Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 130 (2005) 101–128
www.elsevier.com/locate/entcs

A New Method for Transforming Timed Automata

Ahmed Khoumsi[1] Lucien Ouedraogo[2]

Dept. of Elect. and Comp. Eng.
University of Sherbrooke
Sherbrooke J1K2R1, CANADA

Abstract

Discrete events systems (DES) are defined by the sequences of events they can execute. For example, communication protocols and computer networks can be seen as DES. Finite state automata (FSA) are convenient for *studying* (i.e., analyzing, designing) DES, and timed automata (TA) are convenient for *describing* real-time DES. An approach for studying a real-time DES, has been to transform a TA describing the real-time DES into an equivalent FSA, and then to study the latter. We propose here a new transformation method of TA into FSA. The method is well suited for conformance testing and supervisory control of real-time DES.

Keywords: Transformation, Timed Automata, Set-Exp-Automata.

1 Introduction

Discrete events systems (DES) interact with their environment by executing events. Here are some examples of DES: communication protocols (events: send message, receive message, . . .), telephone systems (events: hang on, hang up, . . .), mobile robot (events: start moving, stop, . . .). Correctness of a Real-time DES (RTDES) depends not only on *how* but also on *when* the RTDES interacts with its environment. Timed Automata (TA) [4] are convenient to describe RTDES, but a TA has *infinitely* many states. An approach to give a finite representation of the state space of a TA, is to generate a region

[1] Email: Ahmed.Khoumsi@USherbrooke.ca
[2] Email: Lucien.Ouedraogo@USherbrooke.ca

doi:10.1016/j.entcs.2005.03.007

automaton (RA) [2]. The state space of a RA is finite but can suffer from state explosion [2]. To reduce such a state explosion, several *minimization* methods have been proposed to transform a TA into an automaton with much less states than the corresponding RA [3,15,5,6,14,1].

We propose a new method called *SetExp* for transforming a TA into a finite state automaton (FSA), called Set-Exp-Automaton (SEA), which uses two additional types of events: *Set* and *Exp* that model the setting and expiring of clocks, respectively. A TA and the corresponding SEA represent two different ways to specify the same order and timing constraints of events.

In comparison to other minimization methods, *SetExp* is well suited for *conformance testing* and *supervisory control* of RTDES[3]. Conformance testing aims at *checking* whether an implementation conforms to a specification [13], and supervisory control aims at *forcing* an implementation to conform to a specification [12]. The application of *SetExp* to conformance testing (resp., supervisory control) has been demonstrated in [7,10] (resp., in [11,8,9]). In [7,10,11,8,9], *SetExp* is used as a black-box and the focus is on its application. In the present study, our aim is to "see inside *SetExp*"; more precisely, we explain formally how *SetExp* can be implemented. Therefore, this paper is complementary to [7,10,11,8,9].

The remainder of the paper is structured as follows. Sect. 2 introduces the TA model. In Sect. 3, we illustrate *SetExp* by simple examples, and we present the SEA model. Sects. 4 and 5 explain in a formal way how *SetExp* is realized. In Sect. 6, we present a theorem that states correctness and some properties of *SetExp*. And Sect. 7 concludes the paper.

2 Timed Automata (TA)

A clock is a real-valued variable that can be reset (to 0) at the occurrence of an event and such that, between two resets, its derivative (w.r.t. time) is equal to 1. Let $\mathcal{C} = \{c_1, \cdots, c_{N_c}\}$ be a set of clocks.

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 nonnegative integer. Let $\Phi_{\mathcal{C}}$ denote the (infinite) set of clock constraints depending of clocks of $\mathcal{C}$.

And let 2^X denote the set of subsets of a set X.

[3] *SetExp* does not aim at competing with other minimization methods in *Verification*.

2.1 Syntax of TA

A TA is defined by $(\mathcal{L}, \mathcal{A}, \mathcal{C}, \mathcal{T}, l_0)$, where: $\mathcal{L}$ is a finite set of locations, l_0 is the initial location, $\mathcal{A}$ is a finite set of events (*alphabet*), $\mathcal{C}$ is a finite set of clocks, and $\mathcal{T} \subseteq \mathcal{L} \times \mathcal{A} \times \mathcal{L} \times 2^{\Phi_{\mathcal{C}}} \times 2^{\mathcal{C}}$ is a transition relation. A transition of TA is thus defined by $T = \langle q; \sigma; r; G; Z \rangle$, where: q and r are origin and destination locations, σ is an event, G is a finite subset of $\Phi_{\mathcal{C}}$ and is called *guard* of T, and Z is a subset of $\mathcal{C}$ and is called *reset* of T.

The example of Fig. 1 illustrates this definition. Locations are represented by nodes, and a transition Tr $= \langle q; \sigma; r; G; Z \rangle$ is represented by an arrow linking q to r and labeled $(\sigma; G; Z)$. An empty G or Z is represented by "-".

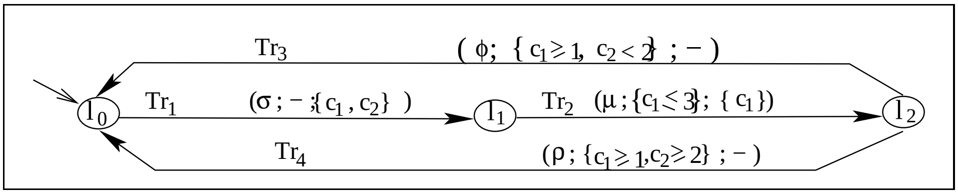

Fig. 1. Example of TA

2.2 Semantics of TA

At time $\tau_0 = 0$, the TA $A = (\mathcal{L}, \mathcal{A}, \mathcal{C}, \mathcal{T}, l_0)$ is at location l_0 with all clocks of $\mathcal{C}$ equal to 0. A transition Tr$=\langle q; \sigma; r; G; Z \rangle$ is *enabled* when q is the current location and *all* the clock constraints of the guard G (if any) evaluate to *true*; otherwise, Tr is *disabled*. From this location q, the event σ is executed only when Tr is enabled; and after the execution of σ, location r is reached and the clocks in Z are reset.

For example, the TA of Fig. 1 is initially in location l_0; it reaches l_1 at the occurrence of σ. From location l_1, the TA reaches l_2 at the occurrence of μ. From location l_2, the TA reaches l_0 at the occurrence of ϕ or ρ. Let $\delta_{u,v}$ denote the delay between events u and v. We have: $\delta_{\sigma,\mu} \leq 3$, $\delta_{\sigma,\phi} < 2$, $\delta_{\sigma,\rho} \geq 2$, $\delta_{\mu,\phi} \geq 1$, and $\delta_{\mu,\rho} \geq 1$.

The semantics of a TA A can also be defined by the set of timed sequences accepted by A. a A *timed sequence* of a TA is a sequence "$(e_1, \tau_1) \cdots (e_i, \tau_i) \cdots$", where $e_1, \cdots, e_i \in \mathcal{A}$, each τ_i is the time of occurrence of e_i, and $0 < \tau_1 < \cdots < \tau_i < \cdots$. Now given a TA A, let us define the acceptance by A of a timed sequence $\lambda = (e_1, \tau_1)(e_2, \tau_2) \cdots$. Let n be the length of λ (n can be infinite), and $\lambda_i = (e_1, \tau_1) \cdots (e_i, \tau_i)$ be the prefix of λ of length i, for $0 \leq i \leq n$ (i is finite). *λ is accepted by A iff*:

- Either λ is the empty sequence λ_0;

- Or A has a sequence of length n of consecutive transitions $Tr_1\, Tr_2 \cdots$ that starts at l_0 and s.t. $\forall i = 1, 2, \cdots, n$: the event of Tr_i is e_i and, after the execution of λ_{i-1}, Tr_i is enabled at time τ_i.

