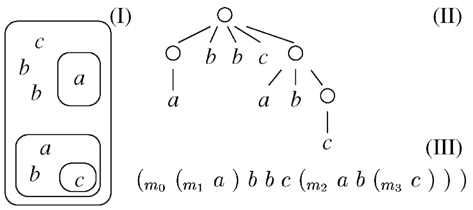
(I)
c
b
b
a
a
b
c
(II)
a
b b c
a b
c
(III)
$(_{m_0}\ (_{m_1}\ a\ )\ b\ b\ c\ (_{m_2}\ a\ b\ (_{m_3}\ c\ )\ )\ )$

Fig. 1. Equivalent representations of a multiset: Venn diagram (I), tree (II), and parenthesised expression (III). In the latter case, labels have been added to indicate the type of each compartment.

2.2. Evolution of a P System State

The state of a P system is represented by a multiset, which may change over time in a discrete fashion. These changes are specified using sets of rules associated with compartment types. A rule $\alpha \to (\beta, \ell)$ consists of the left-hand side α and the right-hand side (β, ℓ). The left-hand side α (the *predecessor*) is a pattern intended to match a sub-multiset of objects that belong to some compartment m. The right-hand side consists of a multiset of objects β (the *successor* or *result*) and a *target location* ℓ. When a rule is applied, the multiset matching α is replaced by the multiset β at location ℓ. The location ℓ is specified by one of the following expressions:

- `here`: the result remains in the same compartment m from which α was taken,
- $\mathtt{in}_{m'}$: the result is *transported* to a compartment m', included in compartment m (this rule can only be applied if m' is (directly) nested in m),
- `out`: the result is transported out of compartment m and added to the parent multiset,
- δ: after replacing α by β as in the case `here`, the boundary surrounding compartment m is removed (compartment m is *dissolved* and all the elements of m are added to its parent compartment).

To shorten the notation, especially when dealing only with elementary objects, we drop the outside brackets enclosing multisets α and β. We also omit the `here` location. Thus a rule $\{|a, b, c|\} \to \{|c, d, d|\}$, `here` is written as $a, b, c \to c, d, d$. This notation is consistent with the definition of comma as an operator that merges elementary objects or multisets into a nested multiset.

Examples of P system rules are given below and illustrated in Fig. 2 under the assumption that each rule applies to compartment m_1:

$a, b \to c$	a and b react to create c
$a \to \{\|\|\}$	a vanishes
$a \to a$, `out`	a `is released into the enclosing compartment`
$a \to a, \mathtt{in}_{m_2}$	a `is transported to` m_2
$a \to \{\|a\|\}_{m_3}$	a is isolated in a newly created compartment
$a \to a, \delta$	the boundary surrounding a is dissolved

2.3. P System Rule Application Strategy

When a rule is applied to a multiset m, the predecessor objects are consumed and deleted from m. Consequently,

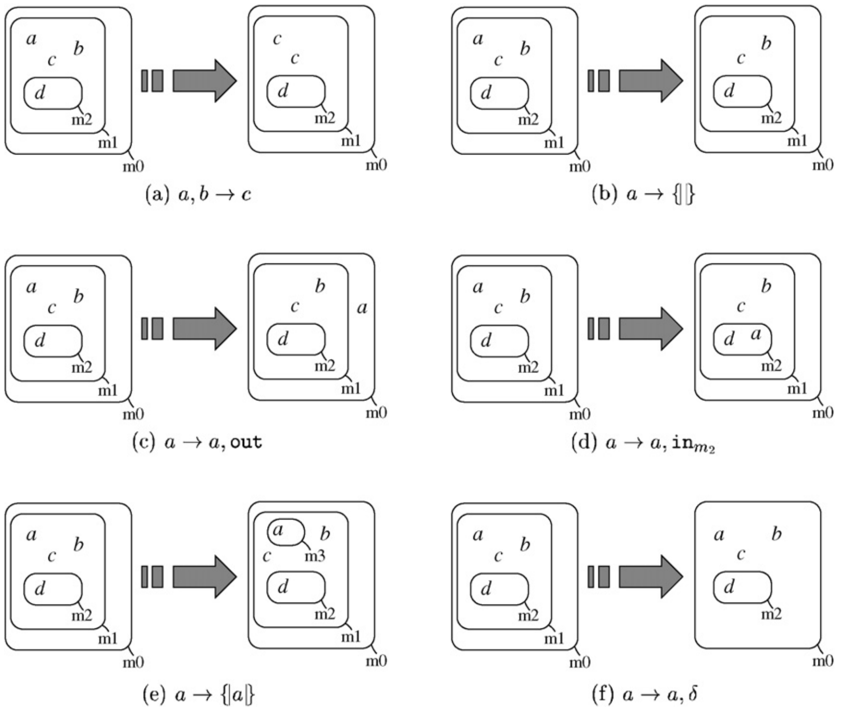

Fig. 2. Example of a rule application in compartment m_1. See text for additional explanations.

two or more rules cannot apply concurrently to the same objects, and one rule has to be chosen. In rewriting systems, the policy for deciding which rule(s) will be applied is called the *application strategy*. For P systems, the *maximal parallel strategy* is commonly used. According to this strategy, rules are applied simultaneously to as many elements as possible, so that no rule matches the remaining elements of the multiset *m*. In the case of conflicts, rules are selected non-deterministically. The motivation for parallel rewriting in P systems is that the passing time affects simultaneously all the elements of the multiset *m*. The same motivation underlies parallel application of productions in L-systems (Lindenmayer, 1968).

The maximal parallel strategy is well suited to the modeling of discrete dynamic systems in which components operate synchronously. It is less well suited to capture events that occur asynchronously in continuous time (Lindenmayer and Jürgensen, 1992), since, as the time interval Δt corresponding to a derivation step decreases, the probability that two events will occur in the same interval decreases as well. This is the case when considering chemical reactions at an atomic scale.

One can consider such situations from the perspective of discrete-event simulation, assuming that events occur one at a time (the probability that two asynchronous events will occur exactly at the same time is equal to zero). This idea underlies Gillespie's algorithm discussed below and leads to an alternative, sequential, rule application strategy for P systems.

3. Stochastic Simulation of Chemical Reactions

Gillespie (1977) developed a stochastic method for simulating well-mixed chemical systems. This method, along with its subsequent improvements and extensions (Gillespie, 2000; Gibson and Bruck, 2000; Gillespie, 2001), has recently found many applications in the area of systems biology. This is due to its suitability for simulating biochemical systems with small numbers of molecules. Such systems cannot be adequately characterized with classical continuous mathematical models of chemical reaction kinetics, because the underlying notion of concentration loses its meaning when the number of molecules is small.

