Fundamenta Informaticae XX (2009) 1–19
IOS Press

Timed P Automata

Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo
Dipartimento di Informatica, Università di Pisa
Largo Pontecorvo 3, 56127 Pisa, Italy
Email: {barbuti, maggiolo, milazzo}@di.unipi.it

Luca Tesei
Dipartimento di Matematica e Informatica
Università di Camerino
Via Madonna delle Carceri 9, 62032 Camerino (MC), Italy
Email: luca.tesei@unicam.it

Abstract. To study systems whose dynamics changes with time, an extension of timed P systems is introduced in which evolution rules may vary with time. The proposed model is a timed automaton with a discrete time domain and in which each state is a timed P system. A result on expressive power and on features of the formalism sufficient for full expressiveness is proved and, as an application example, the model of an ecological system is given.

1. Introduction

P systems were introduced by Păun in [12] as distributed parallel computing devices inspired by the structure and the functioning of a living cell. A P system consists of a *hierarchy of membranes*, each of them containing a multiset of *objects*, representing molecules, a set of *evolution rules*, representing chemical reactions, and possibly other membranes. For each evolution rule there are two multisets of objects, describing the reactants and the products of the chemical reaction. A rule in a membrane can be applied only to objects in the same membrane. Some objects produced by the rule remain in the same membrane, others are sent *out* of the membrane, others are sent *into* the inner membranes, which are

Address for correspondence: Paolo Milazzo
Email address: milazzo@di.unipi.it

identified by their labels. Evolution rules are applied with *maximal parallelism*, meaning that it cannot happen that some evolution rule is not applied when the objects needed for its triggering are available.

Many variants and extensions of P systems exist that include features to increase their expressiveness and that are based on different evolution strategies (for instance, different forms of parallelism). Among the most common extensions we mention P systems with dissolution rules that allow a membrane to disappear and release in its environment all the objects it contains. We mention also P systems with priorities, in which a priority relationship exists among the evolution rules of each membrane and can influence the applicability of such rules, and P systems with promoters and inhibitors, in which the applicability of evolution rules depends on the presence of at least one occurrence and on the absence, respectively, of a specific object. An introduction to P systems in which all the variants we mentioned are described can be found in [14].

In [5] a class of P systems, called timed P systems, has been proposed. In timed P systems, with each rule an integer is associated that represents the time needed by the rule to be entirely executed.

P systems, originally defined as a model of computation inspired by biology, have also been used to model biological systems, in particular biomolecular systems. Recently, in [4], P systems have been used to model ecological systems in the framework of conservation biology. The population dynamics of the Bearded Vulture of the Catalan Pyrenees, and of five subfamilies on which the vulture feeds on, is studied.

In the framework of conservation biology we are interested in the study of the population dynamics when the environment conditions may change. Consider, as an example, populations whose dynamics changes with seasons.

To describe such systems, we introduce an extension of timed P systems in which evolution rules can change over time. This is expressed by means of a timed automaton [1], with a discrete time domain, in which each state (location) is a timed P system. Timed P systems in different locations of the automaton have the same membrane structure, but possibly different evolution rules. As usual in timed automata, a transition of a timed P automaton may have conditions on the value of clocks and may reset some clocks. The execution of a timed P automaton starts from a location designated as the initial one. The timed P system associated with this location is executed and the passage of time in the timed P system coincides with the updates of the clocks of the timed P automaton. When such clocks take values that satisfy the condition of some outgoing transition of the current location of the automaton, such a transition can be performed.

When a transition from a location q to a location q' of the automaton occurs, the timed P system in q stops its execution, all the objects in its membranes are transferred to the same membranes in the target location q', and the timed P system in q', with its own timed evolution rules, is started.

In order to increase the modelling capabilities of timed P automata, we allow some changes to be made in the multisets of objects transferred from the timed P system in q to the timed P system in q' when the transition is performed. In particular, we enrich transitions with operations for adding objects to and removing objects from the skin membrane. Transitions enriched with operations for removing objects can be performed only if such objects are present in the skin membrane.

The possibility of changing rules when moving from a location to another allows us to model changes of dynamics determined by changes in the environmental conditions. As an example, we could have locations corresponding to seasons and population dynamics could be different in different seasons. Moreover, the possibility of adding and removing objects in the skin membrane allows us to model changes of the population composition determined by changes in the environmental conditions.

In [6] a class of contents-timed P systems is defined in which timed evolution rules are used and their execution times may vary as the timed P system evolves, depending on the state of the system itself. This work is related to ours as it also defines a variant of P systems in which the reaction rules can change dynamically. However, the changing is limited to the execution times of the rules, which remain fixed. In timed P automata, instead, the rules can be entirely rewritten, depending on time passing (but not on the state of the timed P system).

In [8, 7] a variant of P systems, called *P automata*, is introduced. Despite the name similarity, the behaviour of P automata is quite different from the one of timed P automata we propose here. P automata are essentially accepting P systems which use, in most cases, only communication rules.

In this paper, after recalling the definition of timed P systems taken from [5], in Section 3.1, we define timed P automata, in Section 3.2. In Section 4 we study the expressive power of the model and we show that in order to have universality it is sufficient to consider timed P automata whose locations are associated with systems of only one membrane, and in which all objects are equal, but for one particular object whose use in reaction rules is constrained (a bi-stable catalyst). In Section 5 we show an application of timed P automata to the modelling of an ecological system, namely the population of Saddleback birds on Mokoia Island. The timed P automaton we consider is derived from the mathematical model given in [2] to guide the reintroduction of extirpated birds in the New Zealand mainland. Finally, in Section 6 we draw some conclusions.

This paper is a revised and extended version of [3]. In particular, in the present version, we prove the universality of Timed P Automata using only context-free rules.

2. Background

In this section we recall the definitions of n-register machine and of matrix grammar with appearance checking. We shall use these notions in proofs of theorems.

2.1. Register Machines

An n-register machine [11] is a tuple $M = (n, B, \mathtt{i}, \mathtt{f})$ in which n is a natural number expressing the number of registers that the machine can use, B is a set of labelled instructions, $\mathtt{i}$ is the label of the initial instruction of the program and $\mathtt{f}$ is the label of the final instruction. Each register of the machine holds a natural number (in a unary representation) that can be increased by 1 or, conditionally (if not zero), decreased by 1. Each labelled instruction can be of the form:

- $\mathtt{k} : (S(i), \mathtt{l}, \mathtt{m})$, where $\mathtt{k}$ is the label and i is the register affected by the operation S (for Subtract 1). If the content of the register i is greater than zero then the machine subtracts 1 and continues at instruction $\mathtt{l}$, else it does nothing and continues at instruction $\mathtt{m}$.

- $\mathtt{k} : (A(i), \mathtt{l})$, where $\mathtt{k}$ is the label and i is the register affected by the operation A (for Add 1). The machine adds 1 to register i and continues at instruction $\mathtt{l}$.

- $\mathtt{f} : halt$, where $\mathtt{f}$ is the label of the final instruction of the program. The machine does nothing and halts.

The machine computes a partial function $f\colon \mathbb{N} \to \mathbb{N}$ as follows. Let $h \in \mathbb{N}$ be the initial content of register 1. If the machine halts and the content of register 1 is r, then the machine has computed $f(h) = r$. If the machine does not halt, then $f(h)$ is undefined. In [11] it is proved that a 5-register machine can simulate any Turing machine. Moreover, by using a suitable representation of data and instructions, universality can be proved for 2-register machines. In [10] it is proved that universality can be obtained also with a 3-register machine without any particular coding of data and operations.

