CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

SetExp: A Method of Transformation of Timed Automata into Finite State Automata

Lucien Ouedraogo
Ahmed Khoumsi
Mustapha Nourelfath

February 2008

CIRRELT-2008-05

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

SetExp: A Method of Transformation of Time Automata into Finite State Automata

Lucien Ouedraogo[1,2,*], Ahmed Khoumsi[2], Mustapha Nourelfath[1,3]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Département de génie électrique et de génie informatique, Université Sherbrooke, 2500, boul. Université, Sherbrooke, Canada J1K 2R1

[3] Département de génie mécanique, Université Laval, Pavillon Adrien-Pouliot, Québec, Canada G1K 7P4

Abstract. The behavior of a discrete event system is described by the sequence of events it can execute and can be modeled by finite state automata. Real-time discrete event systems are discrete event systems with timing constraints, and can be modeled by timed automata. Timed automata are convenient for modeling real-time discrete event systems, but are not suitable for studying them, as opposed to finite state automata that are convenient for studying non-real-time discrete event systems. To take into account the advantages of finite state automata, an approach for studying real-time discrete event systems is to transform the timed automata modeling them into the equivalent finite state automata, and then make the study on the finite state automata model. In this paper, we present a method of transforming timed automata into special finite state automata called Set-Exp automata. The method, called SetExp, models the passing of time as real events in two types: Set events that correspond to resets with programming of clocks, and Exp events that correspond to expiring of clocks. The alphabet of a Set-Exp automaton is therefore formed of the one of the timed automaton at which are added the new types of events. SetExp allows thus to obtain a Set-Exp automaton equivalent to the original timed automaton, such that each state of the Set-Exp automaton is defined by a location in which the value of each clock belongs to a determined interval of time, and each transition corresponds to either the occurrence of an event of the timed automaton simultaneously, if necessary, to the occurrence of Set and Exp events, or only the occurrence of Exp events. SetExp limits the state space explosion problem that occurs in similar transformation methods, notably when the magnitudes of the constants used to express the timing constraints are great. Moreover, SetExp is suitable, for example, in supervisory control and conformance testing of real-time discrete event systems.

Keywords. Modeling, real-time, discrete event systems, timed automata, set-exp-automata, transformation.

---

* Corresponding author: Lucien.Ouedraogo@USherbrooke.ca

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 2008

1 Introduction

A discrete event system (DES) is described by the sequences of events it can execute, where the occurrence of each event is instantaneous, spontaneous and asynchronous, for example as with communication protocols (events: send message, receive message...). A real-time DES (RTDES) is a DES with timing constraints, i.e. its correctness depends not only on the order of events but also on the time of occurrence of events, as for example for real-time protocols. We can then, design and study a DES and a RTDES by analyzing the sequence of events it can execute. It is therefore necessary to model the system in a suitable way for the desired study. Consequently, there exist various modeling methods for DES. These modeling methods are classified as untimed or timed, respectively for non-real-time DES and RTDES. A survey of some modeling and analysis methods such as Temporal Logic, Petri Nets and Minimax Algebra can be found for example in [1]. Automata based modeling have been widely studied and used, for non-real-time DES, for example in supervisory control [2, 3] and faults diagnosis [4]. This model of course has been extended to RTDES, where a discrete or continuous model of time can be used. With the discrete time model in [5, 6], the time is measured by a global clock and increase by integer values, and this model has been used in supervisory control, for example in [7, 8], and with the dense time model in [9], the time is measured by a set of asynchronous clocks that can be reset and that take real values, and this model has also been used in supervisory control, for example in [10, 11, 12]. A comparison of these two modeling of time can be found in [13].

Considering the automata model and the dense time approach, a non-real-time DES can be modeled by a finite state automaton (FSA) [14, 15], and a RTDES by a timed automaton (TA) [9], that can be viewed as a FSA to which are added timing constraints using real-valued clocks. The TA model has been extensively studied and is suitable for describing RTDES, but the state space of a TA is infinite because it uses dense time. For the study of RTDES (supervisory control, diagnosis, verification, testing...), it is practical to have a finite representation of the state space, in order to be able to adapt the same methods used for non-real-time DES. It is therefore commonly admitted, for the study of a RTDES modeled by a TA, to transform the TA into an equivalent FSA. The issue with this type of transformation is that it results in an explosion of the state space of the equivalent FSA. The transformation of TA into region automata (RA) [9, 16] is the first and most used transformation of TA into FSA. The basic principle of the RA construction is to merge equivalents states, i.e. states from which we have the same progression possibilities, determined by a suitable equivalence relation. However, the practical usefulness of the RA transformation is limited because it induces a state space explosion. Indeed, the number of states of a RA is exponential in the number of clocks and polynomial in the magnitudes of constants used to express the timing constraints of the TA. To reduce this inconvenient, some methods of minimization of the state space of RA have been proposed, for example zone automata [16] and the methods proposed in [17, 18, 19, 20, 21], as well as others transformation methods as for example the one in [22].

In this paper, we present a new method, called $SetExp$, which transforms a TA into a particular FSA called *Set-Exp automaton* (SEA). The basic principle of $SetExp$ is that it allows to express the timing constraints as order constraints of events. This is realized by the addition to the alphabet of the TA of two new types of events: Set events and Exp events that model respectively the resets with programming of clocks and their expirations when their value reach constants used to express the timing constraints. The alphabet of a SEA is therefore different from the one of its corresponding TA, and each event of the SEA can have three components: an event of the alphabet of the TA, a set of Set events and a set of Exp events, and its occurrence corresponds to the simultaneous occurrence of the events that compose it. Therefore, in a SEA, the transition from a state to another state with the passage of time occurs only when a clock variable reaches a value used in a timing constraint of the TA. Each state of a SEA is defined by a location where the value of each clock respects constraints defined by the values to which it is compared in the timing constraints of the TA. Therefore, compared to the RA construction, in $SetExp$, only the difference between constants for a given clock is considered, and not the difference for each unit of time. As a consequence of this, in practice, the state space of a SEA does not increase polynomially with the magnitudes of constants used in the timing constraints of the TA, and this constitutes an advantage of $SetExp$ in comparison to the RA transformation. The applicability of $SetExp$ has been demonstrated for supervisory control [23, 24, 12, 25, 26] and conformance testing [27, 28], for which it offers a suitable architecture for implementation. Supervisory control, introduced by Ramadge and Wonham [2, 3], aims at restricting the behavior of a system, called a plant, so that it conforms to a specification using a supervisor, and conformance testing aims at checking whether an implementation conforms to a specification, using a test unit [29].

In [23, 24, 12, 25, 26, 27, 28], $SetExp$ is used as a black-box and the focus is on its application, whereas in this paper, we explain formally how $SetExp$ is realized. Therefore, this paper is complementary to [23, 24, 12, 25, 26, 27, 28].

Previous versions of $SetExp$ are presented in [30, 31] and are improved in the present paper. The main additional contributions of [31] in comparison with [30] are:

- the generalization of $SetExp$ to TA with location invariants;
- the taking into account of AT with marked location;
- the minimization of the state space of the generated SEA by the identification and fusion of equivalent states. This contribution also resolves a problem of termination of $SetExp$ in some cases where equivalent states are constructed infinitely in the version of [30].

On the other hand, the main contributions of the present paper in comparison with [31] are:

- we provide detailed explanations such as explanations of algorithms;
- we propose, under some realistic assumptions, a method for reducing of the number of transitions of a SEA;
- we give proofs of all propositions and theorems;
- we give details of the complexity of $SetExp$.

The rest of the paper is organized as follows: in Section 2, we present the formal definition of TA, and in Section 3, we present the principle of construction of the SEA, which is realized in two steps detailed in sections 4 and 5. Section 6 details the procedures of construction of states and transitions of the SEA, while section 7 summarizes some properties and a simplification of $SetExp$. In Section 8, we present the application of $SetExp$ in supervisory control of RTDES, and in Section 9, we conclude. Note that we have opted to put the proofs of propositions and theorems in the appendix.

2 Timed automata and timed language

2.1 Timed automata

To model a RTDES, we use TA [9, 16]. More precisely, we use TA with location invariants [32, 16] defined below, that allow to model rigorously a RTDES. We take also into account the marking of locations. This TA model that uses dense time is suitable for modeling many real-world entities; its manipulation is relatively simple and its expressivity adequate for modeling RTDES. The use of invariants allows to specify progress properties. To define TA with invariants, we need the following definitions of *Clock* and *Clock Constraints*. In the rest of the paper, for a set X, 2^X denotes the set of subsets of X.

Definition 2.1 *:* **Clock**
A Clock *is a positive real-valued variable that can be reset to 0 at any time, and such that, between two resets, its derivative with respect to (w.r.t.) time is equal to 1. Let $\mathcal{C} = \{c_1, \cdots, c_{N_c}\}$ be a finite set of N_c clocks.*

Definition 2.2 *:* **Clock Constraint**
A Clock Constraint *is a formula in the form "$c_i \sim k$", where c_i is a clock, $\sim \in \{<, >, \leq, \geq, =\}$ and k is a positive integer value. Let $\Phi_{\mathcal{C}}$ be the (infinite) set of clock constraints using clocks of $\mathcal{C}$.*

Note that a Clock Constraint takes a boolean value ($true$ or $false$), depending on the fact that its value satisfies or not the constraint.

Definition 2.3 *:* **Timed Automata *(TA)***
A TA *$\mathcal{A}$ is defined by $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}, \mathcal{T}, l_0, \mathcal{L}_m)$ where: $\mathcal{L}$ is a finite set of locations; $\mathcal{L}_m \subseteq \mathcal{L}$ is the set of marked locations; $l_0 \in \mathcal{L}$ is the initial location; Σ is a finite set of events (also called* alphabet*); $\mathcal{C}$ is a finite set of clocks; $\mathcal{I}: \mathcal{L} \mapsto 2^{\Phi_{\mathcal{C}}}$ associates to each location $\ell \in \mathcal{L}$ a set of Clocks Constraints called* invariant *of ℓ, and noted $\mathcal{I}_\ell$ ($\mathcal{I}_\ell \subset \Phi_{\mathcal{C}}$); and $\mathcal{T} \subset \mathcal{L} \times \Sigma \times \mathcal{L} \times 2^{\Phi_{\mathcal{C}}} \times 2^{\mathcal{C}}$ is the set of transitions.*

According to Definition 2.3, a transition T of $\mathcal{A}$ is defined by $T = \langle q; \sigma; r; G; Z \rangle$, where: q and r are the origin and destination locations of T, $\sigma \in \Sigma$ is the event of Tr, $G \subset \Phi_{\mathcal{C}}$ is a set of Clocks Constraints called *guard* of T and $Z \subseteq \mathcal{C}$ is a set of clocks called *reset* of T. For brevity, we say that a guard or an invariant is satisfied when all its temporal

constraints are evaluated to $true$, otherwise it is unsatisfied. When a guard or an invariant is empty, this means that it is evaluated to $true$ at any time.

The **semantics** of a TA $\mathcal{A}=(\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}, \mathcal{T}, l_0, \mathcal{L}_m)$ is as follows: at time $\tau_0 = 0$, $\mathcal{A}$ is at l_0 with all clocks equal to 0. When q is the current location, a transition $T = \langle q; \sigma; r; G; Z\rangle$ is *enabled* when G is satisfied, i.e. all its clocks' constraints are evaluated to $true$ (if G is empty, it is always satisfied); otherwise, Tr is *disabled*. From q, the event σ can be executed only when Tr is enabled, and after the execution of σ, location r is reached and all the clocks in Z are reset. During an execution of a TA, the invariant of the current location is always satisfied, i.e. all its clocks' constraints are evaluated to $true$. In particular, the invariant of a marked location is satisfied at any time when the location is occupied, because we consider marked location like location where the TA can stop its execution for an unspecified duration.

A constraint "$c_i < k$" or "$c_i \leq k$" in the invariant of a location ℓ specifies an upper time bound of clock c_i after which ℓ must be left. In the other hand, a constraint "$c_i > k$" or "$c_i \geq k$" in the invariant of ℓ specifies a lower time bound of clock c_i before which ℓ cannot be reached. A similar result can be obtained by inserting these later constraints in the guard of any transition reaching ℓ. We also assume that a constraint of the form "$c_i = k$" cannot be in the invariant of ℓ, because in this case ℓ is irrelevant since it must be left at the time where it is reached. Therefore, we make the following non restrictive hypothesis:

Hypothesis 2.1 *Each constraint in an invariant is in the form "$c_i \sim k$" with $\sim \in \{<, \leq\}$.*

As an example of TA, consider the TA $\mathcal{P}$ of Figure 1 representing a machine in a manufacturing system. Each node represents a location with its invariant, if any. Nodes representing marked locations are represented by two circles and the initial location is indicated by an arrow. Each edge linking q to r and labeled $(\sigma; G; Z)$ represents a transition $\langle q; \sigma; r; G; Z\rangle$. An empty G or Z is represented by "-". In a location, each transition can be executed only when it is enabled, otherwise it will be never executed. For this TA, we have : $\Sigma = \{\alpha, \beta, \lambda, \mu, \gamma\}$; $\mathcal{L} = \{I, W, B, R\}$; $\mathcal{C} = \{c1, c2\}$; $\mathcal{L}_m = \{I\}$ and $\ell_0 = I$.

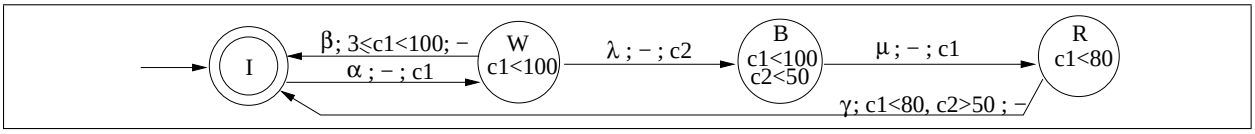


Figure 1: TA modeling a machine in a manufacturing system

Initially, the machine (or the TA $\mathcal{P}$) is in location I (Idle). It can reach location W (working) by executing the event α that means "the machine starts working" and resetting the clock $c1$. From the location W, the machine can reach locations I or B (machine is broken down) by executing respectively the events β (machine finishes working) or λ (machine breaks down). In any case, W must be left before $c1 = 100$, in order to respect its invariant. From the location B, the machine can reach the location R (machine under repair) by executing the event μ (begin the repair of the machine). B must be left before $c1 = 100$ and $c2 = 50$, as indicated by its invariant. From the location R, the machine can reach the initial location I by executing the event γ (end of the repair of the machine). The location R must be left before $c1 = 80$, in order to respect its invariant.

Let $\delta_{u,v}$ be the delay between the events u and v. The guards of the transition of $\mathcal{P}$ imply : $3 \leq \delta_{\alpha,\beta} < 100$, $\delta_{\lambda,\gamma} > 50$, $\delta_{\mu,\gamma} < 80$. Each constraint $\delta_{u,v} \sim k$ does not guarantee the occurrence of v, but indicates that if v occurs, this occurrence respects $\delta_{u,v} \sim k$, otherwise v does not occur. To guarantee the occurrence of v, we need to use invariants. For example the invariant of B guarantees that B is left before $c1 = 100$ and $c2 = 50$, i.e. μ occurs certainly with $\delta_{\alpha,\mu} < 100$ and $\delta_{\lambda,\mu} < 50$.

2.2 Timed language of TA

Definition 2.4 *:* **Timed Trace**
A timed trace is a sequence "$(e_1, \tau_1)\cdots(e_i, \tau_i)\cdots$", where $e_1, \cdots, e_i \in \Sigma$ are events and each τ_i ($0 < \tau_1 < \cdots < \tau_i < \cdots$) is the time of occurrence of e_i.

Notation 2.1 *If λ is a trace, $\overline{\lambda_i}$ is the prefix of λ of length i.*

Definition 2.5 *:* **Acceptance of Timed Trace**
Let $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}, \mathcal{T}, l_0, \mathcal{L}_m)$ *be a TA.* $\mathcal{A}$ accepts *the empty timed trace* λ_0 iff *the invariant of* l_0 *is satisfied whenever all the clocks evaluate to the same value. Intuitively,* $\mathcal{A}$ *can remain forever in* l_0*. A finite timed trace* $\lambda = (e_1, \tau_1) \cdots (e_n, \tau_n)$ *is* accepted *by* $\mathcal{A}$ iff *there exists a sequence of consecutive transitions* $Tr_1 \cdots Tr_n$ *of* $\mathcal{A}$ *that starts in* l_0 *and such that: (i)* $\forall i \in \{1, \cdots, n\}$*, the event of* Tr_i *is* e_i *and after the execution of* $\overline{\lambda_{i-1}}$*,* Tr_i *is enabled at time* τ_i *and the invariant of the* current location *(destination location of* Tr_{i-1}*) is satisfied at every time* $\tau \in [\tau_{i-1}, \tau_i]$*; and (ii) after the execution of* λ*, the invariant of the current location (i.e., invariant of the destination location of* Tr_n*) is satisfied at every time* $\tau \geq \tau_n$*. Intuitively,* $\mathcal{A}$ *can execute* λ *and stop forever its execution.* Acceptance *of an* infinite *timed trace* λ *is defined from acceptance of finite timed trace by removing Item (ii) and replacing* n *by* ∞*.*

Definition 2.6 *:* **Marked Timed Trace**
A finite timed trace accepted by a TA $\mathcal{A}$ *is* marked *if its execution leads to a marked location.*

Definition 2.7 *:* **Timed Language and Timed Marked Language of TA**
The timed language of a TA $\mathcal{A}$*, noted* $L^t(\mathcal{A})$*, is the set of finite and infinite timed traces accepted by* $\mathcal{A}$*. The* timed marked language of $\mathcal{A}$*, noted* $L^t_m(\mathcal{A})$ *is the set of* finite *marked timed traces accepted by* $\mathcal{A}$*.*

For a correct execution of TA (containing invariants), we need the next three requirements.

Requirement 2.1 *At the time when a location is reached, its invariant must be satisfied.*

Requirement 2.2 *During the execution of a TA, when the invariant of the current location is about to be unsatisfied with elapse of time, a transition must be enabled and the invariant of its destination location must be satisfied.*

Requirement 2.3 *Each marked location has an empty invariant, i.e. its invariant is satisfied at any time.*

Requirement 2.1 ensures that the invariant of a location is satisfied when the location is reached. Requirement 2.2 ensures that when a location must be left due to imminent dissatisfaction of its invariant, at least one enabled transition exists. Therefore, this requirement allows to avoid situations where a location must be left imperatively (because its invariant is about to be unsatisfied) and in the same time no transition is enabled or the destination locations of all enabled transitions have their invariant unsatisfied. This requirement 2.2 is respected if for each location ℓ having a non empty invariant $\mathcal{I}_\ell$ that becomes unsatisfied at the moment τ, (i) there exists at least one transition Tr having ℓ as origin location; (ii) the guard of Tr is satisfied just before moment τ and (iii) the invariant of the destination location of Tr is satisfied just before moment τ. For example, suppose the TA uses one clock h, and at the location l_1 having the invariant $h < x$, we have one outgoing transition Tr whose guard is $y < h < z$, and the invariant of its destination location l_2 is $h < w$. l_1 respects requirement 2.2 if $y < x$, $x \leq z$ and $w \geq x$ because: (i) the invariant of l_1 becomes unsatisfied when $h = x$; (ii) the guard of Tr is satisfied at the moment just before $h = x$ *iff* $y < x \leq z$; and (iii) the invariant of l_2 is satisfied at the moment just before $h = x$ *iff* $w \geq x$. We shall show later how to formally verify if requirements 2.1 and 2.2 are respected in a TA. Requirement 2.3 is a logic one, because we consider marked locations as locations where the TA can stop its execution for an unspecified duration, and this also allows to have the timed marked language of the TA included in its timed language.

2.3 Transformation of timed automata into finite state automata

A TA is suitable to model a RTDES, but for its study, this model is not suitable. The reason is that the methods for studying RTDES (and DES) are based on the analysis of their state space which must therefore be finite, i.e. representable with a FSA. However, a TA has an infinite state space. This comes from the fact that a state of a TA is defined by a pair (ℓ, ϑ) where ℓ is a location and ϑ is a function that associates each clock to its current value, called *clock assignment*. As the clocks have real values, the number of possible values for each clock is infinite, and therefore the number of states is also infinite. For this reason, to study a RTDES modeled by a TA, as for example verification, supervisory control, conformance testing or faults diagnosis, it is convenient to transform the TA into a FSA. Such transformation allows thus to adapt methods designed for FSA to solve problems expressed with TA.

One of the most known transformation method is the one which transforms TA into region automata (RA) [9, 16]. a region is defined by an equivalence relation over clocks assignments. The RA is constructed by generating the

reachable region graph, and each state of the RA is defined by a location of the TA and a region. In a region, the value of each clock is either equal to an integer or belongs to an interval bounded by two consecutive integer, so that the number of state of a RA is exponential in the number of clocks and polynomial in the magnitudes of constants used to express the timing constraints. The RA transformation induces therefore a state space explosion. To illustrate this disadvantage of the RA transformation, let us consider the TA of Figure 2. The number of states of its corresponding RA depends on the value of k. For example, if $k = 10$, the RA will have 20 states and if $k = 50$, the RA will have 100 states. The practical usefulness of this transformation is reduced, especially when the magnitudes of the constants in the timing constant are big. To overcome this issue, several minimizations methods have been proposed, such as those in [16, 17, 18, 19, 20, 21]. These methods allow to mitigate the state space explosion problem and most of them are designed essentially for model checking and reachability analysis.

In the following sections, we present a method of transformation of TA into particular FSA called *Set-Exp Automata* (SEA). The transformation "TA↦SEA", which is called $SetExp$, generates less states than the transformation "TA↦ RA", in particular when the magnitudes of the constants used to express the timing constraints are big. For example, the SEA corresponding to the TA of Figure 2 has only 6 states, whatever the value of k is. Contrary to the minimizations methods of RA, $SetExp$ is well suited for supervisory control and conformance testing of RT-DES [23, 24, 12, 25, 26, 27, 28], because it provides a concrete implementation architecture. We will present an overview of the application of $SetExp$ in supervisory control in Section 8. We present in Section 3 the principle of $SetExp$ and the SEA model, and in Sections 4, 5 and 6, we present the formal computation of SEA from TA.

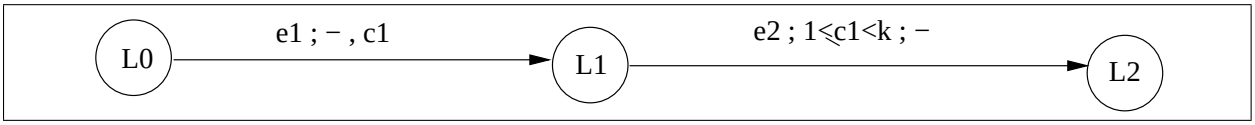


Figure 2: Simple AT to compare $SetExp$ and Region automata transformation

3 Principle of $SetExp$ and Set-Exp automata

$SetExp$ transforms a TA into a FSA called *Set-Exp Automaton* (SEA). $SetExp$ adds to the alphabet of the TA two new types of events called Set and Exp, that model respectively the setting and expiring of the clocks, and thus allow to express temporal constraints as order constraints of events. Therefore, a TA and the corresponding SEA represent two different ways of specifying the same order and timing constraints of events.

3.1 Events Set and Exp

Let us first present the events Set and Exp. For a clock c_i and a positive integer k, the events $Set(c_i; k)$ and $Exp(c_i, k)$ have the following meaning:

- An event $Set(c_i; k)$ means that c_i is reset to 0 and it will expire when its value is equal to k. If c_i must expire several times at the values $k_1, k_2 \cdots, k_p$, with $k_1 < k_2 < \cdots < k_p$, then the corresponding Set event is $Set(c_i; k_1, \ldots, k_p)$.
- An event $Exp(c_i, k)$ means that c_i expires and its current value is k.

In a TA, a clock c_i is reset in order to compare later its value to at least one constant, say k. The event $Set(c_i; k)$ is very convenient for that purpose, because it resets c_i and programs $Exp(c_i, k)$ which is a notification when $c_i = k$. Therefore, $Set(c_i; k)$ is followed, after a delay k, by $Exp(c_i, k)$, and $Set(c_i; k_1, \ldots, k_p)$ is followed, after the delays $k_1, k_2, \ldots, k_p$, by $Exp(c_i, k_1), Exp(c_i, k_2), \ldots, Exp(c_i, k_p)$. The expression "$c_i$ expires" is equivalent to "c_i reaches a value programmed at its last reset". $Set(c_i; k)$ can then be seen as the programming of an alarm that must ring after k units of time, and $Exp(c_i, k)$ corresponds to the ringing of this alarm after the delay k. When a $Set(c_i; *)$ occurs, then all expirations of c_i which were foreseen before this $Set(c_i; *)$ are canceled.

3.2 Intuition of $SetExp$

To illustrate the intuition of $SetExp$, let us consider the following two specifications:

Specification 1: a task ω must be realized in less than k units of time, where k is a positive integer.

Specification 2: at the beginning of the task ω, an alarm is programmed so that it rings after k units of time, and ω must be terminated before the ringing of the alarm.

The above two specifications describe the same timing constraint for the task ω. Specification 1 can be modeled by a TA, and Specification 2 be modeled by a SEA. $SetExp$ can be used to obtain Specification 2 from Specification 1. The programming of the alarm corresponds to a Set event, and the ringing of the alarm corresponds to an Exp event.

3.3 Events and transitions of SEA

For a TA $\mathcal{A}$ over the alphabet Σ, let $\mathcal{B} = SetExp(\mathcal{A})$ be the SEA (we shall present later the formal definition of a SEA) obtained by applying $SetExp$ to $\mathcal{A}$. In the following, σ denotes an event of Σ, $\mathcal{S}$ (resp. $\mathcal{E}$) denotes a set of Set (resp. Exp) events, and occurrence of $\mathcal{S}$ (resp. $\mathcal{E}$) means the simultaneous occurrences of all the events in $\mathcal{S}$ (resp. $\mathcal{E}$). We can categorize the events of $\mathcal{B}$ into the following three types:

Type 1: an event $\gamma = \mathcal{E}$ is of type 1 if it is formed of a non empty set $\mathcal{E}$ of Exp events.