From the computer science point of view, Gillespie's method relies on a discrete-event simulation (Kreutzer, 1986) of reactions between individual molecules. A reaction R_μ, for instance $A + B \to C$, may occur when the reacting molecules (A and B) collide with sufficient energy to yield the product (molecule C). The probability $P(\mu, d\tau)$ that reaction R_μ will take place over an infinitesimal time interval $d\tau$ is proportional to

- the stochastic reaction constant c_μ, which depends on the type of reaction and temperature;
- the number h_μ of distinct combinations of reacting molecules (for example, if the total number of molecules of type A is equal to [A], and the total number of molecules of type B is equal to [B], the number of combinations h_μ is equal to [A][B]; see Gillespie (1976) for further discussion); and
- the length of the time interval $\mathrm{d}\tau$.

We thus have:

$$P(\mu, \mathrm{d}\tau) = h_\mu c_\mu \,\mathrm{d}\tau = a_\mu \,\mathrm{d}\tau, \tag{1}$$

where the product $a_\mu = h_\mu c_\mu$ is called the *propensity* of reaction R_μ.

Let $X(t)$ denote the state of the considered system at time t. We will characterize this state in terms of N multisets X_i of molecules of different species $i = 1, 2, \ldots, N$. Gillespie (1977) showed that the probability $\tilde{p}(\tau, \mu)\mathrm{d}\tau$, with which next reaction R_μ will occur in the infinitesimal time interval $(t + \tau, t + \tau + \mathrm{d}\tau)$, is equal to

$$\tilde{p}(\tau, \mu)\mathrm{d}\tau = a_\mu \,\mathrm{e}^{-a_\mu \tau} \,\mathrm{d}\tau, \tag{2}$$

Eq. (2) differs from Eq. (1) by the term $\mathrm{e}^{-a_\mu \tau}$, which captures the probability that no reaction R_μ will take place in the interval $(t, t + \tau)$. If the total number of different reaction types is M, the probability that the next reaction will be of type μ and will occur in the time interval $(t + \tau, t + \tau + \mathrm{d}\tau)$ is

$$p(\tau, \mu)\mathrm{d}\tau = a_\mu \,\mathrm{e}^{-a_0 \tau}\mathrm{d}\tau, \tag{3}$$

where $a_0 = \Sigma_{\nu=1}^{M} a_\nu$ is the combined propensity of all M reactions (Gillespie, 1977). Adding up the probabilities expressed by Eq. (3) for all reaction types, we obtain the probability $p_1(\tau)\mathrm{d}\tau$ that the first reaction of an arbitrary type will occur in the time interval $(t + \tau, t + \tau + \mathrm{d}\tau)$:

$$\begin{aligned} p_1(\tau)\mathrm{d}\tau &= \sum_{\mu=1}^{M} p(\tau, \mu)\mathrm{d}\tau \\ &= \sum_{\mu=1}^{M} a_\mu \,\mathrm{e}^{-a_0 \tau} \,\mathrm{d}\tau = a_0 \,\mathrm{e}^{-a_0 \tau}\mathrm{d}\tau. \end{aligned} \tag{4}$$

The evolution of the system state over time is simulated by iterating the following steps:

- given system state $X(t)$, determine the type μ of the next reaction and the *inter-reaction time* τ before this reaction takes place,
- modify the state $X(t)$, taking into account the reactants removed from the system and products added to the system by reaction R_μ, and
- advance simulation time t by τ.

Gillespie proposed two methods to determine the reaction type μ and the inter-reaction time τ in a manner consistent with the distribution of probabilities given by Eq. (3). They are called the *direct* method and the *first-reaction* method. In the direct method, the time of the next reaction is chosen using Eq. (4), considering all reaction types at once. A particular reaction is then chosen on the basis of the reaction propensities. In the first reaction method, on the other hand, the time of the first reaction of each type μ is chosen using Eq. (2). The earliest reaction (with the smallest reaction time) is then applied to update the system state, and the simulation time is advanced accordingly.

Specifically, the direct method is based on the conditional probability formula (Gillespie, 1977, p. 418):

$$p(\tau, \mu)\mathrm{d}\tau = p_1(\tau)\mathrm{d}\tau P_2(\mu|\tau), \tag{5}$$

where $p_1(\tau)\mathrm{d}\tau$ is the probability that the next reaction will occur in the time interval $(t + \tau, t + \tau + \mathrm{d}\tau)$, as given by Eq. (4), and $P_2(\mu|\tau)$ is the conditional probability that the next reaction will be R_μ, if the time of the next reaction is $t + \tau$. This conditional probability is obtained by dividing Eq. (3) by Eq. (4):

$$P_2(\mu|\tau) = \frac{p(\tau, \mu)}{p_1(\tau)} = \frac{a_\mu}{a_0}. \tag{6}$$

The inter-reaction time τ and the next reaction R_μ are chosen according to the probabilities given by Eqs. (4) and (6) using the inversion method (Ross, 1989, p. 564). Specifically, given two independent random numbers r_1 and r_2 generated with uniform distribution in the interval [0, 1], the inter-reaction time is obtained using the formula:

$$\tau = \frac{1}{a_0} \ln \frac{1}{r_1}, \tag{7}$$

and the reaction index μ is determined by solving the equation:

$$\sum_{\nu=1}^{\mu-1} a_\nu < r_2 a_0 \le \sum_{\nu=1}^{\mu} a_\nu. \tag{8}$$

In the first reaction method, the time τ_ν of the first reaction of type ν is chosen independently of other reactions for each $\nu = 1, 2, \ldots, M$ with the inversion method applied to Eq. (2). To this end, M independent random numbers r_ν are generated with uniform distribution in the interval [0, 1], and times τ_ν are calculated using Eq. (9), similar to Eq. (7):

$$\tau_\nu = \frac{1}{a_\nu} \ln \frac{1}{r_\nu} \quad \text{for } \nu = 1, 2, \ldots, M. \tag{9}$$

The smallest value τ_ν is then chosen as the inter-reaction time, and the system state X is updated using the corresponding reaction R_ν.

4. Compartmentalized SSA and Stochastic P Systems

4.1. *Gillespie's Algorithm as a Multiset Rewriting Strategy*

Gillespie's algorithm makes it possible to simulate reactions in a well-mixed chemical solution. If such a solution is represented by a multiset whose elementary objects are molecules (Banâtre and Le Métayer, 1986; Dittrich et al., 2001), then chemical reactions can be expressed as multiset rewriting rules. Gillespie's algorithm leads to a *sequential application strategy* for these rules: only one rule is applied in each derivation (simulation) step.

The sequential application strategy represents a considerable departure from the maximal parallel application strategy usually considered for P systems. In the theory of formal languages, the distinction between sequential and parallel rewriting plays a fundamental role, leading to different hierarchies of languages: Chomsky versus Lindenmayer (Herman and Rozenberg, 1975; Rozenberg and Salomaa, 1980). This distinction may also be relevant to the formal properties of P systems and deserves a further study. Nevertheless, here we only consider the modeling applications of stochastic P systems.

4.2. *Handling Compartments*

The potential presence of nested compartments in P systems violates the assumption of homogeneous spatial distribution of molecules on which Gillespie's algorithm is based. Nevertheless, the SSA can be extended to nested compartments as follows:

- Reactions taking place within compartments are simulated by considering each compartment individually (we assume here that molecules within each compartment are distributed homogeneously);
- P system rules involving transport of molecules and creation and dissolution of membranes are assigned their own propensities and treated as reactions, although they may affect two compartments at a time.