2.2. Matrix Grammars with Appearance Checking

A (context-free) matrix grammar with appearance checking [14, 15] is a tuple $G = (N, T, S, M, F)$, where N and T are disjoint alphabets of non–terminals and terminals, respectively, $S \in N$ is the axiom, M is a finite set of matrices, namely sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$ of context–free rules over $N \cup T$ with $n \geq 1$, and F is a set of occurrences of rules in the matrices of M. For a string w, a matrix $m : (r_1, \ldots r_n)$ can be executed by applying its rules to w sequentially in the order in which they appear in m. Rules of a matrix occurring in F can be skipped during the execution of the matrix if they cannot be applied, namely if the symbol in their left–hand side is not present in the string.

Formally, given $w, z \in (N \cup T)^*$, we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*$ with $1 \leq i \leq n+1$ such that $w = w_1$, $z = w_{n+1}$ and, for all $1 \leq i \leq n$, either (1) $w_i = w'_i A_i w''_i$ and $w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or (2) $w_i = w_{i+1}$, A_i does not appear in w_i and the rule $A_i \to x_i$ appears in F. We remark that F consists of *occurrences* of rules in M, that is, if the same rule appears several times in the matrices, it is possible that only some of these occurrences are contained in F.

The language generated by a matrix grammar with appearance checking G is defined as $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$, where $\Longrightarrow^* w$ is the reflexive and transitive closure of $\Longrightarrow$. The family of languages of this form is denoted by MAT^{λ}_{ac}, when rules having the empty string λ as right hand side (λ–rules) are allowed, and by MAT_{ac} when such rules are not allowed. Moreover, the family of languages generated by matrix grammars without appearance checking (i.e. with $F = \varnothing$) is denoted by MAT^{λ}, when λ–rules are allowed, and by MAT, when such rules are not allowed. It is known that (i) $MAT \subset MAT_{ac} \subset CS$; (ii) $MAT^{\lambda} \subset MAT^{\lambda}_{ac} = RE$, where CS and RE are the families of languages generated by context–sensitive and arbitrary grammars, respectively.

Let $ac(G)$ be the cardinality of F in G, and let $|x|$ denote the length of the string x. A matrix grammar with appearance checking $G = (N, T, S, M, F)$ is said to be in the *strong binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these sets mutually disjoint, and the matrices in M are in one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$;
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$;
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$;
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2, x \in T^*, |x| \leq 2$.

Moreover, there is only one matrix of type 1, F consists exactly of all rules $A \to \#$ appearing in matrices of type 3 and $ac(G) \leq 2$. We remark that $\#$ is a trap symbol, namely once introduced it cannot be removed, and a matrix of type 4 is used only once, in the last step of a derivation.

For each matrix grammar (with or without appearance checking) there exists an equivalent matrix grammar in the strong binary normal form. Consequently, for each language $L \in RE$ there exists a matrix grammar with appearance checking G satisfying the strong binary normal form and such that $L(G) = L$.

A matrix grammar with appearance checking in the strong binary normal form is always given as $G = (N, T, S, M, F)$, with $N = N_1 \cup N_2 \cup \{S, \#\}$ and with $n + 1$ matrices in M, injectively labeled with $m_0, m_1, \ldots, m_n$. The matrix $m_0 : (S \to X_{init}A_{init})$ is the initial one, with X_{init} a given symbol from N_1 and A_{init} a given symbol from N_2; the next k matrices are without appearance checking rules, $m_i : (X \to \alpha, A \to x)$, with $1 \leq i \leq k$, where $X \in N_1, \alpha \in N_1 \cup \{\lambda\}, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$ (if $\alpha = \lambda$, then $x \in T^*$); the last $n-k$ matrices, with $n-k \leq 2$, have rules to be applied in the appearance checking mode, $m_i : (X \to Y, A \to \#)$, with $k + 1 \leq i \leq n, X, Y \in N_1$, and $A \in N_2$.

Since the grammar is in the strong binary normal form, we have (at most) two symbols $B^{(1)}$ and $B^{(2)}$ in N_2 such that the rules $B^{(j)} \to \#$ appear in matrices m_i with $k + 1 \leq i \leq n$.

We remark that in matrix grammars in strong binary normal forms we can assume that all symbols $X \in N_1$ appear as the left-hand side of a rule from a matrix: otherwise, the derivation is blocked after introducing such a symbol, hence we can remove these symbols and the matrices involving them.

3. The Model

In this section we introduce timed P automata, a computational model that brings together timed automata [1] and P systems [13]. In this model time plays a fundamental role both by expressing a duration for the evolution rules of a P system and by specifying the changing of rules over time. Roughly speaking, a timed P automaton (TPA for short) consists of a timed automaton, with a discrete time domain, in which each state (location) is a timed P system. Different timed P systems [5], in the states of the automaton, have the same membrane structure but different rules.

A timed P system, introduced formally in Section 3.1, is a P system in which evolution rules require a given number of time units to be completed. A timed P automaton switches from a location to another one depending on the values of clocks (as in timed automata).

We use natural numbers $\mathbb{N}$ for representing units of time. While the automaton stays in a location its clocks increase by one time unit simultaneously and with the same speed. The time passing measured by clocks is synchronized and consistent with the time passing used by the timed evolution rules of the timed P system running in the location. The formal machinery of timed automata that we need for our purposes is introduced in Section 3.2.

3.1. Timed P Systems

A P system consists of a hierarchy of membranes that do not intersect, with a distinguishable membrane – called the skin membrane – surrounding them all. We assume membranes to be labelled by natural numbers. Membranes contain multisets of objects, evolution rules and possibly other membranes. In the biological metaphor, objects represent molecules swimming in a chemical solution, and evolution rules represent chemical reactions that may occur inside the membrane containing them. An evolution rule is a pair of multisets of objects, denoted $u \to v$, describing reactants and products of a chemical reaction. In timed P systems the evolution rules take the form $u \xrightarrow{n} v$, where $n \in \mathbb{N}^{>0}$ (positive natural numbers) represents the time units needed for the application of the rule to be completed.

An evolution step of a timed P system can be described as follows. Rules in a membrane can be applied only to objects in the same membrane, and they cannot be applied to objects contained in inner membranes. The rules must contain target indications specifying the membranes where the new objects, obtained by the rule application, are sent. The new objects either remain in the same membrane, or can be sent out of the membrane, or can be sent into one of the inner membranes precisely identified by its label.

The multiset of an evolution rule describing the products of the represented chemical reaction contains objects having one of the following forms: a_{here} – the new object a remains in the same membrane of the applied rule; a_{out} – the new object a is sent outside; a_{in_j} – the new object a is sent into the membrane labelled by j.

Rules application is done by using *maximal parallelism*: at each evolution step a multiset of instances of evolution rules is chosen non–deterministically in such a way that no other rules can be further applied to the system. In a timed P system the application of timed rules with the completion time $n = 1$ is identical to that of P systems. It consists of removing all the reactants of the chosen rules from the system when they are applied. After 1 time unit the products of the rules are added to the system by taking into account the target indications. If the completion time n is greater than 1 the reactants are immediately removed and the products of the rule are added to the system exactly after n time units.

Let us formally define timed P systems. A multiset over an alphabet $V = \{a_1, \ldots, a_h\}$ is a mapping $M\colon V \to \mathbb{N}$. $M(a_i)$ denotes the number of copies of the element a_i in the multiset M. We often represent a multiset M by a string $w \in V^*$ such that if $M(a_i) = n$ then a_i occurs n times in the string w, in any order. In the following we use the usual notation V^+ to denote the set of non-empty strings of elements of V, and ϵ to denote the empty string. Moreover, we overload the set operators $\in, \cup, \setminus, \cap, \subseteq$ with their multiset counterparts.