Type 2: an event $\gamma = (\sigma, \mathcal{S})$ is of type 2 if it is formed of an event $\sigma \in \Sigma$ and a set $\mathcal{S}$ ($\mathcal{S}$ can be empty) of Set events.

Type 3: an event $\gamma = (\mathcal{E}, \sigma, \mathcal{S})$ is of type 3 if it is formed of an event $\sigma \in \Sigma$, a non empty set $\mathcal{E}$ of Exp events and a set $\mathcal{S}$ ($\mathcal{S}$ can be empty) of Set events. Note that type 3 is a combination of type 1 and type 2 and thus, an event of type 3 is composed by an event of type 1 and an event of type 2.

Each event of the SEA $\mathcal{B} = SetExp(\mathcal{A})$ is therefore formed by at most an event $\sigma \in \Sigma$, by 0 to many Set events, and 0 to many Exp events. With this characterization, we can process as a unique event the simultaneous occurrences of several events. In the sequel, Σ and Γ will refer respectively to the alphabet of TA and SEA.

In accordance with the three types of events, we have three types of transitions. Each transition has the same type as its labeling event.

Notations 3.1 *:*

- *If Γ is the alphabet of a SEA, then Γ_i denotes the set of events of type i of Γ, for $i = 1, 2, 3$. Therefore we have $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$.*
- *For an event $\gamma \in \Gamma$, $\gamma_{\downarrow\Sigma}$ is the component of γ that belongs to Σ (one element, or the empty set in the case where $\gamma \in \Gamma_1$); $\gamma_{\downarrow Exp}$ the set of Exp events of γ (empty set if $\gamma \in \Gamma_2$); and $\gamma_{\downarrow Set}$ is the set of Set events of γ (empty set if $\gamma \in \Gamma_1$).*
- *An event $\gamma = (\mathcal{E}, \sigma, \mathcal{S}) \in \Gamma$ will be noted (also on figures) $\sigma\mathcal{S}\mathcal{E}$, and in a sequence of events of Γ, the events will be separated by a ".".*

Now, let us define formally a SEA. We shall clarify some of its components later.

Definition 3.1 *:* **Set-Exp-Automaton (SEA)**
A SEA $\mathcal{B}$ is defined as follows: $\mathcal{B} = (Q, \Gamma, \delta, \Pi, q^0, Q_m)$ where:

- *Q is a set of states;*
- *$q_0 \in Q$ is the initial state;*
- *$Q_m \subseteq Q$ is a set of marked states;*
- *Γ is a set of events (i.e. an alphabet);*
- *$\delta : Q \times \Gamma \times Q$ is the set of transitions;*
- *$\Pi : Q \rightarrow 2^{\Gamma_1}$ is a function that associates to each state q a set of events of type 1 that are preempted (see Definition 3.2).*

If we apply $SetExp$ to the TA $\mathcal{A}$ of Figure 1, we obtain the SEA $\mathcal{B} = SetExp(\mathcal{A})$ of Figure 3. The construction of this SEA will be presented later. This SEA has 12 states, named S1,S2,...,S12. S1 is the initial state and the only marked one. The SEA has 25 transitions, each labeled by an event of its alphabet Γ. For the different types of transitions, we have, for example, transitions between the states S1 and S2 and between S4 and S5 which are of type 2 (respectively labeled by $\alpha Set(c1; 3, 100)$ and β), transitions between states S2 and S4 and between S7 and S11 which are of type 1 (respectively labeled by $Exp(c1, 3)$ and $Exp(c2, 50)$) and transitions between states S2 and S5 and between S2 and S6 which are of type 3 (respectively labeled by $\beta Exp(c1, 3)$ and $\lambda Set(c2; 50)Exp(1, 3)$).

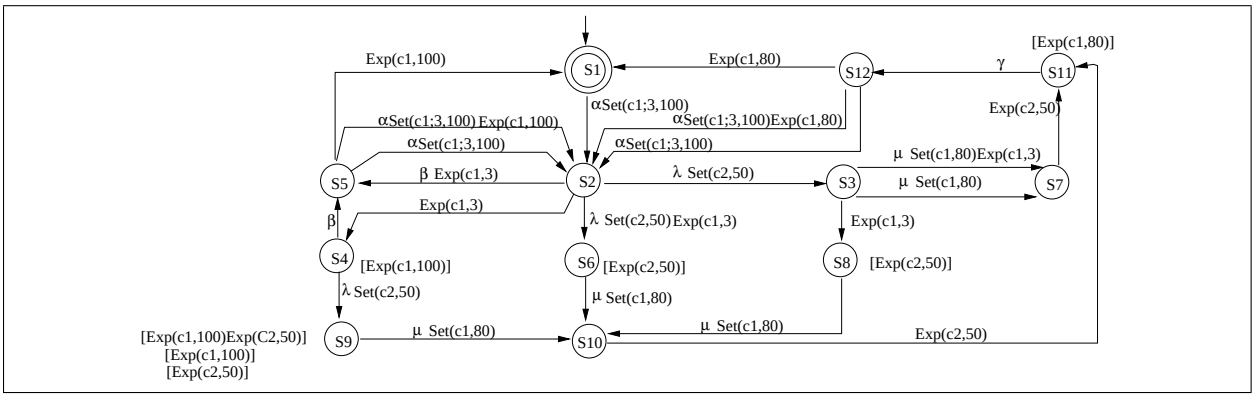


Figure 3: SEA of the TA of Figure 1

We shall show later that a state of a SEA $\mathcal{B} = SetExp(\mathcal{A})$ is defined by a location of $\mathcal{A}$, some inequations on clocks and some inequations on differences of clocks.

Recall that in a TA $\mathcal{A}$, each location must be left before its invariant become unsatisfied. On the SEA $\mathcal{B} = SetExp(\mathcal{A})$, this requirement is taken into account by *preempting transitions of type 1 and 3*. The preemption of a transition in a SEA is defined as follows:

Definition 3.2 *:* **preempted transition**
A transition Tr of type 1 or 3 is said to be preempted if Tr is never executed because another transition having the same origin state as Tr is always executed before Tr. If Tr is labeled by γ and preempted, then the event $\gamma\rfloor_{Exp}$ is also said preempted. The set of preempted events at each state q of the SEA is given by $\Pi(q)$.

Intuitively, a transition Tr is preempted if the system, in its execution, will not have the possibility, when the origin location of Tr is reached, to execute Tr because it is forced to execute another transition. In other words, Tr is preempted if the origin location of Tr is left (by executing another transition) before Tr becomes enabled. In Figure 3, the preempted events at a given state q are indicated between square brackets near q. For this SEA, the preempted events, given by the function Π, are the following:

- $\Pi(S4) = \{Exp(c1, 100)\}$;
- $\Pi(S6) = \{Exp(c2, 50)\}$;
- $\Pi(S9) = \{Exp(c1, 100), Exp(c2, 50), Exp(c1, 100)Exp(c2, 50)\}$;
- $\Pi(S11) = \{Exp(c1, 80)\}$.

Definition 3.3 *:* **active clock**
At a given state q of a SEA, a clock c_i is active when at least an expiration of c_i is expected. More precisely, while a $Set(c_i; k_1, \cdots, k_p)$ is the last $Set(c_i; *)$ that has occurred, c_i is active iff *$Exp(c_i, k_p)$ has not occurred. Otherwise, c_i is inactive.*

Definition 3.4 *:* **quiescent state**
A state q of a SEA is said a quiescent state *if all clocks are inactive in q, i.e. $\Pi(q) = \emptyset$ and q has no outgoing transition of type 1 or 3.*

For the SEA of Figure 3, only S1 is a quiescent state. Intuitively, a quiescent state is a state where it is possible to stay indefinitely without any constraint to leave it with the evolution of time, by the occurrence of Exp events. Such states will be useful to define the language of a SEA.

Definition 3.5 *:* **marked state**
A state q of a SEA $\mathcal{B} = SetExp(\mathcal{A})$ is marked if and only if q is a quiescent state and its location is marked in $\mathcal{A}$.

For the SEA of Figure 3, only S1 is a marked state, because it is a quiescent state and its location in the TA of Figure 1, i.e.. I, is marked. In practice, a marked state is usually a state that represents the end of a task. This can justify the

fact that this state must be a quiescent state, i.e. a state where the process can stop indefinitely its execution because no clock expiration is expected.

Definition 3.6 *:* **acceptance of finite and infinite sequence**
Let $\mathcal{B} = (Q, \Gamma, \delta, \Pi, q^0, Q_m)$ be a SEA and $\gamma = \gamma_1\gamma_2\cdots\gamma_n$ be a finite sequence of events belonging to Γ. γ is said accepted by $\mathcal{B}$ if there exists a set of states $q_0, q_1 \cdots, q_n \in Q$, starting in q^0, such that: $(q_{i-1}, \gamma_i, q_i) \in \delta$ for $i \in \{1, \cdots, n\}$, and q_n is a quiescent state. An infinite sequence $\gamma = \gamma_1\gamma_2\cdots\gamma_i\cdots$ of events belonging to Γ is accepted by $\mathcal{B}$ if it labels a sequence of consecutive transitions in $\mathcal{B}$ that starts in q^0.

For the SEA of Figure 3, only the finite sequences ending in S1 are accepted. For example, the sequence $\alpha Set(c1; 3, 100).\beta Exp(c1, 3).Exp(c1, 100)$ is accepted.

Definition 3.7 *:* **language and marked language of SEA**
The language *generated by a SEA $\mathcal{B}$, noted $L(\mathcal{B})$, is the set of finite and infinite sequences accepted by $\mathcal{B}$. The* marked language *of $\mathcal{B}$, noted $L_m(\mathcal{B})$, is the set of finite sequences accepted by $\mathcal{B}$ and that lead to a marked state. Note that $L_m(\mathcal{B}) \subseteq L(\mathcal{B})$.*

The construction of the SEA $\mathcal{B} = SetExp(\mathcal{A})$ is realized in two steps. We shall present these two steps in sections 4 and 5, respectively.

4 First step of $SetExp$

The first step of $SetExp$ transforms a TA $\mathcal{A}$ into an automaton $\mathcal{A}^I$ that we call *Intermediate Automaton (IA)*. This step does not modify the structure of $\mathcal{A}$, and consists of two substeps that are:

Substep 1: it replaces resets of clocks by Set events. For example, a clock c_i which is reset is replaced by $Set(c_i; k)$ if c_i is compared to k before its next expected reset. If for the same transition, we obtain several $Set(c_i; k_1)$, $Set(c_i; k_2), \cdots, Set(c_i; k_n)$, we replace them by a single $Set(c_i; k_1, k_2, \cdots, k_n)$.

Substep 2: it rewrites the guards (of transitions) and the invariants (of locations) in the form of order constraints relatively to Exp events. More precisely, each constraint "$c_i \sim k$" is rewritten in the form "$\sim Exp(c_i, k)$", with $\sim \in \{<, \leq, >, \geq, =\}$ if the constraint is in a guard, and $\sim \in \{<, \leq\}$ if the constraint is in an invariant (see hypothesis 2.1). We shall call *Exp-Condition* a constraint rewritten under this form.

For the formal definitions of these two substeps, we need the following definition and notations.

Definition 4.1 *:* **path, reachable transition, reachable location**
In a TA $\mathcal{A}$, a path is a transition or sequence of consecutive transitions of $\mathcal{A}$, and a transition T' is said reachable from a transition T iff *$\mathcal{A}$ contains a path $T \cdots T'$. Similarly, a location ℓ is said reachable from a transition T* iff *$\mathcal{A}$ contains a path $T \cdots T'$ such that ℓ is the destination location of T'.*

Notations 4.1 *: Let $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}, \mathcal{T}, l_0, \mathcal{L}_m)$ be a TA, and $T \in \mathcal{T}$ a transition of $\mathcal{A}$.*

- *$\mathcal{R}^t_T$ is the set of transitions reachable from T.*
- *$\mathcal{R}^l_T$ is the set of locations reachable from T.*
- *$\mathcal{P}_{T \to T'}$ is the set of paths linking the destination location of T to the origin location of $T' \in \mathcal{T}$.*
- *$\mathcal{P}_{T \to \ell}$ is the set of paths linking the destination location of T to the location $\ell \in \mathcal{L}$.*
- *Z_P is the set of clocks that are reset in a path P.*
- *Z_T is the set of clocks that are reset by T.*
- *G_T is the guard of T.*
- *$\mathcal{I}_\ell$ is the invariant of location ℓ, and $\mathcal{I}_\ell(c_i)$ is the part of $\mathcal{I}_\ell$ using the clock c_i.*
- *$\mathcal{Z}_T$ is the set of Set events of T, and $\mathcal{Z}_T(c_i)$ is the part of $\mathcal{Z}_T$ using the clock c_i.*
- *$\mathcal{G}_T$ is the set of* Exp-Conditions *of T, and $\mathcal{G}_T(c_i)$ is the part of $\mathcal{G}_T$ using the clock c_i.*

The formal definition of the first substep is as follows, for a TA $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}, \mathcal{T}, l_0, \mathcal{L}_m)$:

1. $\forall T \in \mathcal{T}, \forall c_i \in \mathcal{C} : \mathcal{Z}_T(c_i) := \{Set(c_i, X \cup Y) : c_i \in Z_T,$
2. $\quad X = \{k : \exists T' \in \mathcal{R}^t_T, \exists (c_i \sim k) \in G_{T'}, \exists P \in \mathcal{P}_{T \to T'}, c_i \notin Z_P\},$
3. $\quad Y = \{k : \exists \ell \in \mathcal{R}^\ell_T, \exists (c_i \sim k) \in \mathcal{I}_\ell, \exists P \in \mathcal{P}_{T \to \ell}, c_i \notin Z_P\}\}$
4. $\forall T \in \mathcal{T} : \mathcal{Z}_T := \bigcup_{c_i \in \mathcal{C}} \mathcal{Z}_T(c_i)$

Explanations of the formal definition of substep 1:

Line 1-3: for each transition T of $\mathcal{A}$ and for each clock c_i that is reset by T, we add the event $Set(c_i; X \cup Y)$ to the label of T and we remove c_i from the reset of T, where:

- X is the set of values k such that there exists a transition T' reachable from T and: (i) there exists an inequation "$c_i \sim k$" in the guard of T'; (ii) there exists a trace P linking the destination location of T to the origin location of T' and (iii) P contains no reset of the clock c_i.
- Y is the set of values k such that there exists a location ℓ reachable from T and: (i) there exists an inequation "$c_i \sim k$" in the invariant of ℓ; (ii) there exists a trace P linking the destination location of T to ℓ and (iii) P contains no reset of the clock c_i.

Line 4: for each transition T, the set of Set events of T is the union of Set events for each clock that is reset by T.

The substep 2 is formally defined as follows:

1. $\forall T \in \mathcal{T}, \forall c_i \in \mathcal{C} : \mathcal{G}_T(c_i) := \{$"$\sim Exp(c_i, k)$": $(c_i \sim k) \in G_T\}$
2. $\forall T \in \mathcal{T} : \mathcal{G}_T := \bigcup_{c_i \in \mathcal{C}} \mathcal{G}_t(c_i)$
3. $\forall \ell \in \mathcal{L}, \forall c_i \in \mathcal{C} : \mathcal{I}_\ell(c_i) := \{$"$\sim Exp(c_i, k)$": $(c_i \sim k) \in \mathcal{I}_\ell\}$
4. $\forall \ell \in \mathcal{L} : \mathcal{I}_\ell := \bigcup_{c_i \in \mathcal{C}} \mathcal{I}_\ell(c_i)$

Explanations of the formal definition of substep 2:

Line 1: for each transition T and for each clock c_i such that there exists an inequation "$c_i \sim k$" in the guard of T, we replace this inequation by the *Exp-Condition* "$\sim Exp(c_i, k)$".
Line 2: the set of *Exp-Conditions* of T is the union of the *Exp-Conditions* of each clock, determined at line 1.
Line 3: for each location ℓ and for each clock c_i such that there exists an inequation "$c_i \sim k$" in the invariant of ℓ, we replace this inequation by the *Exp-Condition* "$\sim Exp(c_i, k)$".
Line 4: the set of *Exp-Conditions* of the invariant of ℓ is the union of *Exp-Conditions* of each clock, determined at line 3.

Notation 4.1 *:*

- $\mathcal{G}_\mathcal{C}$ *is the set of* Exp-Conditions *using the clocks of* $\mathcal{C}$.
- $\mathcal{S}_\mathcal{C}$ *is the set of* Set *events using the clocks of* $\mathcal{C}$.
- $StepOne(\mathcal{A})$ *denotes the step 1 (substep 1 and substep 2) applied to* $\mathcal{A}$*, i.e.* $\mathcal{A}^I = StepOne(\mathcal{A})$

Definition 4.2 *:* **Intermediary Automaton (IA)**
Let $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}, \mathcal{T}, l_0, \mathcal{L}_m)$ *be a TA.* $\mathcal{A}^I = StepOne(\mathcal{A})$ *is defined by*
$\mathcal{A}^I = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}^I, \mathcal{T}^I, l_0, \mathcal{L}_m)$ *with* $\mathcal{L}$*,* Σ*,* $\mathcal{C}$*,* l_0 *and* $\mathcal{L}_m$ *idem as in* $\mathcal{A}$*.* $\mathcal{I}^I : \mathcal{L} \to 2^{\mathcal{G}_\mathcal{C}}$ *associates to each location* $\ell \in \mathcal{L}$ *an invariant* $\mathcal{I}_\ell \in \mathcal{G}_\mathcal{C}$*.* $\mathcal{T}^I \subset \mathcal{L} \times \Sigma \times \mathcal{L} \times 2^{\mathcal{G}_\mathcal{C}} \times 2^{\mathcal{S}_\mathcal{C}}$ *is the set of transitions.*

As $\mathcal{A}^I$ is only a different way of representation of $\mathcal{A}$, it accepts the same timed trace than $\mathcal{A}$. Therefore, $\mathcal{A}$ and $\mathcal{A}^I$ generate the same timed language.

If we apply $StepOne$ to the TA of Figure 1, we obtain the IA of Figure 4. For example, since the transition T linking location I to location W resets the clock $c1$, then an event $Set(c1; *)$ ($*$ will be determined) is added to the label of this transition. The clock $c1$ is compared to 100 in the invariant of W and to 3 and again 100 in the guard of the transition linking W to I, and there is no reset of $c1$ between the present reset of $c1$ and these constraints. We add therefore $Set(c1; 3, 100)$ to the label of T. When all the Set events of $c1$ are recorded, we remove $c1$ from the reset of T.

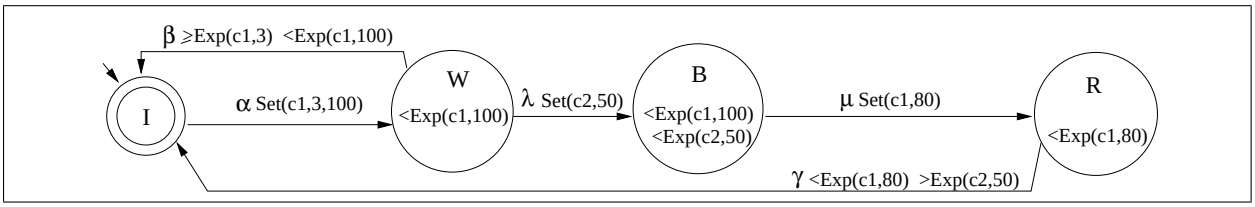


Figure 4: IA corresponding to the TA of Figure 1

5 Second step of $SetExp$

Given the TA $\mathcal{A}$ and the IA $\mathcal{A}^I = StepOne(\mathcal{A})$, the second step of $SetExp$ computes the SEA $\mathcal{B} = SetExp(\mathcal{A})$, also denoted $\mathcal{B} = StepTwo(\mathcal{A}^I)$. We shall present in Section 5.1 the principle of construction of the SEA $\mathcal{B}$, and in Section 5.2 the formal construction of $\mathcal{B}$.

5.1 Principle of the Set-Exp Automaton construction

Each state of a SEA has three parts, that give information on the position in the TA relatively to locations, and information on the time. The following definitions are useful to define these three parts.

Definition 5.1 *: **Clock-Cond***
Let $Set(c_i; k_1, k_2, \cdots, k_p)$ be a Set event for a given clock c_i. A Clock-Cond *on c_i w.r.t. $Set(c_i; k_1, k_2, \cdots, k_p)$ is a clock constraint in one of the following three forms, for $1 \leq u < p$:*

- *$0 < c_i < k_1$, which holds between $Set(c_i; k_1, \cdots, k_p)$ and $Exp(c_i, k_1)$;*
- *$k_u < c_i < k_{u+1}$, which holds between $Exp(c_i, k_u)$ and $Exp(c_i, k_{u+1})$, for $1 \leq u < p$; and*
- *$k_p < c_i$, which holds after $Exp(c_i, k_p)$.*

Definition 5.2 *: **ExpSeq***
Let $Set(c_i; k_1, k_2, \cdots, k_p)$ be a Set event for a given clock c_i. An ExpSeq *is a sequence $k_u k_{u+1} \cdots k_p$, for $1 \leq u \leq p$, which specifies:*

- *if $u > 1$: the remaining values to which c_i will expire after $Exp(c_i, k_{u-1})$;*
- *if $u = 1$: the values to which c_i will expire after $Set(c_i; k_1, k_2, \cdots, k_p)$.*

Definition 5.3 *: **ΔClock-Cond***
Let c_i and c_j be two clocks. A ΔClock-Cond *of c_i and c_j is an expression in one of the following four forms, where k, k_1 and k_2 are integers:*

- *$c_i - c_j < k$;*
- *$k < c_i - c_j$;*
- *$k_1 < c_i - c_j < k_2$;*
- *$c_i - c_j = k$.*

According to Definition 5.1, a *Clock-Cond* indicates in which interval of time the value of the clock is. From Definition 5.2, an *ExpSeq* indicates the next expected expiration(s) of a clock, before its next reset. From Definition 5.3, a *ΔClock-Cond* indicates in which interval of time the difference of two clocks is.

For a TA $\mathcal{A}$ and the SEA $\mathcal{B} = SetExp(\mathcal{A})$, the three parts of each state q of $\mathcal{B}$ are:
Part 1: is the location of q.
Part 2: is formed, for each clock in $\mathcal{C}$, of a *Clock-Cond* and an *ExpSeq*.
Part 3: is formed, for each pair of clocks in $\mathcal{C}$, of zero or one *ΔClock-Cond*.

Notations 5.1 *Let $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}, \mathcal{T}, l_0, \mathcal{L}_m)$ be a TA, $\mathcal{B} = SetExp(\mathcal{A}) = (Q, \Gamma, \delta, \Pi, q^0, Q_m)$ a SEA, $q \in Q$, $c_i \in \mathcal{C}$ and $c_j \in \mathcal{C}$.*

- *$C_q(c_i)$ denotes the* Clock-Cond *of c_i in q, $K_q(c_i)$ denotes the* ExpSeq *of c_i in q, $CE_q(c_i)$ denotes the pair $(C_q(c_i), K_q(c_i))$, and $\Delta C_q(c_i, c_j)$ denotes the* ΔClock-Cond *of c_i and c_j in q.*

- *C_q is the set of* Clock-Conds *of q, i.e.* $C_q = \bigcup_{c_i \in \mathcal{C}} C_q(c_i)$, *$CE_q$ is the set of pairs* $(C_q(c_i), K_q(c_i))$, *i.e. we have* $CE_q = \bigcup_{c_i \in \mathcal{C}} CE_q(c_i)$, *and* ΔC_q *is the set of* ΔClock-Conds *of q, i.e.* $\Delta\mathcal{C}_q = \bigcup_{c_i \in \mathcal{C}, c_j \in \mathcal{C}} \Delta\mathcal{C}_q(c_i, c_j)$.

A state q of $\mathcal{B}$ is therefore given by $q = (L_q, CE_q, \Delta\mathcal{C}_q)$, where L_q is the location of q. For a state $q = (L_q, CE_q, \Delta C_q)$, if $Set(c_i; k_1, k_2, \cdots, k_p)$ is the last Set event of c_i before q is reached, $CE_q(c_i) = (C_q(c_i), K_q(c_i))$ is in one of the following forms:

- If q is reached before $Exp(c_i, k_1)$: $C_q(c_i) = (0 < c_i < k_1)$, $K_q(c_i) = k_2 k_3 \cdots k_p$.
- If q is reached between $Exp(c_i, k_u)$ and $Exp(c_i, k_{u+1})$, $u = 1, \cdots, p-2$: $C_q(c_i) = (k_u < c_i < k_{u+1})$, $K_q(c_i) = k_{u+2} \cdots k_p$.
- If q is reached between $Exp(c_i, k_{p-1})$ and $Exp(c_i, k_p)$: $C_q(c_i) = (k_{p-1} < c_i < k_p)$, $K_q(c_i) = \epsilon$ (i.e., it is empty).
- If q is reached after $Exp(c_i, k_p)$: $C_q(c_i) = (k_p < c_i)$, $K_q(c_i) = \epsilon$.

The above definition of a state, with three parts, is necessary for the construction of the SEA, but for some studies, these parts are not useful, and then can be ignored after the construction of the SEA. For example, if the SEA is used in *language* based supervisory control [3], the state definition is not used for the synthesis of a supervisor, because this latter is based on the traces generated by the SEA. But in the case of *predicate* based supervisory control [8], this state definition can be useful for checking predicate satisfaction.

For the construction of the SEA, the usefulness of the three parts of a state q can be explained as follows:

- The location L_q is used to determine the transitions of $\mathcal{A}$ that are the candidates of enabling in q, which are the outgoing transitions of L_q in $\mathcal{A}$.
- C_q is used to determine the transitions of $\mathcal{A}$ that are enabled in q among the candidate ones, and the expected next expiration (if any) of each clock.
- K_q is used to determine the *Clock-Cond* and the *ExpSeq* of the state reached by each transition of $\mathcal{A}$ that is enabled in q.
- $\Delta\mathcal{C}_q$ is used to determine the expirations or set of simultaneous expirations that can really occur in q among the expected ones determined from C_q.