Our extension preserves the discrete-event simulation character of Gillespie's method and treats reactions and transport events as occuring instantaneously.

We define a derivation step in a stochastic P system by analogy to the direct or first reaction method. In the *direct method*, reactions of the same type, but associated with different compartments, are formally treated as distinct reactions with their own propensities. This distinction is achieved by renaming identical molecules, and their associated reactions, that appear in different compartments. After this renaming, the next reaction is selected, and the simulation time advanced, as in the single compartment situation.

In the *first reaction method*, the SSA is applied to each compartment c separately, yielding reaction R_c and reaction time τ_c for each compartment. The compartment with the smallest reaction time is then selected and the corresponding reaction is applied. A detailed algorithm for the first reaction method is given below.

Let `split()` be the function that divides a nested multiset m into two parts: the multiset of elementary objects belonging to $\mathcal{O}$ and the multiset of the remaining multisets:

$$\texttt{split}(m) = \langle m';m'' \rangle$$
$$\text{where } m' = \{|x, x \in m \text{ and } x \neq \in \mathcal{O}|\},$$
$$m'' = \{|x, x \in m \text{ and } x \notin \mathcal{O}|\}.$$

Furthermore, let R_m denote the set of rules applicable to m, and $\langle \tau; p \rangle = SSA(m)$ be the result of the application of one of these rules according to the original Gillespie's algorithm. In the pair $\langle \tau; p \rangle$, τ is the time increment related to the application of the selected rule to m, and p is the new multiset. A simulation step of a stochastic P system is then given by the following recursive function:

```
function $nestedSSA$ ($m$: nested multiset)
  $\langle m'; m'' \rangle := \texttt{split}(m)$
  $\langle \tau_0; n_0 \rangle := SSA(m')$
  let $N = \textbf{size}\,(m'')$
  for $i = 1$ to $N$ do
    $\langle \tau_i; n_i \rangle := nestedSSA(m''_i)$
  let $j$ such that $\tau_j = \min_{(0 \le i \le N)} \tau_i$
  if $j = 0$ then return $\langle \tau_0; (n_0, m'') \rangle$
  else return $\langle \tau_j; m' :: m''_1 :: \ldots :: m''_{j-1} :: n_j :: m''_{j+1} :: \ldots :: m''_N \rangle$
```

The above pseudo-code can be implemented in various programming environments. An example is given in the next section.

5. Implementation of Stochastic P Systems in MGS

MGS is a domain-specific programming language supporting the modeling and simulation of dynamical systems with a dynamical structure (Giavitto et al., 2003). Numerous examples of such systems are found in the area of biology. Computation in MGS consists of the application of rewriting rules to dynamic data structures. The rules and data structures are defined in local terms, using the notion of neighborhood rather than global coordinates or indexing schemes. Different types of neighborhood (Giavitto and Michel, 2002) can be specified within MGS, leading to a unified treatment of collections of objects with different topologies (called *topological collections*).

Below we present the features of MGS that are relevant to the implementation of stochastic P systems.

5.1. Representation of P Systems States

As defined in Section 2, the state of a P system is a nested multiset, called *bag* in the context of MGS. Each element of a bag is a neighbor of all other elements. Elements of bags can be any values supported by MGS including *numbers* and *symbols*. Symbols are denoted by back-quoted identifiers as ‘*X*.

The empty bag is written `bag : ()`. The operations on bags include `cons` (::) and `comma`, as defined in Section 2. For example, the nested multiset of Fig. 1 can be specified using the following MGS expression:

```
‘c :: #2 ‘b :: (‘a :: +bag : ())
:: (‘a :: ‘b :: (‘c :: bag : ()) :: bag : ()) :: bag : ().
```

The nesting of compartments is specified by the parentheses. The syntax #2 ‘X :: m is an abbreviation for ‘X :: ‘X :: m.

To handle P systems with typed compartments (cf. Figs. 1 and 2), we rely on the notion of *sub-typing* provided by MGS. Sub-typing in MGS associates different sub-types to various instances of objects of the same type. For example, the following statements create bags of two sub-types A and B:

collection A = bag; ;

collection B = bag; ;

The expressions A:() and B:() refer to empty bags of different sub-types within the common type bag.

5.2. Transformations

To manipulate topological collections, MGS provides a unifying construct, called *transformation*. A transformation is a function defined by cases. Each case corresponds to a specific rewriting rule. An MGS rewriting rule consists of the *left-hand side*, a rule *qualifier*, and the *right-hand side*. The left-hand side is a pattern that specifies a sub-collection to which the rule may be applied. The qualifier characterizes conditions of rule application. The right-hand side evaluates to the sub-collection that will replace the sub-collection matched by the left-hand side.

The pattern syntax follows the grammar:

```
Atom ::= l|id|id : t,
Pattern ::= Atom|Atom,Pattern|Atom \ /Pattern
```

An Atom matches a literal value (l) or a pattern variable bound to an element and used in the right-hand side of the rule (id). The construct id:*t* matches a variable id of type *t*. A Pattern is a finite sequence of Atoms.

The comma operator in the left-hand side of a rule denotes the neighborhood relationship. For example, the pattern x, y matches two elements that are neighbors. In the context of bags, in which each element is a neighbor of any other element, x, y matches any pair of elements.

The \/ construct, termed *down*, is used to descend into a multiset nested within the current one. For example, if $m = \{|a, \{|b|\}, \{|c, d|\}, e|\}$, the pattern a,n \ /(c,d) will match the sub-collection $\{|a, \{|c, d|\}|\}$ of m.

As a simple example of MGS code, let us consider a variant of the sieve of Eratosthenes that computes the bag of all prime numbers between 2 and n, given the bag that contains all integers from 2 to n. The idea is to iterate the transformation that substitutes y for a pair x, y such that y divides x:

```
trans prime = {x,y⇒ if (x%y) == 0 then
                  y else x, y fi}
```

The prime transformation consists of only one rule. The operator % computes the remainder from the division of x by y. If any two values x and y in the bag are such that y divides x then x is removed. If y does not divide x then the pair x, y is replaced by itself.

Once defined, this transformation can be applied in several ways:

(1) only once, like an ordinary function: `prime(M)`;
(2) n times, using the `iter` option: `prime[iter=`n`](M)`;
(3) until some predicate P holds: `prime[iter = `P`](M)` (the argument of the predicate P is the result returned by the last application of the transformation); and
(4) until the fixed point has been reached: `prime[iter = ‘fixpoint](M)`.

By default, MGS transformations are applied using the maximal parallel rewriting strategy. However, MGS also supports a parameterized sequential application strategy (Spicher et al., 2006; Spicher and Michel, 2006), which is suitable for implementing stochastic P systems.