Definition 2.1 *The* Timed language of a TA *A (TL_A^{TA}) is the set of timed sequences accepted by A. That is, TL_A^{TA} models the behavior of A.*

A TA allows to express constraints on delays between events. For example, to specify that the delay d between two transitions Tr_1 and Tr_2 is such that $d \in [1, 3]$, we use a clock c_1 as follows: the reset of Tr_1 is $\{c_1\}$, and the guard of Tr_2 is $\{(c_1 \geq 1), (c_1 \leq 3)\}$.

The class of TA that we will consider respects the following hypothesis:

Hypothesis 2.1 *Each TA is assumed deterministic, i.e., if Tr1 and Tr2 are two transitions executing the same event from the same location, and are enabled at the same time, then they lead to the same location and reset the same clock(s).*

3 Illustration of *SetExp*, Set-Exp-Automata (SEA)

The aim of this paper is to present a method *SetExp* for transforming a TA into a finite state automaton, called Set-Exp-Automaton (SEA), which uses two additional types of events: *Set* and *Exp* that model the setting and expiring of clocks, respectively. A TA and the corresponding SEA represent two different ways to specify the same order and timing constraints of events. In this section, we first introduce the events *Set* and *Exp*, followed by a trivial example that illustrates *SetExp*. Then, we present the SEA model, followed by a running example of *SetExp* corresponding to the TA of Fig. 1.

3.1 Events Set and Exp

An event $Set(c_i; k)$ means: clock c_i is reset (to 0) and will expire when its value is equal to k. And $Set(c_i; k_1, k_2, \cdots, k_p)$ means that c_i is reset and will expire several times, when its value is equal to $k_1, k_2, \cdots, k_p$, respectively. We assume without loss of generality that $k_1 < k_2 < \cdots < k_p$.

An event $Exp(c_i; k)$ means: clock c_i expires and its current value is k.

Therefore, $Set(c_i; k)$ is followed (after a delay k) by $Exp(c_i; k)$, and $Set(c_i; k_1, k_2, \cdots, k_p)$ is followed (after delays $k_1, \cdots, k_p$) by $Exp(c_i; k_1)$, $Exp(c_i; k_2), \cdots, Exp(c_i; k_p)$. When a $Set(c_i; *)$ occurs, then all expirations of c_i which were foreseen before this $Set(c_i; *)$ are canceled.

3.2 Trivial example of transformation

To explain the intuition of *SetExp*, let us consider the following two specifications. Specification 1: a task must be realized in less than two units of time. Specification 2: at the beginning of the task an alarm is programmed so that it occurs after two time units, and the task must be terminated before the alarm. It is clear that these two expressions define the same timing constraint. In this example, *SetExp* can be used to obtain the second specification from the first one. The programming of the alarm corresponds to a *Set* event, and the occurrence of the alarm corresponds to an *Exp* event.

3.3 Transitions of SEA

Let A be a TA and $\mathit{SetExp}(A)$ be the corresponding SEA. In the following: σ denotes an event of the alphabet of the TA A, $\mathcal{S}$ (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. $\mathcal{E}$). Let us categorize the transitions of the SEA $\mathit{SetExp}(A)$ into three types:

Type 1 : a transition labeled $(\mathcal{E})$ represents the occurrence of $\mathcal{E}$.

Type 2 : a transition labeled (σ) or $(\sigma, \mathcal{S})$: (σ) represents the occurrence of σ, and $(\sigma, \mathcal{S})$ represents the simultaneous occurrences of σ and $\mathcal{S}$. A transition TR of Type 2 in the SEA $\mathit{SetExp}(A)$ corresponds to a transition of A.

Type 3 : a transition labeled $(\mathcal{E}, \sigma)$ or $(\mathcal{E}, \sigma, \mathcal{S})$: $(\mathcal{E}, \sigma)$ represents the simultaneous occurrences of $\mathcal{E}$ and σ, $(\mathcal{E}, \sigma, \mathcal{S})$ represents the simultaneous occurrences of $\mathcal{E}$, σ and $\mathcal{S}$. A transition TR of Type 3 in the SEA $\mathit{SetExp}(A)$ corresponds to the simultaneous executions of $\mathcal{E}$ and a transition Tr of A.

3.4 Syntax and semantics of SEA

A SEA B is a particular FSA and its syntax can be simply defined by (Q, Γ, δ, q^0), where Q is a set of states, q^0 is the initial state, and Γ is an alphabet consisting of labels of transitions of types 1, 2 or 3. $\delta \subseteq Q \times \Gamma \times Q$ is a set of transitions.

Let a *sequence* denote a (finite or infinite) sequence "$E_1 \cdots E_i \cdots$", where $E_i \in \Gamma$. From a behavioral point of view, let us define the semantics of a SEA $B = (Q, \Gamma, \delta, q^0)$ by the set of sequences accepted by B:

- A finite sequence $\mu_n = E_1 \cdots E_n$ is accepted by B *iff* it labels a sequence of n consecutive transitions of B that starts in q^0 and terminates in a state with no outgoing transition of Type 1 or 3. Intuitively, B can execute μ_n

and then stops only if there is no expiration after μ_n. This is so, because expirations cannot be delayed.

- An infinite sequence μ_∞ is accepted by B *iff* it labels a sequence of consecutive transitions of B that starts in q^0.

We can now introduce the notion of Language of a SEA:

Definition 3.1 *The language of a SEA B (L_B^{SEA}) consists of the (finite and infinite) sequences accepted by B. That is, L_B^{SEA} models the behavior of B.*

Given a SEA B, note that L_B^{SEA} implicitly respects the following condition, called the *Consistency condition*: every $Set(c;k)$ and its corresponding $Exp(c;k)$ are effectively separated by time k.

3.5 Running example

For the TA A of Fig. 1, we obtain the SEA $SetExp(A)$ of Fig. 2. We will explain in Sect. 5 how this SEA is computed. Transitions of type 1 are those labeled $Exp(c_1;1)$, $Exp(c_1;3)$, $Exp(c_2;2)$ and $(Exp(c_1;1), Exp(c_2;2))$. Transitions of Type 2 are those labeled $(\sigma, Set(c_1;3), Set(c_2;2))$, $(\mu, Set(c_1;1))$, ϕ and ρ; they correspond to transitions Tr1, Tr2, Tr3, and Tr4, respectively, of the TA A of Fig. 1. Transitions of Type 3 are those labeled
$(Exp(c_2;2), \sigma, Set(c_1;3), Set(c_2;2))$, $(Exp(c_1;1), \phi)$,
$(Exp(c_1;1), \rho)$, $(Exp(c_2;2), \rho)$, $(Exp(c_1;1), Exp(c_2;2), \rho)$,
$(Exp(c_2;2), \mu, Set(c_1;1))$, $(Exp(c_1;3), \mu, Set(c_1;1))$. Finite sequences accepted by the SEA $SetExp(A)$ (i.e., $\in L_{SetExp(A)}^{SEA}$) start in q_0 and terminate in a shaded state (i.e., state without outgoing Exp event).