If two clocks c_i and c_j are inactive in a state q, $\Delta\mathcal{C}_q(c_i, c_j)$ is irrelevant, because the two clocks have no expected expirations. $\Delta\mathcal{C}_q(c_i, c_j)$ can then be removed from $\Delta\mathcal{C}_q$. In particular, when all the clocks are inactive in q, $\Delta\mathcal{C}_q$ is irrelevant, and thus, Part 3 is empty.

For a TA $\mathcal{A}$, the principle of construction of $\mathcal{B} = StepTwo(\mathcal{A}^I)$ from the IA $\mathcal{A}^I = StepOne(\mathcal{A})$, is as follows:

1. First, we construct the initial state $q^0 = \langle L_{q^0}, C_{q^0}, \Delta\mathcal{C}_{q^0} \rangle = \langle l_0, C_0, \emptyset \rangle$ where:
   a) l_0 is the initial location of $\mathcal{A}^I$;
   b) C_0 is a $N_c - uplet$ (N_c is the number of clocks) of pairs $(0 < c_i, \epsilon)$, $i = 1, \cdots, N_c$ and ϵ an empty *ExpSeq*.
   c) $\Delta\mathcal{C}_{q^0} = \emptyset$, because all clocks are initially inactive.
2. From the initial state and iteratively, we construct transitions and states as follows:
   a) For each constructed state q, we determine the set of events enabled in q;
   b) For each enabled event γ, we construct the state r reached after the occurrence of γ (r is constructed only once, then if it has been constructed in a previous iteration, no new construction of r is made) and the transition (q, γ, r).

   The iterative process terminates when no new state and new transition is constructed. Item 2a and 2b will be detailed further.

In a state q, when an expiration (or set of expirations) can occur, we obtain an enabled transition of type 1. When an event σ labeling a transition of $\mathcal{A}$ can occur, we obtain an enabled transition of type 2, and when expiration(s) and a σ can occur simultaneously, we obtain an enabled transition of type 3. To construct iteratively the SEA $\mathcal{B}$, we need then a procedure that allows to determine the set of events enabled at each constructed state (i.e. a procedure that realizes the above item 2a), and a procedure to construct the state reached after the occurrence of each enabled event and the corresponding transition (i.e. a procedure that realizes the above item 2b).

In a descriptive way, an event $\gamma \in \Gamma$ (of type 1, 2 or 3) is said enabled in state q if it is possible that at the same moment τ:

- The invariant of L_q is satisfied.
- The simultaneous occurrence of each event composing γ can take place, and this without the obligation for another event to occur before.
- The location invariant of the state reached after the occurrence of γ is satisfied.

In other words, for the event γ to be enabled, its occurrence must extend a trace, which converted into a timed trace (each event associated with its occurrence time), gives a timed trace accepted by the TA $\mathcal{A}$. In particular, if γ is of type 2, it is enabled if the guard of the corresponding transition labeled by $\gamma\rfloor_\Sigma$ in $\mathcal{A}$ is satisfied in q. If γ is of type 1, it is enabled if:

- at a given moment in q, all expirations in $\gamma\rfloor_{Exp}$ must occurs, because the delay programmed at their resets expire;
- if r is the state reached after the occurrence of γ, the invariant of L_r is satisfied at the time of occurrence of γ.

If γ is of type 3, it is enabled if:

- the guard of the corresponding transition labeled by $\gamma\rfloor_\Sigma$ in $\mathcal{A}$ is satisfied in q;
- the invariant of q is satisfied at the time of occurrence of γ.

We shall define more formally the enabling of each type of event in Section 6. In Section 5.2, we present formally the iterative construction of $\mathcal{B}$, supposing we have a procedure to determine which events are enabled at a given state (i.e. a procedure that realizes item 2a), and a procedure to compute the state reached after the occurrence of each enabled event (i.e. a procedure that realizes item 2b). These procedures will be presented in detail in Section 6.

5.2 Construction of the SEA

In this section, we shall present the formal procedure of construction of a SEA. The main contribution of this procedure in comparison to the one in [30] is: first the support of location with *invariants* and second the identification and fusion of equivalent states. Let us first present the identification of equivalent states.

Definition 5.4 *:* **prefix and suffix**
For the language $L(\mathcal{B})$ of $\mathcal{B}$, the set of finite prefixes *of $L(\mathcal{B})$, noted $\overline{L(\mathcal{B})}$, is the set of finite sequences of events of Γ that can be extended to sequences belonging to $L(\mathcal{B})$. Formally, the sequence s is a prefix of $L(\mathcal{B})$* iff *there exists a sequence s' of events of Γ such that $ss' \in L(\mathcal{B})$. The set of finite* suffixes *of $L(\mathcal{B})$, noted $\underline{L(\mathcal{B})}$, is the set of finite sequences of events of Γ that extend prefixes of $L(\mathcal{B})$ to sequences belonging to $L(\mathcal{B})$. Formally, the sequence s is a suffix of $L(\mathcal{B})$* iff *there exists a sequence s' of events of Γ such that $s's \in L(\mathcal{B})$.*

Notations 5.2 *: Let $\mathcal{B} = (Q, \Gamma, \delta, \Pi, q^0, Q_m)$ be a SEA, $q \in Q$, $\gamma \in \Gamma$, $\mathcal{E} \in \Gamma_1$:*

- *$(q, \gamma)!$ means γ is enabled in q, and (q, γ) denotes the state reached after the occurrence of γ from the state q, when $(q, \gamma)!$..*
- *$(q, \mathcal{E})_1!$ means that the simultaneous occurrence of the Exp of $\mathcal{E}$ is possible in q, but would lead to a state where an invariant is unsatisfied.*
- *Let $s = E_1E_2 \cdots E_m$ be a sequence of events belonging to Γ, $(q, s)!$ means that s is a suffix of $\mathcal{B}$ starting in q, i.e. for the states $q_0 = q$ and $q_i = (q_{i-1}, E_i)$, $i = 1, 2, ..., m$, we have $(q_{i-1}, E_i)!$.*
- *$C_q \approx \emptyset$ if all the clocks are inactive in q.*
- *$EXP(q)$ is the set of next expected expirations of clocks active in q. Therefore, if in q, $c_1, c_2, \cdots, c_p$ are the active clocks and $Exp(c_i, k_i)$ is the next expiration of c_i, for $i = 1, 2, \cdots, p$, then $EXP(q) = \{Exp(c_1, k_1), \cdots, Exp(c_p, k_p)\}$.*
- *For a location ℓ of a IA $\mathcal{A}^I = StepOne(\mathcal{A})$, $OUT(\ell)$ is the set of events of type 2 labeling the outgoing transitions of ℓ.*

Definition 5.5 *:* **same future**
For the SEA $\mathcal{B} = (Q, \Gamma, \delta, \Pi, q^0, Q_m)$ and two states q_1 and q_2 in Q, q_1 and q_2 have the same future iff *$\forall s \in \underline{L(\mathcal{B})}$: $(q_1, s)! \Leftrightarrow (q_2, s)!$.*

Definition 5.6 *:* **location equivalent states, equivalent SEA**
For $\mathcal{B} = (Q, \Gamma, \delta, \Pi, q^0, Q_m) = SetExp(\mathcal{A})$:

- *Two states q_1 and q_2 in Q are* location equivalent *relatively to $\mathcal{B}$, noted $q_1 \simeq q_2$, if $L_{q_1} = L_{q_2}$ and q_1 and q_2 have the same future.*

- *Two SEA $\mathcal{B}_1$ and $\mathcal{B}_2$ are equivalent, noted $\mathcal{B}_1 \simeq \mathcal{B}_2$,* iff $L(\mathcal{B}_1) = L(\mathcal{B}_2)$ *and* $L_m(\mathcal{B}_1) = L_m(\mathcal{B}_2)$.

Definition 5.6 states that two states of the SEA are *location equivalent* if they have the same location and the set of possible sequences from these states is the same. This definition is used for detecting states that are location equivalent during the construction of the SEA.

Remarque 5.1 *If $q_1 \simeq q_2$, then $(q_1 \in Q_m) \Leftrightarrow (q_2 \in Q_m)$. The reason is that if q_1 and q_2 have the same location and the same future, then if q_1 is a quiescent state, q_2 is also a quiescent state (the events enabled at q_1 are also enabled at q_2) and if q_1 is marked, its location is marked in $\mathcal{A}$ and then q_2 is also marked.*

Proposition 5.1 *If q_1 and q_2 are two states of a SEA such that: (i) $L_{q_1} = L_{q_2}$, (ii) $C_{q_1} \approx \emptyset$ and (iii) $C_{q_2} \approx \emptyset$, then q_1 and q_2 are location equivalent.*

Proposition 5.2 *If q_1 and q_2 are two states of a SEA $\mathcal{B}$ such that: (i) $L_{q_1} = L_{q_2}$, (ii) $C_{q_1} = C_{q_2}$ and (iii) $\forall \Delta C_{q_1}(c_i, c_j) \in \Delta C_{q_1}$, $\Delta C_{q_2}(c_i, c_j) \in \Delta C_{q_2}$: ($\Delta C_{q_1}(c_i, c_j) = \Delta C_{q_2}(c_i, c_j)$) OR ($\Delta C_{q_1}(c_i, c_j) \neq \Delta C_{q_2}(c_i, c_j)$ AND (c_i OR c_j is inactive)), then q_1 and q_2 are location equivalent.*

Proposition 5.1 states that two states are location equivalent if they have the same location and all clocks are inactive in these states. It can thus be deduced that location equivalent states have the same future, i.e., permit the same set of sequences. Proposition 5.2 states that if two states are differentiated uniquely by Δ*Clock-Conds* using one or two inactive clocks, then they are location equivalent. This can be justified intuitively by the fact that a Δ*Clock-Cond* using $c_i - c_j$ allows to determine the order of occurrences of the next expirations of c_i and c_j, and thus if one of c_i or c_j is inactive then this Δ*Clock-Cond* has no influence on the future because it does not influence the enabling of events.

Notation 5.1 *Let $\mathcal{B}$ be a SEA. $\mathcal{B}_\partial$ denotes the SEA obtained from $\mathcal{B}$ by merging the states that are location equivalent.*

Proposition 5.3 *For the SEA $\mathcal{B}$ and $\mathcal{B}_\partial$, we have: $\mathcal{B}_\partial \simeq \mathcal{B}$.*

Proposition 5.3 states that if in a SEA, we merge all the location equivalent states, the obtained SEA is equivalent to the original one. We take into account this fact by enclosing the merging of the location equivalent states in the procedure of construction of the SEA. This allows us to minimize the state space of the generated SEA. The merging of two states q_1 and q_2 is straightforwardly made by deleting q_2 and "re-directing" to q_1 all the transitions leading to q_2.

Remarque 5.2 *The merging of states used above do not take into account updating of the* ΔClock-Cond *of the states. This has no consequence in a study where* ΔClock-Conds *are not used. For a study where the exact expressions of the* ΔClock-Conds *of states are required, the merging of q_1 and q_2 can take into account the update of the* ΔClock-Cond *of q_1, by adding to the* ΔClock-Cond *of q_1 the one of q_2 and combining each $\Delta C_{q_1}(c_i, c_j)$ and $\Delta C_{q_2}(c_i, c_j)$ into one* ΔClock-Cond*. This is possible because the* ΔClock-Cond *using $c_i - c_j$ of two equivalent states defines two contiguous intervals.*

Let $\mathcal{A}^I = StepOne(\mathcal{A})$ be an IA, and $\mathcal{B}_0 = (Q_0, \Gamma_0, \delta_0, \Pi_0, q^0, Q_m^0) = (\{q^0\}, \emptyset, \emptyset, \Pi_0, q^0, Q_m^0)$ a SEA, with $Q_m^0 = \{q^0\}$ if q^0 is marked, else $Q_m^0 = \emptyset$, and Π_0 is such that $\Pi_0(q^0) = \emptyset$. q^0 has been specified in Section 5.1. Consider the series $\mathcal{B}_i = (Q_i, \Gamma_i, \delta_i, \Pi_i, q^0, Q_m^i)$, defined by $\mathcal{B}_{i+1} = \Omega(\mathcal{B}_i)$, for $i \geq 0$, where the operator Ω is formally defined below, and where the notation $\sharp$ denotes an empty label.

Definition of $\mathcal{B}_{i+1} = \Omega(\mathcal{B}_i)$:

1. $Q_{i+1} = Q_i \cup \{(q, \gamma) \,|\, q \in Q_i, \gamma \in (2^{EXP(q)} \times (OUT(L_q) \cup \{\sharp\})) \setminus \{\emptyset, \sharp\}, (q, \gamma)!\}$
2. $\delta_{i+1} = \delta_i \cup \{(q, \gamma, r) \,|\, q \in Q_i, (q, \gamma)!, r = (q, \gamma) \in Q_{i+1}\}$
3. $\Gamma_{i+1} = \Gamma_i \cup \{\gamma \,|\, \exists (q, \gamma, r) \in \delta_{i+1}\}$
4. $Q_m^{i+1} = Q_m^i \cup \{q \in Q_{i+1} \,|\, (L_q \in \mathcal{L}_m) \wedge (C_q \approx \emptyset)\}$
5. $\forall q \in Q_{i+1},\ \Pi_{i+1}(q) = \{\mathcal{E} \,|\, ((q, \mathcal{E})_1!)\}$

Explanation of the procedure of the operator Ω

Line 1: the set of states at the iteration $i+1$ is equal to the set of states at iteration i to which is added the new constructed states, which are states reached by transitions enabled in states of iteration i. ***Line 2:*** the set of transitions at iteration $i+1$ is equal to the set of transitions at iteration i to which are added the new constructed transitions, which are transitions enabled in states constructed at iteration i and that lead to states of iteration $i+1$. ***Line 3:*** the alphabet of the SEA at iteration $i+1$ is equal to the alphabet at iteration i to which are added the events of the new transitions constructed at line 2. ***Line 4:*** the set of marked states at iteration $i+1$ is equal to the set of marked states at iteration i to which are added the new marked states constructed at line 1. ***Line 5:*** at iteration $i+1$, for a given state q, the set of preempted events is the set of events of type 1 that, if they occur, can lead to a state where an invariant is unsatisfied.

Theorem 1 *The operator Ω has a fixpoint obtained after a finite number of iterations. Formally: $\exists p \geq 0 | \forall n \geq p : \mathcal{B}_n = \mathcal{B}_p$.*

The SEA $\mathcal{B} = SetExp(\mathcal{A})$ is the fixpoint of Ω.

According to the semantics of a SEA, a location must be left before its invariant becomes unsatisfied. In the construction of the SEA, considering that requirements 2.1 and 2.2 are satisfied, the generation of the SEA by the operator Ω constructs only the states where invariants are satisfied. To avoid construction of states where invariants are not satisfied, the procedure operates as follows:

- disable every transition T of type 1 if the invariant of the location of its destination state is unsatisfied.
- disable every transition of type 3 if at the moment where its occurrence is possible, the invariant of the location of its origin state is unsatisfied.

Notation 5.2 *:*

- *If $\mathcal{X}$ is a system of linear equations and inequations using a set of variables, $\mathcal{S}ol_{\geq 0}(\mathcal{X})$ denotes the set of non negative solutions of $\mathcal{X}$.*
- *Let q be a state of a SEA over the alphabet Γ. $\delta(q) \subseteq \Gamma$ is the set of events enabled in q.*

During the construction of the SEA $\mathcal{B}$, we can ensure the satisfaction of requirements 2.1 and 2.2 by checking at each iteration of the operator Ω the satisfaction of the following two conditions:

1. $\forall q, q' \in Q_i$:
   - **If** $\exists \gamma \in \Gamma_2 \cup \Gamma_3 | ((q,\gamma) = q') \wedge (\mathcal{S}ol_{\geq 0}(C_q \cup \mathcal{I}_{L_q}) \neq \emptyset) \wedge (\mathcal{S}ol_{\geq 0}(C_{q'} \cup \mathcal{I}_{L_{q'}}) = \emptyset)$,
   - **Then** requirement 2.1 is not satisfied.
2. $\forall q \in Q_i$:
   - **If** $\big(\Pi_i(q) \neq \emptyset\big) \wedge \big(\delta(q) \cap \Gamma_2 = \emptyset\big) \wedge \big(\exists \gamma \in \Pi_i(q) | (\nexists \beta \in \delta(q) \cap \Gamma_3 | \beta\rfloor_{Exp} = \gamma)\big)$,
   - **Then** requirement 2.2 is not satisfied.

Intuitively, for the first condition, requirement 2.1 is unsatisfied if there exists two states q and $q' = (q,\gamma)$ such that (i) γ is of type 2 or 3 and (ii) the invariant of L_q is satisfied whereas the one of $L_{q'}$ is unsatisfied. Translated in the TA, this means that there exists a transition which can be enabled in location L_q at a time when the invariant of its destination location (i.e $L_{q'}$) is unsatisfied, which means that $L_{q'}$ can be reached when its invariant is unsatisfied.

For the second condition, requirement 2.2 is not satisfied if at least one event (of type 1) must be preempted but there exists no transition able to make this preemption. Note that the transitions able to preempt an event $\mathcal{E} \in \Gamma_1$ are those labeled either by an event $\sigma \in \Gamma_2$ or $\gamma \in \Gamma_3$ such that $\gamma\rfloor_{Exp} = \mathcal{E}$.

In the generation of the SEA $\mathcal{B} = SetExp(\mathcal{A})$, we do not construct the states having the invariants of their locations unsatisfied, and the transitions that would lead to them are preempted. The satisfaction or dissatisfaction of the invariant of a location L_q of $\mathcal{A}$ can be checked from the inequations of the invariant and the *Clock-Conds* of q. If the system of inequations $\mathcal{X}_{L_q,C_q}$, formed of the invariant of L_q and the *Clock-Conds* of q has no positive solution, then the invariant of L_q is not satisfied. This statement is justified by the fact that for each clock c_i:

- $C_q(c_i)$ defines an interval denoted $I_{C_q(c_i)}$ which specifies the values each clock can have when in q;
- $\mathcal{I}_{L_q}(c_i)$ (i.e. the Clock Constraint of the invariant of L_q using c_i) defines an interval denoted $I_{\mathcal{I}_{L_q}(c_i)}$ which specifies the values each clock can when when in L_q, and

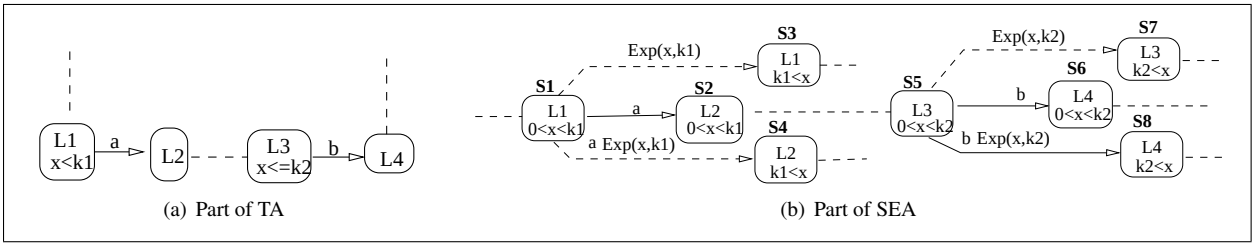
- if $I_{C_q(c_i)} \cap I_{\mathcal{I}_{L_q}(c_i)} = \emptyset$ (i.e $\mathcal{X}_{L_q,C_q}$ has no positive solution), then there exists no time in q during which the invariant of L_q is satisfied and thus, $\mathcal{I}_{L_q}$ is unsatisfied.

A transition of type 1 or 3 must be preempted if one of the following two conditions is satisfied:

Condition a: the location invariant of the destination state is unsatisfied;
Condition b: when the transition is enabled, the location invariant of its origin state is unsatisfied.

As only the execution of an event of Σ in a TA leads to a different location, then in a SEA, only transitions of type 2 and 3 have the locations of their origin and destination states different. When requirement 2.1 is satisfied, we can deduce that transitions of type 2 and 3 never satisfy Condition a. Thus, without preemption, only a transition of type 1 can have a destination state with location invariant unsatisfied and then satisfies Condition a. Only transitions of type 3 can satisfy Condition b, and in this case the transition of type 1 formed of the same expirations as those of the transition of type 3 satisfies Condition a.

To illustrate Condition a and b, let us consider the part of a TA represented on Figure 5(a). We suppose that the TA has one clock named x. The transformation in SEA of this part of TA is represented on Figure 5(b), where preempted transitions are represented by dashed lines. In state S1, the transition labeled by $Exp(x, k1)$ satisfies Condition a because the location invariant of its destination state (S3) is unsatisfied ($x > k1$ whereas the invariant of L1 requires $x < k1$). This transition will therefore be preempted. In the same way, the transition labeled by $aExp(x, k1)$ satisfies Condition b because when the transition is enabled (i.e. when $x = k1$), the location invariant of S1 is unsatisfied (L1 must be left before $x = k1$). This transition will also be preempted. In state S5, the transition labeled by $Exp(x, k2)$ satisfies Condition a (invariant of L3 is unsatisfied at S7) and will be preempted, but the transition labeled by $bExp(x, k2)$ does not satisfy Condition b (and Condition a) because of Requirement 2.1) and thus must not be preempted (when $x = k2$, the invariant of L3 is still satisfied). For this example, we shall have, for the function Π: $\Pi(S1) = \{Exp(x, k1)\}$ and $\Pi(S5) = \{Exp(x, k2)\}$.



(a) Part of TA (b) Part of SEA

Figure 5: Part of a TA and its SEA for illustration of preemption

If we apply the second step of $SetExp$ to the IA of Figure 4, we obtain the SEA of Figure 6, which is the same as the one of Figure 3, but where each state is defined by its three parts (see Section 3). The behavior of this SEA can be described as follows: from the initial state S1=$q^0 = (I, \{(0 < c_1, -), (0 < c_2, -)\}, \emptyset)$, the process can move to the state S2=$(W, \{(0 < c_1 < 3, 100), (0 < c_2, -)\}, \{0 < c_2 - c_1\})$ by the execution of the event $\alpha Set(c_1; 3, 100)$ (simultaneous occurrence of α and $Set(c_1; 3, 100)$). In S2, the invariant of W is always satisfied because $c_1 < 3$, while the invariant of W is satisfied as long as $c_1 < 100$. In S2, four events are enabled: $\lambda Set(c_2; 50)$ leading to S3, $Exp(c_1, 3)$ leading to S4, $\beta Exp(c_1, 3)$ leading to S5, and $\lambda Set(c_2; 50)Exp(c_1, 3)$ leading to S6. If for example the event $\lambda Set(c_2; 50)$ occurs, the process moves to the state S3=$(B, \{(0 < c_1 < 3, 100), (0 < c_2 < 50)\}, \{0 < c_2 - c_1 < 3\})$ and continues its move to S7 or S8 and so on. The occurrence of $Exp(c_1, 3)$ in S2 leads to S4=$(W, \{(3 < c_1 < 100, -), (0 < c_2, -)\}, \{0 < c_2 - c_1\})$. In S4, the expiration $Exp(c_1, 100)$ is preempted because if not, its occurrence would lead the process to the state S4'=$(W, \{(100 < c_1, -), (0 < c_2, -)\}, \{0 < c_2 - c_1\})$ where the invariant of W is unsatisfied because $c_1 > 100$. Therefore in S4, at least one of the event β or $\lambda Set(c_2; 50)$ occurs inevitably before $Exp(c_1, 100)$, in order to avoid the dissatisfaction of the invariant. The rest of the behavior can be described in a similar way.

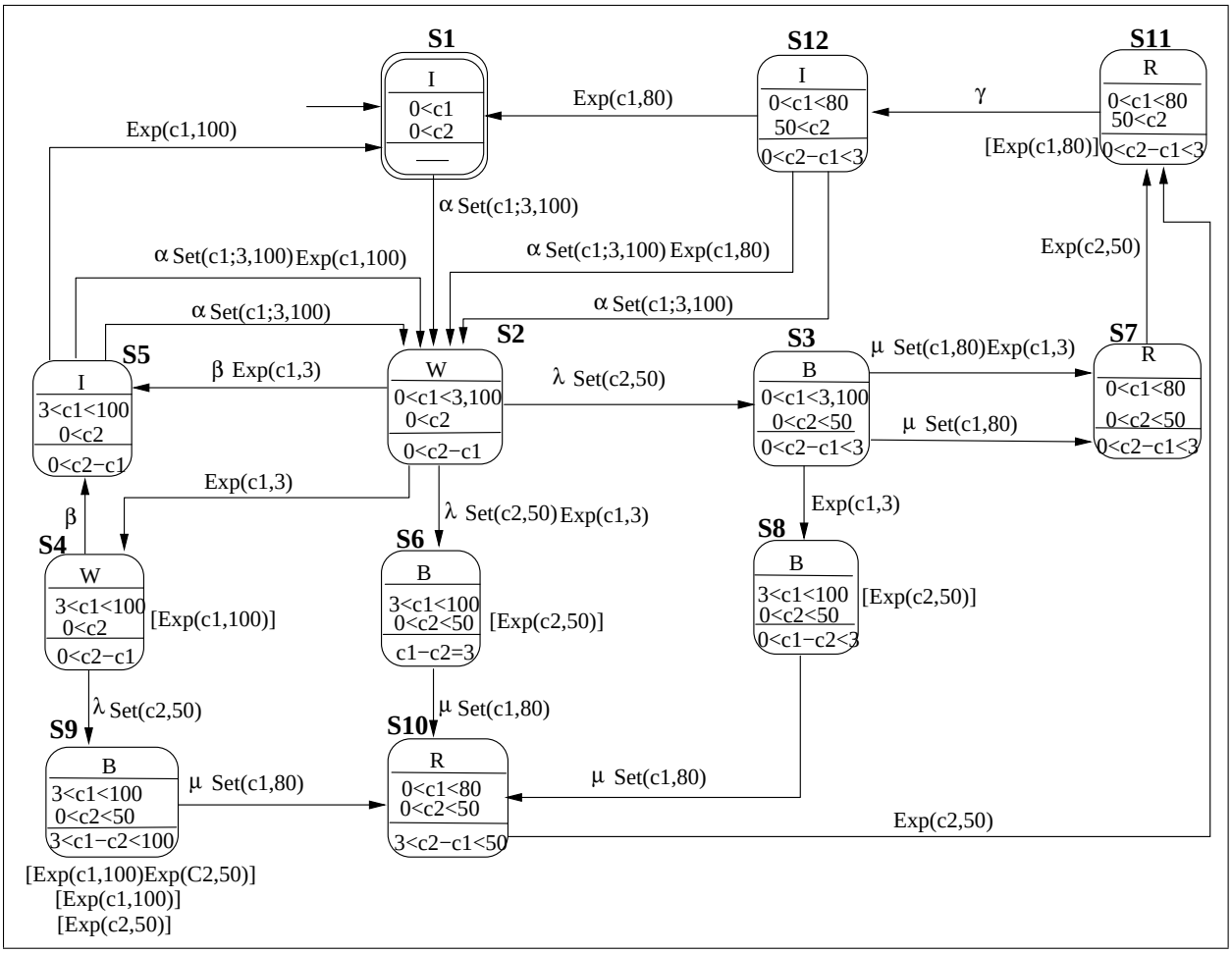


Figure 6: SEA corresponding to the IA of Figure 4

6 Construction of States and Transitions

In the previous section, we have presented the formal construction of the SEA, but we do not show: a) how to determine if an event is enabled in a given state, and b) how to compute the state reached by a given enabled event. In sections 6.1 and 6.2 below, we study items a) and b) respectively.