5.3. *Gillespie's SSA in* MGS

A sequential stochastic rule is specified using arrow qualifiers. The following two forms are available:

(1) $=\{\mathtt{C}=c_\mu\}\Rightarrow$ to explicitly give the stochastic reaction constant c_μ for the rule;
(2) $=\{\mathtt{A}=\backslash\mathtt{self.f(self)}\}\Rightarrow$ to specify the propensity of the rule.

In the second case, `f(self)` is a function of the multiset to which the transformation applies. This function is specified using a notation based on lambda-calculus, \x.*exp* is a function of argument x with body *exp*. For example, the propensity of the rule:

$$\text{`X, `Y} = \{\mathtt{A} = \backslash\mathtt{self.count(self, `X)}$$
$$\mathtt{*count(self, `Y)}\} \Rightarrow \text{`Z}$$

is computed by evaluating the function that returns the number of symbols ‘X multiplied by the number of symbols ‘Y in the current bag self.

The use of the stochastic sequential application strategy is indicated by the transformation option [strategy = ‘gillespie]. For example,

T[strategy = ‘gillespie](m)

applies transformation T to the bag m. It is assumed that each rule of T is qualified by either a stochastic reaction constant or a propensity function. The elapsed time is available through a global variable ‘tau. The applied reacting rule is chosen using the first reaction method.

5.4. *Stochastic P Systems in* MGS

The full implementation of stochastic P systems that operate on nested multisets with dynamic membranes is based on the *nestedSSA* algorithm presented in Section 4.1. The translation of a stochastic P system into MGS raises two problems: (1) P system rules can be constrained to specific compartments while MGS transformations are defined globally, and (2) there are no MGS transformations that correspond directly to the P system transport, compartment creation and dissolution rules. In other words, only P system rules of type `here` are supported in MGS.

The first problem is solved by considering as many bag types as there are rule sets attached to specific compartments. Thus, for each rule set M, there is an associated bag type M and an associated MGS transformation $\mathtt{T}_M$. The MGS implementation of the *nestedSSA* algorithm is then modified so that for a bag of type M only the transformation $\mathtt{T}_M$ applies.

The second problem is properly addressed by coding the P system rules that transport into a compartment, out of a compartment, or dissolve a compartment. This is achieved by including `out` and δ rules in each transformation T. Table 1 gives the translation of all possible stochastic P system rules.

The first case is obvious. For the `in` rule, we match a bag m of type M′ (the destination of the result) and the pattern α, then we replace the matched elements by the bag m with β added. For the `out` rule, we match a bag m of type M containing an occurrence of α, and we replace m by β and m with α removed (cf. the previous description of the down pattern \/). The propensity of the translated rule is explicitly computed by counting the number of occurrences of α in bag m. This rule is added in each transformation. The rule for the dissolution makes use of the `flat` qualifier (Giavitto and Michel, 2001): the elements of the collection β that appears on the right-hand side are added to the current collection, instead of being nested into the current collection as a single element. Using the `flat` feature, it is easy to translate a dissolution rule: we match a bag m of type M that contains

Table 1
Translation for stochastic P system rules into an MGS transformation

Rule in M	Corresponding MGS rule	Appears in
$\alpha\rightarrow_{c_\mu}\beta$, `here`	$\alpha=\{\mathtt{C}=c_\mu\}\Rightarrow\beta$	$\mathtt{T}_M$ only
$\alpha\rightarrow_{c_\mu}\beta$, $\mathtt{in}'_{\mathtt{M}}$	$\mathtt{m}:\mathtt{M}', \alpha=\{\mathtt{C}=c_\mu\}\Rightarrow\beta::\mathtt{m}$	$\mathtt{T}_M$ only
$\alpha\rightarrow_{c_\mu}\beta$, `out`	$\mathtt{m}:\mathtt{M}\backslash/\alpha=\{\mathtt{A}=\backslash x.c_\mu*\mathtt{count(m,}\alpha\mathtt{)}\}\Rightarrow\beta::\mathtt{m}$	Each T
$\alpha\rightarrow_{c_\mu}\beta, \delta$	$\mathtt{m}:\mathtt{M}\backslash/\alpha=\{\mathtt{A}=\backslash x.c_\mu*\mathtt{count(m,}\alpha\mathtt{)flat}\}\Rightarrow\beta::\mathtt{m}$	Each T
$\alpha\rightarrow_{c_\mu}\{\lvert\beta\rvert\}_{M'}$	$\alpha=\{\mathtt{C}=c_\mu\}=>\beta::M'$	$\mathtt{T}_M$ only

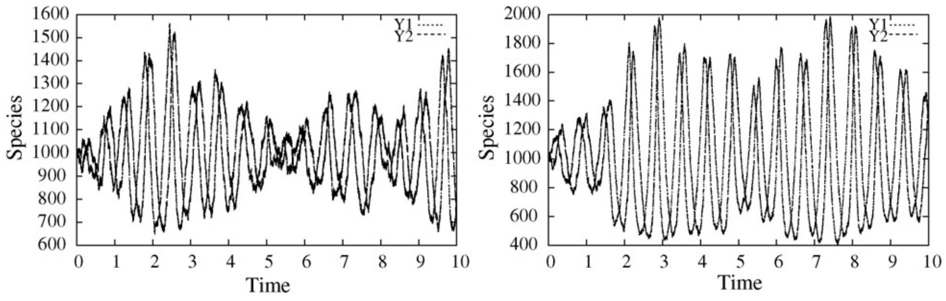

Fig. 3. Results of two simulations using the Lotka–Volterra model.

an occurrence of α, and we replace it by inserting the elements of the bag m with α removed and β added. The propensity of the translated rule is explicitly computed by counting the occurrences of α in bag m. This rule is added in each transformation. Finally, the last rule builds a new collection of type M′ with elements β within the current collection.

6. Examples

Below we present two examples of stochastic P systems and their MGS implementations. The first example is an application of the Gillespie's algorithm coded in MGS. The second example illustrates the use of dynamic compartments.

6.1. A model of the Lotka–Volterra Process

The Lotka–Volterra process was introduced by Lotka as a model of coupled auto-catalytic chemical reactions, and was investigated by Volterra as a model for studying an ecosystem of predators and prey (Edelstein-Keshet, 1988). The reaction rules are as follows:

$$X + Y_1 \rightarrow X + Y_1 + Y_1,$$
$$Y_1 + Y_2 \rightarrow Y_2 + Y_2, \quad Y_2 \rightarrow Z$$

The dynamics of these reactions is conveniently characterized using the predator–prey interpretation. The first rule states that a prey Y_1 reproduces after feeding on a food resource X; this resource is renewable and thus its concentration does not change as a result of feeding. The second rule states that a predator Y_2 reproduces after feeding on prey Y_1. Finally, the last rule specifies that predators Y_2 die of natural causes.