SetExp, that transforms a TA $A = (\mathcal{L}, \mathcal{A}, \mathcal{C}, \mathcal{T}, l_0)$ into a SEA $SetExp(A)$, is realized in two steps. Step 1 does not modify the structure of A: it replaces the resets of transitions by *Set* events (substep 1a), and rewrites the guards of transitions in the form of order constraints relatively to *Exp* events (substep 1b). Step 2 generates $SetExp(A)$ that describes explicitly all the possible orders of events, including *Set* and *Exp* events in addition to the events of the alphabet of A. Steps 1 and 2 are presented in Sects. 4 and 5, respectively. The results of Steps 1 and 2 will be denoted $A' = StepOne(A)$ and $B = SetExp(A) = StepTwo(A')$, respectively.

4 First step of *SetExp*

Definition 4.1 *In a TA A: a* path *is a transition or sequence of consecutive transitions of A.*

Definition 4.2 *In a TA A: a transition T' is said* reachable *from a transition*

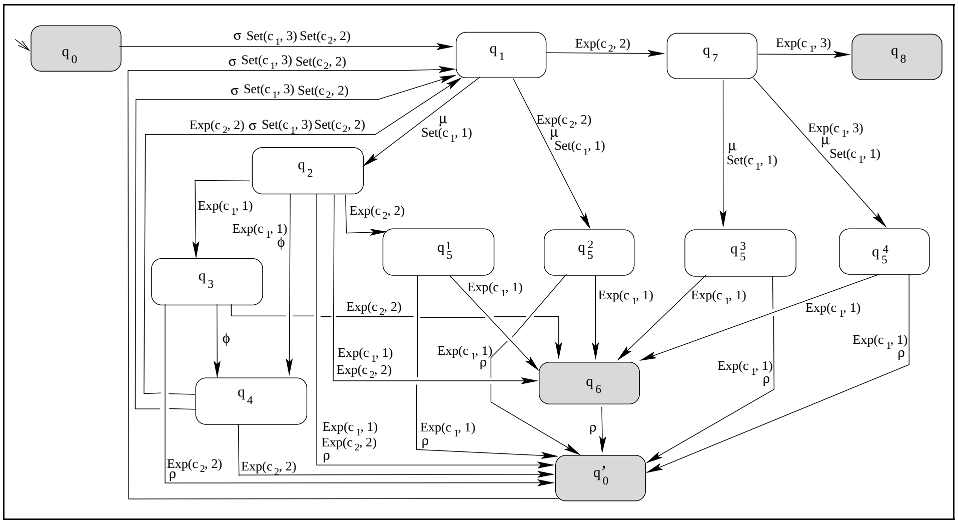

Fig. 2. Example of SEA

T iff *A contains a path* $T \cdots T'$.

Remark 4.1 *In Def. 4.1 the guards of transitions are not taken into account, and thus, a path can be* unexecutable. *Therefore in Def. 4.2, "reachability of transition" is actually an abuse of the language. We need this non-standard definition to define in a simple way Substep 1a (in Sect. 4.1).*

Notations 4.1 *In a TA A:*

- $\Re_T$ *denotes the set of transitions that are reachable from a transition* T.
- $\wp_{T \to T'}$ *denotes the set of paths between two transitions* T *and* T', excluding T *and* T'. *That is,* $\wp_{T \to T'}$ *contains every path* $T_1 \cdots T_m$ *such that: the destination location of* T *is the origin location of* T_1, *and the origin location of* T' *is the destination location of* T_m.
- Z_P *is the set of clocks that are reset in a path* P.
- Z_T *denotes the set of clocks that are reset in* T.
- G_T *denotes the guard of* T.

Since Step 1 replaces the resets of transitions by *Set* events and rewrites the guards of transitions in the form of order constraints relatively to *Exp* events, we will use the following Notations 4.2:

Notations 4.2 *For a transition* T *of a TA A:*

- $\mathcal{Z}_T$ *is the set of Set events associated to* T, *and* $\mathcal{Z}_T(c_i)$ *is the part of* $\mathcal{Z}_T$ *using* c_i.
- $\mathcal{G}_T$ *is the rewritten guard associated to* T, *and*

$\mathcal{G}_T(c_i)$ is the part of $\mathcal{G}_T$ using c_i.

Notations 4.3 *In a formal expression: Symbol "$\coloneqq$" denotes an assignment; and Symbol ":" have two possible meanings:*

- *In a definition of a set, $\{A : B\}$ denotes the set of A such that B is satisfied. Example: $\{(x, y) : x > y\}$ is the set of pairs (x, y) s.t. $x > y$.*
- *An expression "$\forall A : B$" means that B is applied whenever A is satisfied. Example: $\forall x > 3 : A[x] \coloneqq 0$ means that $A[x]$ is set to zero $\forall x > 3$.*

4.1 Substep 1a: replacing clock resets by Set events

In a TA, a clock c is reset with the objective to compare later its value to (at least) one constant, say k. The event $Set(c; k)$ is very convenient for that purpose, because it resets c and programs $Exp(c; k)$ which is a notification when $c = k$. The aim of Substep 1a is indeed to replace by $Set(c_i; k)$, each reset of c_i if it is compared later to k before a subsequent reset. If for the same transition, we obtain several $Set(c_i; k_1), Set(c_i; k_2), \cdots$, we replace them by a single $Set(c_i; k_1, k_2, \cdots)$. Without loss of generality, we consider that $k_1 < k_2 < \cdots$. Here is a formal definition of Substep 1a:

$$\boxed{\begin{array}{l} \forall T \in \mathcal{T}, \forall c_i \in \mathcal{C} : \mathcal{Z}_T(c_i) \coloneqq \{Set(c_i; U) : \\ \qquad c_i \in Z_T, U = \{k : \exists T' \in \Re_T, \exists (c_i \sim k) \in G_{T'}, \exists P \in \wp_{T \to T'}, c_i \notin Z_P\}\} \\ \forall T \in \mathcal{T} : \mathcal{Z}_T \coloneqq \cup_{c_i \in \mathcal{C}} \mathcal{Z}_T(c_i) \end{array}}$$

Note that $\mathcal{Z}_T(c_i)$ is either *empty* if $c_i \notin Z_T$, or a *singleton* $\{Set(c_i; *)\}$ if $c_i \in Z_T$.

4.2 Substep 1b: rewriting the guards

Every guard $c_i \sim k$ is rewritten in the form $\sim Exp(c_i; k)$. Formally:

$$\boxed{\begin{array}{l} \forall T \in \mathcal{T}, \forall c_i \in \mathcal{C} : \mathcal{G}_T(c_i) \coloneqq \{\sim Exp(c_i; k) : (c_i \sim k) \in G_T\} \\ \forall T \in \mathcal{T} : \mathcal{G}_T \coloneqq \cup_{c_i \in \mathcal{C}} \mathcal{G}_T(c_i) \end{array}}$$

If we apply Step 1 to the TA A of Fig. 1, we obtain the $StepOne(A)$ of Fig. 3. Labels are represented in one or two lines. The first line contains the event of the transition and *Set* event(s) if any. The second line contains the rewritten guard(s) if any.