Notations 6.1 *:*

- *Let $\lambda = e_1 e_2 \cdots e_n$ be a sequence of events, $|\lambda|$ is the length of λ (i.e. $|\lambda| = n$); $\lambda[i]$ is the ith element of λ (i.e. $\lambda[i] = e_i$) and $\lambda[\not i]$ is a sequence obtained from λ by removing $\lambda[i]$ (i.e. $\lambda[\not i] = e_1 \cdots e_{i-1} e_{i+1} \cdots e_i$).*
- *If ΔC_1 and ΔC_2 are two* ΔClock-Conds *using $c_i - c_j$, where c_i and c_j are two clocks, then $\Delta C_1 \sqcap \Delta C_2$ is a* ΔClock-Cond *obtained by combining ΔC_1 and ΔC_2 into one* ΔClock-Cond *which corresponds to the conjunction of ΔC_1 and ΔC_2. For example, if $\Delta C_1 = k_1 < c_i - c_j < k_2$ and $\Delta C_2 = k_3 < c_i - c_j < k_4$, then $\Delta C_1 \sqcap \Delta C_2 = \max(k_1, k_3) < c_i - c_j < \min(k_2, k_4)$.*
- *Let c_i and θ_i be two clocks, q a state and $C_q(c_i)$ the* Clock-Cond *of q using c_i. $C_q\langle c_i \mapsto \theta_i - c_i\rangle$ is a* ΔClock-Cond *obtained from $C_q(c_i)$ by replacing c_i by $\theta_i - c_i$. For example, if $C_q(c_i) = k_1 < c_i < k_2$, then $C_q\langle c_i \mapsto \theta_i - c_i\rangle = \Delta C_q(c_i, \theta_i) = k_1 < c_i - \theta_i < k_2$*
- *Let ΔC be a set of* ΔClock-Conds*, and θ_i a clock used in ΔC. $\Delta C\langle \not\theta_i\rangle$ is a set of* ΔClock-Conds *obtained from ΔC by eliminating θ_i. The elimination of θ_i is made by firstly removing all* ΔClock-Cond *in the form $\Delta C_q(\theta_i, \theta_j)$ and by combining all pairs of* ΔClock-Cond *using θ_i to form one* ΔClock-Cond*. To combine two* ΔClock-Conds *$\Delta C_q(c_i, \theta_i)$ and $\Delta C_q(c_j, \theta_i)$, we express the first* ΔClock-Cond *using $\theta_i - c_i$ and the second using $c_j - \theta_i$, and we add member by member these two inequations to obtain one inequation using $c_j - c_i$. For example, if the first* ΔClock-Cond *is $k_1 < \theta_i - c_i < k_2$ and the second is $k_3 < \theta_i - c_j < k_4$, we rewrite the first inequation as $-k_2 < c_i - \theta_i < -k_1$ and we add it with the second to obtain $k_3 - k_2 < c_i - c_j < k_4 - k_1$. If ΔC contains only one* ΔClock-Cond *using θ_i, the elimination of θ_i consists in removing this* ΔClock-Cond*.*

6.1 Processing of transition of type 1

The following system of inequations will be used to determine when an event of type 1 is enabled, for a state q and an event $\mathcal{E}$ of type 1.

$$\begin{aligned}\mathcal{X}_{q,\mathcal{E}} = \Delta C_q \cup &\{c_i - k_i < 0 \mid Exp(c_i, k_i) \in \mathcal{E}\} \\ \cup &\{c_i - k_i = c_j - k_j \mid Exp(c_i, k_i), Exp(c_j, k_j) \in \mathcal{E}\} \\ \cup &\{c_i - k_i < c_j - k_j \mid Exp(c_j, k_j) \in \mathcal{E}, Exp(c_i, k_i) \in EXP(q) \setminus \mathcal{E}\}\end{aligned}$$

The above formula is a set of four sets of inequations: the first set ΔC_q indicates for each pair of clocks the bound of their difference and thus in which order their expirations may occur, the second set $\{c_i - k_i < 0 \mid Exp(c_i, k_i) \in \mathcal{E}\}$ indicates that the expirations of $\mathcal{E}$ have not still occurred, the third set $\{c_i - k_i = c_j - k_j \mid Exp(c_i, k_i), Exp(c_j, k_j) \in \mathcal{E}\}$ indicates that the expirations of $\mathcal{E}$ will occur simultaneously, and the fourth set $\{c_i - k_i < c_j - k_j \mid Exp(c_j, k_j) \in \mathcal{E}, Exp(c_i, k_i) \in EXP(q) \setminus \mathcal{E}\}$ indicates that the expirations of $\mathcal{E}$ are preceded by no other expiration. It is clear that the conjunction of these conditions indicates that $\mathcal{E}$ can be the next set of simultaneous expirations.

The following proposition allows to determine when an event $\mathcal{E} \in \Gamma_1$ is enabled in state q while invariant of L_q is satisfied.

Proposition 6.1 $(q, \mathcal{E})!$ iff *the following three conditions are satisfied:*

1. $\mathcal{E} \subseteq EXP(q)$*;*
2. $\mathcal{S}ol_{\geq 0}(\mathcal{X}_{q,\mathcal{E}}) \neq \emptyset$*;*
3. $\forall Exp(c_i, k) \in \mathcal{E} : \exists (\sim Exp(c_i, k')) \in \mathcal{I}_{L_q} \Rightarrow k' > k$*, with* $\sim \in \{<, \leq\}$.

Intuitively, $(q, \mathcal{E})!$ if it is possible that at the same moment τ, all the expirations of $\mathcal{E}$ occur and without leading to a state whose location invariant is unsatisfied. If $\mathcal{E}$ satisfies the first two conditions of Proposition 6.1 and does not satisfy the last condition, then its occurrence would lead to a state violating a location invariant, which is noted $(q, \mathcal{E})_1!$ (see Notation 5.2). In this case, $\mathcal{E}$ is preempted and we can record the preempted events in q as follows:

$\Pi(q) = \{\mathcal{E} \subseteq EXP(q) \mid (\mathcal{S}ol_{\geq 0}(\mathcal{X}_{q,\mathcal{E}}) \neq \emptyset) \wedge$
$(\exists Exp(c_i,k) \in \mathcal{E}, \exists(\sim Exp(c_i,k')) \in \mathcal{I}_{L_q} \mid k' \leq k)\}$

For an enabled event $\mathcal{E}$ in state q, the following procedure allows to compute the state $r = (q,\mathcal{E})$ reached after the occurrence of $\mathcal{E}$:

1. $r := q$
2. $\forall Exp(c_i,k) \in \mathcal{E} : |K_r(c_i)| = 0 \Rightarrow C_r(c_i) := (k < c_i)$;
3. $|K_r(c_i)| > 0 \Rightarrow C_r(c_i) := (k < c_i < K_r(c_i)[1])$,
4. $|K_r(c_i)| > 0 \Rightarrow K_r(c_i)[\not{1}]$.
5. $\forall Exp(c_i,k_i), Exp(c_j,k_j) \in \mathcal{E}, \Delta C_r(c_i,c_j) := (c_i - c_j = k_i - k_j)$.
6. $\forall Exp(c_i,k_i) \in \mathcal{E}, \forall Exp(c_j,k_j) \in EXP(q) \setminus \mathcal{E}, \Delta C_r(c_i,c_j) := \Delta C_r(c_i,c_j) \sqcap (k_i - k_j < c_i - c_j)$.
7. $(\forall c_i \in \mathcal{C}, \exists k > 0, C_r(c_i) = (k < c_i)) \Rightarrow \Delta C_r := \emptyset$.

Explanation of the above procedure of computation of $r = (q,\mathcal{E})$

Line 1 : r is initialized to q.
Lines 2-4 : for each $Exp(c_i,k) \in \mathcal{E}$, the objective is to update $C_r(c_i)$ and $K_r(c_i)$.
Line 2 : we consider the case where $Exp(c_i,k) \in \mathcal{E}$ is the last expiration of c_i, and thus we assign $k < c_i$ to $C_r(c_i)$ because $Exp(c_i,k)$ has occurred and no other expiration of c_i is programmed.
Lines 3-4 : we consider the case where there exists $Exp(c_i,k')$ programmed after $Exp(c_i,k)$. At ***Line 3***, we update $C_r(c_i)$ considering that $Exp(c_i,k)$ has occurred and $Exp(c_i,k')$ is the next programmed expiration of c_i. At ***line 4***, we remove k' from $K_r(c_i)$ because $K_r(c_i)$ is a sequence which indicates the values to which c_i will expire after its next expiration, which is in this case $Exp(c_i,k')$.
Line 5 : for all $Exp(c_i,k_i) \in \mathcal{E}$, we have the equality $c_i - k_i = 0$ at the time of occurrence of $\mathcal{E}$. Thus, for all $Exp(c_i,k_i) \in \mathcal{E}$ and $Exp(c_j,k_j) \in \mathcal{E}$, we have $c_i - k_i = c_j - k_j = 0$ at the time of occurrence of $\mathcal{E}$. As all the clocks increase at the same rate, then $c_i - k_i = c_j - k_j$, (i.e. $c_i - c_j = k_i - k_j$) is true before and after the occurrence of $\mathcal{E}$ and then, we replace $\Delta C_r(c_i,c_j)$ by this equality which is a more precise information
Line 6 : for all $Exp(c_i,k_i) \in \mathcal{E}$, we have the equality $c_i - k_i = 0$ at the time of occurrence of $\mathcal{E}$. In the same way, for all $Exp(c_j,k_j) \in EXP(q) \setminus \mathcal{E}$, we have the inequality $c_j - k_j < 0$ at the time of occurrence of $\mathcal{E}$. As all the clocks increase at the same rate, then $c_j - k_j < c_i - k_i$ (i.e. $k_i - k_j < c_i - c_j$) is true before and after the occurrence of $\mathcal{E}$. We combine this inequation with $\Delta C_r(c_i,c_j)$.
Line 7 : we empty ΔC_r if all the clocks are inactive.

6.2 Processing of transitions of type 2 and 3

Let q be a state of the SEA $\mathcal{B} = SetExp(\mathcal{A}) = StepTwo(\mathcal{A}^I)$, and $\sigma \in \Gamma_2$ labeling a transition T of $\mathcal{A}^I$. The following proposition allows to determine when σ is enabled in q.

Proposition 6.2 $(q,\sigma)!$, *for* $\sigma \in \Gamma_2$, iff*:*

1. $\sigma \in OUT(L_q)$*;*
2. $\forall(< Exp(c_i,k))$ *or* $(\leq Exp(c_i,k)) \in \mathcal{G}_T : \exists k' \leq k, \exists u < k'$ *such that* $C_q(c_i) = (u < c_i < k')$*;*
3. $\forall c_i \in \mathcal{Z}_T, (= Exp(c_i,k)) \notin \mathcal{G}_T$*;*
4. $\forall(> Exp(c_i,k))$ *or* $(\geq Exp(c_i,k)) \in \mathcal{G}_T : \exists k' \geq k, \exists u > k'$ *such that* $(C_q(c_i) = (k' < c_i < u)$ *or* $C_q(c_i) = (k' < c_i))$*;*

Proposition 6.2 allows to determine if $(q,\sigma)!$, by checking if the guard of Transition is satisfied in state q. More precisely:
Condition 1: means that σ labels an outgoing transition of Location L_q in $\mathcal{A}^I$.
Condition 2: means that if there exists an *Exp-Condition* in the form "$< Exp(c_i,k)$" or "$\leq Exp(c_i,k)$" in the guard of T, then the expiration $Exp(c_i,k)$ has not occurred yet.
Condition 3: means that there exist no *Exp-Condition* in the form "$= Exp(c_i,k)$" in the guard of T.

Condition 4: means that if there exists an *Exp-Condition* in the form "$> Exp(c_i, k)$" or "$\geq Exp(c_i, k)$" in the guard of T, then the expiration $Exp(c_i, k)$ has already occurred.

If $\gamma = (\mathcal{E}, \sigma) \in \Gamma_3$, the following proposition allows to determine when $(q, \gamma)!$, i.e. γ is enabled, where T is the transition labeled by σ in the IA.

Proposition 6.3 $(q, \gamma)!$, *with* $\gamma = (\mathcal{E}, \sigma)$, *where* $\mathcal{E} \in \Gamma_1$ *and* $\sigma \in \Gamma_2$, iff*:*

1. $\sigma \in OUT(L_q)$*;*
2. $\mathcal{E} \subseteq EXP(q)$ *and* $\mathcal{S}ol_{\geq 0}(\mathcal{X}_{q,\mathcal{E}}) \neq \emptyset$*;*
3. $\forall (< Exp(c_i, k)) \in \mathcal{G}_T : Exp(c_i, k) \notin \mathcal{E}$ *and* $\exists k' \leq k, \exists u < k'$ *such that* $C_q(c_i) = (u < c_i < k')$*;*
4. $\forall (\leq Exp(c_i, k)) \in \mathcal{G}_T : \exists k' \leq k, \exists u < k'$ *such that* $C_q(c_i) = (u < c_i < k')$*;*
5. $\forall (= Exp(c_i, k)) \in \mathcal{G}_T : Exp(c_i, k) \in \mathcal{E}$ *and* $\exists u < k$ *such that* $C_q(c_i) = (u < c_i < k)$*;*
6. $\forall (> Exp(c_i, k)) \in \mathcal{G}_T : \exists k' \geq k, \big(\exists u > k'$ *such that* $C_q(c_i) = (k' < c_i < u)\big)$ *or* $C_q(c_i) = (k' < c_i)$*;*
7. $\forall (\geq Exp(c_i, k)) \in \mathcal{G}_T$:
   - ***Either*** $\exists k' \geq k, (\exists u > k'$ *such that* $C_q(c_i) = (k' < c_i < u))$ *or* $C_q(c_i) = (k' < c_i)$*;*
   - ***Or*** $Exp(c_i, k) \in \mathcal{E}$ *and* $\exists u < k$ *such that* $C_q(c_i) = (u < c_i < k)$.
8. $\forall Exp(c_i, k) \in \mathcal{E} : \exists (< Exp(c_i, k')) \in \mathcal{I}_{L_q} \Rightarrow k < k'$ *and* $\exists (\leq Exp(c_i, k')) \in \mathcal{I}_{L_q} \Rightarrow k \leq k'$.

Conditions 1-7 of Proposition 6.3 allow to determine if γ may be enabled if the satisfaction of invariants is not taken into account, and condition 8 allows to exclude the preempted events. More precisely:
Condition 1: means that σ labels an outgoing transition of the location L_q in $\mathcal{A}^I$.
Condition 2: means that all expirations in $\mathcal{E}$ can occur simultaneously in q, i.e. $\mathcal{E}$ satisfies the conditions for an event of type 1 to be enabled (point 1 and 2 of Proposition 6.1), without taking account satisfaction of invariants (point 3 of Proposition 6.1).
Condition 3: means that if there exists an *Exp-Condition* in the form "$< Exp(c_i, k)$" in the guard of T, then the expiration $Exp(c_i, k)$ has not yet occurred and is not in $\mathcal{E}$.
Condition 4: means that if there exists an *Exp-Condition* in the form "$\leq Exp(c_i, k)$" in the guard of T, then the expiration $Exp(c_i, k)$ has not yet occurred.
Condition 5: means that if there exists an *Exp-Condition* in the form "$= Exp(c_i, k)$" in the guard of T, then the expiration $Exp(c_i, k)$ is in $\mathcal{E}$ and is the next programmed expiration of c_i.
Condition 6: means that if there exists an *Exp-Condition* in the form "$> Exp(c_i, k)$" in the guard of T, then the expiration $Exp(c_i, k)$ has already occurred.
Condition 7: means that if there exists an *Exp-Condition* in the form "$\geq Exp(c_i, k)$" in the guard of T, then either $Exp(c_i, k)$ has already occurred, or it is in $\mathcal{E}$ and is the next programmed expiration of c_i.
Condition 8: means that γ is not preempted, i.e. at the time of its occurrence, the invariant of L_q is satisfied.

When $\gamma = (\mathcal{E}, \sigma) \in \Gamma_3$ is enabled in accordance with Proposition 6.3, the following procedure allows to compute the state $r = (q, \gamma)$ reached after the occurrence of γ.

1. $r := (q, \mathcal{E})_{1 \to 6}$ /*execution of lines 1-6 of the procedure of Section 6.1*/
2. $L_r :=$ destination location of T
3. $\forall Set(c_i, k_1, k_2, \cdots, k_m) \in \mathcal{Z}_T$:
4. $\quad \Delta C_r \langle c_i \mapsto \theta_i \rangle$
5. $\quad \nexists Exp(c_i, k) \in \mathcal{E} \Rightarrow \Delta C_r := \Delta C_r \cup \{C_r(c_i) \langle c_i \mapsto \theta_i - c_i \rangle\}$
6. $\quad \exists Exp(c_i, k) \in \mathcal{E} \Rightarrow \Delta C_r := \Delta C_r \cup \{\theta_i - c_i = k\}$
7. $\quad \forall c_j \notin Z_T$:
8. $\quad\quad \nexists Exp(c_j, k) \in \mathcal{E} \Rightarrow \Delta C_r := \Delta C_r \cup \{C_r(c_j) \langle c_j \mapsto c_j - c_i \rangle\}$
9. $\quad\quad \exists Exp(c_j, k) \in \mathcal{E} \Rightarrow \Delta C_r := \Delta C_r \cup \{c_j - c_i = k\}$
10. $\quad C_r(c_i) := (0 < c_i < k_1)$
11. $\quad K_r(c_i) := k_2 \cdots k_m$
12. $\forall c_i, c_j \in Z_T : \Delta C_r := \Delta C_r \cup \{c_j - c_i = 0\}$
13. $\forall \theta_i, \Delta C_r \langle \nexists \theta_i \rangle$
14. $r := (q, \mathcal{E})_7$ /*execution of line 7 of the procedure of Section 6.1*/

Explanation of the above procedure of computation of $r = (q, \gamma)$

Line 1 : we compute the intermediate state $v = (q, \mathcal{E})$ which would be obtained after the occurrence of $\mathcal{E}$ at q, using the procedure of Section 6.1 without executing its last line, because this later is executed at the end of the procedure in order to finalize the computation of each state. The justification of this intermediate computation is that the simultaneity of σ and $\mathcal{E}$ can be interpreted as the occurrence of $\mathcal{E}$ immediately followed by the one of σ. r is initialized to this intermediary state v.
Line 2 : we update the location of r which becomes the destination location of T.
Lines 3-13 : these lines are of use to compute the effect of the *resets* of T. Therefore, if T has no *reset*, then T has no influence on the values of the clocks and thus, r is determined by lines 1, 2 and 14.
Lines 3-11 : these lines allow to compute the effect of the *resets* of T, for each clock c_i, on r.
Line 4 : we replace in the Δ*Clock-Conds* each clock c_i which is reset by T by a fictitious clock θ_i which has the same value as c_i before its reset. This fictitious clock is not reset by T and thus, all the Δ*Clock-Conds* using c_i remain satisfied after the execution of T if we replace c_i by θ_i.
Line 5 : if c_i does not expire, i.e. $Exp(c_i, k) \notin \mathcal{E}$, we add to ΔC_r the Δ*Clock-Conds* obtained by replacing c_i by $\theta_i - c_i$. This is justified by the fact that if τ is the time when T occurs and c_i is reset, then $C_r(c_i)$ is satisfied before the reset of c_i and at the moment τ if c_i has not been reset. At τ, θ_i has the value that c_i would have if it has not been reset, and c_i is null. We deduce that $C_r(c_i)$ is satisfied after the execution of T and before the next reset of c_i, if we replace c_i by $\theta_i - c_i$. This information is inserted in ΔC_r.
Line 6 : if c_i expires, i.e. $Exp(c_i, k) \in \mathcal{E}$,we add to ΔC_r the Δ*Clock-Cond* "$\theta_i - c_i = k$". This is justified by the fact that if τ is the time of occurrence of T, we have $\theta_i = k$ and $c_i = 0$ at τ. Thus $\theta_i - c_i = k$ at τ, and this equality remains true until the next reset of c_i. This information is inserted in ΔC_r.
Line 7 : we consider every c_j that is not reset by T.
Line 8 : if c_j does not expire, i.e. $Exp(c_j, k) \notin \mathcal{E}$, we add to ΔC_r the Δ*Clock-Conds* obtained by replacing c_j by $c_j - c_i$ in the *Clock-Cond* of r using c_i. This is justified by the fact that if τ is the time when T occurs and c_i is reset, then $C_r(c_j)$ is satisfied at τ because c_j is not reset by T. At τ, $c_j - c_i$ has the value of c_i because c_i is reset at this moment. We deduce that $C_r(c_j)$ is satisfied after the occurrence of T and before the next reset of c_i, if we replace c_j by $c_j - c_i$. This information is inserted in ΔC_r.
Line 9 : if c_j expires, i.e. $Exp(c_j, k) \in \mathcal{E}$, we add to ΔC_r the Δ*Clock-Cond* $c_j - c_i = k$. This is justified by the fact that if τ is the time of occurrence of T, we have $c_j = k$ and $c_i = 0$ at τ. Thus $c_j - c_i = k$ at τ, and this equality remains true before the next reset of c_i. This information is inserted in ΔC_r.
Line 10 : we update $C_r(c_i)$ by its status just after the reset of c_i. If $Set(c_i; k_1, k_2 \cdots, k_m)$ is the Set event of c_i associated to T, then the new $C_r(c_i)$ indicates that c_i has been reset and that its next expiration is $Exp(c_i, k_1)$.
Line 11 : we update $K_r(c_i)$ by its status just after the reset of c_i. If $Set(c_i; k_1, k_2 \cdots, k_m)$ is the Set event of c_i associated to T, then the new $K_r(c_i)$ indicates that c_i has been reset and that its next expirations values after $Exp(c_i, k_1)$ are $k_2 \cdots k_m$.
Line 12 : for two clocks c_i and c_j which are reset by T, the equality $c_j - c_i = 0$ is satisfied before the next reset of one of the two clocks. This information is inserted in ΔC_r.
Line 13 : We eliminate each θ_i from ΔC_r, because θ_i has been used just to keep the information on the clocks which have been reset. For this elimination we add each pair of Δ*Clock-Cond* using θ_i as indicated in Notation 6.1.
Line 14 : we finalize the computation of r by executing the last line of the procedure of construction for events of type 1, i.e. we empty ΔC_r if all the clocks are inactive.

7 Correctness, properties and simplification of $SetExp$

We shall present in this section some ideas on the correctness of $SetExp$ and some of its properties. Moreover, we shall present a way of simplification of $SetExp$.

7.1 Correctness of $SetExp$

Let $L(\mathcal{B})$ be the regular language of a SEA $\mathcal{B}$ over the alphabet Γ. By construction, $L(\mathcal{B})$ respects the following condition, called the ***consistency condition***: each $Set(c; k)$ and its corresponding expiration $Exp(c, k)$ are separated

by a delay k. Recall that a trace in $L(\mathcal{B})$ is a sequence in the form $\gamma_1\gamma_2\cdots\gamma_n$ which terminates in a quiescent state, where each $\gamma_i \in \Gamma$ is the event of a transition of $\mathcal{B}$.

Definition 7.1 *:* **timed language of SEA**
The timed language of a SEA $\mathcal{B}$, noted $L^t(\mathcal{B})$, is defined as follows, where each τ_i ($i = 1, \cdots, n$) is a real value: $(\gamma_1, \tau_1)(\gamma_2, \tau_2)\cdots(\gamma_n, \tau_n) \in L^t(\mathcal{B})$ iff*:* $\gamma_1\gamma_2\cdots\gamma_n \in L(\mathcal{B})$, $0 < \tau_1 < \tau_2 < \cdots < \tau_n$, *and* $\forall\gamma_i, \gamma_j, i < j$:
If *γ_i contains $Set(c; k)$, γ_j contains $Exp(c, k)$ and no γ_m ($i < m < j$) contains $Set(c; *)$,*
then $\tau_j = \tau_i + k$.

Definition 7.1 means that $L^t(\mathcal{B})$ contains all the timed traces obtained from a trace of $L(\mathcal{B})$ by associating a time respecting the consistency condition to each event.