In the MGS expression of these rules, the members of (ecological or chemical) species are represented by symbols in a bag. Stochastic reaction constants are specified as the C qualifiers of the rules. In the example below we assumed that these constants are equal to 0.001, 0.01 and 10, respectively:

```
trans lotka_volterra = {
  ‘X,  ‘Y1  ={ C = 0.001 }=> #2‘Y1, ‘X;
  ‘Y1, ‘Y2  ={ C = 0.01  }=> #2‘Y2;
  ‘Y2       ={ C = 10    }=> ‘Z
} ;;
```

A simulation of the Lotka–Volterra system consists of an iterative application of the lotka_volterra transformation, beginning with the initial state of the system. Such an application can be specified by the following MGS code:

```
lotka_volterra[iter\ = x.(tau >= tmax),
  strategy = ‘gillespie]
  (#10000‘X,#1000‘Y1,#1000‘Y2,bag : ());;
```

We assumed here that the iteration will proceed until the elapsed time 'tau reaches tmax = 10. The initial state of the system consists of 10,000 members of species X, 1000 members of species Y_1, and 1000 members of species Y_2. Traces of two stochastic simulations, generated using different seeds for the random number generator, are shown in Fig. 3. The simulations reveal oscillations in the populations of both species Y_1 and Y_2, which is consistent with the dynamics of the Lotka–Volterra model (Edelstein-Keshet, 1988). The use of stochastic simulations reveals random variation in the process, which is absent from deterministic simulations based on differential equations.

6.2. A Model of Viral Infection

We present a high-level model of a viral infection that follows the process outlined by Alberts et al. (1994, pp. 273–280). The example involves the formation and dissolution of membranes, as well as the transport of individual molecules and entire compartments. This pro-

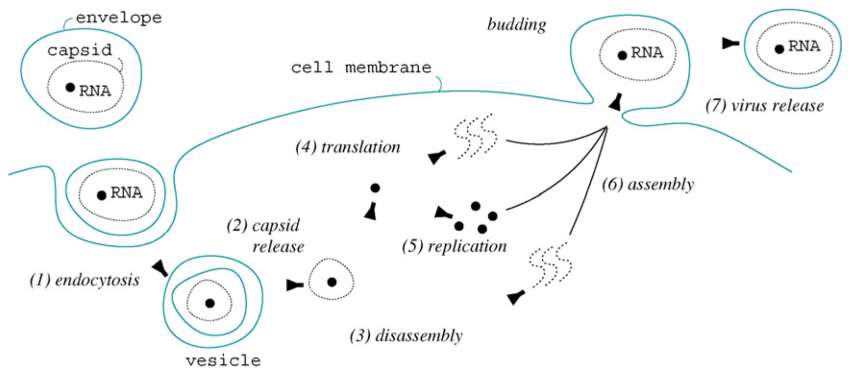

Fig. 4. Rough sketch of the seven steps describing a viral infection of the Semliki Forest virus. The description of each step is given in the text.

cess has previously been modeled using *brane calculi* (Cardelli, 2004), which treats dynamic nested compartments in a manner similar to P systems. However, brane calculi do not capture the stochastic aspect of molecular reactions.

6.2.1. Biological Background

Viruses are genetic elements enclosed in a protein coat, which makes it possible for them to move from one cell to another. The structure and life cycle of the *Semliki Forest* virus are shown in Fig. 4. The virus consists of a single strand of *viral RNA* surrounded by a shell called *capsid*. The capsid is composed of many virus is surrounded by a second shell called the *envelope*. An infection is initiated when the virus binds to a receptor protein in the membrane of the host cell (Phase 1 in Fig. 4.

copies of the same *C* protein. Outside a cell, the The virus then enters a healthy cell following a standard cellular endocytosis pathway. Upon entering, the virus acquires an additional membrane, called a *vesicle*, which is derived from the cell membrane. Subsequently, both the vesicle and the envelope dissolve, releasing the capsid (Phase 2). The capsid is then disassembled into the viral RNA and the *C* proteins that formed the capsid (Phase 3). The viral RNA is translated into the structural proteins of the virus (Phase 4), and it is replicated (Phase 5). The old and the newly synthesized *C* proteins then bind to the viral RNA to form new capsids (Phase 6). When a capsid comes into contact with the cellular membrane, it is lined with the viral envelope and buds out to recreate the initial virus structure outside of the cell. This virus may now infect another healthy cell (Phase 7).

6.2.2. Stochastic P Systems Model

We model infection by the Semliki Forest virus in the following way. A multiset of type *Universe* represents the whole system comprised of cells and viruses. A healthy cell is an empty multiset of type *Cell* (we ignore the internal structure of the healthy cell, as it does not play a role in our model). In contrast, an infected cell contains viruses and their components. A virus outside of a cell is a multiset of type *Envelope*, which contains a single multiset of type *Capsid*. The capsid, in turn, contains one *RNA* molecule. Inside a cell, an *Envelope* multiset may be further contained in a *Vesicle* multiset.

With the multiset representation of the biological compartments involved in the model, endocytosis (Phase 1) corresponds to an `in` rule:

$$\Big\{\!\!\Big|_{Envelope} \{\!|_{Capsid}\, RNA |\!\}\, \Big|\!\!\Big\} \longrightarrow_{C_1} \Big\{\!\!\Big|_{Vesicle} \Big\{\!\!\Big|_{Envelope} \{\!|_{Capsid}\, RNA |\!\}\, \Big|\!\!\Big\} \Big|\!\!\Big\}, \mathtt{in}_{Cell}$$

This is the only rule associated with the *Universe* multiset. The stochastic reaction constant C_1 is proportional to the probability that a virus will encounter a cell in the universe. This probability has the form:

$c_1[Envelope][Cell]$

where c_1 is a constant coefficient.

The dissolution of the vesicle and the envelope (Phase 2), as well as the disassembly of the capsid (Phase 3), are captured by the P system dissolution rules. The translation (Phase 4) and replication (Phase 5) of RNA are reactions taking place inside a cell. The assembly of a new capsid (Phase 6) is a multiset creation rule, and the release of the virus (Phase 7) is an `out` rule. The entire set of rules associated with a cell thus has the form:

$$
\begin{aligned}
\Big\{\!\!\Big|_{Vesicle} \; \Big\{\!\!\Big|_{Envelope} \; \{\!|_{Capsid} RNA |\!\} \; \Big|\!\!\Big\} \; \Big|\!\!\Big\} &\longrightarrow_{C_2} \{\!|_{Capsid} RNA |\!\} \\
\{\!|_{Capsid} RNA |\!\} &\longrightarrow_{C_3} RNA, \#5C \\
RNA &\longrightarrow_{C_4} C, RNA \\
RNA &\longrightarrow_{C_5} \#2RNA \\
\#5C, RNA &\longrightarrow_{C_6} \{\!|_{Capsid} RNA |\!\} \\
\{\!|_{Capsid} RNA |\!\} &\longrightarrow_{C_7} \Big\{\!\!\Big|_{Envelope} \; \{\!|_{Capsid} RNA |\!\} \; \Big|\!\!\Big\}, \mathtt{out}
\end{aligned}
$$

We assume here that a capsid consists of five C proteins. Now that the stochastic P systems rules have been defined, we can implement them in MGS.