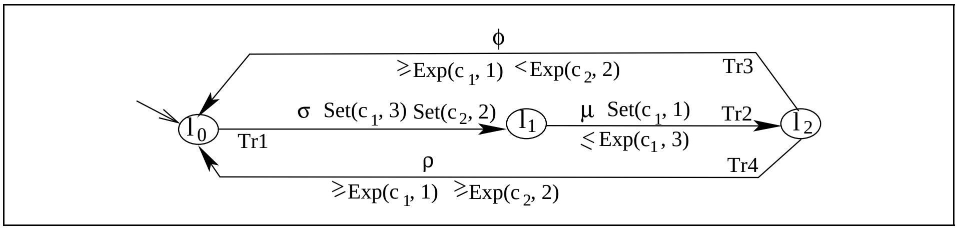

Fig. 3. Step 1 applied to the TA of Fig. 1

5 Second step of *SetExp*

Step 2 transforms $A' = StepOne(A)$ into a SEA $B = SetExp(A) = StepTwo(A')$. B describes explicitly all the possible sequences of events, including *Set* and *Exp* events in addition to the events of the alphabet of A.

5.1 States of SEA

Each state of a SEA is defined by a location and inequation(s) of clock(s). The latter definition, which we call *state definition* and present in this subsection, is not necessary for defining the syntax and semantics of a given SEA; but it is useful for algorithmic purpose, during the construction of $SetExp(A)$ from $StepOne(A)$. Once $SetExp(A)$ constructed, this state definition can be ignored. Here are a few necessary definitions.

Definition 5.1 *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.

Note that the set of current active clocks depends on the current time and on the history of the automaton up to the current time.

Definition 5.2 *For a $Set(c_i; k_1, \cdots, k_p)$, a* Clock-Cond *is an expression 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 $Exp(c_i; k_{u+1})$; and
"$k_p < c_i$", which holds after $Exp(c_i; k_p)$.

Definition 5.3 *For a $Set(c_i; k_1, \cdots, k_p)$, an* ExpSeq *is a sequence $k_u k_{u+1} \cdots k_p$, for $1 \leq u \leq p$; it 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 5.4 *A* ΔClock-Cond *is an expression in one of the following four forms, where m, m_1, m_2 are integers (they can be positive, negative, or null):*

"$c_i - c_j < m_2$", "$m_1 < c_i - c_j$", "$m_1 < c_i - c_j < m_2$", and "$c_i - c_j = m$".

Let A be a TA and $B = SetExp(A)$ be the corresponding SEA to be computed. A *state definition* of a state q of B consists of three parts:

Part 1 indicates the location of A that corresponds to q, and is noted L_q.

Part 2 consists of a Clock-Cond and an ExpSeq for each clock in $\mathcal{C}$. More precisely, for each c_i, Part 2 contains $(C_q(c_i), K_q(c_i))$ which is in one of the following forms, considering that $Set(c_i; k_1, k_2, \cdots, k_p)$ is the last *Set* event of c_i before q is reached:

- 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 $Exp(c_i; k_p)$: $C_q(c_i) = (k_p < c_i)$, $K_q(c_i) = \epsilon$.

Note that c_i is active in the three first cases, and inactive in the fourth case.

Part 3 consists of zero or one ΔClock-Cond $\Delta C_q(c_i, c_j)$ for each pair of clocks $c_i, c_j \in \mathcal{C}$.

Notations 5.1 *To recapitulate, a state q of $B = SetExp(A)$ is defined by three parts, which we note $q = \langle L_q, C_q, \Delta C_q \rangle$. L_q is a location; C_q is a N_c-tuple of pairs $(C_q(c_i); K_q(c_i))$, for $i = 1, \cdots, N_c$, where $C_q(c_i)$ is a Clock-Cond and $K_q(c_i)$ is an ExpSeq; and ΔC_q is a set of ΔClock-Conds.*

The usefulness of the three parts can be explained as follows:

- The location L_q is used to determine the transitions of A that are candidate in q, which are the outgoing transitions of L_q in A.
- Clock-Conds of q are used to determine:
  -the transitions of A that are enabled in q (among the candidate ones), and
  -the expected next expiration (if any) of each clock.
- ExpSeqs of q are used to determine the Clock-Conds and ExpSeqs of the state reached by each transition of A that is enabled in q.
- ΔClock-Conds of q are used to determine the expirations (or set of simultaneous expirations) that can *really* occur at q, among the expected ones determined from Clock-Conds of q. Therefore when all the clocks are inactive, ΔClock-Conds are irrelevant, and thus, Part 3 is empty.

When an expiration (or set of expirations) can occur in q, we obtain an enabled transition of type 1. When an event σ labelling a transition of A can occur in q, 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.

The state definition for the SEA of Fig. 2 is represented in Fig. 4. ExpSeqs are empty, and thus not represented, because in every $Set(c_i;*)$, $*$ consists of a single value.

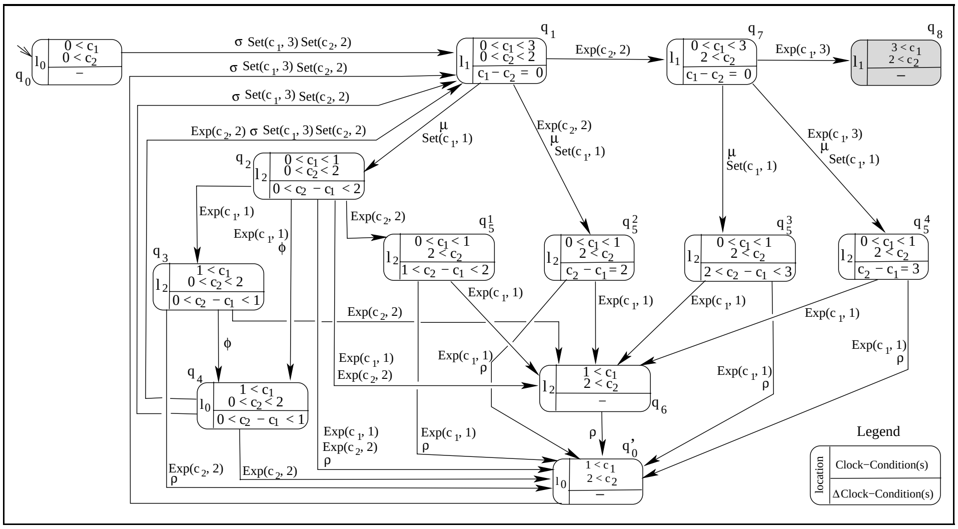

Fig. 4. Example of SEA with state definition

5.2 *Principle of construction of SEA*

Let $A' = StepOne(A)$ be the automaton computed in the first step (see Sect. 4). The principle of construction of $B = StepTwo(A')$ from A' is as follows:

- First of all, we construct its initial state $q^0 = \langle l_0, C_0, - \rangle$ where: l_0 is the initial location of A', C_0 is the N_c-tuple of pairs $(0 < c_i; \epsilon)$, for $i = 1, \cdots, N_c$. "$0 < c_i$" and ϵ are a Clock-Cond and an empty ExpSeq, respectively; they mean that c_i has been reset and no expiration is expected, i.e., c_i is initially inactive. There is no ΔClock-Cond because all clocks are inactive.
- Then, we proceed iteratively as follows until no new state or transition is constructed: For every constructed state q, we determine every event or set of (simultaneous) events that can label a transition (of type 1, 2 or 3) enabled in q. And for every determined (enabled) label l, we construct: the state r reached by l from q, and the corresponding transition.