Definition 7.2 *:* **equivalence of TA and SEA**
A TA $\mathcal{A}$ is said equivalent to a SEA $\mathcal{B}$ iff *$L^t(\mathcal{A})$ (the timed language of $\mathcal{A}$) is obtained from $L^t(\mathcal{B})$ by removing all the Set and Exp events.*

Intuitively, a TA $\mathcal{A}$ is equivalent to a SEA $\mathcal{B}$ *iff* the behavior of $\mathcal{A}$ and $\mathcal{B}$ cannot be distinguished by an observer who does not observe Set and Exp events.

Theorem 2 *Each TA $\mathcal{A}$ and its corresponding SEA $SetExp(\mathcal{A})$ are equivalent.*

Theorem 2 ensures the correctness of $SetExp$. This implies that it is possible to transform the study of a RTDES modeled by a TA $\mathcal{A}$ under an untimed form, by making the study on the SEA $\mathcal{B} = SetExp(\mathcal{A})$.

7.2 Properties and complexity of $SetExp$

We express in this section some properties of $SetExp$ and determine its complexity in term of maximum number of states.

Property 7.1 *If in a TA $\mathcal{A}$, we multiply by the same value k all the constants used to express the timing constraints, the SEA $\mathcal{B} = SetExp(\mathcal{A})$ does not change, modulo the multiplication by k of every constants in the Set and Exp events.*

Property 7.1 is justified by the fact that multiplying by the same constant all the values in the timing constraints implies the multiplication by the same value of all the constants in the *Clock-Cond* and *ΔClock-Cond* of the states of the SEA, and this does not influence the solutions of these inequations. Intuitively, multiplying all the constants in the timing constraints by the same value is equivalent to changing the unit of time. This change has no influence on the generation of Set and Exp events. We can then switch from one unit of time to a more fine one, for example from minutes to seconds by multiplying the constants by 60, without changes in the structure of the SEA. Property 7.1 is interesting because: (i) state space in the obtained SEA does not increase when we multiply by the same value all the constants used in the guards and invariants, and (ii) if $SetExp$ has been applied to generate a SEA $\mathcal{B}$ from a TA $\mathcal{A}$, we can deduce straightforwardly every SEA corresponding to any TA $\mathcal{A}'$ that is similar to $\mathcal{A}$ modulo a multiplication of constant(s) used in guards and invariants.

Property 7.2 *We consider a class of TA such that for every TA $\mathcal{A}$ of this class: let $k_1, k_2, \cdots, k_p$ (where $k_i < k_{i+1}$, for $i = 1, 2, \cdots, p_{-1}$) be all the constants used in the guards and invariants of $\mathcal{A}$.*
If *every k_i is replaced by m_i such that $m_i < m_{i+1}$, for $i = 1, 2, \cdots, p_{-1}$,*
Then *the SEA $SetExp(\mathcal{A})$ is not modified, modulo the replacement of every k_i by m_i.*

Property 7.2 holds for certain TA for which if we change the constants in the guards and invariants but we let them in the same order, the corresponding SEA does not change, modulo the replacement of every constant in the Set and Exp events by its new value. The reason for which Property 7.2 cannot be satisfied for a given TA $\mathcal{A}$ is as follows: in a state q where the next expirations of the clocks c_i and c_j (supposing only these two clocks are active in q) are respectively $Exp(c_i, k_i)$ and $Exp(c_j, k_j)$, the order of occurrence of $Exp(c_i, k_i)$ and $Exp(c_j, k_j)$ depends on the difference $c_i - c_j$,

i.e. the Δ*Clock-Cond* of q, which is defined using the constants used in the guards and invariants of $\mathcal{A}$. Then, changing the values in guards and transitions can change the number of states and transitions because we shall not have the same number of states and transitions in the case where only $Exp(c_i, k_i)Exp(c_j, k_j)$ is enabled and in the case where one or both individually of $Exp(c_i, k_i)$ and $Exp(c_j, k_j)$ is enabled. Therefore, in particular, Property 7.2 holds for every TA that use one clock. Even if Property 7.2 does not hold in general, we studied several examples where the state space of the corresponding SEA change, but its size does not increase significantly with the magnitudes of constants used in guards and invariants. For certain examples, we even noted a decrease of the state space despite the increase of the magnitudes of constants, and we have noted that in general, there is no relation between the magnitude of the constants and the size of the state space.

Properties 7.1 and 7.2 and our observations in various examples show an advantage of using SEA instead of RA because contrary to RA, in practice the state space of SEA does not increase with the magnitudes of the constants used in timing constraints.

Definition 7.3 *:* **deterministic TA**
A TA is deterministic if:

- *it has one initial location;*
- *two transitions having the same origin location and the same event and which can be enabled at the same time, have the same destination location and reset the same clocks.*

$SetExp$ can be applied indifferently to deterministic and non deterministic TA.

Definition 7.4 *:* **deterministic SEA**
A SEA $\mathcal{B}$ is deterministic if at each state q, there exists no pair of transitions labeled by the same event which have distinct destination states.

Property 7.3 *If a TA $\mathcal{A}$ is deterministic, then $\mathcal{B} = SetExp(\mathcal{A})$ is also deterministic.*

Justification of Property 7.3 is evident, because in the procedure of construction of a SEA, only one destination state is constructed for each enabled event. Property 7.3 indicates that $SetExp$ conserves determinism.

Let us now give an outline of the computational complexity of $SetExp$. Given a TA $\mathcal{A}$, let $|\mathcal{L}|$, $|\mathcal{T}|$, $|\Sigma|$ and $|\mathcal{C}|$ be respectively the number of locations, the number of transitions, the number of events and the number of clocks of $\mathcal{A}$.

Proposition 7.1 *The complexity of the first step of SetExp is and thus the computational complexity of this step is $\mathcal{O}(|\mathcal{T}|^2|\mathcal{C}|^3 + |\mathcal{T}||\mathcal{L}||\mathcal{C}|^2)$.*

In the iterative construction of the SEA $\mathcal{B} = SetExp(\mathcal{A})$, we must process each state by determining the enabled events (and thus enabled transitions) and constructing the state reached after the occurrence of each enabled event.

Proposition 7.2 *Let q be a state of $\mathcal{B}$, $\mathcal{E} \in \Gamma_1$, $\sigma \in \Gamma_2$ and $\gamma \in \Gamma_3$.*

1. *The maximum number of* ΔClock-Conds *of q is $|\Delta C| = \mathbb{C}^2_{|\mathcal{C}|} = \frac{|\mathcal{C}|!}{2(|\mathcal{C}|-2)!} = \frac{|\mathcal{C}|(|\mathcal{C}|-1)}{2}$. Then, we have $\mathcal{O}(|\Delta C|) = \mathcal{O}(|\mathcal{C}|^2)$.*
2. *The maximum number of events of type 1 that can be enabled at q, denoted $|\Gamma_{1,q}|$, is the number of non empty subsets in a set of $|\mathcal{C}|$ components. That is $|\Gamma_{1,q}| = \sum_{i=1,\ldots,|\mathcal{C}|} \frac{|\mathcal{C}|!}{i!(|\mathcal{C}|-i)!} = 2^{|\mathcal{C}|} - 1$. Then, we have $\mathcal{O}(|\Gamma_{1,q}|) = \mathcal{O}(2^{|\mathcal{C}|})$.*
3. *The complexity of checking if $(q, \mathcal{E})!$ is $\mathcal{O}(|\mathcal{C}|^3)$.*
4. *The complexity of checking if $(q, \sigma)!$ is $\mathcal{O}(|\mathcal{C}|^2)$.*
5. *The complexity of checking if $(q, \gamma)!$ is $\mathcal{O}(|\mathcal{C}|^3)$.*
6. *The complexity of determining all the events enabled at q is $\mathcal{O}(2^{|\mathcal{C}|}|\mathcal{C}|^2(|\mathcal{C}| + |\Sigma|))$.*

Proposition 7.3 *Let q_1, q_2 and q_3 be three states of $\mathcal{B}$, $\mathcal{E} \in \Gamma_1$, $\sigma \in \Gamma_2$ and $\gamma \in \Gamma_3$ such that $(q_1, \mathcal{E})!$, $(q_2, \sigma)!$ and $(q_3, \gamma)!$.*

1. *The complexity to compute the state $r = (q_1, \mathcal{E})$ is $\mathcal{O}(|\mathcal{C}|^4)$.*
2. *The complexity to compute the state $r = (q_2, \sigma)$ is $\mathcal{O}(|\mathcal{C}|^3)$.*
3. *The complexity to compute the state $r = (q_3, \gamma)$ is $\mathcal{O}(|\mathcal{C}|^4)$.*
4. *The complexity to check if q is equal or equivalent to a state q' is $\mathcal{O}(|\mathcal{C}|^4)$.*

The iterative procedure of construction of $\mathcal{B} = SetExp(\mathcal{A}) = StepTwo(\mathcal{A}')$ given in Section 5.2 process at each iteration one state. The following proposition give the complexity to process a state at iteration $i+1$, where $|Q_i|$ is the number of states of $\mathcal{B}_i$.

Proposition 7.4 *The computational complexity to process a state at each iteration of the procedure of computation of the SEA given in Section 5.2 is $\mathcal{O}(2^{|\mathcal{C}|}|\Sigma|(|\mathcal{C}|^4 + |Q|))$ where $|Q|$ is the maximum number of states of the SEA (see Proposition 7.5).*

To process a state q, we determine all the events enabled in q and for each one, we compute the state reached after its occurrence and in the same time we construct the transition from q to the new state.

Let k be the biggest constant used in the expressions of the timing constraints of $\mathcal{A}$. For each clock $c_i \in \mathcal{C}$, let S_i be the number of distinct Set events associated to c_i, and let $p_{i,j}$ be the number of constants in each of these Set events, for $j = 1, ..., S_i$. Let then $p_i = \sum_{j=1,...,S_i} p_{i,j}$ and $p = \max_{i=1,...,Nc} p_i$.

Proposition 7.5 *The number of states of $\mathcal{B}$ is bounded by $|\mathcal{L}|p^{|\mathcal{C}|}2^{k(2k+1)|\mathcal{C}|^2}$.*

Proposition 7.5 gives an upper bound of the number of states of a SEA w.r.t. its input parameters. This bound is a theoretical one and is far from being reached in practice. The same observation apply to the computational complexity given by Propositions 7.1, 7.2, 7.3 and 7.4.

As we can seen in the above results, the computational complexity and number of states (need of memory) of $SetExp$ increases with the number of clocks. It is therefore important to use in the AT the minimum necessary clocks variables. A method of minimization of the number of clocks of a AT can be found in [33]. Generally in practice, the number of clocks of a TA is not very big and thus the computational complexity of $SetExp$ is not very hard.

7.3 Simplification of a SEA

We present in this section a way of simplification of $SetExp$, useful in some context. For this, we shall classify into two types the transitions of type 3: *conditioned transitions of type 3* and *unconditioned transitions of type 3*.

Definition 7.5 *:* **conditioned and unconditioned transition**
Let $\mathcal{B} = (Q, \Gamma, \delta, \Pi, q^0, Q_m)$ be a SEA obtained from the TA $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}, \mathcal{T}, l_0, \mathcal{L}_m)$, i.e. $\mathcal{B} = SetExp(\mathcal{A})$. Let $\mathcal{A}^I = StepOne(\mathcal{A}) = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}^I, \mathcal{T}^I, l_0, \mathcal{L}_m)$ and $t \in \delta$ be a transition of type 3 of $\mathcal{B}$ labeled by the event $\gamma = (\sigma, \mathcal{E})$, with $\sigma \in \Gamma_2$ and $\mathcal{E} \in \Gamma_1$. Let $t^I \in \mathcal{T}^I$ be the transition corresponding to t in the IA $\mathcal{A}^I$, i.e. t has been constructed from t^I. The transition t is said **conditioned** iff *at least one of the following two conditions is satisfied:*

- *at least one* Exp-Condition *of the guard of t^I or the invariant of its origin location has one of the following three forms: "$= Exp(c, k)$", "$\leq Exp(c, k)$" or "$\geq Exp(c, k)$", i.e. the guard or the invariant allows the simultaneity of the occurrence of an event and an expiration.*
- *t^I resets a non empty subset of the clocks of $\mathcal{C}$, i.e., there are at least two clocks c_i and c_j such that c_i is reset and c_j is not reset.*

In the other hand, t is said **unconditioned** iff *it is not conditioned, i.e. the following two conditions are satisfied:*

- *the guard of t^I and the invariant of its origin location contain only* Exp-Condition *in the form "$< Exp(c, k)$" or "$> Exp(c, k)$", if any, and*
- *t^I resets either no clock or all the clocks of $\mathcal{C}$.*

According to Definition 7.5, a transition t of type 3 is conditioned if it satisfies at least one of the two conditions: (i) t is enabled because of a constraint in the execution of the TA which either requires the simultaneity of the events of t (case where the guard of t^I contains an *Exp-Condition* in the form "$= Exp(c,k)$"), or allows this simultaneity (case where the guard of t^I or the invariant of its origin location contains an *Exp-Condition* "$\leq Exp(c,k)$" or "$\geq Exp(c,k)$") and (ii) the effect of t on the values of the clocks is not uniform, i.e. some clocks are reset by t whereas others are not. As example, on the SEA of Figure 6, the transition between S2 and S5, labeled by $\beta Exp(c1,3)$, is conditioned because its corresponding transition in the IA of Figure 4 (transition between locations W and I) contains the *Exp-Condition* "$\geq Exp(c1,3)$". In the same way, the transition between S2 and S6, labeled by $\lambda Set(c2;50)Exp(c1,3)$ is conditioned because it resets only the clock $c2$ and not $c1$.

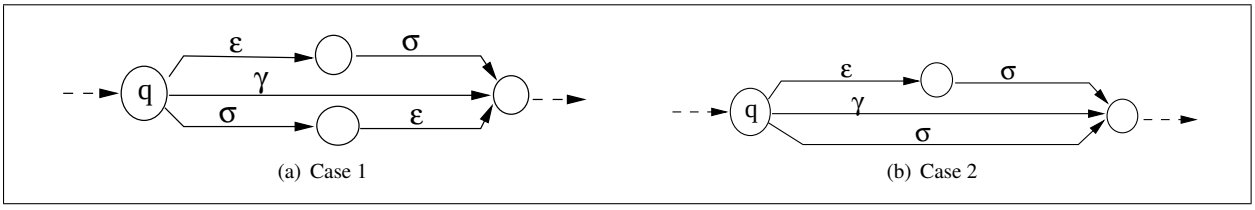
In the other hand, a transition of type 3 is unconditioned if (i) the simultaneity of its events is not imposed by a timing constraint (no equality or non strict inequality in the guard or invariant), and (ii) the transition acts uniformly on the clocks, i.e. it resets all of them or none of them. On the SEA of Figure 6, there are no unconditioned transitions.

Proposition 7.6 *Let $\mathcal{B}$ be a SEA, q a state of $\mathcal{B}$, $\gamma = (\mathcal{E},\sigma) \in \Gamma_3$ with $\sigma = (\rho,\mathcal{S}) \in \Gamma_2$ such that $(q,\gamma)!$ and $t = (q,\gamma,q')$ is unconditioned.*

- *If $\gamma\rfloor_{Set} = \emptyset$, then $(q,\mathcal{E}.\sigma)!$, $(q,\sigma.\mathcal{E})!$ and we have $(q,\gamma) = (q,\mathcal{E}.\sigma) = (q,\sigma.\mathcal{E})$.*
- *If $\gamma\rfloor_{Set} \neq \emptyset$, then $(q,\mathcal{E}.\sigma)!$, $(q,\sigma)!$ and we have $(q,\gamma) = (q,\mathcal{E}.\sigma) = (q,\sigma)$.*

Proposition 7.6 presents the two situations when an unconditioned transition is enabled. These two situations are illustrated on Figure 7. The first case presents the situation when the transition resets no clock, and is illustrated on Figure 7(a), in which the destination state of the unconditioned transition can be reached by the execution of the sequence $\sigma.\mathcal{E}$ or $\mathcal{E}.\sigma$ from the same origin state.

The second case presents the situation when the unconditioned transition resets all the clocks (recall that an unconditioned transition resets no clock or all the clocks of $\mathcal{C}$), and is illustrated on Figure 7(b), in which the destination state of the unconditioned transition can be reached by the execution of σ or the sequence $\mathcal{E}.\sigma$. This result is justified by the fact that since all the clocks are reset with σ, the occurrence of the expirations in the event has no influence on the future, and the construction of the destination state of the transition does not depend on the occurrence of these expirations, and for this reason this state is the same in the following three cases of execution: $\mathcal{E}$ followed by σ, σ simultaneous with $\mathcal{E}$, and σ.



(a) Case 1 (b) Case 2

Figure 7: Illustration of the behaviors for an unconditioned type 3 transition

Proposition 7.6 can be intuitively justified by the fact that if $\gamma \in \Gamma_3$ is unconditioned and enabled in q, then σ is not constrained by the expirations in $\mathcal{E}$, and thus the occurrence of σ can arrive indifferently before or after the one of $\mathcal{E}$, and this implies that σ and $\mathcal{E}$ are enabled at the same state. In addition, the same state is reached after the occurrence of γ and the execution of $\mathcal{E}.\sigma$ and $\sigma.\mathcal{E}$ (respectively σ) for case 1 (respectively case 2), because the values of all the clocks are uniformly affected during the execution of these events and sequences.

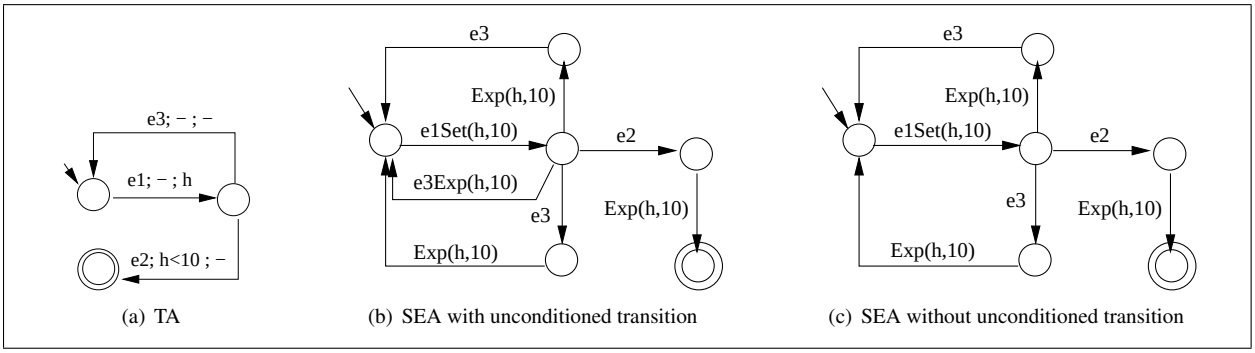
Definition 7.6 *:* **equivalent occurrence of sequence of events**
Let $\mathcal{B}$ be a SEA over the alphabet Γ, q a state of $\mathcal{B}$, $\gamma \in \Gamma_3$, $\sigma \in \Gamma_2$ and $\mathcal{E} \in \Gamma_1$.

- *In q, the occurrence of γ is* equivalent *to the one of σ followed by $\mathcal{E}$* iff*: (i) $\gamma\rfloor_{Exp} = \mathcal{E}$; (ii) $\gamma\rfloor_{Set} = \sigma\rfloor_{Set}$; (iii) $\gamma\rfloor_{\Sigma} = \sigma\rfloor_{\Sigma}$; (iv) $(q,\gamma)!$ and $(q,\sigma.\mathcal{E})!$; (v) $(q,\gamma) = (q,\sigma.\mathcal{E})$.*
- *In q, the occurrence of γ is* equivalent *to the one of $\mathcal{E}$ followed by σ* iff*: (i) $\gamma\rfloor_{Exp} = \mathcal{E}$; (ii) $\gamma\rfloor_{Set} = \sigma\rfloor_{Set}$; (iii) $\gamma\rfloor_{\Sigma} = \sigma\rfloor_{\Sigma}$; (iv) $(q,\gamma)!$ and $(q,\mathcal{E}.\sigma)!$; (v) $(q,\gamma) = (q,\mathcal{E}.\sigma)$.*

- *In q, the occurrence of γ is* equivalent *to the one of σ* iff*: (i) $\gamma\rfloor_{Set} = \sigma\rfloor_{Set}$; (ii) $\gamma\rfloor_{\Sigma} = \sigma\rfloor_{\Sigma}$; (iii) $(q,\gamma)!$ and $(q,\sigma)!$; (iv) $(q,\gamma) = (q,\sigma)$.*

If the occurrence of two events or sequences of events is equivalent, we shall say that the two transitions or sequences of transitions labeled by these events are equivalent.

Let $\mathcal{B}$ be a SEA and t an unconditioned type 3 transition of $\mathcal{B}$, labeled by the event $\gamma = (\sigma, \mathcal{E})$. If we delete all the traces of $\mathcal{B}$ containing t, all the others traces of $\mathcal{B}$ not containing t remain unchanged, because the deletion of t does not imply the deletion of its destination state because the later remains reachable. Furthermore, if s and s' are such that the sequence $s.\gamma.s'$ is accepted by $\mathcal{B}$, then the sequence $s.\mathcal{E}.\sigma.s'$ and $s.\sigma.\mathcal{E}.s'$ for case 1 (respectively $s.\sigma.s'$ for case 2) is also accepted by $\mathcal{B}$. We can then deduce that for any unconditioned transition t, there exists two transitions or sequences of (two) transitions that are equivalent (following Definition 7.6) to t. By removing t, all the states of the SEA remain reachable and this has no influence on its remaining behavior. If we consider the TA of Figure 8(a), we obtain normally the SEA of Figure 8(b) by applying $SetExp$. If we do not construct the unconditioned transitions, we obtain the SEA of Figure 8(c), which has the same number of states as the original one, but without the transition on the event $e3Exp(h, 10)$.



(a) TA (b) SEA with unconditioned transition (c) SEA without unconditioned transition

Figure 8: Illustration of removing of unconditioned transitions

Remarque 7.1 *If $\mathcal{B}$ is a SEA and $\mathcal{B}'$ the SEA obtained from $\mathcal{B}$ by removing the unconditioned transitions, then $\mathcal{B}$ and $\mathcal{B}'$ are not equivalent because they do not accept the same timed language, due to the fact that every timed trace containing the timed event corresponding to the event modeling the removed unconditioned transition become non accepted by the SEA.*

Depending on the study for which a SEA is used, the removing of unconditioned transition can has no influence on the final results. This is particularly true in studies where for an event $(\sigma, \mathcal{E})$ labeling an unconditioned transition, the simultaneous occurrence of σ and $\mathcal{E}$ can be considered as the occurrence of σ (respectively $\mathcal{E}$) followed by the one of $\mathcal{E}$ (respectively σ). This is the case especially in real implementation where events occurrence detection is not done exactly at their occurrence time. In such situation, we can simplify $SetExp$ by not constructing the unconditioned transitions. This simplification allows to decrease the number of transitions of the SEA and simplify its implementation.

8 Application of $SetExp$ in supervisory control of RTDES

In this section, we shall first present first the principle of the supervisory control of a RTDES modeled by a TA, and after this we shall present why and how $SetExp$ can be used to solve supervisory control problems.

8.1 Supervisory control of RTDES

The objective of supervisory control is to design a supervisor $\mathcal{S}up$ which interacts with a given DES P called *plant* in order to restrict its behavior so that it respects a given specification S. When P and S are not subject to any timing constraint and are described by two FSA over the same alphabet, a method has been proposed in order to automatically

synthesize $\mathcal{S}up$ [3]. Let $\mathcal{S}up/\mathcal{P}$ be the behavior of P under supervision of $\mathcal{S}up$. When P and S are subject of timing constraints and are modeled respectively by two TA $\mathcal{P}$ and $\mathcal{K}$ over the same alphabet Σ, the objective of supervisory control consists in synthesizing a supervisor $\mathcal{S}up$ which, by interacting with $\mathcal{P}$, ensures the following two constraints:

$\mathcal{S}up/\mathcal{P}$ **respects S:** i.e. all timed traces accepted by $\mathcal{S}up/\mathcal{P}$ are also accepted by $\mathcal{K}$.

$\mathcal{S}up/\mathcal{P}$ **is non blocking:** intuitively, in its evolution, $\mathcal{S}up/\mathcal{P}$ always have the possibility to reach a marked location of $\mathcal{P}$. More formally, every prefix of a timed trace of $\mathcal{S}up/\mathcal{P}$ is also a prefix of a timed marked trace of $\mathcal{S}up/\mathcal{P}$.

$\mathcal{S}up$ can be viewed as a module which observes the evolution of $\mathcal{P}$, and forbids and forces some specific events in order to ensure the respect of the above two constraints. As $\mathcal{S}up$ must forbid events, we shall use the notion of *controllable event* which means that $\mathcal{S}up$ has the ability to forbid it, otherwise the event is classified as *uncontrollable* [3]. In the same way, as $\mathcal{S}up$ must force some events, we shall use the notion of *forcible event* which means that $\mathcal{S}up$ has the ability to force the occurrence of the event when this one is enabled, otherwise the event is said unforcible [34].

Figure 9 illustrates an architecture for supervisory control of P by $\mathcal{S}up$. The interactions between the plant (P) and the environment (E) are modeled by the TA $\mathcal{P}$. $\mathcal{S}up$ forbids or forces only the outputs of P, because we suppose that $\mathcal{S}up$ has no control on E, and thus has no control on the input of P, generated by E.