Definition 3.1. (Timed P Systems)
A *timed P system* Π is a tuple

$$\langle V, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m \rangle$$

where

- V is an alphabet whose elements are called *objects*.
- μ is a membrane structure consisting of a hierarchy of m membranes labelled by $1, 2, \ldots, m$. The skin membrane is labelled by 1.
- w_i ($i = 1, 2, \ldots, m$) is a string of V^* representing the multiset of objects enclosed in membrane i.
- R_i ($i = 1, 2, \ldots, m$) is a finite set of *timed evolution rules* associated with the membrane i. The rules are of the form $u \xrightarrow{n} v$, $n \in \mathbb{N}^{>0}$, $u \in V^+$, and $v \in \{a_{\mathrm{here}}, a_{\mathrm{out}}, a_{\mathrm{in}_j} \mid a \in V, 2 \leq j \leq m\}^*$.

A timed evolution rule is *cooperative* if $|u| > 1$, where $|u|$ is the number of elements in u, it is *non-cooperative* otherwise. A *catalyst* c is a special object which is involved only in rules of the form $ca \xrightarrow{n} v$, where $a \in V$, $v \in \{a_{\mathrm{here}}, a_{\mathrm{out}}, a_{\mathrm{in}_j} \mid a \in V, 2 \leq j \leq m\}^*$. A $bi-stable$ catalyst is a pair of objects, $c, \overline{c} \in V$, which can be used only in rules $ca \xrightarrow{n} \overline{c}v$, or $\overline{c}a \xrightarrow{n} cv$, where $a \in V$, $v \in \{a_{\mathrm{here}}, a_{\mathrm{out}}, a_{\mathrm{in}_j} \mid a \in V, 2 \leq j \leq m\}^*$. A *timed P structure* P is a timed P system without the sets of timed evolution rules: $P = \langle V, \mu, w_1, \ldots, w_m \rangle$. A *timed P frame* F is a timed P structure without the

multisets of objects contained in the membrane structure: $F = \langle V, \mu \rangle$. Given an alphabet V, we denote by $\mathbb{P}_V$ the class of all timed P systems on V. In the following, when it is clear from the context we use simply $\mathbb{P}$.

Note that, for the sake of simplicity while introducing a new model, we use a "basic" version of P systems, without priority among rules and without membrane dissolving. Moreover, note that the timed extension of P systems we use is essentially the same introduced in [5].

To describe precisely the dynamics of a timed P system we introduce a multiset of pending rules called $\mathcal{U}$.

Definition 3.2. (Pending Rules)
A multiset of *pending rules* $\mathcal{U}$ is a multiset of elements of the form $\xrightarrow{k}_i v$, with $k > 0$. Every element of this form is originated by a timed evolution rule $u \xrightarrow{n} v$, $n > 1$, in a set R_i of a timed P system.
One element $\xrightarrow{k}_i v$ stores the information that an instance of the evolution rule was fired in the past and is waiting k time steps to be completed, with $0 < k < n$.
We denote by $\mathbb{U}$ the set of all multisets of pending rules.

We describe the possible evolutions of a timed P system Π by a transition relation $\xrightarrow{1}$ between configurations in the set $\{(\Pi, \mathcal{U}) \mid \Pi \in \mathbb{P}, \mathcal{U} \in \mathbb{U}\}$.

Definition 3.3. (Timed P Step)
A configuration $(\Pi, \mathcal{U})$ can perform a *timed P step* $\xrightarrow{1}$ to a configuration $(\Pi', \mathcal{U}')$ if and only if:

- Π' is a timed P system resulting from an evolution step of Π using maximal parallelism and such that:
  - the effects of the rules of the form $u \xrightarrow{1} v$ that have been fired in the evolution step are visible in Π', i.e., the reactants have disappeared and the products of the rules are available
  - the effects of the rules $u \xrightarrow{n} v$ with $n > 1$ that have been fired in the evolution step are half visible in Π'. More precisely, the reactants have disappeared, but the products are not yet available (they are stored as pending rules)
  - for every element $\xrightarrow{1}_i v$ in $\mathcal{U}$, the objects of v are inserted into the membrane structure of Π' as if they had been generated from a timed evolution rule with a completion time 1 in the membrane labelled by i;
- $\mathcal{U}'$ is the multiset union of
  - the multiset of all elements $\xrightarrow{k-1}_i v$ derived from all elements $\xrightarrow{k}_i v$, $k > 1$, in $\mathcal{U}$; and
  - the multiset of all elements $\xrightarrow{n-1}_i v$, $n > 1$, representing that an instance of a timed evolution rule $u \xrightarrow{n} v \in R_i$, for some i, has been fired in the evolution step of Π.

We write $(\Pi, \mathcal{U}) \not\xrightarrow{1}$ if it does not exist any $(\Pi', \mathcal{U}')$ such that $(\Pi, \mathcal{U}) \xrightarrow{1} (\Pi', \mathcal{U}')$. This means that the system is steady: no evolution rule can be fired and there are no pending rules left.

In Figure 1 an example of a timed P system is shown. Graphically, boxes represent membranes and their nesting reflects the hierarchy. Inside each box we depict the evolution rules for the corresponding

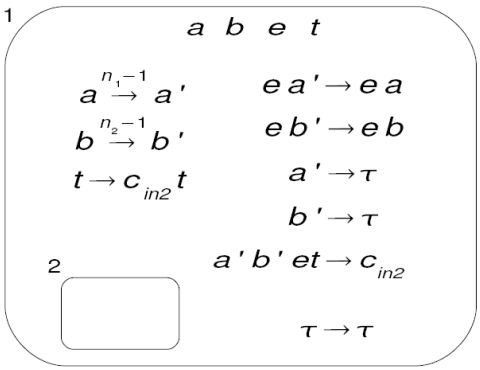

Figure 1. A timed P system calculating the least common multiple of n_1 and n_2

membrane. In evolution rules we omit the completion time if it is 1 and we omit in the products the subscript $_{\text{here}}$ for objects that remain in the same membrane. The free symbols inside the box represent the objects that are present at the beginning of the computation.

The defined system computes the least common multiple of n_1 and n_2, namely $lcm(n_1, n_2)$. The idea is to let an object a and an object b appear in membrane 1 for one time unit every n_1 and n_2 time units, respectively. The first time when a and b appear together is exactly after $lcm(n_1, n_2)$ time units. It is hence sufficient to count the time units until a and b appear together to obtain the results. The system consists of two membranes: membrane 1, in which the computation takes place, and membrane 2, where one object c is sent every time unit. The number of objects c in membrane 2 will represent the result of the computation.

The initial configuration has one a and one b in membrane 1, together with an object e and an object t. Object t is used at each step to send one c into membrane 2. Object e is used to check whether a and b appear together. Actually, the rules consuming a and b remove them for $n_1 - 1$ and $n_2 - 1$ time units, respectively. After these times they appear again as a' and b', respectively. If they do not appear together, the rule $ea' \to ea$ ($eb' \to eb$, resp.) transforms in one time unit a' into a (b' into b, resp) and the computation continues. If a' and b' appear together, neither $ea' \to ea$ nor $eb' \to eb$ can be applied because, by maximal parallelism, this would cause the application of either $b' \to \tau$ or $a' \to \tau$. Note that the production of the trap symbol τ makes the computation infinite, hence not considered as a valid computation. Therefore, when a' and b' appear together, the only rule that can be applied without starting an infinite computation is $a'b'et \to c_{in_2}$, that stops the computation and sends the last c into membrane 2.