We see that this iterative process requires to determine if a label is enabled in a state, and to compute the state reached by an enabled label and the corresponding transition. Here are some related notations and definitions:

Notations 5.2 *($OUT(l)$, $EXP(q)$, $\sharp$) :*

(i) *For a location l of A' ($= StepOne(A)$) , $OUT(l)$ denotes the set of labels of the outgoing transitions of l, including the empty label noted $\sharp$.*

(ii) *If in a state q of SEA, $c_{u_1}, c_{u_2}, \cdots, c_{u_p}$ are the active clocks and $Exp(c_{u_i}; k_{u_i})$ is the expected next expiration of c_{u_i}, for $i = 1, \cdots, p$, then:*
$EXP(q) = \{Exp(c_{u_1}; k_{u_1}), \cdots, Exp(c_{u_p}; k_{u_p})\}$.

Let q be a state of the SEA $B = StepTwo(A')$ and $\ell \in (2^{EXP(q)} \times OUT(L_q)) \setminus \{(\emptyset, \sharp)\}$ be a label of a transition of type 1, 2 or 3.

Definition 5.5 ℓ is said to be enabled in q, *which we note* $(q \xrightarrow{\ell})!$, iff *it is possible that at a same time in q,* all *the events of ℓ occur simultaneously before any other event.*

Definition 5.6 *Assuming $(q \xrightarrow{\ell})!$, we can define the state reached from q after the occurrence of ℓ, which we note $(q \xrightarrow{\ell})$.*

In Sect. 5.3, we explain in a formal way how to determine whether $(q \xrightarrow{\ell})!$ and how to compute the state $(q \xrightarrow{\ell})$. Subsection 5.3.1 (resp. 5.3.2) corresponds to the case where ℓ labels a transition of Type 1 (resp. Types 2 and 3). The reader who is not interested by these details, may directly go to Sect. 5.4 which presents in a formal way the iterative process of SEA construction.

5.3 Construction of states and transitions

From Sect. 3.3, a transition TR of B is labeled: $\mathcal{E}$ if TR is of Type 1, ℓ_T if TR is of Type 2, and $(\mathcal{E}, \ell_T)$ if TR is of Type 3; where ℓ_T is in the form (σ) or $(\sigma, \mathcal{S})$. Recall that if TR is of Type 2 or 3, then it has a corresponding transition in A' (and A). The index T in ℓ_T indicates that: T is the transition in A' (and A) corresponding to TR, and ℓ_T consists of σ and *Set* events (if any) that label T in A'. The general form of a label of a transition TR of SEA can be written $(\mathcal{E}, \ell_T)$ where: $\ell_T = \sharp$ if TR is of Type 1, and $\mathcal{E} = \emptyset$ if TR is of Type 2.

Notations 5.3 *:*

(i) *Let X be a system of linear equations and inequations using a set of variables $x_1, x_2, \cdots, x_N$. A nonnegative solution of X is any N-tuple $(a_1, a_2, \cdots, a_N)$ that satisfies X, where a_i is a nonnegative value of x_i. Let $Sol_{\geq 0}(X)$ denote the set of nonnegative solutions of X.*

(ii) *If $S = e_1 e_2 \cdots e_n$ is a sequence, then:*
$|S|$ denotes the length *of S (i.e., $|S| = n$),*
$S[i]$ denotes the ith element of S (i.e., $S[i] = e_i$),

$S[\not{i}]$ is obtained from S by removing *the ith element of S.*

(iii) *Let c_i and c_j be two clocks, and ΔC_1 and ΔC_2 be two ΔClock-Conds using $c_i - c_j$ (see Def. 5.4). $\Delta C_1 \sqcap \Delta C_2$ is obtained by* combining *ΔC_1 and ΔC_2 into a single ΔClock-Cond that corresponds to the conjunction of ΔC_1 and ΔC_2. For example, if $\Delta C_1 = (m_1 < c_i - c_j < m_2)$ and $\Delta C_2 = (m < c_i - c_j)$, then the result of* combination *is $\max(m_1, m) < c_i - c_j < m_2$.*

(iv) *Let c_i and θ_i be two clocks, and C be a Clock-Cond using c_i.*
$C\langle c_i \mapsto \theta_i - c_i\rangle$ is obtained from C by replacing *c_i by $\theta_i - c_i$.*

(v) *Let ΔC be a set of ΔClock-Conds, and θ_i be a clock possibly used in ΔC. $\Delta C\langle \not{\theta}_i\rangle$ is obtained from ΔC by* eliminating *θ_i (if any) from ΔC by rewriting its ΔClock-Conds. For that purpose, every pair of ΔClock-Conds in ΔC that use a θ_i, are added with each other in order to eliminate θ_i. The addition of two $\Delta C_r(\theta_i, c_j)$ and $\Delta C_r(\theta_i, c_k)$ for the elimination of θ_i is realized as follows: 1) $\Delta C_r(\theta_i, c_j)$ is expressed using $\theta_i - c_j$; 2) $\Delta C_r(\theta_i, c_k)$ is expressed using $c_k - \theta_i$; 3) the members of the two ΔClock-Conds are added in order to obtain a ΔClock-Cond expressed using $c_k - c_j$; 4) the obtained ΔClock-Cond is combined with $\Delta C_r(c_j, c_k)$, if any.*
If θ_i is used in a single ΔClock-Cond, the elimination consists of removing this ΔClock-Cond.

5.3.1 Study of transitions labeled ($\mathcal{E}$) (i.e., of Type 1)

Let q be a state of the SEA $B = \mathit{StepTwo}(A')$ and $\mathcal{E}$ be a set of expirations.

Let X be the following system of inequations (formally described as a *set* of inequations) where the variables are clocks:

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

The following lemma can be used to determine whether $\mathcal{E}$ is enabled in q. It states that $\mathcal{E}$ is enabled in q *iff*: each $\mathit{Exp}(c_i; k) \in \mathcal{E}$ is the expected next expiration of c_i, and X has nonnegative solutions.

Lemma 5.1 [4] *$(q \xrightarrow{\mathcal{E}})! \Leftrightarrow (\mathcal{E} \subseteq \mathit{EXP}(q)$ and $\mathit{Sol}_{\geq 0}(X) \neq \emptyset)$.*

When $\mathcal{E}$ is enabled in q (noted $(q \xrightarrow{\mathcal{E}})!$), let us express formally the procedure for computing the state r, noted $(q \xrightarrow{\mathcal{E}})$, reached from q after the occurrence of $\mathcal{E}$:

[4] Proof in Section A.1

```
1. $r \coloneqq q$
2. $\forall Exp(c_i; k) \in \mathcal{E}$: $|K_r(c_i)| = 0 \Rightarrow C_r(c_i) \coloneqq (k < c_i)$;
3.                 $|K_r(c_i)| > 0 \Rightarrow C_r(c_i) \coloneqq (k < c_i < K_r(c_i)[1])$,
4.                                  $K_r(c_i)[\cancel{1}]$.
5. $\forall Exp(c_i; k_i), Exp(c_j; k_j) \in \mathcal{E}$: $\Delta C_r(c_i, c_j) \coloneqq (c_i - c_j = k_i - k_j)$.
6. $\forall Exp(c_i; k_i) \in \mathcal{E}, \forall Exp(c_j; k_j) \in EXP(q) \setminus \mathcal{E}$:
7.            $\Delta C_r(c_i, c_j) \coloneqq \Delta C_r(c_i, c_j) \sqcap (k_i - k_j < c_i - c_j)$.
8. $( \forall c_i \in \mathcal{C}, \exists k > 0, C_r(c_i) = (k < c_i) ) \Rightarrow \Delta C_r \coloneqq \emptyset$.
```

Let us give some explanations of the above computation of $(q \xrightarrow{\mathcal{E}})$:

Line 1 : r is initialized with q.