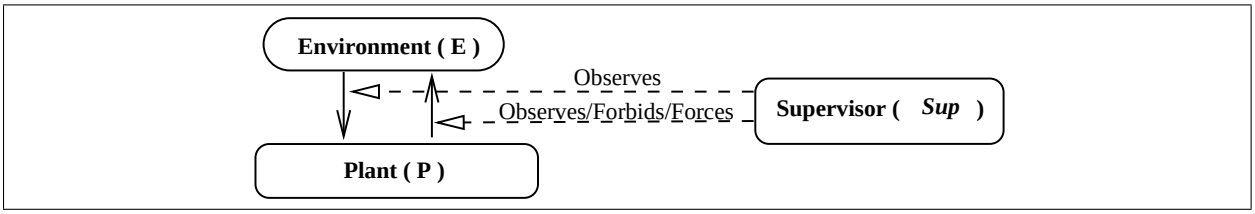


Figure 9: Architecture of Supervisory control of a RTDES

Recall that the state space of a TA is infinite. However, in order to solve the supervisory control problem, i.e. find $\mathcal{S}up$, it is necessary to have a finite representation of the state space of $\mathcal{P}$ and $\mathcal{K}$. To obtain this finite representation of the state space, we must transform the TA $\mathcal{P}$ and $\mathcal{K}$ into FSA. In [10, 35], the authors use the transformation into region automata (RA) to solve the supervisory control problem. But, in our knowledge, this transformation into RA, and also other transformations found in the literature, are subject to state space explosion, because among others aspects, the state space increases with the magnitude of constants used in the timing constraints. We propose in this study the use of the transformation $SetExp : TA \mapsto SEA$ rather than $TA \mapsto RA$, because in practice, following all the examples we have studied, the state space of a SEA does not increase with the magnitude of the constants used in the timing constraints of the TA.

8.2 Supervisory control of a RTDES based on $SetExp$

Given two TA $\mathcal{P}$ and $\mathcal{K}$ modeling the plant and the specification respectively, as we cannot synthesize straightly $\mathcal{S}up$ from $\mathcal{P}$ and $\mathcal{K}$, we shall use $SetExp$ in order to bring back the synthesis in the field of SEA, by transforming the TA $\mathcal{P}$ and $\mathcal{K}$ into SEA $\mathcal{P}_s$ and $\mathcal{K}_s$ over the same alphabet Γ. Then, we compute the supervisor $\mathcal{S}upSEA$ from $\mathcal{P}_s$ and $\mathcal{K}_s$. In the sequel $\mathcal{S}upSEA/\mathcal{P}_s$ denotes the plant under the control of $\mathcal{S}upSEA$. $\mathcal{S}upSEA/\mathcal{P}_s$ must ensures the following constraints:

$\mathcal{S}upSEA/\mathcal{P}_s$ **respects** $\mathcal{K}_s$**:** i.e. every trace accepted by $\mathcal{S}upSEA/\mathcal{P}_s$ is also accepted by $\mathcal{K}_s$.

$\mathcal{S}upSEA/\mathcal{P}_s$ **is non blocking:** in its evolution, $\mathcal{S}upSEA/\mathcal{P}_s$ always has the possibility to reach a marked state of $\mathcal{P}_s$. Formally, every prefix of a trace of $\mathcal{S}upSEA/\mathcal{P}_s$ is also a prefix of a marked trace of $\mathcal{S}upSEA/\mathcal{P}_s$.

On Figure 10, we propose an architecture for realizing supervisory control in the field of SEA. Compared to Figure 9, $\mathcal{S}up$ is formed of two modules: a *Clock-Handler* (CH) and a supervisor of SEA $\mathcal{S}upSEA$, which have the following roles:

Clock-Handler (CH) : observes the interactions between P and E, generates the Set and Exp events and sends them to $\mathcal{S}upSEA$. CH can be considered as a module which sets timers at given moments determined by the interactions between P and E, and send these sets (Set events) as well as their corresponding expirations (Exp events) to $\mathcal{S}upSEA$.

Supervisor ($\mathcal{S}upSEA$**)** : receives Set and Exp events from CH, observes the interactions between P and E, and forbids/forces controllable/forcible events when this is necessary for the respect of the specification $\mathcal{K}_s$.

Now, we shall show why the above supervisory control architecture is suited for $SetExp$. Recall that the objective of the supervisory control is to synthesize a model of $\mathcal{S}up$ which ensures that the plant P respects the specification S, where P and S are modeled by two TA $\mathcal{P}$ and $\mathcal{K}$. The transformation $SetExp$ allows us to reformulate this objective in another way as follows:

- the plant under supervision is the system formed of P and CH, denoted ⟨P,CH⟩, which is modeled by the SEA $SetExp(\mathcal{P})$;
- the specification to respect is defined by the SEA $SetExp(\mathcal{K})$;
- the objective is to synthesize a model of $\mathcal{S}upSEA$ which ensures that the system ⟨P,CH⟩ respects the specification modeled by $SetExp(\mathcal{K})$.

It can be showed that this new objective guarantees the initial one. In other words:

If $\mathcal{S}upSEA$ forces ⟨P,CH⟩ to respect $SetExp(\mathcal{K})$,
Then $\mathcal{S}up$ forces P to respect $\mathcal{K}$.

The advantage of this transformation is that we have transformed a problem of supervisory control of RTDES under the form of a problem of supervisory control of non-real time DES. The system ⟨P,CH⟩ can be seen as a non-real time DES, owing to the fact that its behavior is only defined by the sequences of events of P and CH. The use of Set and Exp events has allowed to represent the timing constraints under the form of order constraints of events. This approach has been used to solve centralized and modular supervisory control problem respectively in [25] and [26]. These two papers are complementary to this one, because they illustrate the application of $SetExp$ to supervisory control, whereas here we present the internal mechanism of $SetExp$, i.e. how $SetExp$ is realized.

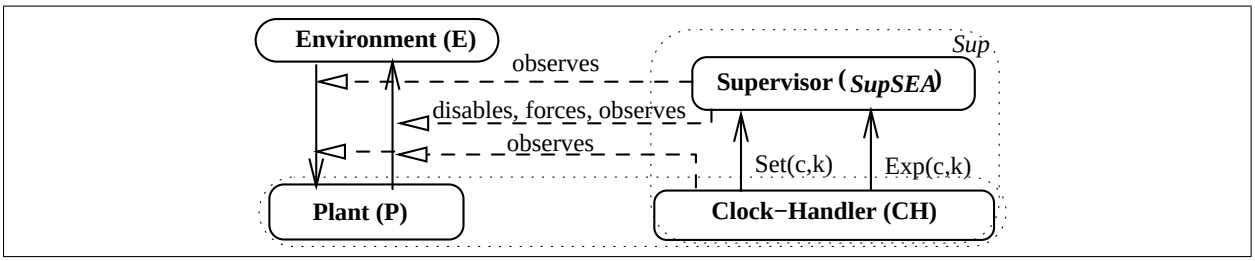


Figure 10: Control architecture with SEA

9 Conclusion

In this paper, we have presented a method of transformation of timed automata (TA) into finite state automata (FSA) and illustrated its application in supervisory control of real-time discrete events systems (RTDES). The method of transformation, called $SetExp$, allows to transform a TA into a particular FSA called Set-Exp Automaton (SEA), by adding to its alphabet two new events called Set and Exp. These events, which represent respectively resets and programming of clocks and expirations of clocks, represent the progression of time and allow to express timing constraints as order constraints on events. This representation of time progression by Set and Exp implies that the value of each clock is considered only when it reaches a value used to express the timing constraints of the TA. Therefore, the transition from a state to another state by the passage of time can occur only when a clock variable reaches a considered value, unlike the region automata (RA) transformation, for which the value of each clock is considered for each unit of time. As a consequence of this, the state space of a SEA does not increase in practice with the magnitudes of constants used in in the timing constraints of the TA, and this constitutes an advantage of $SetExp$ in comparison to other transformation methods such as RA. We have illustrated the application of $SetExp$ in supervisory control of RTDES, and given an architecture its realization.

In comparison to other transformation methods, such as RA and its minimization methods, which is more applicable in model checking, $SetExp$ is suited for supervisory control and conformance testing of RTDES. This paper is an

extension and improvement of previous versions of $SetExp$ presented in [30, 31]. [31] essentially extends [30] to TA with location invariants. The present paper extends [31], which is written in French, by providing detailed explanations of algorithms, proofs of propositions and theorem, and a method for reducing the number of transitions of SEA.

We have recently developed a software tool called SEAGenerator that implement $SetExp$. In a near future, we project to: (i) use SEAGenerator to develop concrete applications of $SetExp$ for the centralized and modular supervisory control method of [12, 31]; (ii) develop a decentralized supervisory control method based on $SetExp$ and (ii) explore the use of $SetExp$ for other RTDES studies such as fault diagnosis.

References

[1] E. C. Yamalidou, E. P. Patsidou, and J. C. Kantor. Modelling discrete-event dynamical systems for chemical process control - a survey of several new techniques. *Computers and Chemical Engineering*, 14(3):281–299, 1990.

[2] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM journal control and optimization*, 25(1):206–230, January 1987.

[3] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings IEEE*, 77:81–98, January 1989.

[4] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE transactions on automatic control*, 40(9):1555–1575, September 1995.

[5] J.S. Ostroff and W.M. Wonham. A framework for real-time discrete event control. *IEEE Transaction on Automatic Control*, 35(4):386–397, 1990.

[6] J.S. Ostroff. Deciding properties of timed transition models. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):170–183, April 1990.

[7] B.A. Brandin and W.M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–342, 1994.

[8] T.-J Ho. A method for the modular synthesis of controllers for timed discrete-event systems. *International journal of control*, 76(5):520–535, 2003.

[9] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[10] H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. In *Proc. of 30th IEEE Conference on Decision and Control*, pages 1527–1528, Brighton, England, December 1991.

[11] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *IFAC Symposium on System Structure and Control*, pages 469–474, 1998.

[12] A. Khoumsi. A supervisory control method for ensuring the conformance of real-time discrete event systems. *Journal of Discrete Event Dynamic Systems*, 15(4):397–431, December 2005.

[13] R. Alur and T.A. Henzinger. Logic and models of real time : A survey. *J.W. de Bakker, C. Huizing, W.-P. de Rover, and G. Rozenberg, editors, Real-Time: Theory in Practice. REX Workshop Proceedings, LNCS*, 600:74–106, 1991.

[14] J.E. Hopcroft and Ullman J.D. *Introduction to Automata Theory, Language and Computation*. Addisson-Wesley, 1979.

[15] Carroll J. and J.D. Long. *Theory of Finite Automata*. Printice-Hall, 1989.

[16] R. Alur. Timed automata. In *11th International Conference on Computer Aided Verification*, volume 1633, pages 8–22, Spring-Verlag, 1999. Lecture Notes in Computer Science.

[17] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transitions systems. In *CONCUR*, pages 340–354, 1992.

[18] I. Kang and I. Lee. State minimization for concurrent system analysis based on state space exploration. In *Proceedings of Conference on Computer Assurance*, pages 123–134, 1994.

[19] I. Kang and I. Lee. An efficient state space generation for analysis of real-time systems. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA'96)*, 1996.

[20] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstraction bisimulations. *Formal methods in systems design*, 18(1):25–68, January 2001.

[21] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In *Proceedings, Fifth International Conference on Computer Aided Verification, volume 697 of LNCS, pages 210-224*, Springer-Verlag, 1993.

[22] M. P. Spathopoulos. On a simplified untiming procedure for supervisory control of timed automata when the time increases strictly monotonically. *International Journal of Control*, 76(17):17391748, November 2003.

[23] A. Khoumsi and M. Nourelfath. An efficient method for the supervisory control of dense real-time discrete event systems. In *Proc. 8th Intern. Conf. on Real-Time Computing Systems (RTCSA)*, Tokyo, Japan, March 2002.

[24] A. Khoumsi. Supervisory control of dense real-time discrete event systems with partial observation. In *6th IEEE International Workshop on Discrete Event Systems (WODES)*, Zaragoza, Spain, October 2002.

[25] A. Khoumsi, L. Ouédraogo, and M. Nourelfath. Supervisory control of real-time discrete event systems modeled by timed

automata with invariants. In *IASTED International Conference on Intelligent Systems and Control (ISC)*, Cambridge, Massachussets, USA, October 2005. http://www.gel.usherbrooke.ca/khoumsi/Research/Public/IASTED05.pdf.

[26] L. Ouédraogo, M. Nourelfath, and A. Khoumsi. A new method for centralized and modular supervisory control of real-time discrete event systems. In *IEEE International Workshop on Discrete Event Systems (WODES)*, Ann Arbor, Michigan, USA, July 2006. http://www.gel.usherbrooke.ca/khoumsi/Research/Public/WODES06.pdf.

[27] A. Khoumsi. A method for testing the conformance of real-time systems. In *IEEE Internatinal Symposium on Formal Technics in Real-Time and Fault-Tolerent Systems (FTRTFT)*, Oldenburg, Germany, September 2002.

[28] A. Khoumsi. Complete test graph generation for symbolic real-time systems. In *Brasilian Symposium on Formal Methods (SBMF)*, Recife, Brazil, November 2004. http://www.gel.usherbrooke.ca/khoumsi/Research/Public/SBMF04-Test.pdf.

[29] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software-Concepts and Tools*, 17(3):103–120, 1996.

[30] A. Khoumsi and L. Ouédraogo. A new method for transforming timed automata. In *Proc. Brazilian Symposium on Formal Methods (SBMF)*, Recife, Brazil, November 2004.
http://www.gel.usherb.ca/khoumsi/Research/Public/SBMF04-SetExp.pdf.

[31] L. Ouédraogo, A. Khoumsi, and M. Nourelfath. Méthode de transformation d'automates temporisés avec invariants de localités. In *Conférence francophone de Modélisation et Simulation (MOSIM)*, Rabat, Morroco, April 2006. http://www.gel.usherbrooke.ca/khoumsi/Research/Public/MOSIM06.pdf.

[32] T.A Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Journal of Information and Computation*, 111(2):193–244, 1994.

[33] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proceedings, 17th IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, 1996.

[34] C.H. Golaszewski and P.J. Ramadge. Control of discrete event processes with forced events. In *Proceedings 26th IEEE Conference on Decision and Control*, pages 247–251, Los Angeles, CA, December 1987.

[35] A. Gouin and J.L Ferrier. Commande supervisée de systèmes à événements discrets temporisés : synthèse basée sur les automates de τ-regions. In *conférence internationale francophone d'automatique*, Lille, France, July 2000.

Appendix

A Proof of Proposition 5.1

Let q_1 and q_2 be two states such that $L_{q_1} = L_{q_2}$ and where all the clocks are inactive, i.e. no expirations of any clock is expected. As q_1 and q_2 have the same location, it remains to show that they have the same future. Under the conditions of Proposition 5.1, the following items are satisfied:

1. there are no events of type 1 or type 3 enabled at q_1 and q_2;
2. any event of type 2 enabled at q_1 is also enabled at q_2;
3. the destination states of two transitions labeled by the same event and enabled at q_1 and q_2 are either equal (i.e. the same state) or equivalent.

The justification of the above item 1 is evident, following the fact that all clocks are inactive.

For item 2: let γ be an event of type 2 enabled at q_1, and let us show that γ is also enabled at q_2. For this, we must verify that the conditions of Proposition 6.2 have the same effect in the two states.

- For item 1 of Proposition 6.2: we have $L_{q_1} = L_{q_2}$ and as $(q_1, \gamma)!$, then $\gamma \in OUT(L_{q_2}) = OUT(L_{q_1})$.
- For item 2 of Proposition 6.2: as all clocks are inactive, the condition $\exists k' \leq k$ cannot be satisfied in q_1 and q_2, then this condition has the same effect in q_1 and q_2.
- For item 3 of Proposition 6.2: as γ is enabled at q_1, then there is no equality in the guard of T and thus, this condition has no effect in q_2 too.
- For item 4 of Proposition 6.2: γ enabled at q_1 implies $\exists k' \geq k$ such that $C_{q_1}(c_i) = k' < c_i$. The same condition is satisfied in q_2, because c_i is also inactive at q_2, and as the states have the same location implying that the clocks are subject to the same resets in the two states, the *Clock-Conds* of the two states are either equal (the states are perfectly equal in this case), or one of the state is the initial state and in this case, γ does not gave a guard and the condition does not apply in the two states.

We can then conclude that the influence of the 4 item of Proposition 6.2 on the enabling of γ from q_1 and q_2 is the same, and thus if γ is enabled at q_1, then it is enabled at q_2.

For item 3: this item follows from the procedure of construction of the destination state of the transition on γ given in Section 6.2. For each event γ enabled at q_1 and q_2, let $r_1 = (q_1, \gamma)$ and $r_2 = (q_2, \gamma)$. As all the clocks are inactive, we have that:

(i) clearly, $L_{r_1} = L_{r_2}$ (in the TA, execution of the same event leads to one location);

(ii) the changes in the *Clock-Conds* of r_1 and r_2 are determined by the Set events of γ, and then these changed *Clock-Conds* are the same in r_1 and r_2;

(iii) the changes in the Δ*Clock-Conds* of r_1 and r_2 are determined by the Set events of γ, and then these changed Δ*Clock-Conds* are the same in r_1 and r_2;

(iv) (i), (ii) and (iii) imply that either $r_1 = r_2$ or they satisfy Proposition 5.1 or Proposition 5.2 and thus are equivalent.

B Proof of Proposition 5.2

Let q_1 and q_2 be two different states satisfying the conditions of Proposition 5.2, i.e. q_1 differs from q_2 by one or many Δ*Clock-Conds* having at least one inactive clock. As q_1 and q_2 have the same location, it remains to show that they have the same future. For this, we shall use the following lemma:

Lemma B.1 *If two states q_1 and q_2 have the same location, then they have the same future* iff *any event enabled at q_1 is also enabled at q_2.*

The proof of Lemma B.1 comes straightforwardly from the fact that as q_1 and q_2 have the same location, they have the "same future" relatively to the original TA, and thus the only thing that can differentiate their future in the SEA is that at least one event is enabled in one of the state and not in the other. This can arrive if the *Clock-Conds* or the

Δ*Clock-Conds* of the two states are different, dues to the fact that they have been reached by the execution of two traces that differ by their Exp events.

Let γ be an event enabled at q_1. We must show that γ is also enabled at q_2 to conclude that q_1 and q_2 have the same future and thus, are location equivalent.

If $\gamma \in \Gamma_2$, γ is enabled at q_2 because its enabling depends only on the location and the *Clock-Cond* of q_2, and these ones are the same as those of q_1. If $\gamma \in \Gamma_1$, γ is enabled at q_2 if the effect of the Δ*Clock-Conds* of q_2 on its enabling is the same as those of q_1. If a clock is inactive, the Δ*Clock-Conds* using it have no effect on the enabling of event of type 1, because these Δ*Clock-Conds* does not influence the solution of the system of inequations $\mathcal{X}_{q,\mathcal{E}}$ defined in Section 6.1. This comes from the fact that these Δ*Clock-Conds* has no interference with the second part of $\mathcal{X}_{q,\mathcal{E}}$, which clearly apply to active clocks, i.e. clocks which have programmed expirations. As the Δ*Clock-Conds* of q_1 and q_2 differ only on those using inactive clocks, then the effect of the Δ*Clock-Conds* of q_2 is the same as those of q_1 and thus, γ is enabled at q_2. If $\gamma \in \Gamma_3$, it is enabled in q_2 because its enabling conditions is the composition of those of event of type 1 and event of type 2, and this is the same as showed above. We can then conclude that any event enabled at q_1 is also enabled at q_2, and following Lemma B.1, q_1 and q_2 have the same future.

C Proof of Proposition 5.3

From the definition of equivalent states and equivalent SEA, Proposition 5.3 is evident and can be proved straightforwardly. Let q_1 and q_2 be two states of $\mathcal{B}$ such that $q_1 \simeq q_2$. In $\mathcal{B}_\partial$, q_1 and q_2 are merged, and suppose that q_1 is conserved in $\mathcal{B}_\partial$. Let $s_1 = s'_1 s \in L(\mathcal{B})$ and $s_2 = s'_2 s \in L(\mathcal{B})$, where s'_1 and s'_2 are prefix of $L(\mathcal{B})$ and s a suffix of $L(\mathcal{B})$, and such that $q_1 = (q_,^0 s'_1)$ and $q_2 = (q_,^0 s'_2)$ are location equivalent. In $\mathcal{B}_\partial$ (q_2 is removed and the transitions having it as destination state are redirected on q_1), s_1 and s_2 remain in $L(B)$, because the only thing that change is that $(q_,^0 s'_2) = q_1$ instead of q_2. Also, by definition, location equivalent states have the same status with regard to marking. We can then conclude that $\mathcal{B} \simeq \mathcal{B}_\partial$.

D Proof of Theorem 1

We have defined the suite $\mathcal{B}_{i+1} = \Omega(\mathcal{B}_i)$, for $i \geq 0$. Let $Q_i, \Gamma_i, \delta_i, \Pi_i$ and Q_m^i be such that $\mathcal{B}_i = (Q_i, \Gamma_i, \delta_i, \Pi_i, q^0, Q_m^i)$. We have, by definition of the operator Ω (see Section 5.2): $Q_{i+1} = Q_i \cup \Delta Q_i$, $\Gamma_{i+1} = \Gamma_i \cup \Delta\Gamma_i$, $\delta_{i+1} = \delta_i \cup \Delta\delta_i$, $Q_m^{i+1} = Q_m^i \cup \Delta Q_m^i$, where:
$\Delta Q_i = \{(q,\gamma) \,|\, q \in Q_i, \gamma \in (2^{EXP(q)} \times (OUT(L_q) \cup \{\sharp\})) \setminus \{\emptyset, \sharp\}, (q,\gamma)!\}$;
$\Delta\Gamma_i = \{\gamma \,|\, \exists (q,\gamma,r) \in \delta_{i+1}\}$;
$\Delta\delta_i = \{(q,\gamma,r) \,|\, q \in Q_i, (q,\gamma)!, r = (q,\gamma) \in Q_{i+1}\}$;
$\Delta Q_m^i = \{q \in \Delta Q_i \,|\, (L_q \in \mathcal{L}_m) \wedge (C_q \approx \emptyset)\}$.
We shall also use the notation $|U|$ for the cardinal (i.e., number of elements) of a set U. In order to prove Theorem 1, let us prove the following five lemmas:

Lemma D.1 $\forall i \geq 0 : Q_i \subseteq Q_{i+1}$.

Lemma D.2 $\forall p \geq 0 : ((Q_p = Q_{p+1}) \Rightarrow (\mathcal{B}_{p+2} = \mathcal{B}_{p+1}))$.

Lemma D.3 $\forall p \geq 0 : ((Q_p = Q_{p+1}) \Rightarrow (\forall n > p : \mathcal{B}_n = \mathcal{B}_{p+1}))$.

Lemma D.4 $\exists K$ *finite such that* $\forall i \geq 0 : |Q_i| \leq K$.

Lemma D.5 $\lim_{i\to\infty} |Q_i|$ *exists and is finite.*

D.1 Proof of Lemma D.1

From the fact that $\forall i \geq 0 : Q_{i+1} = Q_i \cup \Delta Q_i$, we deduce straightforwardly that $Q_i \subseteq Q_{i+1}$.

D.2 Proof of Lemma D.2

(i) By definition: $Q_{p+1} = Q_p \cup \Delta Q_p$ and $Q_{p+2} = Q_{p+1} \cup \Delta Q_{p+1}$.
(ii) By definition: $\Delta Q_p = \{(q,\gamma) \,|\, q \in Q_p, \gamma \in (2^{EXP(q)} \times (OUT(L_q) \cup \{\sharp\})) \setminus \{\emptyset, \sharp\}, (q,\gamma)!\}$,
$\Delta Q_{p+1} = \{(q,\gamma) \,|\, q \in Q_{p+1}, \gamma \in (2^{EXP(q)} \times (OUT(L_q) \cup \{\sharp\})) \setminus \{\emptyset, \sharp\}, (q,\gamma)!\}$.
(iii) Item (ii) implies that if $Q_p = Q_{p+1}$, then $\Delta Q_p = \Delta Q_{p+1}$.
(iv) Items (i) and (iii) imply that if $Q_p = Q_{p+1}$, then $Q_{p+2} = Q_{p+1} = Q_p$.
(v) By definition: $\delta_{p+1} = \delta_p \cup \Delta\delta_p$ and $\delta_{p+2} = \delta_{p+1} \cup \Delta\delta_{p+1}$.
(vi) By definition : $\Delta\delta_p = \{(q,\gamma,r) \,|\, q \in Q_p, (q,\gamma)!, r = (q,\gamma) \in Q_{p+1}\}$ and
$\Delta\delta_{p+1} = \{(q,\gamma,r) \,|\, q \in Q_{p+1}, (q,\gamma)!, r = (q,\gamma) \in Q_{p+2}\}$.
(vii) Items (iv) and (vi) imply that if $Q_p = Q_{p+1}$, then $\Delta\delta_p = \Delta\delta_{p+1}$.
(viii) Items (v) and (vii) imply that if $Q_p = Q_{p+1}$, then $\Delta\delta_{p+2} = \Delta\delta_{p+1}$.
(ix) By definition: $\Gamma_{p+1} = \Gamma_p \cup \Delta\Gamma_p$, and $\Gamma_{p+2} = \Gamma_{p+1} \cup \Delta\Gamma_{p+1}$.
(x) By definition: $\Delta\Gamma_p = \{\gamma \,|\, \exists(q,\gamma,r) \in \delta_{p+1}\}$ and $\Delta\Gamma_{p+1} = \{\gamma \,|\, \exists(q,\gamma,r) \in \delta_{p+2}\}$.
(xi) Items (viii) and (x) imply that if $Q_p = Q_{p+1}$, then $\Delta\Gamma_p = \Delta\Gamma_{p+1}$.
(xii) Items (ix) and (xi) imply that if $Q_p = Q_{p+1}$, then $\Gamma_{p+2} = \Gamma_{p+1}$.
(xiii) Items (iv), (viii) and (xii) imply that if $Q_p = Q_{p+1}$, then: $\mathcal{B}_{p+2} = \mathcal{B}_{p+1}$.