3.2. Timed P Automata

A timed P automaton is a timed automaton [1] with a discrete time domain in which a set of timed evolution rules of a timed P system is associated with every location q. When the control is in q the rules represent the dynamics of a timed P system which runs while the timed automaton let the time elapse in the location.

The transitions of the timed automaton are guarded by clock constraints and by a multiset of objects,

called *extracted objects*. When a transition fires, the target location receives the timed P structure of the running timed P system – in which the extracted objects specified in the guard have been removed – together with a description of the pending evolution rules, i.e., those rules with a completion time n ($n > 1$) which were activated k time units before the current time and such that $n > k$. The extracted objects are removed from the skin membrane.

The transitions specify also a clock reset set and a multiset of objects, that we call *inserted objects*, to be added into the skin membrane of the received timed P structure. In the target location the clocks are reset to zero and the inserted objects are put in the skin membrane of the timed P structure. Then, this is completed with the timed evolution rules associated with the location, yielding a new timed P system with the same membrane structure but a different set of rules. This system starts its running in the new location, as time restarts to elapse, taking into account the pending evolution rules of the previous location to be completed.

Let us formally describe timed P automata. Clock variables, or simply clocks, are ranged over by $x, y, z, \ldots$ and we use $\mathcal{X}, \mathcal{X}', \ldots$ to denote sets of clocks. A *clock valuation* over $\mathcal{X}$ is a function that assigns a natural number to every clock. The set of valuations of $\mathcal{X}$, denoted by $\mathcal{V}_{\mathcal{X}}$, is the set of total functions from $\mathcal{X}$ to $\mathbb{N}$. Clock valuations are ranged over by $\nu, \nu', \ldots$. Given $\nu \in \mathcal{V}_{\mathcal{X}}$ and $k \in \mathbb{N}^{>0}$, we use $\nu + k$ to denote the valuation that maps each clock $x \in \mathcal{X}$ into $\nu(x) + k$.

Given a set $\mathcal{X}$ of clocks, a *reset* γ is a subset of $\mathcal{X}$. The set of all resets of clocks in $\mathcal{X}$ is denoted by $\Gamma_{\mathcal{X}}$ and reset sets are ranged over by $\gamma, \gamma', \ldots$ Given a valuation $\nu \in \mathcal{V}_{\mathcal{X}}$ and a reset γ, we let $\nu\backslash\gamma$ be the valuation that assigns the value 0 to every clock in γ and assigns $\nu(x)$ to every clock $x \in \mathcal{X}\backslash\gamma$.

Given a set $\mathcal{X}$ of clocks, the set $\Psi_{\mathcal{X}}$ of *clock constraints* over $\mathcal{X}$ are defined by the following grammar: $\psi \ ::= \ true \mid x \,\#c \mid x - y \,\#\, c \mid \psi \wedge \psi \mid \neg\psi$ where $x, y \in \mathcal{X}$, $c \in \mathbb{N}$, and $\# \in \{<, >, \leq, \geq, =\}$. A satisfaction relation $\models$ is defined such that $\nu \models \psi$ if the values of the clocks in ν satisfy the constraint ψ in the natural interpretation.

Definition 3.4. (Timed P Automaton)
A *timed P automaton* is a tuple $T = \langle Q, \Sigma, q_0, \mathcal{E}, \mathcal{X}, F, \mathcal{R}, Inv\rangle$, where: Q is a finite set of locations, Σ is a finite alphabet of symbols, q_0 is the initial location, $\mathcal{E}$ is a finite set of edges, $\mathcal{X}$ is a finite set of clocks, $F = \langle V, \mu\rangle$ is a timed P frame, $\mathcal{R}$ is a function assigning to every $q \in Q$ a set of timed evolution rules $\{R_1^q, \ldots R_m^q\}$, and Inv is a function assigning to every $q \in Q$ an *invariant*, i.e., a clock constraint ψ such that for each clock valuation $\nu \in \mathcal{V}_{\mathcal{X}}$ and for each $k \in \mathbb{N}^{>0}$, $\nu + k \models \psi \Rightarrow \nu \models \psi$. Constraints having this property are called *past-closed*.
Each edge $e \in \mathcal{E}$ is a tuple in $Q \times \Psi_{\mathcal{X}} \times V^* \times \Gamma_{\mathcal{X}} \times \Sigma \times V^* \times Q$. If $e = (q, \psi, u, \gamma, \sigma, v, q')$ is an edge, q is the source location, q' is the target location, ψ is the clock constraint, u represents the extracted objects, $\sigma \in \Sigma$ is the label, v represents the inserted objects and γ is the clock reset set.

Note that the symbols of the alphabet Σ of the TPA are purely descriptive (they are used to attach a label to a transition in order to specify an observable information for that transition), and have nothing to do with the symbols in the alphabet V of the timed P frame F, which are the symbols for the objects.

Note, also, that we use invariants on the locations of the automaton to control the progress of time, as introduced in [9] for timed automata. Invariants must be true while the automaton stays in the locations. Past-closed invariants either represent deadlines or are equivalent to the constraint always *true*. In the former case a live behaviour of the automaton is guaranteed by the fact that the control cannot stay in the location beyond the deadline. The latter case allows the possibility of a divergence of time in a location, which we use below to define a proper run of a TPA.

$$\text{T1}\qquad \frac{\nu+1 \models Inv(q) \qquad (\Pi,\mathcal{U}) \xrightarrow{1} (\Pi',\mathcal{U}')}{\langle q,\nu,\Pi,\mathcal{U}\rangle \xrightarrow[\text{TP}]{1} \langle q,\nu+1,\Pi',\mathcal{U}'\rangle}$$

$$\text{T2}\qquad \frac{\nu+1 \models Inv(q) \qquad (\Pi,\mathcal{U}) \not\xrightarrow{1}}{\langle q,\nu,\Pi,\mathcal{U}\rangle \xrightarrow[\text{TP}]{1} \langle q,\nu+1,\Pi,\mathcal{U}\rangle}$$

$$\text{T3}\qquad \frac{\begin{array}{c}\Pi = \langle V,\mu,w_1,\ldots,w_m,R_1,\ldots,R_m\rangle \\ (q,\psi,u,\gamma,\sigma,v,q') \in \mathcal{E}, \quad \nu \models \psi, \quad u \subseteq w_1 \quad w_1' = (w_1 \backslash u) \cup v \\ \Pi' = \langle V,\mu,w_1',w_2,\ldots,w_m,\mathcal{R}(q')\rangle\end{array}}{\langle q,\nu,\Pi,\mathcal{U}\rangle \xrightarrow[\text{TP}]{\sigma} \langle q',\nu\backslash\gamma,\Pi',\mathcal{U}\rangle}$$

Figure 2. Rules for the transition relation $\xrightarrow[\text{TP}]{\cdot}$

To define the behaviour of a timed P automaton T we define a labelled transition system $\mathcal{S}(T)$ whose states are tuples $\langle q,\nu,\Pi,\mathcal{U}\rangle$, where $q \in Q$ is a location of T, $\nu \in \mathcal{V}_{\mathcal{X}}$ is a clock valuation, and $(\Pi,\mathcal{U})$ is a configuration of the associated timed P system. The transition relation $\xrightarrow[\text{TP}]{\cdot}$ is defined in Figure 2.