Lines 2-4 : For each $Exp(c_i; k) \in \mathcal{E}$, the aim is to update $C_q(c_i)$ and $K_q(c_i)$.

Line 2 : We consider the case where an $Exp(c_i; k) \in \mathcal{E}$ is the last expected expiration of c_i. The assignment models the fact that $Exp(c_i; k)$ has occurrred and that no other expiration of c_i is foreseen.

Lines 3-4 : We consider the case where: $Exp(c_i; k)$ is the expected next expiration of c_i, and another $Exp(c_i; k')$ becomes expected after $Exp(c_i; k)$. In ***Line 3*** the assignment models the fact that $Exp(c_i; k)$ has occurred and that $Exp(c_i; k')$ is the expected new expiration. (k' is indicated by $K_r(c_i)[1]$.) In ***Line 4***, k' is removed from $K_r(c_i)$ because $K_r(c_i)$ is a sequence that specifies the values to which c_i will expire *after* its next expiration $Exp(c_i; k')$.

Line 5 : Let τ be the time of occurrence of $\mathcal{E}$. For every $Exp(c_i; k_i) \in \mathcal{E}$, $c_i - k_i = 0$ at time τ. Therefore for every $Exp(c_i; k_i)$ and $Exp(c_j; k_j) \in \mathcal{E}$, $c_i - k_i = c_j - k_j = 0$ at time τ. Since all the clocks increase at the same rate, $c_i - k_i = c_j - k_j$ (i.e., $c_i - c_j = k_i - k_j$) is also satisfied before and after τ. This equality replaces $\Delta C_q(c_i, c_j)$. Intuitively, from the observation of the simultaneity of $Exp(c_i; k_i)$ and $Exp(c_j; k_j)$, we can replace $\Delta C_q(c_i, c_j)$ by a more precise information.

Lines 6-7 : Let τ be the time of occurrence of $\mathcal{E}$. For every $Exp(c_i; k_i) \in \mathcal{E}$, $c_i - k_i = 0$ at time τ. For every $Exp(c_j; k_j) \in EXP(q) \setminus \mathcal{E}$, $c_j - k_j < 0$ at time τ, because we are in the case where the expirations of $EXP(q) \setminus \mathcal{E}$ occur *after* $\mathcal{E}$. This case is possible because $\mathcal{E}$ is enabled in q. Therefore $c_j - k_j < c_i - k_i$ at time τ. Since all the clocks increase at the same rate, $c_j - k_j < c_i - k_i$ (i.e., $k_i - k_j < c_i - c_j$) is also satisfied before and after τ. This ΔClock-Cond is *combined* with $\Delta C_q(c_i, c_j)$. For example, if $\Delta C_q(c_i, c_j)$ is in the form $m_1 < c_i - c_j < m_2$, then the result of *combination* is $\max(m_1, k_i - k_j) < c_i - c_j < m_2$. Intuitively, from the observation that $Exp(c_i; k_i)$ is before $Exp(c_j; k_j)$, we replace $\Delta C_q(c_i, c_j)$ by a more precise information.

Line 8 : ΔC_r is emptied when all the clocks are inactive, because in this case ΔClock-Conds are irrelevant (and thus, useless).

5.3.2 Study of transitions of types 2 and 3

Let q be a state of the SEA $B = \mathit{SetExp}(A) = \mathit{StepTwo}(A')$, ℓ_T consist of σ and Set events (if any) that label a transition T in A', and $\mathcal{E}$ be a set of expiration(s).

The following lemma can be used to determine whether ℓ_T is enabled in q.

Lemma 5.2 [5] $(q \xrightarrow{\ell_T})!$ iff*:*

(i) $\ell_T \in OUT(L_q)$;

(ii) $\forall$"$< \mathit{Exp}(c_i;k)$" *or* "$\leq \mathit{Exp}(c_i;k)$" $\in \mathcal{G}_T$, $\exists k' \leq k$ *such that* $C_q(c_i) = (u < c_i < k')$;

(iii) $\not\exists$"$= \mathit{Exp}(c_i;k)$" $\in \mathcal{G}_T$;

(iv) $\forall$"$> \mathit{Exp}(c_i;k)$" **or** "$\geq \mathit{Exp}(c_i;k)$" $\in \mathcal{G}_T$, $\exists k' \geq k$ *such that* $C_q(c_i) = (k' < c_i < u)$ **or** $C_q(c_i) = (k' < c_i)$.

The following lemma can be used to determine whether $(\mathcal{E}, \ell_T)$ (i.e., label of a transition of Type 3) is enabled in q:

Lemma 5.3 [6] $(q \xrightarrow{\mathcal{E},\ell_T})!$ iff*:*

(i) $\ell_T \in OUT(L_q)$;

(ii) $(q \xrightarrow{\mathcal{E}})!$;

(iii) $\forall$"$< \mathit{Exp}(c_i;k)$" $\in \mathcal{G}_T$: $\mathit{Exp}(c_i;k) \notin \mathcal{E}$, **and** $\exists k' \leq k$ *such that* $C_q(c_i) = (u < c_i < k')$;

(iv) $\forall$"$\leq \mathit{Exp}(c_i;k)$" $\in \mathcal{G}_T$, $\exists k' \leq k$ *such that* $C_q(c_i) = (u < c_i < k')$;

(v) $\forall$"$= \mathit{Exp}(c_i;k)$" $\in \mathcal{G}_T$: $\mathit{Exp}(c_i;k) \in \mathcal{E}$, **and** $C_q(c_i) = (u < c_i < k)$;

(vi) $\forall$"$> \mathit{Exp}(c_i;k)$" $\in \mathcal{G}_T$, $\exists k' \geq k$ *such that* $C_q(c_i) = (k' < c_i < u)$ **or** $C_q(c_i) = (k' < c_i)$;

(vii) $\forall$"$\geq \mathit{Exp}(c_i;k)$" $\in \mathcal{G}_T$:
$\exists k' \geq k$ *such that* $C_q(c_i) = (k' < c_i < u)$ **or** $C_q(c_i) = (k' < c_i)$; **or**
$\mathit{Exp}(c_i;k) \in \mathcal{E}$ **and** $C_q(c_i) = (u < c_i < k)$.