D.3 Proof of Lemma D.3

(i) Let us consider a $p \geq 0$ such that $Q_p = Q_{p+1}$. LemmaD.2 implies that $\mathcal{B}_{p+2} = \mathcal{B}_{p+1}$.
(ii) Let us consider a $n \geq p+2$ such that $\mathcal{B}_n = \mathcal{B}_{n-1}$. Therefore, $Q_n = Q_{n-1}$. From Lemma D.2, we deduce that $\mathcal{B}_{n+1} = \mathcal{B}_n$.
(iii) Items (i) and (ii) imply by induction that: if ($\mathcal{B}_p = \mathcal{B}_{p+1}$) for a given $p \geq 0$, then $\forall n \geq p+2 : \mathcal{B}_n = \mathcal{B}_{p+1}$.
(iv) Item (iii) is equivalent to: if $\mathcal{B}_p = \mathcal{B}_{p+1}$ for a given $p \geq 0$, then $\forall n \geq p+2 : \mathcal{B}_n = \mathcal{B}_{n-1}$.
(v) Since $\mathcal{B}_n = \mathcal{B}_{p+1}$ for $n = p+1$, Item (iv) can be written: if $\mathcal{B}_p = \mathcal{B}_{p+1}$ for a given $p \geq 0$, then $\forall n \geq p+1 : \mathcal{B}_n = \mathcal{B}_{n-1}$.

D.4 Proof of Lemma D.4

Let $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{C}, \mathcal{I}, \mathcal{T}, l_0, \mathcal{L}_m)$ be the TA from which we construct $\mathcal{B}_0$, $\mathcal{B}_1$, $\mathcal{B}_2, \cdots$. Recall that each state of a SEA, and thus of $\mathcal{B}_i$, is defined by three parts which are:

- Part 1: a location of $\mathcal{L}$.
- Part 2: a *Clock-Cond* and an *ExpSeq* for each clock of $\mathcal{C}$.
- Part 3: zero or one Δ*Clock-Cond* for each pair of clocks of $\mathcal{C}$.

(i) The cardinal of Part 1 has a *finite* upper bound equal to $|\mathcal{L}|$.
(ii) The (integer) values to which a clock $c_i \in \mathcal{C}$ is compared (i.e.., the values used in guards using c_i) are finite and their number is finite.
(iii) Item (ii) implies that the number of pairs (*Clock-Cond*, *ExpSeq*) using a clock $c_i \in \mathcal{C}$ has a *finite* upper bound which we denote U. For each clock c_i, let us consider the distinct Set events associated to c_i, and let $k_{i,j}$ be the number of constants in each of these Set, for $j = 1, \cdots$. Let $p_i = \sum_{j=1,\cdots} k_{i,j}$ and $p = \max_{i=1,\cdots,|\mathcal{C}|} p_i$. It can be easily proved that U can be taken equal to p.
(iv) Item (iii) and the fact that the number of clocks of $\mathcal{A}$ is finite ($= |\mathcal{C}|$) imply that $p^{|\mathcal{C}|}$ is a (finite) upper bound of the cardinal of Part 2.
(v) For each pair of clocks $c_i, c_j \in \mathcal{C}$, if M_i (resp. M_j) is the greatest value to which c_i (resp. c_j) is compared in the timing constraints of $\mathcal{A}$, then every Δ*Clock-Cond* using $c_i - c_j$ is expressed by using (integer) values that fall within the interval $[-M_j, M_i]$. This is justified by the fact that although we make sometimes additions of the members of two Δ*Clock-Conds* (for the elimination of the fictitious clock θ_i in the procedure of Section 6.2), the difference $c_i - c_j$ is:

a) in the interval $[-M_j, M_i]$ and $\Delta C_q(c_i, c_j)$ is in the form $k_1 < c_i - c_j < k_2$ or $c_i - c_j = k$ in each state q where c_i and c_j are active (i.e. q is reached by a path where the two clocks have been reset), because in this case we know in which interval the value of each of the two clocks is.

b) in the interval $[-M_j, M_i]$ or if not, $\Delta C_q(c_i, c_j)$ is in the form $k_1 < c_i - c_j$ with $k_1 \in [-M_j, M_i]$, in each state q where c_i or c_j is inactive because in such state:

- either q is not in a cycle in which only one of the two clocks c_i or c_j is reset and in this case, the difference $c_i - c_j$ is constant and in the interval $[-M_j, M_i]$ (if the active clock have been reset before the last expiration of the inactive clock) or $\Delta C_q(c_i, c_j)$ is in the form $k_1 < c_i - c_j$ (if the active clock have been reset after the last expiration of the inactive clock).
- or q is in a cycle in which only one of the two clocks c_i and c_j is reset and in this case, $\Delta C_q(c_i, c_j)$ is either in the form $k_1 < c_i - c_j$ ($k_1 \in [-M_j, M_i]$) or in the form $k_1 < c_i - c_j < k_2$ ($k_1, k_2 \in [-M_j, M_i]$) even if the difference $c_i - c_j$ can take any value not in $[-M_j, M_i]$ (the cycle can be executed many times an as only one of the two clocks is reset, their difference increase at each execution of the cycle) because of Proposition 5.2 which ensures that when in q, $\Delta C_q(c_i, c_j)$ takes values not in $[-M_j, M_i]$, q is equivalent to a state q' where $\Delta C_{q'}(c_i, c_j)$ uses values in $[-M_j, M_i]$ and by construction, q is merged with q'.

(vi) Items (ii) and (v) imply that for each pair of clocks $c_i,\ c_j \in \mathcal{C}$, the number of possible Δ*Clock-Conds* using $c_i - c_j$ has a finite upper bound which we denote V. Since the width of interval $[-M_j, M_i]$ is $\leq 2k$, V can be taken equal to $2k(2k+1)$, where k is the greatest constant used in timing constraints, because:

- for the Δ*Clock-Conds* in the form $k_1 < c_i - c_j < k_2$ or $c_i - c_j = k_1$, we have at most $2k * 2k = 4k^2$ possibilities.
- for the Δ*Clock-Conds* in the form $k_1 < c_i - c_j$, we have at most $2k$ possibilities.

(vii) Item (vi) and the fact that the number of pairs of clocks of $\mathcal{A}$ is finite ($< \frac{|\mathcal{C}|^2}{2}$) imply that $2^{2k(2k+1)\frac{|\mathcal{C}|^2}{2}}$ (= $2^{k(2k+1)|\mathcal{C}|^2}$) is a (finite) upper bound of the cardinal of Part 3.

(viii) Items (i), (iv) and (vii) imply that $|\mathcal{L}|p^{|\mathcal{C}|}2^{k(2k+1)|\mathcal{C}|^2}$ is a (finite) upper bound of the number of states of every $\mathcal{B}_i$.

D.5 Proof of Lemma D.5

Lemma D.1 implies that $\lim_{i\to\infty} |Q_i|$ exists. And from Lemma D.4, we deduce that this $\lim_{i\to\infty} |Q_i|$ is finite.

D.6 Proof of Theorem 1

(i) Lemmas D.1 and D.5 imply: $\exists i \geq 0$ such that $Q_i = Q_{i+1}$.

(ii) Item (i) and Lemma D.2 imply $\exists p \geq 0$ such that $\forall n > p : \mathcal{B}_n = \mathcal{B}_{p+1}$.

E Proof of Proposition 6.1

Item 1 of Proposition 6.1 ensures that only next remaining expirations of clocks are candidate to enabling. Let us define:

$$\mathcal{X}_1 = \{c_i - k_i < 0 \mid Exp(c_i, k_i) \in \mathcal{E}\}$$
$$\mathcal{X}_2 = \cup \{c_i - k_i = c_j - k_j \mid Exp(c_i, k_i), Exp(c_j, k_j) \in \mathcal{E}\}$$
$$\mathcal{X}_3 = \cup \{c_i - k_i < c_j - k_j \mid Exp(c_j, k_j) \in \mathcal{E}, Exp(c_i, k_i) \in EXP(q) \setminus \mathcal{E}\}$$

We have therefore : $\mathcal{X}_{q,\mathcal{E}} = \Delta C_q \cup \mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3$.

(i) Since all the clocks progress at the same rate, the system $\mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3$ has positive solution *iff* with the passing of time and if no clock is reset, the following items (ii) and (iii) will be satisfied simultaneously.

(ii) $\forall Exp(c_i, k_i) \in \mathcal{E} : c_i - k_i = 0$.

(iii) $\forall Exp(c_j, k_j) \in EXP(q) \setminus \mathcal{E} : c_j - k_j < 0$.

(iv) Item (ii) holds when and only when the expirations of $\mathcal{E}$ occur simultaneously.

(v) Item (iii) holds while and only while no expiration of $EXP(q) \setminus \mathcal{E}$ has occurred.

(vi) Items (i), (iv) and (v) mean that the system $\mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3$ has positive solution *iff* at the same future instant of time and if no clock is reset: all the expirations of $\mathcal{E}$ occur and no expiration of $EXP(q) \setminus \mathcal{E}$ occurs.

(vii) Item (vi) implies that the system $\mathcal{X}_{q,\mathcal{E}} = \Delta C_q \cup \mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3$ has positive solution *iff* ΔC_q allows that at a same future time: all expirations of $\mathcal{E}$ occur and no expiration of $EXP(q) \setminus \mathcal{E}$ occurs.

(viii) Let $q' = (q, \mathcal{E})$ (i.e. the state reached after the occurrence of $\mathcal{E}$). $\forall Exp(c_i, k) \in \mathcal{E}$, $C_{q'}(c_i)$ is in the form "$k < c_i < u$" or "$k < c_i$", where u is an integer. As $\mathcal{I}_{L_q} = \mathcal{I}_{L_{q'}}$, if $\exists$" $\sim Exp(c_i, k')''$ $\in \mathcal{I}_{L_q}$, then $\mathcal{I}_{L_q}$ is satisfied in q' if $Exp(c_i, k')$ does not occur again, i.e. if $k' > k$, because in q', $Exp(c_i, k)$ has already occurred and any other expiration of c_i has occurred. Thus, $\forall Exp(c_i, k) \in \mathcal{E}$: if the condition "$\exists Exp(c_i, k') \in \mathcal{I}_{L_q} \Rightarrow k' > k$" is satisfied, then $\mathcal{I}_{L_q}$ is satisfied in q' (notes that $\mathcal{I}_{L_q} = \mathcal{I}_{L_{q'}}$) .

(ix) Item (vii) and (viii) imply that $(q, \mathcal{E})!$ *iff* all expirations of $\mathcal{E}$ occur and no expiration of $EXP(q) \setminus \mathcal{E}$ occurs, and the invariant of L_q remains satisfied.

F Proof of Proposition 6.2

(i) Item 3 of Proposition 6.2 means that every guard of T is in one of the following forms : "$< Exp(c_i, k)$", "$\leq Exp(c_i, k)$", "$> Exp(c_i, k)$", "$\geq Exp(c_i, k)$".

(ii) Items 2 and 4 of Proposition 6.2 mean that every guard of T is satisfied in q if it is in one of the following forms: "$< Exp(c_i, k)$", "$\leq Exp(c_i, k)$", "$> Exp(c_i, k)$", "$\geq Exp(c_i, k)$".

(iii) The above Items (i) and (ii) mean that every guard of T is satisfied in q.

(iv) Item 1 of Proposition 6.2 and the above Items (i) and (iii) mean that T can be executed at any time in q, without being simultaneous to any expiration.

G Proof of Proposition 6.3

(i) Item 1 and 2 of Proposition 6.3 ensures that basically, σ is candidate to enabling in q, and this will depend on its guard, and that $\mathcal{E}$ is candidate to occur simultaneously with σ.

(ii) In Item 3 of Proposition 6.3: "$C_q(c_i) = (u < c_i < k')$, for $k' \leq k$" means that every guard "$< Exp(c_i, k)$" of T is satisfied in q; "$Exp(c_i, k) \notin \mathcal{E}$" and Item 2 of Proposition 6.3 implies that such a guard "$< Exp(c_i, k)$" remains satisfied at the occurrence of $\mathcal{E}$.

(iii) In Item 4 of Proposition 6.3:

a) "$C_q(c_i) = (u < c_i < k')$, for $k' \leq k$" implies that every guard "$\leq Exp(c_i, k)$" of T is satisfied in q;
b) "$C_q(c_i) = (u < c_i < k')$, for $k' \leq k$" and Item 2 of Proposition 6.3 imply that $\mathcal{E}$ occurs before or contains $Exp(c_i, k')$.

The above items (iiia) and (iiib) imply that every guard "$\leq Exp(c_i, k)$" of T *remains* satisfied at the occurrence of $\mathcal{E}$.

(iv) Item 5 of Proposition 6.3 means that with the passing of time, every guard "$= Exp(c_i, k)$" of T will be satisfied when and only when $c_i = k$, i.e. simultaneously to $Exp(c_i, k) \in \mathcal{E}$.

(v) Item 6 of Proposition 6.3 means that every guard "$> Exp(c_i, k)$" of T is satisfied in q. This guard remains satisfied at the occurrence of any expiration, and thus, at the occurrence of $\mathcal{E}$.

(vi) In Item 7 of Proposition 6.3, for every guard "$\geq Exp(c_i, k)$" of T:

a) "$C_q(c_i) = (k' < c_i < u)$ or $C_q(c_i) = (k' < c_i)$, for $k \leq k'$" implies that "$\geq Exp(c_i, k)$" is and remains satisfied at the occurrence of $\mathcal{E}$.
b) "$Exp(c_i, k) \in \mathcal{E}$ **and** $C_q(c_i) = (u < c_i < k)$" implies that "$\geq Exp(c_i, k)$" will be satisfied when and only when $c_i = k$, i.e. simultaneously to $Exp(c_i, k) \in \mathcal{E}$.

Item 7 of Proposition 6.3 means (via) or (vib) holds.

(vii) In the above Items (ii), (iii), (iv),(v), (via), and (vib), all the guards of T are considered.

(viii) The above items (ii), (iii), (v), and (via) correspond to the guards of T that are satisfied in q. These guards remain satisfied at the occurrence of $\mathcal{E}$.

(ix) The above items (iv) and (vib) correspond to the guards of T that will be satisfied with the passing of time. Each of these guards will be satisfied simultaneously to an Exp event of $\mathcal{E}$.

(x) Item 2 of Proposition 6.3 and the above Item (ix) mean that the guards of T that will be satisfied with the passing of time, can all be satisfied at the same time, simultaneously to $\mathcal{E}$.

(xi) The above items (vii), (viii) and (ix) mean that there is no non-satisfied guard of T that will not be satisfied with the passing of time.

(xii) $\mathcal{I}_{L_q}$ is satisfied in q if $\forall c_i \in \mathcal{C} : \exists \text{“} \sim Exp(c_i, k')'' \in \mathcal{I}_{L_q} \Rightarrow C_q(c_i)$ is in the form $u < c_i < k$ with $k \leq k'$, i.e. in q, $Exp(c_i, k')$ still not occurs.
$\forall Exp(c_i, k) \in \mathcal{E}$ if:

a) $\exists \text{“} < Exp(c_i, k')'' \in \mathcal{I}_{L_q}, \mathcal{I}_{L_q}$ become unsatisfied when the value of c_i is equal to k'; and if
b) $\exists \text{“} \leq Exp(c_i, k')'' \in \mathcal{I}_{L_q}, \mathcal{I}_{L_q}$ become unsatisfied when the value of c_i become greater than k'.

a) and b) imply that if the condition $\forall Exp(c_i, k) \in \mathcal{E} : \exists(< Exp(c_i, k')) \in \mathcal{I}_{L_q} \Rightarrow k < k'$ and $\exists(\leq Exp(c_i, k')) \in \mathcal{I}_{L_q} \Rightarrow k \leq k'$ is satisfied in q, then at the moment of the simultaneous occurrence of $\mathcal{E}$ and σ, the invariant of q is satisfied.

(xiii) The above items (viii), (x), (xi), (xii) and item 1 of Proposition 6.3 mean that the simultaneous executions of T and $\mathcal{E}$ is possible in q and at a moment where the invariant of q is still satisfied.

H Proof of Theorem 2

Let $\mathcal{A}$ be a TA, $\mathcal{A}^I = StepOne(\mathcal{A})$ be obtained from $\mathcal{A}$ by applying Step 1, and $\mathcal{B} = StepTwo(\mathcal{A}^I)$ be the SEA obtained by applying Step 2 to $\mathcal{A}^I$.

(i) Step 1 of $SetExp$ relabels transitions of the TA $\mathcal{A}$ without changing its structure. The fact to reset a clock c_i in a transition Tr1, in order to compare further in a transition Tr2 the value of c_i with k, is clearly equivalent to: resetting c_i and programming it such that it expires after a delay k (Substep 1), and then checking in Tr2 whether the expiration of c_i has occurred (Substep 2). Therefore, $\mathcal{A}$ and $\mathcal{A}^I$ describe exactly the same order and timing constraints of the events other than Set and Exp events.

(ii) $SetExp$ is realized by a fix-point method that constructs iteratively all the reachable states and the transitions of the SEA. This fix-point method converges after a finite number of iterations (Theorem 1) and necessitates in each iteration:

a) to determine the events enabled in every state, and
b) to construct the states reached by the occurrence of these enabled events.

Therefore, this fix-point method generates a correct result *iff* (iia) and (iib) are realized correctly.

(iii) Lemmas 6.1, 6.2 and 6.3 imply that the events enabled in every state are determined correctly.

(iv) The detailed explanations of the procedures in sections 6.1 and 6.2 can be considered as a proof sketch that the states reached by the enabled labels are constructed correctly.

(v) Items (ii), (iii) and (iv) imply that the fixpoint method is correct, i.e., Step 2 of $SetExp$ is correct. Therefore, $\mathcal{B}$ accepts all and only the possible order of events that respects the specification of $\mathcal{A}^I$.

(vi) Item v and the fact that $L^t(\mathcal{B})$ (see Definition 7.1) models the behavior of $\mathcal{B}$, mean that $L^t(\mathcal{B})$ contains all and only the possible order of events that respects the specification of $\mathcal{A}^I$.

(vii) Items (i) and (vi) mean that $L^t(\mathcal{A})$ (Definition 2.7) is obtained from $L^t(\mathcal{B})$ by removing all Set and Exp events, i.e $\mathcal{A}$ and $\mathcal{B}$ are equivalent.

I Proof of Proposition 7.1

Recall that the first step of $SetExp$ is realized in two subsbteps. In the first substep, we construct the Set events, and in the second substep, we rewrite the timing constraints in the form of *Exp-Condition*.

1. For the first substep, we make the following computations.
   a) For each transition T_i (maximum $|\mathcal{T}|$ steps) and for each clock c_i that is reset (maximum $|\mathcal{C}|$ steps):

(i) scan the others transitions T_j, $j = 1, ...$ (maximum $|\mathcal{T}|$ steps) and locations l_j, $j = 1, ...$ (maximum $|\mathcal{L}|$ steps) to record the constants;
(ii) for each scanned transition T_j, we have to find in its guard the constraint using the clock c_i (maximum $|\mathcal{C}|$ steps to find it) and to record its constant(s), and we have to check if c_i is reset by T_j (maximum $|\mathcal{C}|$ steps);
(iii) for each scanned location l_j, we have to find in its invariant the constraint using the clock c_i (maximum $|\mathcal{C}|$ steps to find it) and to record its constant(s).

b) Items a), (i), (ii) and (iii) imply that the number of steps to process all the Set events is bounded by $|\mathcal{T}||\mathcal{C}|(|\mathcal{T}||\mathcal{C}|^2 + |\mathcal{L}||\mathcal{C}|)$.

2. For the second substep, we make the following operations:
   a) For each transition (maximum $|\mathcal{T}|$ steps): rewrite each inequation of its guard in the form of *Exp-Condition* (maximum $|\mathcal{C}|$ steps);
   b) For each location (maximum $|\mathcal{L}|$ steps): rewrite each inequation of its invariant in the form of *Exp-Condition* (maximum $|\mathcal{C}|$ steps);
   c) Items a) and b) imply that the number of step to rewrite all the constraints in the form of *Exp-Condition* is bounded by $|\mathcal{T}||\mathcal{C}| + |\mathcal{L}||\mathcal{C}|$.
3. Items 1b) and 2c) imply the complexity of the first step of $SetExp$ is: $\mathcal{O}(|\mathcal{T}|^2|\mathcal{C}|^3 + |\mathcal{T}||\mathcal{L}||\mathcal{C}|^2)$ (complexity of substep 2 is in a negligible order).

J Proof of Proposition 7.2

1. For Item 1 of Proposition 7.2, each Δ*Clock-Cond* use two different clocks and thus, the maximum number of Δ*Clock-Conds* is the set of combination of 2 elements among $|\mathcal{C}|$.
2. For Item 2 of Proposition 7.2, the number of events of type 1 that can be enabled is the sum of combination of i elements among $|\mathcal{C}|$, for $i = 1, ..., |\mathcal{C}|$. That is, $|\Gamma_{1,q}| = \sum_{i=1,...,|\mathcal{C}|} \complement^i_{|\mathcal{C}|} = \sum_{i=1,...,|\mathcal{C}|} \frac{|\mathcal{C}|!}{i!(|\mathcal{C}|-i)!}$.
3. For Item 3 of Proposition 7.2, we have to verify the three Items of Proposition 6.1. That is:
   a) For Item 1 of Proposition 6.1, i.e. check if $\mathcal{E} \in EXP(q)$, we normally check if for each $Exp(c_i, k) \in \mathcal{E}$ (maximum $|\mathcal{C}|$ expirations), c_i is active and k is its next expiration (comparison of k and the upper bound of $C_q(c_i)$, that is one operation). But in practice, we check if $(q, \mathcal{E})!$ only for each $\mathcal{E} \subseteq EXP(q)$ and we do not need to check it again. That is, we have no complexity for this Item.
   b) For item 2 of Proposition 6.1, we have to check if $\mathcal{X}_{q,\mathcal{E}}$ has a positive solution. That is, we have to check if the four sets of $\mathcal{X}_{q,\mathcal{E}}$ are satisfied. The second set of $\mathcal{X}_{q,\mathcal{E}}$ is always satisfied by construction, because we have $\mathcal{E} \in EXP(q)$. The satisfaction of the 3 others sets can be checked as follows. For each Δ*Clock-Cond* $\Delta C_q(c_i, c_j) \in \Delta C_q$:
      (i) check if $c_i \in \mathcal{E}$ (maximum $|\mathcal{C}|$ steps);
      (ii) check if $c_j \in \mathcal{E}$ (maximum $|\mathcal{C}|$ steps);
      (iii) if $c_i \in \mathcal{E}$ and $c_j \in \mathcal{E}$, check if $\Delta C_q(c_i, c_j)$ and the third set of $\mathcal{X}_{q,\mathcal{E}}$ are compatible, i.e. $\Delta C_q(c_i, c_j) \wedge (c_i - c_j = k_i - k_j)$ is not empty. This can be done in 2 steps.
      (iv) if $c_i \in \mathcal{E}$ and $c_j \in EXP(q) \setminus \mathcal{E}$, check if $\Delta C_q(c_i, c_j$ and the fourth set of $\mathcal{X}_{q,\mathcal{E}}$ are compatible, i.e. $\Delta C_q(c_i, c_j) \wedge (c_i - c_j < k_i - k_j)$ is not empty. This can be done in 2 steps.

      The first paragraph of this Item and the above Items (i)-(iv) imply that the maximum steps needed to check if $\mathcal{X}_{q,\mathcal{E}}$ has a positive solution is $|\Delta C|(2|\mathcal{C}| + 4)$.
   c) For Item 3 of Proposition 6.1, we have to make the following computations: For each $Exp(c_i, k) \in \mathcal{E}$ (maximum $|\mathcal{C}|$ components), we have to check if there exists a constraint using c_i in the location invariant of q (maximum $|\mathcal{C}|$ steps) and to compare the constants k and k'. That is, the maximum number or steps to check this Item is $|\mathcal{C}|^2$.
   d) The above Items a), b) and c) imply that the maximum number of steps to check if $(q, \mathcal{E})!$ is $|\Delta C|(2|\mathcal{C}| + 4) + |\mathcal{C}|^2$, and thus of complexity $\mathcal{O}(|\mathcal{C}|^3)$ (recall that $|\Delta C| = \frac{|\mathcal{C}|(|\mathcal{C}|-1)}{2}$).
4. For Item 4 of Proposition 7.2, we have to verify the four Items of Proposition 6.2. That is, we do the following computations:

a) For Item 1 of Proposition 6.2, we normally check if $\sigma \in OUT(L_q)$. In practice, we check if $(q,\sigma)!$ only for every $\sigma \in OUT(L_q)$ and thus, this condition is basically satisfied and do not imply a computational complexity.
b) For Item 2 of Proposition 6.2, we check for each "$< Exp(c_i,k)$" or "$\leq Exp(c_i,k)$"$\in \mathcal{G}_T$ (maximum $|\mathcal{C}|$ components) if $k' \leq k$ (one operation). That is, this Item can be check in maximum $|\mathcal{C}|$ steps.
c) For Item 3 of Proposition 6.2, we check for each $c_i \in \mathcal{Z}_T$ (maximum $|\mathcal{C}|$ components) if "$= Exp(c_i,k)$"$\notin \mathcal{G}_T$ (maximum $|\mathcal{C}|$ components). That is, this Item can be check in maximum $|\mathcal{C}|^2$ steps.
d) For Item 4 of Proposition 6.2, we check for each "$> Exp(c_i,k)$" or "$\geq Exp(c_i,k)$"$\in \mathcal{G}_T$ (maximum $|\mathcal{C}|$ components) if $k' \geq k$ (one operation, k' is the lower bound of $C_q(c_i)$). That is, this Item can be checked in maximum $|\mathcal{C}|$ steps.
e) The above Items a)-d) imply that the maximum number of steps to check if $(q,\sigma)!$ is $|\mathcal{C}|^2 + 2|\mathcal{C}|$ and thus of complexity $\mathcal{O}(|\mathcal{C}|^2)$.