Rules T1 and T2 can only be applied if the passing of one time unit still allows the valuation of clocks ν to satisfy the invariant of the location q. If rule T1 is applied, one time unit passes for the timed automaton and the associated timed P system performs a timed P step as in Definition 3.3. If rule T2 is applied one time unit passes for the timed automaton even if the associated timed P system is unable to perform a timed P step. This allows time to go on when the timed P system has completed its computation. We call 1-transitions the transitions performed by using rules T1 and T2.

Rule T3 describes a transition, labelled by σ, that is possible only if the current clock valuation ν satisfies the clock constraint ψ and u, the multiset of extracted objects is a subset (using the subset operation of multisets) of the current content of the skin membrane w_1. The effect of the transition is that the automaton enters the target location q', where: 1) the extracted objects u are removed from the skin membrane w_1 (using multiset difference); 2) the inserted objects v are added to the skin membrane w_1 (using multiset union); 3) the clocks in the reset set γ are assigned to 0; and 4) the rules of the associated timed P systems are substituted with those of q', $\mathcal{R}(q')$. This kind of transition is instantaneous. We call σ-transitions the transitions performed by using this rule.

Given $w_1, w_2, \ldots, w_m$ multisets of objects, an *initial state* s_0 of the transition system $\mathcal{S}(T)$ is a tuple $\langle q_0,\nu_0,\Pi_0,\oslash\rangle$, where ν_0 is the clock valuation assigning 0 to all clocks, $\oslash$ is the empty multiset of pending rules, and Π_0 is the timed P system composed of the timed P frame F of T, the given multisets of objects, and the set of rules associated with the initial location q_0 of T, namely $\Pi_0 = \langle V,\mu,w_1,w_2,\ldots,w_m,\mathcal{R}(q_0)\rangle$.

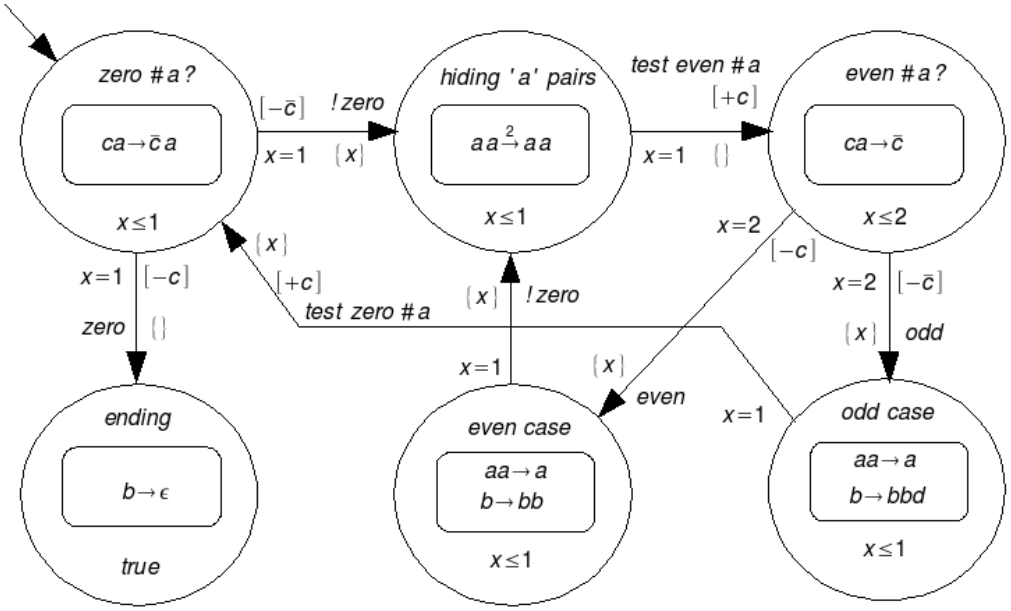

Figure 3. A TPA executing the ancient Egyptian multiplication

Definition 3.5. (Input, Run, Output)

Let $T = \langle Q, \Sigma, q_0, \mathcal{E}, \mathcal{X}, F, \mathcal{R}, Inv \rangle$ be a timed P automaton. Given $w_1, w_2, \ldots, w_m$ multisets of objects, called *input objects*, we can construct, as described above, an initial state s_0 of $\mathcal{S}(T)$. Let ς be an infinite sequence of transitions of $\mathcal{S}(T)$ starting from s_0:

$$\varsigma = s_0 \xrightarrow[\text{TP}]{\ell_0} s_1 \xrightarrow[\text{TP}]{\ell_1} s_2 \xrightarrow[\text{TP}]{\ell_2} s_3 \xrightarrow[\text{TP}]{\ell_3} \cdots$$

We say that ς is a *run* of T if and only if there exists $J \in \mathbb{N}$ such that:

- for all $j > J$, $\ell_j = 1$; i.e., the automaton enters a location, say q_{stop}, from which it never exits and in which the time goes on infinitely (the only possible action is "1", which corresponds to increase the clocks by one time unit); and
- for all $j > J$, $s_j = \langle q_{\text{stop}}, \nu_{j-1} + 1, \Pi_{\text{stop}}, \oslash \rangle$; i.e., the timed P system Π_{stop} in location q_{stop} does not evolve any more as time passes.

We say that the *output* produced by the run is the content of the skin membrane of the timed P system Π_{stop}.

Figure 3 shows an example of a TPA. Graphically, we use circles to represent the locations of the TPA and arrows to represent the edges. In each location we depict the structure of the timed P frame of the TPA together with the evolution rules associated with the membranes in the location. A description label and the invariant of the location are also depicted inside the circle. For instance, in location labelled "zero

	a	b
d ← 0		
while (a ≠ 0)		
if (a is even)		
a ← a / 2	35 +	**42**
b ← b * 2	17 +	**84**
else	8	168
d ← d + b	4	336
a ← ⌊a / 2⌋	2	672
b ← b * 2	1 +	**1344**
endif		-------

Figure 4. Ancient Egyptian multiplication algorithm

$\#a$?" of the TPA of Figure 3 the invariant is $x \leq 1$, where x is a clock of the automaton. The membrane structure of the timed P frame of the given TPA is composed only of the skin membrane. When the automaton is in location "zero $\#a$?" the only evolution rule of the skin membrane is "$ca \rightarrow \overline{c}a$".

Attached to every arrow there are the various components of the corresponding edge. For instance, the arrow from location "zero $\#a$?" to location "hiding 'a' pairs" has the "!zero" label, the clock constraint $x = 1$, the c multiset of extracted objects (we write $[-u]$ for extracted objects u and $[+v]$ for inserted objects v), and the $\{x\}$ clock reset set. Where, as in the previous case, there are no inserted (or extracted) objects we omit to attach $[+\epsilon]$ (or $[-\epsilon]$) to the arrow.

The TPA of Figure 3 calculates the product of two natural numbers, n and m, by using the ancient Egyptian multiplication algorithm[1]. The peculiarity of this algorithm, described in Figure 4, is that it uses, as basic operations, only addition, doubling, and division in two halves.

To start the computation we insert in the skin membrane of the initial location ("zero $\#a$?") n 'a' objects, m 'b' objects, and one bi-stable catalyst c. These correspond to the input objects of Definition 3.5. The result is given in location "ending" where $n \cdot m$ copies of object 'd' will be present at the end of the computation.