6.2.3. MGS *Implementation*

We represent compartments involved in this model as bag collections of different types:

collection Universe = bag; ;

collection Capsid = bag; ;

collection Envelope = bag; ;

collection Cell = bag; ;

collection Vesicle = bag; ;

A virus outside a cell is defined as:

(‘RNA :: Capsid : ()) :: Envelope : (); ;

The processes describing the viral infection take place in two compartments: the Universe, where the virus enters or leaves a cell, and a Cell. The first MGS transformation describes the activities in the Universe:

```
trans t_Universe = {
  P1 =
    e:Envelope, ce:Cell
    ={A = \x.(c1 * count(Envelope,x) * count(Cell,x))}=>
    (e::Vesicle:()) :: ce;
  P7 =
    (ce : Cell) \/ (ca : Capsid)
    ={A = \x.(c7 * countAll(Cell,Capsid,x))}=>
    (ca::Envelope:()), ce;
} ;;
```

The numbering of the rules corresponds to the numbering of phases in Fig. 4. The virus entering a cell is described by the in rule P1. The virus exiting a cell is described by the out rule P7. The propensities are computed explicitly. The function countAll(Cell,Capsid,x) counts all the capsids present in all the cells within the universe x. A comma operator is used in the right-hand side of rule P7 to incorporate a virus that has left a cell into the universe.

The second MGS transformation describes processes taking place in a cell:

```
trans t_Cell = {
  P2 =  (v:Vesicle) \/ ((e:Envelope) \/ (ca:Capsid))
        ={A = \x.(c2 * count(Vesicle,x))}=>
        ca;
  P3 =  (ca:Capsid) \/ ‘RNA
        ={A = \x.(c3 * count(Capsid,x))}=>
        ‘RNA, #5 ‘C;
  P4 =  ‘RNA ={C=c4}=> ‘C,‘RNA;
  P5 =  ‘RNA ={C=c5}=> #2 ‘RNA;
  P6 =  #5 ‘C,‘RNA ={C=c6}=> ‘RNA::Capsid:();
};;
```

In rule P2, the \/ operator has been used twice to match a Vesicle that contains an Envelope containing a Capsid. The propensities of the first two rules are computed explicitly. The propensities of the remaining three rules are computed automatically by MGS, given the stochastic reaction constants.

6.2.4. *A Simulation Example*

Fig. 5 shows the result of four simulations that began with 20 healthy cells and 1, 10 or 100 viruse molecules. The initial state was specified by the expression:

initial_state := (#n(‘RNA :: Capside : ()) ::

Envelope : ()) :: #20 Cell : () :: Universe : (); ;

where #n is the number of viruses. In the absence of quantitative data, all stochastic reaction constants were set to the same value of 1.0. Fig. 5(a) highlights the discrete-event nature of the stochastic simulation algorithm, with the infrequent events separated in time for small virus molecule numbers. Individual runs significantly differ from each other in this case. Fig. 5(b) and (c) show two typical runs beginning with 10 virus molecules. These simulations differ in details, but generally proceed in a similar manner. The variance between runs is further reduced for larger initial numbers of molecules (Fig. 5(d)). The mean time between consecutive events decreases as the number of molecules grows.

As expected, the simulations show that the total numbers of RNA molecules, vesicles, capsids, and viruses

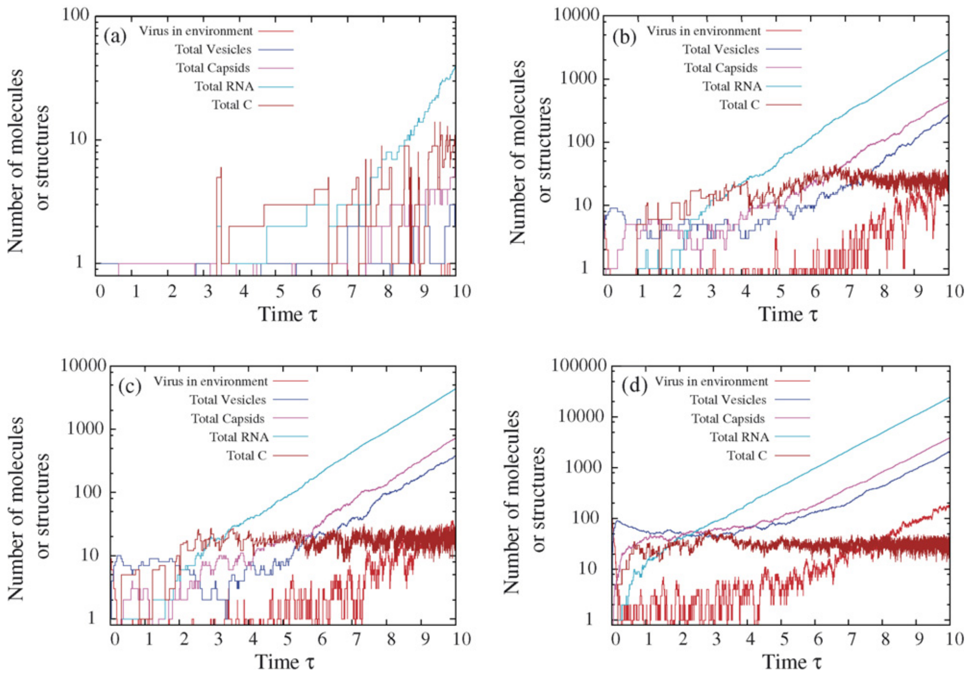

Fig. 5. Sample simulation results for the Semliki Forest virus infection. Simulation begins with 20 healthy cells and (a) 1, (b and c) 10, or (d) 100 virus molecules. Each curve shows the total number of molecules or structures.

increase exponentially with time. In contrast, after an initial increase, the number of free C proteins appears to saturate. We interpret this as the effect of almost immediate reincorporation of free C proteins into newly formed capsids. More in-depth applications of the presented model will be possible once experimental data related to the reaction times become available.

6.2.5. Performance Analysis

Simulation times for the Semliki Forest virus infection model are shown in Fig. 6. The model was expressed in the `MGS` language and implemented using the `MGS`system.[1] All simulations were performed on a Dell GX 260 computer with the Intel Pentium IV 1800 MHz processor and 1 GB of RAM, running under Linux Debian Sarge (Debian, 2006) with kernel 2.4.26.

Simulations of up to several thousand events are executed in a few seconds, which makes it possible to explore the model interactively. As the number of events increases, the simulation times grow exponentially. This reflects the linear relation between the exponentially increasing number of molecules in the Semliki Forest virus infection model and the search time for the next reaction (Eq. (8)). The simulation times could be reduced using binary search to determine the next reaction (Gibson and Bruck, 2000), or using further extensions of Gillespie's algorithm, such as τ-leaping (Gillespie, 2001) or R-leaping (Auger et al., 2006).