5. For Item 5 of Proposition 7.2, we have to verify the 8 Items of Proposition 6.3. That is, we do the following computations:

a) The computational complexity of Item 1 and 2 of Proposition 6.3 can be deduced from the one of the above Items 3a), 3b) and 4a). That is, the maximum number of steps to check Items 1 and 2 of Proposition 6.3 is $|\Delta\mathcal{C}|(2|\mathcal{C}|+4)$.
b) For Item 3 of Proposition 6.3, we check that for each "$< Exp(c_i,k)$"$\in \mathcal{G}_T$ (maximum $|\mathcal{C}|$ components): $Exp(c_i,k) \notin \mathcal{E}$ (maximum $|\mathcal{C}|$ steps) and we compare k' (upper bound of $C_q(c_i)$ and k (one step). That is, this Item can be checked in maximum $|\mathcal{C}|(|\mathcal{C}|+1)$ steps.
c) For Item 4 of Proposition 6.3, we check that for each "$\leq Exp(c_i,k)$"$\in \mathcal{G}_T$ (maximum $|\mathcal{C}|$ components) if $k' \leq k$ (one step), where is k' the upper bound of $C_q(c_i)$. That is, this Item can be checked in maximum $|\mathcal{C}|$ steps.
d) For Item 5 of Proposition 6.3, we check that for each "$= Exp(c_i,k)$"$\in \mathcal{G}_T$ (maximum $|\mathcal{C}|$ components): $Exp(c_i,k) \in \mathcal{E}$ (maximum $|\mathcal{C}|$ steps) and $u < k$ (one step) where u is the lower bound of $C_q(c_i)$. That is, this Item can be checked in maximum $|\mathcal{C}|(|\mathcal{C}|+1)$ steps.
e) For Item 6 of Proposition 6.3, we check that for each "$> Exp(c_i,k)$"$\in \mathcal{G}_T$ (maximum $|\mathcal{C}|$ components) if $k' \geq k$ (one step), where k' is the lower bound of $C_q(c_i)$. That is, this Item can be checked in maximum $|\mathcal{C}|$ steps.
f) For Item 7 of Proposition 6.3, we check that for each "$\geq Exp(c_i,k)$"$\in \mathcal{G}_T$ (maximum $|\mathcal{C}|$ components): $k' \geq k$ (one operation), where k' is the lower bound of $C_q(c_i)$, and in case this condition is not satisfied, we check if $Exp(c_i,k) \in \mathcal{E}$ (maximum $|\mathcal{C}|$ steps) and if $u < k$ (one step) where u is the lower bound of $C_q(c_i)$. That is, this Item can be checked in maximum $|\mathcal{C}|(|\mathcal{C}|+2)$ steps.
g) For Item 8 of Proposition 6.3, we check that for each $Exp(c_i,k) \in \mathcal{E}$ (maximum $|\mathcal{C}|$ steps): if there exists "$< Exp(c_i,k')$" or "$\leq Exp(c_i,k')$" in the location invariant of q (maximum $|\mathcal{C}|$ steps), then we check if $k' < k$ or $k' \leq k$ (one operation). That is, this Item can be checked in maximum $|\mathcal{C}|^2$ steps.
h) The above Items a)-g) imply that the maximum number of steps to check if $(q,\gamma)!$ is $2|\Delta\mathcal{C}||\mathcal{C}| + 4|\Delta\mathcal{C}| + 4|\mathcal{C}|^2 + 6|\mathcal{C}|$ where $|\Delta\mathcal{C}| = \frac{|\mathcal{C}|(|\mathcal{C}|-1)}{2}$ and thus of complexity $\mathcal{O}(|\mathcal{C}|^3)$.

6. For Item 6 of Proposition 7.2, we do the following computations.

a) We determine all the events of type 1 that are enabled, i.e. we check $|\Gamma_{1,q}|$ times if $(q,\mathcal{E})!$ (complexity of Item 3 of 7.2). That is, the number of steps to determine all the events of type 1 enabled in q is $|\Gamma_{1,q}|(|\mathcal{C}|^2 + |\Delta\mathcal{C}|(2|\mathcal{C}|+4))$.
b) We determine all the events of type 2 that are enabled, i.e. we check for each event σ enabled in L_q (maximum $|\Sigma|$ components) if $(q,\sigma)!$ (complexity of Item 4 of of 7.2). That is, the number of steps to determine all the events of type 2 enabled in q is $|\Sigma|(|\mathcal{C}|^2 + 2|\mathcal{C}|)$.
c) We determine all the events of type 3 that are enabled, i.e. we check for each event $\mathcal{E} \in \Gamma_1$ that satisfy Items 1 and 2 of Proposition 6.1 (maximum $|\Gamma_{1,q}|$ components, Items 1 and 2 of Proposition 6.1 are checked in Item a) above) and for each event $\sigma \in \Gamma_2$ enabled in L_q (maximum $|\Sigma|$ events) the satisfaction of Items 3-8 of Proposition 6.3 (maximum $4|\mathcal{C}|^2 + 6|\mathcal{C}|$ steps, see proof of Item 5 of Proposition 7.2). That is, the number of steps to determine all the events of type 3 enabled in q is $|\Gamma_{1,q}||\Sigma|(4|\mathcal{C}|^2 + 6|\mathcal{C}|)$.

The above Items a)-c) imply that the maximum number of steps to determine all the events enabled in q is $|\Gamma_{1,q}|(|\mathcal{C}|^2 + |\Delta\mathcal{C}|(2|\mathcal{C}|+4)) + |\Sigma|(|\mathcal{C}|^2 + 2|\mathcal{C}|) + |\Gamma_{1,q}||\Sigma|(4|\mathcal{C}|^2 + 6|\mathcal{C}|)$ where $|\Delta\mathcal{C}| = \frac{|\mathcal{C}|(|\mathcal{C}|-1)}{2}$ and $|\Gamma_{1,q}| = 2^{|\mathcal{C}|} - 1$ and thus of complexity $\mathcal{O}(2^{|\mathcal{C}|}|\mathcal{C}|^2(|\mathcal{C}| + |\Sigma|))$.

K Proof of Proposition 7.3

1. For Item 1 of Proposition 7.3, we compute $r = (q, \mathcal{E})$ as described by the procedure in Section 6.1. That is, we do the following computations.
   a) We compute lines 2-4 as follows: for each $Exp(c_i, k) \in \mathcal{E}$ (maximum $|\mathcal{C}|$ components), we compare $K_r(c_i)$ to 0 (one step) and we update $C_r(c_i)$ (maximum 2 steps to update the 2 bound) and $K_r(c_i)$ (one step). That is the maximum steps to compute line 2-4 is $4|\mathcal{C}|$.
   b) We compute line 5 as follows: for each $Exp(c_i, k) \in \mathcal{E}$ and $Exp(c_j, k) \in \mathcal{E}$ (maximum $|\Delta C|$ components), we search $\Delta C_r(c_i, c_j)$ (maximum $|\Delta C|$ steps) and update it (4 operations to update the 2 bounds and the 2 comparison operators). That is, this line is computed in maximum $|\Delta C|(|\Delta C| + 4)$.
   c) We compute line 6 as follows: for each $Exp(c_i, k) \in \mathcal{E}$ (maximum $|\mathcal{C}|$ components) and for each $Exp(c_j, k) \in EXP(q) \setminus \mathcal{E}$ (maximum $|\mathcal{C}|$ components), we search $\Delta C_r(c_i, c_j)$ (maximum $|\Delta C|$ steps) and combine it with $k_i - k_j < c_i - c_j$ (one operation to update one bound). That is, this line is computed in maximum $|\mathcal{C}|^2(|\Delta C| + 1)$.
   d) We compute line 7 as follows: we check if the upper bound of each $C_r(c_i)$ is infinite ($|\mathcal{C}|$ steps) and if so we empty ΔC_r (maximum $|\Delta C|$ steps). That is, this line is computed in maximum $|\mathcal{C}| + |\Delta C|$.
   e) The above Items a)-d) imply that the the maximum number of steps to compute $r = (q, \mathcal{E})$ is $5|\mathcal{C}| + |\Delta C|(|\Delta C| + 5) + |\mathcal{C}|^2(|\Delta C| + 1)$, where $|\Delta C| = \frac{|\mathcal{C}|(|\mathcal{C}|-1)}{2}$, and thus of complexity $\mathcal{O}(|\mathcal{C}|^4)$.
2. For Item 2 of Proposition 7.3, $r = (q, \sigma)$ is computed by lines 2-13 of the procedure in Section 6.2. That is, we do the following computations.
   a) Line 2 is computed in one operation and thus can be ignored.
   b) For the loop of line 3-11, we do for each Set event (maximum $|\mathcal{C}|$ components) the followings computations.
      (i) For line 4, we replace c_i by θ_i in ΔC_r, and this is done in maximum $|\Delta C|$ steps (necessary to scan all the Δ*Clock-Conds*. That is, the maximum number of steps necessary to compute this line is $|\Delta C|$.
      (ii) For lines 5 and 6, we check if there exists $Exp(c_i, k) \in \mathcal{E}$ (maximum $|\mathcal{C}|$ steps) and if so or not, we create a new Δ*Clock-Cond* (one operation). That is, the maximum number of steps necessary to compute these lines is $|\mathcal{C}|$.
      (iii) For line 7-9, we determine each $c_j \notin \mathcal{E}$ (maximum $|\mathcal{C}|^2$ steps, because we check if each $c_j \in \mathcal{C}$, that is for $|\mathcal{C}|$ components, is in $\mathcal{Z}_T$, done in maximum $|\mathcal{C}|$ steps). Then, for each $c_j \notin \mathcal{E}$ that we have determined, we check if there exists or not exists $Exp(c_j, k) \in \mathcal{E}$ (maximum $|\mathcal{C}|$ steps) and we create a new Δ*Clock-Cond* depending on the result (line 8 if $Exp(c_j, k) \in \mathcal{E}$ and line 9 if $Exp(c_j, k) \notin \mathcal{E}$. That is, the maximum number of steps to compute lines 7-9 is $|\mathcal{C}|^2 + |\mathcal{C}|$.
      (iv) For lines 10 and 11, we update $C_r(c_i)$ (2 operations to update the two bound) and $K_r(c_i)$ (2 operations, supposing we affect the values in the Set event and remove the first value).
   c) The above Items (i)-(iv) imply that the maximum number of steps necessary to compute the loop of lines 3-11 is $|\mathcal{C}|(|\Delta C| + |\mathcal{C}| + |\mathcal{C}|^2 + |\mathcal{C}| + 4)$.
   d) For line 12, for each pair of clocks c_i and c_j, both in $\mathcal{Z}_T$ (maximum $|\Delta C|$ components), we create a new Δ*Clock-Cond* (one step).
   e) For line 13, we eliminate each θ_i in ΔC_q (maximum $|\mathcal{C}|$ θ_i to eliminate). For each Δ*Clock-Cond* using a θ_i, we must combine it with all the others Δ*Clock-Conds* using θ_i (maximum $|\Delta C|$ pairs of Δ*Clock-Conds* to combine) and for each combination, we create a new Δ*Clock-Cond* (one step). That is, the maximum number of steps to compute line 13 is $|\mathcal{C}||\Delta C|$.
   f) Line 14 is irrelevant for events of type 2 and nothing is computed at this step.
   g) The above Items a)-f) imply that the maximum number of steps to compute $r = (q, \sigma)$ is $2|\mathcal{C}||\Delta C| + 2|\mathcal{C}|^2 + |\mathcal{C}|^3 + 5|\mathcal{C}|$, where $|\Delta C| = \frac{|\mathcal{C}|(|\mathcal{C}|-1)}{2}$, and thus of complexity $\mathcal{O}(|\mathcal{C}|^3)$.
3. For Item 3 of Proposition 7.3, $r = (q, \gamma)$ is computed by lines 1-14 of the procedure in Section 6.2. That is, we do the following computations.
   a) For line 1 and 14, we compute respectively lines 1-6 and line 7 of the procedure in Section 6.1 and following the proof of Item 1 of Proposition 7.3 given in Item 1 above, these lines are computed in maximum $5|\mathcal{C}| + |\Delta C|(|\Delta C| + 5) + |\mathcal{C}|^2(|\Delta C| + 1)$ steps.

b) The maximum steps necessary to compute lines 2-13 is, following the proof of Item 2 of Proposition 7.3 given in Item 2 above, bounded by $2|\mathcal{C}||\Delta C| + 2|\mathcal{C}|^2 + |\mathcal{C}|^3 + 5|\mathcal{C}|$.
c) The above Items a) and b) imply that the maximum number of steps to compute $r = (q, \gamma)$ is $|\Delta C|(|\Delta C| + |\mathcal{C}|^2 + |\mathcal{C}| + 5) + 3|\mathcal{C}|^2 + 10|\mathcal{C}|$, and thus of complexity $\mathcal{O}(|\mathcal{C}|^4)$.

4. For Item 4 of Proposition 7.3, to check if $q = q'$, we do the following computations.
   a) We check if $L_q = L_{q'}$, and this is done in one operation and thus negligible.
   b) We check if $C_q = C_{q'}$ by comparing each $C_q(c_i)$ and $C_{q'}(c_i)$ ($|\mathcal{C}|$ comparisons), where for each comparison we make at most 2 steps (comparison of the two bounds). That is, the maximum necessary steps to check if $C_q = C_{q'}$ is bounded by $2|\mathcal{C}|$.
   c) We check if $\Delta C_q = \Delta C_{q'}$ by comparing each $\Delta C_q(c_i, c_j)$ and $\Delta C_{q'}(c_i, c_j)$. That is, for each $\Delta C_q(c_i, c_j)$ (maximum $|\Delta C|$ components), we search $\Delta C_{q'}(c_i, c_j)$ (maximum $|\Delta C|$ steps) and we compare them (4 steps to compare the 2 bounds and the 2 comparison operators). If $\Delta C_q \neq \Delta C_{q'}$, we check if c_i or c_j is inactive (2 steps and thus negligible). That is, the maximum necessary steps to check if $\Delta C_q = \Delta C_{q'}$ is bounded by $|\Delta C|(|\Delta C| + 4)$.
   d) The above Items a)-c) imply that the maximum number of steps to check if $q = q'$ or $q \simeq q'$ is $|\Delta C|(|\Delta C| + 4) + 2|\mathcal{C}|$ and thus of complexity $\mathcal{O}(|\mathcal{C}|^4)$.

L Proof of Proposition 7.4

To process a state q at iteration $i + 1$, we make the followings computations.

1. We determine all the events that are enabled in q. This can be done in maximum $|\Gamma_{1,q}|(|\mathcal{C}|^2 + |\Delta C|(2|\mathcal{C}| + 4)) + |\Sigma|(|\mathcal{C}|^2 + 2|\mathcal{C}|) + |\Gamma_{1,q}||\Sigma|(4|\mathcal{C}|^2 + 6|\mathcal{C}|)$ steps following Items 6 of Proposition 7.2.
2. We compute the state reached after the occurrence of each enabled transition. We can have respectively at most $|\Gamma_{1,q}|$, $|\Sigma|$ and $|\Gamma_{1,q}||\Sigma|$ events of type 1, 2 and 3 enabled in q. That is, following Items 1-3 of Proposition 7.3, the maximum number of steps to make this computation is $|\Gamma_{1,q}|(5|\mathcal{C}| + |\Delta C|(|\Delta C| + 5) + |\mathcal{C}|^2(|\Delta C| + 1)) + |\Sigma|(2|\mathcal{C}||\Delta C| + 2|\mathcal{C}|^2 + |\mathcal{C}|^3 + 5|\mathcal{C}|) + |\Gamma_{1,q}||\Sigma|(|\Delta C|(|\Delta C| + |\mathcal{C}|^2 + |\mathcal{C}| + 5) + 3|\mathcal{C}|^2 + 10|\mathcal{C}|)$.
3. We check for each new constructed state if the same state or an equivalent state has not been already constructed. Therefore, for each new state, we scan Q_i (maximum $|Q|$ steps) and for each $q_i \in Q_i$, we check if $q = q_i$ or $q \simeq q_i$. Thus, following the above Item 1 and Item 4 of Proposition 7.3, the maximum steps necessary to make this computation is bounded by $(|\Gamma_{1,q}| + |\Sigma| + |\Gamma_{1,q}||\Sigma|)(|Q| + |\Delta C|(|\Delta C| + 4) + 2|\mathcal{C}|)$.
4. The above Items 1-3 imply that the maximum number of steps to process a state q is $|\Gamma_{1,q}|(|\mathcal{C}|^2 + |\Delta C|(2|\mathcal{C}| + 4)) + |\Sigma|(|\mathcal{C}|^2 + 2|\mathcal{C}|) + |\Gamma_{1,q}||\Sigma|(4|\mathcal{C}|^2 + 6|\mathcal{C}|) + |\Gamma_{1,q}|(5|\mathcal{C}| + |\Delta C|(|\Delta C| + 5) + |\mathcal{C}|^2(|\Delta C| + 1)) + |\Sigma|(2|\mathcal{C}||\Delta C| + 2|\mathcal{C}|^2 + |\mathcal{C}|^3 + 5|\mathcal{C}|) + |\Gamma_{1,q}||\Sigma|(|\Delta C|(|\Delta C| + |\mathcal{C}|^2 + |\mathcal{C}| + 5) + 3|\mathcal{C}|^2 + 10|\mathcal{C}|) + (|\Gamma_{1,q}| + |\Sigma| + |\Gamma_{1,q}||\Sigma|)(|Q_i| + |\Delta C|(|\Delta C| + 4) + 2|\mathcal{C}|)$, where $|\Delta C| = \frac{|\mathcal{C}|(|\mathcal{C}|-1)}{2}$ and $|\Gamma_{1,q}| = 2^{|\mathcal{C}|} - 1$, and thus, of complexity $\mathcal{O}(2^{|\mathcal{C}|}|\Sigma|(|\mathcal{C}|^4 + |Q|))$.

M Proof of Proposition 7.5

See proof of Theorem 1.

N Proof of Proposition 7.6

Let $\gamma = (\mathcal{E}, \sigma) \in \Gamma_3$, with $\sigma = (\rho, \mathcal{S}) \in \Gamma_2$ such that $(q, \gamma)!$ and $t = (q, \gamma, r)$ is unconditioned.

1. For the first point of Lemma 7.6, i.e. $\gamma|_{Set} = \emptyset$:
   (i) From Proposition 6.3, $(q, \gamma)!$ implies $\mathcal{E}$ satisfies item 1 and 2 of Proposition 6.1 (these later are the same as item 2 of Proposition 6.3.
   (ii) The fact that $(q, \gamma)!$ implies that $\mathcal{E}$ satisfies item 8 of Proposition 6.3 and thus satisfies item 3 of Proposition 6.1, because as t is unconditioned, $\nexists \text{“} \leq Exp(c_i, k)'' \in \mathcal{I}_{L_q}$.
   (iii) Item (1i) and (1ii) imply that $(q, \mathcal{E})!$.
   (iv) Let $q' = (q, \mathcal{E})$. As q' is reached from q by a transition of type 1, then $L_{q'} = L_q$ and thus, σ satisfies item 1 of Proposition 6.2 at q'.

(v) As t is unconditioned, we have that $\nexists$" $\leq Exp(c_i,k)''\in\mathcal{G}_t$. As $(q,\gamma)!$, from item 3 of Proposition 6.3, we have that $\exists$" $< Exp(c_i,k)''\in\mathcal{G}_t \Rightarrow Exp(c_i,k)\notin\mathcal{E}$ and for such c_i, we have that $C_q(c_i)=C_{q'}(c_i)$ and the condition "$\exists k'\leq k,\exists u<k'$ such that $C_{q'}(c_i)=u<c_i<k'$" remains satisfied at q'. Thus, item 2 of Proposition 6.2 for σ is satisfied at q'.

(vi) As t is unconditioned, then $\nexists$" $= Exp(c_i,k)''\in\mathcal{G}_t$ and thus, item 3 of Proposition 6.2 is satisfied for σ at q'.

(vii) a) As t is unconditioned, then $\nexists$" $\geq Exp(c_i,k)''\in\mathcal{G}_t$.
b) Item 6 of Proposition 6.3 implies that in q, "$\forall$" $> Exp(c_i,k)''\in\mathcal{G}_t : \exists k'\geq k,\exists u>k'$ such that $C_q(c_i)=(k'<c_i<u)$ *or* $C_q(c_i)=(k'<c_i)$".
c) b) implies that at q', $\exists$" $> Exp(c_i,k)''\in\mathcal{G}_t \Rightarrow Exp(c_i,k)\notin\mathcal{E}$ and thus, $\forall$" $> Exp(c_i,k)''\in\mathcal{G}_t : C_q(c_i)=C_{q'}(c_i)$.
d) a) and c) imply that at q', "$\forall(> Exp(c_i,k))\in\mathcal{G}_t : \exists k'\geq k,\exists u>k'$ such that $C_q(c_i)=(k'<c_i<u)$ *or* $C_q(c_i)=(k'<c_i)$" and thus, item 4 of Proposition 6.2 is satisfied for σ at q'.

(viii) (1iv), (1v), (1vi) and (1vii) imply that $(q',\sigma)!$.

(ix) The procedure of computation of $r=(q,\gamma)$ in Section 6.2 can clearly be decomposed as the computation of $q'=(q,\mathcal{E})$ and $r=(q',\sigma)$.

(x) To have $(q,\sigma)!$, the four items of Proposition 6.2 must be satisfied for σ in q. The satisfaction of item 1 of Proposition 6.3 for γ implies the satisfaction of item 1 of Proposition 6.2 for σ in q. As t is unconditioned (no "$\sim Exp(c_i,k)$" in $\mathcal{G}_t$, with $\sim\in\{\leq,\geq,=\}$), item 3 of Proposition 6.2 is necessarily satisfied for σ in q, and the satisfaction of item 3 and 6 of Proposition 6.3 for γ implies the satisfaction of items 2 and 4 of Proposition 6.2 for σ in q. Thus $(q,\sigma)!$.

(xi) Let $r'=(q,\sigma)$. As t resets any clock, from the procedure of computation of r' in Section 6.2, we deduce that r' differs from q by its location, i.e. we have $L_q\neq L_{r'}$, $CE_q=CE_{r'}$, $\Delta C_q=\Delta C_{r'}$ (*Clock-Conds* and Δ*Clock-Conds* change only when clocks are reset).

(xii) (1xi) implies that $\mathcal{E}$ satisfies items 1 and 2 of Proposition 6.1 at r', because $\mathcal{E}$ satisfies item 2 of Proposition 6.3, and this depends only on *Clock-Conds* and Δ*Clock-Conds* which are the same in r' and q.

(xiii) As $(q,\gamma)!$, for $r=(q,\gamma)$, we have by semantic that $\mathcal{I}_{L_r}$ is satisfied after the occurrence of $\mathcal{E}$. As r and r' are reached from q after the occurrence of σ in the IA, we have that $L_r=L_{r'}$ and thus, item 3 of Proposition 6.1 is satisfied for $\mathcal{E}$ in r', because it ensures that $\mathcal{I}_{L_{r'}}$ remains satisfied after the occurrence of $\mathcal{E}$ at r'.

(xiv) (1xii) and (1xiii) imply that $(r',\mathcal{E})!$.

(xv) As $\gamma\rfloor_{Set}=\emptyset$, $r=(q,\gamma)$ is constructed by the procedure of Section 6.1 without execution of line 7 (construction of a state reached after the occurrence of an event of type 1) and by line 2 and 14 of the procedure of Section 6.2, and $r'=(q,\sigma)$ is constructed by line 2 and 14 of this later procedure.

(xvi) From (1xv), as line 2 and 14 of the procedure of Section 6.2 are independent of the others line, we deduce that the state $r=(q,\gamma)$ remains unchanged if we execute first line 2 of the procedure of Section 6.2 and next all the lines of the procedure of Section 6.1 instead of the normal one.

(xvii) (1v) and (1xvi) imply that the state $r''=(r',\mathcal{E})$ is equal to $r=(q,\gamma)$ because r is computed by the same instructions as r' and r.

(xviii) (1iii), (1viii) and (1ix) show that $(q,\mathcal{E})!$, $(q',\sigma)!$ (i.e. $(q,\mathcal{E}.\sigma)!$), and $(q,\gamma)=(q,\mathcal{E}.\sigma)$. In the same way, (1x), (1xiv) and (1xvii) show that $(q,\sigma)!$, $(r',\mathcal{E})!$ (i.e. $(q,\sigma.\mathcal{E})!$) and $(q,\gamma)=(q,\sigma.\mathcal{E})$. thus, item 1 of Proposition 7.6 is correct.

2. For the second point of Proposition 7.6, i.e. when $\gamma\rfloor_{Set}\neq\emptyset$ (all the clocks are reset by t):

(i) (1iii), (1viii) and (1ix) remain correct because they are independent of the fact that t resets or not resets clocks, and thus, we have that $(q,\mathcal{E})!$, $(q',\sigma)!$, with $q'=(q,\mathcal{E})$ and $(q,\gamma)=(q,\mathcal{E}.\sigma)$.

(ii) (1x) remains correct because it is independent of the fact that t resets or not resets clocks, and thus, we have that $(q,\sigma)!$.

(iii) Let $r'=(q,\sigma)$ and $r=(q,\gamma)$. As r and r' are reached from q after the occurrence of σ in the IA, we have that $L_r=L_{r'}$. As t resets all the clocks, we have that $CE_r=CE_{r'}$ and $\Delta C_r=\Delta C_{r'}$, because they are all determined by lines 10-12 of the procedure of Section 6.2, i.e. $\forall c_i\in\mathcal{C}$, with $Set(c_i;k_1\cdots,k_p)\in\mathcal{Z}_t$, we have: $C_r(c_i)=C_{r'}(c_i)=(0<c_i<k_1)$, $K_r(c_i)=K_{r'}(c_i)=k_2\cdots k_p$, $\forall c_j\in\mathcal{C}$, $\Delta C_r(c_i,c_j)=\Delta C_{r'}(c_i,c_j)=(c_i-c_j=0)$. Thus $r'=r$.

(iv) (2i) and (2iii) imply that $(q,\gamma)=(q,\mathcal{E}.\sigma)=(q,\sigma)$ and thus, item 2 of Proposition 7.6 is correct.

3. (1xviii) and (2iv) imply that Proposition 7.6 is correct.