The system executes exactly the steps of the algorithm in Figure 4. To test if n is zero (i.e., no 'a' objects are present in the skin membrane), the TPA lets one time unit pass by making the associated timed P system execute one timed P step with the only rule in the initial location: $ca \rightarrow \overline{c}a$. After that, the automaton is forced to exit the location by the invariant $x \leq 1$, using one of the two exiting edges, which are guarded by the mutually exclusive constraints $[-c]$ and $[-\overline{c}]$. If $\overline{c}$ is present after the performed timed P step, then the rule $ca \rightarrow \overline{c}a$ was executed, meaning that there was an 'a' object left, i.e., n was not zero. Else, no 'a' objects were left, otherwise the rule would have been fired because of the maximal parallelism; i.e., n was zero.

To control if n is even in location "hiding 'a' pairs" the rule $aa \xrightarrow{2} aa$ is started. By maximal parallelism, if the number of 'a' objects (i.e., the current value of n) is even, then all the 'a's are involved in an instance of this rule. Otherwise, one 'a' remains unpaired. While the instances of the rules – which

[1]This algorithm has been discovered in a papyrus where the Egyptian scribe Ahmes shows how to multiply 35 and 42. The papyrus dates back to about 2000 B.C. Nowadays the algorithm is used in digital circuits for multiplication.

take 2 time units to be completed – are still executing, that is to say they are pending rules, the automaton is forced to enter location "even $\#a$?" where the same method used in location "zero $\#a$?" to detect if there are 'a' objects left is applied. Note that the rule $ca \rightarrow \overline{c}$, if applied, does not regenerate the unpaired 'a' object, which correctly disappears because, in the following, it would be the remainder of the halving of the 'a's. Depending on the result of the test, the locations "even case" and "odd case" are entered, where the doubling and the dividing into halves is performed (in the meantime, the products of the $aa \xrightarrow{2} aa$ rule instances have appeared). Note that in the odd case 'd' objects are generated in a number equal to the number of the current 'b's, as required by the algorithm. After that, the automaton cycles until all the 'a' objects are consumed.

As stated in Definition 3.5, the output is given in location "ending", where time diverges and the associated timed P system does not perform any other operation after the cancellation of the remaining 'b's. At this time, exactly $n \cdot m$ 'd's are left in the skin membrane.

4. Expressive Power

The universality of TPAs is easily provable by considering that a timed P automaton without clocks and with only one location, whose invariant is *true*, can be considered a timed P system. Then, a timed P system in which all the timed rules are of duration 1 can be considered a "classical" P system. Therefore, our model has a computing power at least equal to that of P systems, which have been shown to be Turing equivalent in [14].

In this section we show that TPAs are still universal also when restrictions on the available resources (number of membranes, size of the alphabet, etc..) or on the form of evolution rules (non-cooperative rules) are introduced. In particular, we show that a timed P automaton can generate any arbitrary recursively enumerable set of natural numbers by using a very restricted set of resources and any arbitrary recursively enumerable set of vectors of natural numbers by using only non-cooperative evolution rules.

Theorem 4.1. For each recursively enumerable set of natural numbers S there exists a timed P automaton – with only one membrane, one symbol for objects, and one bi-stable catalyst – which generates S.

Proof:
We show that by using a TPA with one membrane, one symbol and one bi-stable catalyst we can simulate a 3-register machine, which can generate every recursively enumerable set [10].

Given a 3-register machine $M = (3, B, \mathtt{i}, \mathtt{f})$ we construct a TPA

$$T_M = \langle Q_M, \Sigma_M, q_0, \mathcal{E}_M, \{x\}, \langle \{a, c, \overline{c}\}, \mu_M \rangle, \mathcal{R}_M, Inv_M \rangle$$

that computes the same partial function of M. The membrane structure μ_M consists of only the skin membrane. We use only one clock x that is reset in every edge we construct. At every step of computation the contents of the three registers of the machine – say n_1, n_2, n_3 – is represented by $2^{n_1}3^{n_2}5^{n_3}$ 'a' objects in the skin membrane.

For each instruction of M, we construct one or more locations of T_M, depending on the type of the instruction and on its arguments. Edges are constructed consequentially. The initial location of T_M is the one corresponding to the initial instruction $\mathtt{i}$. Figure 5 shows the three locations and the edges generated

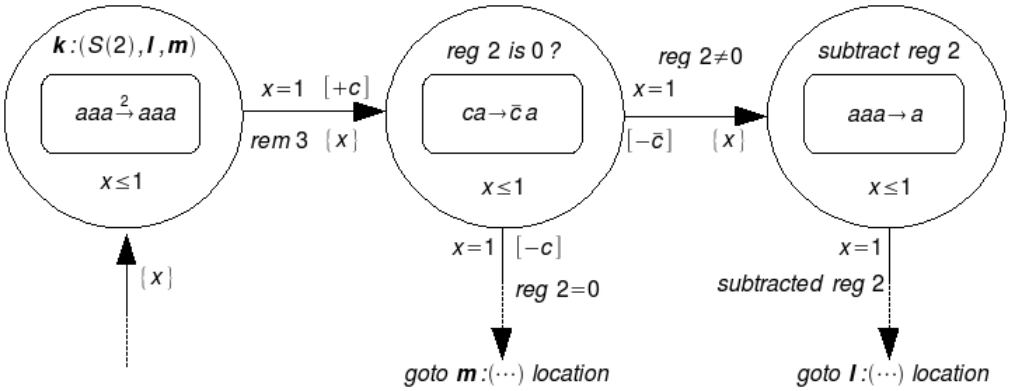

Figure 5. Translation of the subtracting operation

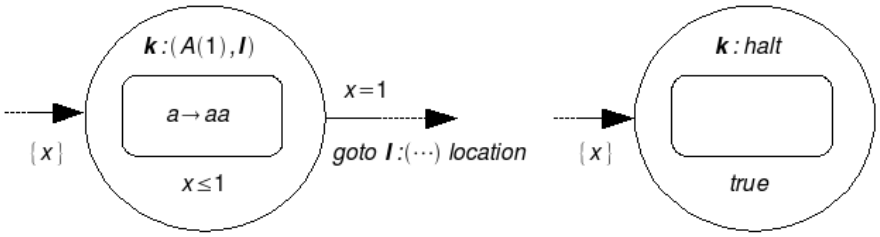

Figure 6. Translation of the adding and the halt operations

by an instruction $\mathtt{k} : (S(2), \mathtt{l}, \mathtt{m})$. The principal location is that labelled with the instruction, while the other two are added for a correct implementation. To test if register 2, in the case shown in Figure 5, is zero we use the same method we described in Section 3.2 for locations "hiding 'a' pairs" and "even $\#a$?" of the TPA in Figure 3. In this case we rewrite all groups of three 'a' by the rule $aaa \xrightarrow{2} aaa$. At the following time unit we check whether a symbol 'a' is present. Since $2^{n_1}3^{n_2}5^{n_3}$, with $n_2 > 0$ is a multiple of 3, while $2^{n_1}3^{0}5^{n_3}$ is not, a symbol 'a' is present at this time if and only if register 2 contains 0. Then, if register 2 is zero the automaton is forced to enter the location associated with the instruction labelled $\mathtt{m}$. Otherwise, register 2 is decreased by 1 by dividing the whole number $2^{n_1}3^{n_2}5^{n_3}$ by 3, yielding $2^{n_1}3^{n_2-1}5^{n_3}$ (location "subtract reg 2") and the automaton is forced to enter the location associated with the instruction labelled $\mathtt{l}$. In case of subtracting registers 1 or 3, the rules to be used are $aa \xrightarrow{2} aa$ and $aaaaa \xrightarrow{2} aaaaa$, respectively.