Let TR be a transition labeled $(\mathcal{E}, \ell_T)$ of Type 2 or 3 ($\mathcal{E} = \emptyset$ if TR is of Type 2). When $(\mathcal{E}, \ell_T)$ is enabled in q (noted $(q \xrightarrow{(\mathcal{E},\ell_T)})!$), let us express

[5] Proof in Section A.2
[6] Proof in Section A.3

formally the procedure for computing the state r, denoted $(q \xrightarrow{\mathcal{E},\ell_T})$, reached from q after the execution of TR:

```
1. $r \coloneqq (q \xrightarrow{\mathcal{E}})_{1\to 7}$        /* Execution of *Lines 1-7* of the procedure of Sect. 5.3.1 */
2. $L_r \coloneqq$ destination location of $T$
3. $\forall Set(c_i; k_1, k_2, \cdots, k_m) \in \mathcal{Z}_T$ (and thus, $c_i \in Z_T$):
4.     $\Delta C_r \langle c_i \mapsto \theta_i \rangle$
5.     $\nexists Exp(c_i; k) \in \mathcal{E} \Rightarrow \Delta C_r \coloneqq \Delta C_r \cup \{C_r(c_i)\langle c_i \mapsto \theta_i - c_i \rangle\}$
6.     $\exists Exp(c_i; k) \in \mathcal{E} \Rightarrow \Delta C_r \coloneqq \Delta C_r \cup \{\theta_i - c_i = k\}$
7.     $\forall c_j \notin Z_T$ (i.e., $\nexists Set(c_j; *) \in \mathcal{Z}_T$):
8.         $\nexists Exp(c_j; k) \in \mathcal{E} \Rightarrow \Delta C_r \coloneqq \Delta C_r \cup \{C_r(c_j)\langle c_j \mapsto c_j - c_i \rangle\}$
9.         $\exists Exp(c_j; k) \in \mathcal{E} \Rightarrow \Delta C_r \coloneqq \Delta C_r \cup \{c_j - c_i = k\}$
10.    $C_r(c_i) \coloneqq (0 < c_i < k_1)$
11.    $K_r(c_i) \coloneqq k_2 \cdots k_m$
12. $\forall c_i, c_j \in Z_T$: $\Delta C_r \coloneqq \Delta C_r \cup \{c_j - c_i = 0\}$
13. $\forall \theta_i$: $\Delta C_r \langle \cancel{\theta}_i \rangle$.
14. $r \coloneqq (q \xrightarrow{\mathcal{E}})_8$        /* Execution of *Line 8* of the procedure of Sect. 5.3.1 */
```

Let us give some explanations of the above computation of $(q \xrightarrow{\mathcal{E},\ell_T})$.

***Lines* 1 and 14** : The simultaneity of $\mathcal{E}$ and T is conceptually equivalent to the execution of $\mathcal{E}$ which is *immediately* followed by T. We first determine the intermediate (virtual) state v reached from q after the execution of $\mathcal{E}$. We compute v by using $(q \xrightarrow{\mathcal{E}})$ without its last *Line 8*. The latter is used at the end of $(q \xrightarrow{\mathcal{E},\ell_T})$ (in *Line 14*) because it is used only to finalize the computation of a "real" state. The aim of *Lines 1-13* is to determine the (real) state r reached from v after the execution of T. In $(q \xrightarrow{\mathcal{E},\ell_T})$, v and r are indicated by the same variable r, because the intermediate state is directly modified for obtaining the final reached state. For the sake of clarity, in the following explanations, the two states are indicated differently by v and r.

Lines 2-13 are executed only if $\ell_T \neq \sharp$, because when $\ell_T = \sharp$, the transition is of Type 1 and the reached state $(q \xrightarrow{\mathcal{E}})$ is computed by Lines 1 and 14 (i.e., the procedure of Sect. 5.3.1). The following explanations assume $\ell_T \neq \sharp$.

Line 2 : The location of the reached state is the destination location of T.

Lines 3-13 : If T resets no clock, then T has no influence on the values of clocks and State r is determined by Lines 1, 2 and 14. The aim of *Lines 3-13* is to compute the effect of the resets of T on r.

Lines 3-11 : The aim is to compute the effect on r, of the reset of each clock.

Line 4 : For every c_i that is reset by T, we use a fictitious clock θ_i that is equal to c_i (before the reset of c_i) and that is not reset by T. Therefore, all the ΔClock-Conds using c_i that were satisfied before T, will continue to be satisfied after T if we replace c_i by θ_i.

Line 5 : Let τ be the time when T is executed and c_i is reset. If c_i does not expire at time τ, then $C_v(c_i)$ is satisfied before the reset of c_i and would be satisfied at instant τ if c_i was not reset. At time τ, θ_i has the value c_i would have if it was not reset, and c_i is null. Therefore at time τ, $\theta_i - c_i$ has the value c_i would have if it was not reset. And since $\theta_i - c_i$ is constant $\underline{\text{from } \tau \text{ and before another reset of } c_i}$, we deduce that in this period, $\theta_i - c_i$ has the value c_i would have at time τ if it was not reset. Therefore, $C_v(c_i)$ is satisfied after T if we replace c_i by $\theta_i - c_i$. This information is inserted into ΔC_r.

Line 6 : Let τ be the time when T is executed. $\theta_i = k$ at time τ because $Exp(c_i; k)$ occurs at time τ, and $c_i = 0$ at time τ because c_i is reset at time τ. Therefore, $\theta_i - c_i = k$ at time τ. Since $\theta_i - c_i$ is constant $\underline{\text{from } \tau \text{ and before another reset of } c_i}$, we deduce that $\theta_i - c_i = k$ in this period. This information is inserted into ΔC_r.

Lines 7-9 are associated to a c_i reset by T and consider every c_j not reset by T.

Line 8 : Let τ be the time when T is executed and c_i is reset. If c_j (*not* reset by T) does not expire at τ, then $C_v(c_j)$ is satisfied at time τ. At time τ, $c_j - c_i$ has the value of c_j, because c_i is reset at time τ. Since $c_j - c_i$ is constant $\underline{\text{from } \tau \text{ and before another reset of } c_i \text{ or } c_j}$, we deduce that in this period, $c_j - c_i$ has the value that c_j has at time τ.

Therefore, $C_v(c_j)$ is satisfied after T if we replace c_j by $c_j - c_i$. This information is inserted into ΔC_r.

Line 9 : Let τ be the time when T is executed. $c_j = k$ at time τ because $Exp(c_j; k)$ occurs at time τ, and $c_i = 0$ at time τ because c_i is reset at time τ. Therefore $c_j - c_i = k$ at time τ. Since $c_j - c_i$ is constant $\underline{\text{from } \tau \text{ and before another reset of } c_i \text{ or } c_j}$, we deduce that $c_j - c_i = k$ in this period. This information is inserted into ΔC_r.

Lines 10-11 : We insert in C_r the Clock-Cond that specifies the status of c_i just after its reset. If $Set(c_i; k_1, \cdots, k_m)$ is the *Set* event of c_i that is associated to T, then: in *Line 10* we insert the information that c_i has been reset and that its next foreseen expiration is $Exp(c_i; k_1)$, and in *Line 11* we insert the information which specifies the expirations of c_i that are foreseen *after* $Exp(c_i; k_1)$.

Line 12 : we insert into ΔC_r the information that the clocks reset by T are equal with each other after the execution of T.

Line 13 : The use of θ_i is just a trick to deal with clocks that are reset while they are active. Now, we can eliminate every θ_i from ΔC_r. For that purpose, every pair of ΔClock-Conds in ΔC_r that use a θ_i, are added with each other in order to eliminate θ_i. (See Item 7 in Notations 5.3 for more details about elimination of θ_i.)