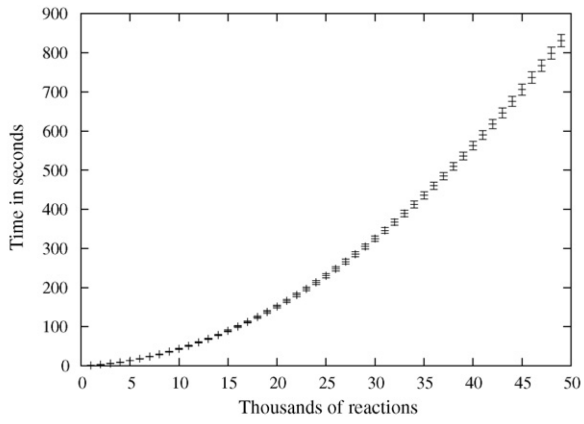

Fig. 6. Performance analysis of the simulation of the Semliki Forest virus infection. The plot represents mean values and variance of execution times for 20 runs.

7. Conclusions

In this paper we presented stochastic P systems as a formalism for modeling and simulating biochemical processes that take place in dynamic, nested compartments.

[1] The source code and executables for the `MGS` system are freely available at http://mgs.ibisc.univ-evry.fr.

We also proposed an implementation of stochastic P systems in MGS, an experimental programming language designed to support computing in topological spaces.

Our objectives are related to those of Pescini et al. (2006), who simulated chemical reactions using probabilistic P systems. While their approach preserves the maximal parallel rule application strategy originally proposed for P systems, our method is based on the sequential application of stochastic rules introduced by Obtułowicz (2003). In contrast to that work, we assumed, as does Bernardini (2005), that the rules may have dynamically computed probabilities. This made it possible to relate the resulting formalism to Gillespie's stochastic simulation algorithm, the fundamental algorithm for stochastic simulation of chemical reactions. We also applied stochastic P systems to model biochemical systems with dynamic and nested compartments. The results are illustrated using two examples: a simulation of the Lotka–Volterra process, based on a straightforward application of Gillespie's algorithm, and a simulation of a virus infection, which involves dynamic nested compartments. Our results show that P systems are relevant not only as a biologically motivated theoretical model of computation, but also as a basis for modeling and simulation in systems biology.

Many problems are open for further work. One direction is the acceleration of computation. A straightforward approach is the replacement of Gillespie's algorithm by its computationally more efficient counterpart, proposed by Gibson and Bruck (2000). Furthermore, in a multiprocessing environment, the simulation of biochemical reactions that take place simultaneously in different compartments can be viewed as an instance of parallel discrete-event simulation. The effectiveness of such simulations can be improved using the notions of *virtual time* (Jefferson, 1985) and *time warp* (Jefferson et al., 1987).

The second direction is the addition of geometric features to stochastic P systems. The modeling and simulation of systems in which compartments can expand and contract represents a theoretical challenge with important practical ramifications (Takahashi et al., 2005; Lemerle et al., 2005). For example, such models may represent fundamental processes in a cell, such as cytokinesis and mitosis. As these processes take place over an extended period of time, a further extension of the model may be needed, lifting the assumption of instantaneous reactions. In addition, inclusion of geometry may provide a basis for considering the impact of the volume of compartments on the propensities of reactions. Stochastic P systems may represent a potentially useful point of departure for modeling and simulating such processes within a well-founded formalism.

Acknowledgements

We thank Brendan Lane for editorial assistance, and the anonymous referees for insightful and helpful comments. The support of the Centre National de la Recherche Scientifique ACI grant "NANOPROG" to O.M., and the Natural Sciences and Engineering Research Council of Canada Discovery Grant RGP 130084 to P.P. is gratefully acknowledged.

References

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., Watson, J., 1994. Molecular Biology of the Cell, 3rd ed. Garland, New York.

Ardelean, I., Cavaliere, M., 2003. Modelling biological processes by using a probabilistic P system software. Nat. Comput. 2 (2), 173–197.

Auger, A., Chatelain, P., Koumoutsakosa, P., 2006. R-leaping: accelerating the stochastic simulation algorithm by reaction leaps. J. Chem. Phys. 125, 084103-1-084103-13.

Banâtre, J.P., Le Métayer, D., 1986. A new computational model and its discipline of programming. Technical Report RR-0566, INRIA.

Bernardini, F., 2005. Membrane systems for molecular computing and biological modelling. Ph.D. thesis, University of Sheffield, Sheffield, UK.

Bernardini, F., Gheorghe, M., Krasnogor, N., Muniyandi, R.C., Pérez-Jiménez, M.J., Romero-Campero, F.J., 2005. On P systems as a modelling tool for biological systems. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (Eds.), Workshop on Membrane Computing, vol. 3850. Lecture Notes in Computer Science. Springer, pp. 114–133.

Bournez, O., Hoyrup, M., pp. 61–75 2003. Rewriting logic and probabilities. In: Nieuwenhuis, R. (Ed.), Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA'03), vol. 2706. Lecture Notes in Computer Science. Springer, Berlin.

Bournez, O., Kirchner, C., pp. 252–266 2002. Probabilistic rewrite strategies. applications to ELAN. In: Tison, S. (Ed.), Proceedings of Rewriting Techniques and Applications, 13th International Conference, vol. 2378. Lecture Notes in Computer Science. Copenhagen, Denmark, Springer.

Cardelli, L., 2004. Brane calculi. In: Danos, V., Schächter, V. (Eds.), Computational Methods in Systems Biology, vol. 3082. Lecture Notes in Computer Science. Springer, Berlin, pp. 257–278.

Cazzaniga, P., Pescini, D., Besozzi, D., Mauri, G., 2006a. Tau leaping stochastic simulation method in P systems. In: Pre-Proceedings of the 7th Workshop on Membrane Computing, WMC7, Leiden, The Netherlands.

Cazzaniga, P., Pescini, D., Romero-Campero, F.J., Besozzi, D., Mauri, G., pp. 145–164 2006b. Stochastic approaches in P systems for simulating biological systems. In: Gutiérrez-Naranjo, M.A., Păun, Gh., Riscos-Núñez, A., Romero-Campero, F.J. (Eds.), Proceedings of the 4th Brainstorming Week on Membrane Computing, vol. I. Fénix Editora. Sevilla, Spain.

Cieslak, M., 2006. Stochastic simulation of pattern formation: an application of L-systems. Master's thesis, University of Calgary.

Debian, 2006. The Debian project web site. http://www.debian.org.

Dittrich, P., Ziegler, J., Banzhaf, W., 2001. Artificial chemistries—a review. Artif. Life 7 (3), 225–275.

Edelstein-Keshet, L., 1988. Mathematical Models in Biology. Random House, New York.

Eichhorst, P., Savitch, W.J., 1980. Growth functions of stochastic Lindenmayer systems. Inform. Control 45 (3), 217–228.

Giavitto, J.-L., Godin, C., Michel, O., Prusinkiewicz, P., 2003. Modeling and simulation of biological processes in the context of genomics. In: Hermes, Dieppe, Ch. (Eds.), Computational Models for Integrative and Developmental Biology.