For an adding instruction and for the halt instruction we construct only one location, as shown in Figure 6. The location on the left represents an adding instruction to register 1. In this case the number of symbols 'a' must be doubled. The halt instruction is represented on the right. It makes the timed P system of T_M stop and in the location the time diverges, as requested in Definition 3.5, to produce the output, which is the current $2^{n_1}3^{n_2}5^{n_3}$ number from which n_1, the calculated value, can be determined.

□

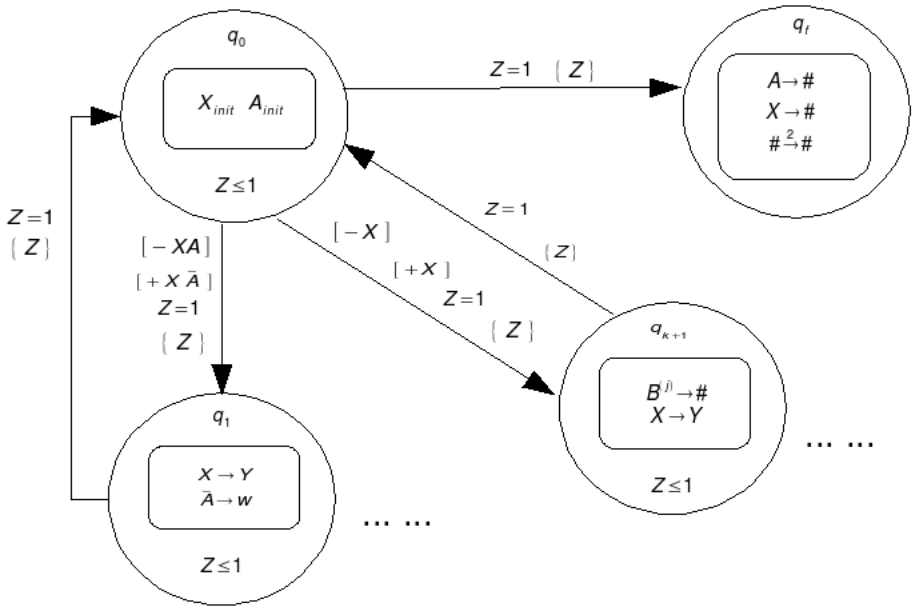

Figure 7. A timed P automaton corresponding to a matrix grammar in MAT^{λ}_{ac}.

Theorem 4.2. For each recursively enumerable set of vectors of natural numbers S there exists a timed P automaton with only non-cooperative rules which generates S.

Proof:
Every recursively enumerable set of vectors of natural numbers can be obtained from a recursively enumerable language by means of the Parikh mapping (defined below). We prove that TPAs with non-cooperative rules are universal by showing that the family of sets of outputs given by these TPAs is equivalent to the family of the images of all the languages in RE obtained through the Parikh mapping (this is the family of recursively enumerable sets of vectors of natural numbers). The proof is carried out by showing that the Parikh mapping of the language generated by any matrix grammar in MAT^{λ}_{ac} can be obtained as an output of a TPA.

Given an alphabet V, an object $a \in V$ and a string x in V^*, we denote with $|x|_a$ the number of occurrences of a in x. If $V = \{a_1, \ldots, a_n\}$ (the ordering is important here), then the Parikh mapping of x is defined by $\Psi_V(x) = (|x|_{a_1}, \ldots, |x|_{a_n})$. The definition is extended in the natural way to languages. The output w of a TPA corresponds to the Parikh mapping of a string x if and only if w contains only and exactly $|x|_{a_1}$ occurrences of a_1, $|x|_{a_2}$ occurrences of a_2, etc.…

Given any matrix grammar G in MAT^{λ}_{ac} in strong binary normal form, a corresponding TPA can be constructed as follows (see Figure 7):

- The TPA consists of $n + 2$ locations, where n is the number of matrices in G of type 2, 3 and 4. The matrix of type 1 ($S \to X_{init}A_{init}$) has no corresponding location.
- Locations are associated with timed P systems consisting only of the skin membrane.

- The initial location (q_0) does not contain any evolution rule, initially contains the two objects X_{init} and A_{init} and has an invariant $Z \leq 1$.

- For every matrix $m_i = (X \to \alpha, A \to x)$ of types 2 and 4 there is a corresponding location q_i. If $\alpha = Y$, the location is associated with evolution rules $X \to Y$ and $\overline{A} \to w$. If $\alpha = \lambda$, then q_i is associated with evolution rules $X \to \varnothing$ and $\overline{A} \to w$. In both cases $\overline{A}$ is a symbol in the alphabet of the TPA that is not in the alphabet of the matrix grammar and w is the multiset such that for every $a \in V$, w contains exactly $|x|_a$ occurrences of a. This location has an invariant $Z \leq 1$ and can be reached from location q_0 via a transition with X and A as extracted and X and $\overline{A}$ as inserted objects. Such a transition is labeled with guard $Z = 1$ and with the clock reset set $\{Z\}$.

- For every matrix $m_i = (X \to Y, B^{(j)} \to \#)$ of type 3 there is a corresponding location q_i associated with evolution rules $X \to Y$ and $B^{(j)} \to \#$. This location has an invariant $Z \leq 1$ and can be reached from location q_0 via a transition with X both as extracted and as inserted objects. Such a transition is labeled with guard $Z = 1$ and with the clock reset set $\{Z\}$.

- There is a transition from every location described so far that corresponds to a matrix of any type to location q_0. Such transitions are labeled with guard $Z = 1$ and with the clock reset set $\{Z\}$.

- There is a final location q_f that can be reached from q_0 by a transition labeled with guard $Z = 1$ and with the clock reset set $\{Z\}$. Location q_f is associated with rules that rewrite every non-terminal symbol of the matrix grammar into $\#$ and rewrite $\#$ into itself every two time units, thus leading to an infinite run in which $\#$ repeatedly appears and disappears.

The TPA we have constructed simulates the corresponding matrix grammar as follows: if in the grammar there is a matrix that can be applied, then in the TPA a transition from location q_0 to the location corresponding to the matrix can be performed. Extracted and inserted objects in the label of the transition ensure that the objects corresponding to the symbols rewritten by the matrix are present. If the matrix is either of type 2 or of type 4, inserted objects transform A into $\overline{A}$ in order to avoid that more than one occurrence of A will be rewritten in the reached location. Subsequently, in such a location rules corresponding to the rules of the matrix are applied. A trap object $\#$ is produced in the locations corresponding to matrices of type 3 when objects corresponding to symbol $B^{(j)}$ are present.

After application of the rules simulating the application of the matrix, the control is sent back to location q_0. The proper use of invariants and clock constraints ensures that at each time unit one of the described transitions is performed and, in turn, this ensures that the TPA always progresses and rules corresponding to matrices are applied only once.

Finally, the control is passed to the final location q_f that produces the trap object $\#$ if some object corresponding to a non terminal symbol is present. Actually, the trap object is produced either when location q_f is reached but the computation is not yet finished, or when in the final string produced by the matrix grammar there is some non terminal symbol (causing the whole derivation to be discarded).

Note that the trap rule in location q_f takes two time units to be completed (hence the trap object continuously appears and disappears) because, by definition of run of a TPA, the configuration of the system must continuously change in order to have an illegal execution.

Moreover, we want to point out that the definition of the TPA corresponding to a matrix grammar could be modified to avoid the use of more than one extracted and inserted object. The idea of the modification is to split the transitions between q_0 and each q_i with $i \in \{1, \ldots, m\}$ into two steps (with

an intermediate location q_i') in which X and A are extracted and inserted and in the first and in the second step, respectively. The transitions from each q_i back to q_0 remain unchanged. A new transition has to be added from each new location q_i' to q_0 to avoid consuming X when A is not present. This approach enables some additional invalid runs (for instance when the automaton cycles between q_i and q_i').

□

5. An Application to Ecological Systems

In this section we present an application of timed P automata. The example we use is a population model taken by the field of "reintroduction biology". In such a research field, population models are useful to guide environment management as they allow to project the persistence of wildlife populations under different conditions.

In [2] a model for guiding the reintroduction of extirpated birds in New Zealand mainland is presented. Population dynamics is simulated by a stochastic, discrete-time female-only model. The model is derived from the observation of the population of Saddleback birds (*Philesturnus rufusater*) on Mokoia Island. The female-only approach assumes that there are sufficient males for all females to be paired. This is a good assumption if the sex ratio is approximately 50:50, as actually happens in the case of Mokoia saddlebacks.

In the model in [2], females are partitioned in two classes, the first-year females and the older females. The two kinds of females are considered to have different fecundity rates, which correspond to a different number in the fledglings produced over a breeding season. There are different survival rates for juveniles and adults. In particular, there is a probability for fledglings to reach the first month of life, and there is a different probability to survive a whole year either for juveniles (older than one month) or adults.

An annual harvest of females is scheduled, with harvesting taking place at the start of breeding season. Such a harvest must maintain the population size in equilibrium without driving it to extinction.

Figure 8 shows the timed P automaton model representing the above scenario: f_1 represents a one-year old female and f_{2+} represents older ones, and analogously, j_1 represents juveniles old less than one month and j_{2+} represents older ones.

In the model, a time unit is considered to correspond to one week. In the "breeding season" location females can either search for a mate – rules $f_1 \xrightarrow{2} f_1$ and $f_{2+} \xrightarrow{2} f_{2+}$ – and such a search takes two weeks, or they can effectively mate, producing offsprings. The mating and breeding process takes four weeks and the number of fledglings is two for one-year old females and it is four for older females. The breeding season lasts for 16 weeks.

The location "juveniles growing" allows all the j_1's to reach the condition j_{2+}, and in the next location ("growing and selection") juveniles and females are subjected to selection. The survived juveniles reach the new breeding season (becoming adult females) together with the survived females. Before the restarting of the breeding season a harvest of 40 females older than one year takes place.

6. Conclusions

An extension of timed P systems has been introduced to study systems whose dynamics changes with time. The proposed model is a timed automaton with a discrete time domain and in which each state is a timed P system. Results on expressive power and on features of the formalism sufficient for full

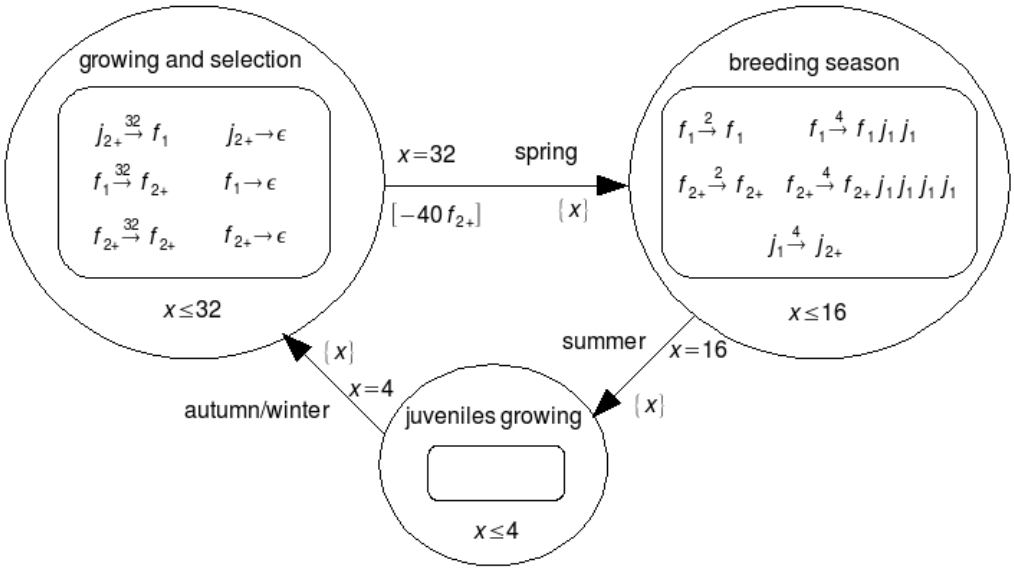

Figure 8. A TPA modelling the population dynamics of Saddleback birds

expressiveness have been proved and, as an application example, the model of an ecological system has been given.

Our model is non-deterministic and there is no notion of probability. This is not a good assumption for a model of a biological system in which events have different probabilities. We plan to add, in the future, probabilities to timed P automata, in order to obtain a stochastic model and to allow more accurate specifications. This would allow the development of a simulator and the study of the dynamics of ecological systems similarly to what done in [4].

References

[1] Alur, R. and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), pp. 183–235.

[2] Armstrong, D. and R. Davidson, *Developing population models for guiding reintroductions of extirpated bird species back to New Zealand mainland*, New Zealand Journal of Ecology **30** (2006), pp. 73–85.

[3] Barbuti, R., A. Maggiolo-Schettini, P. Milazzo and L. Tesei, *Timed P automata*, in: *Membrane Computing and Biologically Inspired Calculi (MeCBIC'08)*, ENTCS **227**, 2008, pp. 21–36.

[4] Cardona, M., M. Colomer, M. Pérez-Jiménez, D. Sanuy and A. Margalida, *Modeling ecosystems using P Systems: The bearded vulture, a case study*, in: *Workshop on Membrane Computing*, LNCS **5391**, 2008, pp. 137–156.

[5] Cavaliere, M. and D. Sburlan, *Time-independent P Systems*, in: *Workshop on Membrane Computing*, LNCS **3365**, pp. 239–258.

[6] Cavaliere, M. and C. Zandron, *Time-driven computations in P Systems*, in: *Proc. of the 4th Brainstorming Week on Membrane Computing*, 2006.

[7] Csahuj-Varjú, E., O. Ibarra and G. Vaszil, *On the computational complexity of P Automata*, in: *Proc. of the International Meeting on DNA Computing*, 2004, pp. 97–106.

[8] Csahuj-Varjú, E. and G. Vaszil, *P automata*, in: *Workshop on Membrane Computing*, LNCS **2597**, 2003, pp. 219–233.

[9] Henzinger, T. A., X. Nicollin, J. Sifakis and S. Yovine, *Symbolic model checking for real-time systems*, Information and Computation **111** (1994), pp. 193–244.

[10] Krishna, S. N. and A. Paun, *Three universality results on P Systems*, in: *Proc. of the 3th Brainstorming Week on Membrane Computing*, 2003.

[11] Minsky, M., "Computation: Finite and Infinite Machines," Prentice-Hall, Englewood Cliffs, NJ, 1967.

[12] Paun, G., *Computing with membranes*, Journal of Computer and System Sciences **61** (2000), pp. 108–143.

[13] Paun, G., *From cells to computers: Computing with membranes (P Systems)*, Biosystems **59** (2001), pp. 139–158.

[14] Paun, G., "Membrane Computing. An Introduction," Springer, Berlin, 2002.

[15] Rozenberg, G. and A. Salomaa, "Handbook of Formal Languages," Springer, Berlin, 1997.