Giavitto, J.-L., Michel, O., 2001. MGS: a rule-based programming language for complex objects and collections. Electr. Notes Theor. Comput. Sci. 4, 59.

Giavitto, J.-L., Michel, O., 2002. The topological structures of membrane computing. Fund. Inform. 49 (1–3), 107–129.

Gibson, M.A., Bruck, J., 2000. Efficient exact stochastic simulation of chemical systems with many species and many channels. J. Chem. Phys. 104, 1876–1889.

Gillespie, D.T., 1976. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J. Comput. Phys. 22, 403–434.

Gillespie, D.T., 1977. Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. 81 (25), 2340–2361.

Gillespie, D.T., 2000. Chemical Langevin equation. J. Chem. Phys. 113, 297–306.

Gillespie, D.T., 2001. Approximate ¡!–¡query¿Please check the deletion of reference Gillespie (2001b) which was the repetition of reference Gillespie (2001a).¡/query¿–¿accelerated stochastic simulation of chemically reacting systems. J. Chem. Phys. 115, 1716–1733.

Herman, G.T., Rozenberg, G., 1975. Developmental Systems and Languages. North-Holland, Amsterdam.

Holland, J.H., 1973. Genetic algorithms and the optimal allocation of trials. SIAM J. Comput. 2 (2), 88–105.

Jefferson, D.R., 1985. Virtual time. ACM Trans. Program. Lang. Syst. 7 (3), 404–425.

Jefferson, D.R., Beckman, B., Wieland, F., Blume, L., 1987. Distributed simulation and the time warp operating system. Oper. Syst. Rev. 21, 77–93.

Jürgensen, H., 1976. Probabilistic L-systems. In: Lindenmayer, A., Rozenberg, G. (Eds.), Automata, Languages, Development. North-Holland, Amsterdam, pp. 211–225.

Koushik, S., Kumar, N., Meseguer, J., Agha, G., 2003. Probabilistic rewrite theories. Technical Report 2343, University of Illinois at Urbana Champaign.

Kreutzer, W., 1986. System Simulation Programming Styles and Languages. Addison-Wesley Publishing Co., Reading, MA.

Lemerle, C., Di Ventura, B., Serrano, L., 2005. Space as the final frontier in stochastic simulations of biological systems. FEBS Lett. 579, 1789–1794.

Lindenmayer, A., 1968. Mathematical models for cellular interaction in development. Parts I and II. J. Theor. Biol. 18, 280–315.

Lindenmayer, A., Jürgensen, H., 1992. Grammars of development: discrete-state models for growth, differentiation, and gene expression in modular organisms. In: Ronzenberg, G., Salomaa, A. (Eds.), Lindenmayer Systems, Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology. Springer, pp. 3–21.

Madhu, M., 2003. Probabilistic rewriting P systems. Int. J. Found. Comput. Sci. 14 (1), 157–166.

McCulloch, W.S., Pitts, W., 1943. A logical calculus of ideas immanent in nervous activity. Bull. Math. Biophys. 5, 115–133.

Nishida, T., 1980. K0L-systems simulating almost but not exactly the same development—the case of Japanese cypress. Memoirs Fac. Sci., Kyoto University, Ser. Biol. 8, 97–122.

Novère, N.L., Shimizu, T.S., 2001. STOCHSIM: modelling of stochastic biomolecular processes. Bioinformatics 17 (6), 575–576.

Obtułowicz, A., 2003. Probabilistic P systems. In: Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (Eds.), Membrane Computing, International Workshop, Curtea de Arges, Romanai, vol. 2597. Lecture Notes in Computer Science. Springer, Berlin, pp. 377–387.

Păun, Gh., 2000. The P system web page: http://psystems.disco.unimib.it/.

Păun, Gh., 2001. From cells to computers: computing with membranes (P systems). Biosystems 59 (3), 139–158.

Pescini, D., Besozzi, D., Mauri, G., Zandron, C., 2006. Dynamical probabilistic P systems. Int. J. Found. Comput. Sci. 17 (1), 183–204.

Prusinkiewicz, P., pp. 534–548 1987. Applications of L-systems to computer imagery. In: Ehrig, H., Nagl, M., Rosenfeld, A., Rozenberg, G. (Eds.), Proceedings of the 3rd International Workshop on Graph Grammars and their Application to Computer Science, vol. 291. Lecture Notes in Computer Science. Springer, Berlin.

Prusinkiewicz, P., 1999. A look at the visual modeling of plants using L-systems. Agronomie 29, 211–224.

Prusinkiewicz, P., Hanan, J., 1989. Lindenmayer Systems, Fractals and Plants. Springer, Berlin.

Prusinkiewicz, P., Lindenmayer, A., Hanan, J.S., Fracchia, F.D., Fowler, D.R., de Boer, M.J.M., Mercer, L., 1990. The Algorithmic Beauty of Plants. Springer, New York.

Prusinkiewicz, P., Lindenmayer, A., Fracchia, F.D., 1991. Synthesis of space-filling curves on the square grid. In: Peitgen, H.-O., Henriques, J.M., Penedo, L.F. (Eds.), Fractals in the Fundamental and Applied Sciences. North-Holland, Amsterdam, pp. 341–366.

Prusinkiewicz, P., Samavati, F.F., Smith, C., Karwowski, R., 2003. L-system description of subdivision curves. Int. J. Shape Model. 9 (1), 41–59.

Regev, A., Panina, E., Silverman, W., Cardelli, L., Shapiro, E., 2004. Bioambients: an abstraction for biological compartments. Theor. Comput. Sci. 325 (1), 141–167.

Ross, S.M., 1989. Introduction to Probability Models, 4th ed. Academic Press.

Rozenberg, G., Salomaa, A., 1980. The Mathematical Theory of L-systems. Academic Press, New York.

Spicher, A., Michel, O., 2006. Stratgie d'application stochastique de rgles de rcritures dans le langage MGS. In: Michel, O. (Ed.), Journes Francophones des Langages Applicatifs. INRIA, Rocquencourt.

Spicher, A., Michel, O., Giavitto, J.-L., 2006. Rewriting and Simulation—Application to the Modeling of the Lambda Phage Switch. No. 5 in Modlisation de systmes biologiques complexes dans le contexte de la gnomique. Genopole, Evry.

Takahashi, K., Arjunan, S.N.V., Tomita, M., 2005. Space in systems biology of signaling pathways–towards intracellular molecular crowding in silico. FEBS Lett. 579, 1783–1788.

Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T.S., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter

III, J.C.C.A.H., 1999. E-cell: software environment for whole-cell simulation. Bioinformatics 15 (1), 72–84.

Ulam, S.M., 1962. On some mathematical problems connected with patterns of growth of figures. Proc. Symp. Appl. Math. 14, 215–224.

Von Neumann, J., 1966. Theory of Self-Reproducing Automata. University of Illinois Press, Urbana and Chicago.

Yokomori, T., 1980. Stochastic characterizations of EOL languages. Inform. Control 45 (1), 26–33.