5.4 Construction of SEA

The principle of the iterative procedure for computing $B = \mathit{StepTwo}(A')$ (where $A' = \mathit{StepOne}(A)$) is explained in 5.2. The basis of this procedure can be formally expressed by the following fixpoint characterization:

Let us set the SEA $B_0 = (Q_0, \Gamma_0, \delta_0, q^0) = (\{q^0\}, \emptyset, \emptyset, q^0)$, where q^0 has been specified in Sect. 5.2. And let us consider the suite of SEA $B_i = (Q_i, \Gamma_i, \delta_i, q^0)$, defined by $B_{i+1} = \Omega(B_i)$, for $i \geq 0$, where the operator Ω: SEA$\rightarrow$ SEA is defined by:

$$Q_{i+1} = Q_i \cup \{(q \xrightarrow{\ell}) : q \in Q_i, \ell \in (2^{EXP(q)} \times OUT(L_q)) \setminus \{(\emptyset, \sharp)\}, (q \xrightarrow{\ell})!\}$$

$$\delta_{i+1} = \delta_i \cup \{q \xrightarrow{\ell} r : q \in Q_i, (q \xrightarrow{\ell})!, r = (q \xrightarrow{\ell}) \in Q_{i+1}\}$$

$$\Gamma_{i+1} = \Gamma_i \cup \{\ell : \exists (q \xrightarrow{\ell} r) \in \delta_{i+1}\}$$

In the above expressions of δ_{i+1} and Γ_{i+1}, $q \xrightarrow{\ell} r$ denotes a transition from state q to state r which is labeled ℓ.

Lemma 5.4 [7] *Ω has a fixpoint that is obtained after a* finite *number of iterations. Formally: $\exists p \geq 0$ s.t. $\forall n \geq p$: $B_n = B_p$.*

The SEA $B = \mathit{StepTwo}(A')$ is indeed the fixpoint of Ω.

6 Correctness and properties of *SetExp*

6.1 Correctness

In this section, we present a theorem which states correctness of *SetExp*.

Let L_B^{SEA} denote the regular language of a SEA B. Each sequence of L_B^{SEA} is in the form "$E_1 E_2 \cdots E_i \cdots$", where each E_i is the label of a transition of B, and thus, consists of one event or several (simultaneous) events (see Sect. 3.3).

Definition 6.1 *The* timed language of a SEA *B (TL_B^{SEA}) is defined as follows, where τ_i are real values: $(E_1, \tau_1)(E_2, \tau_2) \cdots \in TL_B^{SEA}$* iff*:*

[7] Proof in Section A.4

$E_1E_2\cdots \in L_B^{SEA}$ (see Def. 3.1), $\quad 0<\tau_1<\tau_2<\cdots,\quad$ and
$\forall E_i, E_j,\ i<j$: **If** *E_i contains $Set(c;k)$, E_j contains $Exp(c;k)$, and no E_m $(i<m<j)$ contains $Set(c;*)$,*
Then: *$\tau_j=\tau_i+k$.*

That is, TL_B^{SEA} contains every timed sequence obtained from a sequence of L_B^{SEA} by: associating a time to each event, and respecting the consistency condition (i.e., each $Set(c;k)$ and corresponding $Exp(c;k)$ are separated by the delay k).

Definition 6.2 *A TA A is said* equivalent *to a SEA B* iff *TL_A^{TA} (see Def. 2.1) is obtained from TL_B^{SEA} by removing all the Set and Exp events.*

Intuitively, a TA A is equivalent to a SEA B *iff* the behaviors of A and B cannot be differentiated by an observer who does not see (or ignores) *Set* and *Exp* events.

Theorem 6.1 [8] *Every TA A is equivalent to its corresponding SEA $SetExp(A)$.*

Theorem 6.1 states correctness of *SetExp* and implies the possibility to transform a study of a system modelled by a TA A into a non-real-time form (i.e., $SetExp(A)$), and thus, we can adapt non-real-time methods of study. This idea has been used in conformance testing [7,10] and supervisory control [11,8,9]. Therefore, this theorem confirms correctness of the methods in [7,10,11,8,9].

6.2 Properties

Property 6.1 *(General property) If in a TA, we multiply by the same value K all the constants used in the guards, the results of Steps 1 and 2 are not modified, modulo the multiplication by K of every constant used in a Set or Exp event.*

Property 6.1 can be intuitively justified as follows: multiplying all the constants of guards by the same value is equivalent to changing the unit of time, which has no influence on the generation of *Set* and *Exp* events. Here is a mathematical justification: multiplying by the same value all the constants of guards, implies multiplying by the same value all the constants of the inequations (Clock-Conds, ΔClock-Conds) defining the states of SEA, which has no influence on the solutions of these inequations.

Property 6.1 is interesting because: (i) state space in the obtained SEAs does not increase when we multiply by the same value all the constants used in the transitions' guards, and (ii) if *SetExp* has been applied to generate a SEA

[8] Proof in Section A.5

B from a TA A, we can deduce straightforwardly every SEA corresponding to any TA A' that is similar to A modulo a multiplication of constant(s) used in guards.

We define now a more interesting property but that holds only for certain TA, for instance the TA of Fig. 1, and we consider a class of TAs for which this property holds.

Property 6.2 *(Property of a class of TA) We consider a class of TAs such that for every TA 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 of 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(A) is not modified, modulo the replacement of every k_i by m_i.*

Property 6.2 holds for the TA of Figure 1, that is, if we replace the three values $1, 2, 3$ used in the guards of the transitions, by any other integer values k, m, n respectively, such that $k < m < n$, the results of Figures 3 and 4 are not modified, modulo the replacement of $1, 2, 3$ by k, m, n in *Set* and *Exp* events. Note that we have studied several examples, and this property is satisfied in most of them.

Property 6.2 is more interesting than Property 6.1 because it does not restrict the modification of constants of guards by multiplying all of them by the *same* value. The determination of a class of TA in which Property 6.2 holds, is not trivial and we intend to study it in a near future.

Remark 6.1 *(About Property 6.2)*
In all the examples of TA we have studied, we note that, even for a TA where Property 6.2 does not hold, the state space of the corresponding SEA can change but its size does not increase significantly with the magnitudes of constants used in guards.

Properties 6.1 and 6.2, and Remark 6.1 show an advantage of using SEA instead of RA. In fact, contrary to RA, *in practice* the state space of SEA does not increase with the magnitudes of the constants used in timing constraints. We have used the term "in practice" because Remark 6.1 holds for all the examples we have studied, but we have no guarantee that it holds for every TA. We intend to study this aspect in a near future.
Let us now give an upper bound of the number of states of a SEA B obtained from a TA A (i.e., $B = SetExp(A)$). Let k be the greatest constant used in timing constraints. For each clock c_i, let us consider the distinct *Set* events associated to c_i, and let $p_{i,j}$ be the number of constants in each of these *Set* events, for $j = 1, \cdots$. Let then $p_i = \sum_{j=1,\cdots} p_{i,j}$ and $p = \max_{i=1,\cdots,N_c} p_i$. And recall that $|\mathcal{L}|$ and $|\mathcal{C}|$ are the numbers of locations and clocks of A, respectively.

Lemma 6.1 [9] *$|\mathcal{L}|p^{|\mathcal{C}|}2^{k|\mathcal{C}|^2}$ is an upper bound of the number of states of B.*

Lemma 6.1 presents a coarse upper bound which is never reached. We intend in a near future to compute a more accurate upper bound.

7 Conclusion

We propose a method *SetExp* that transforms a TA A into an equivalent SEA B which uses two types of events: *Set* and *Exp*, in addition to the events of the TA. A and B are *equivalent*, in the sense that they specify the same order and timing constraints. Applicability of *SetExp* has been demonstrated in [7,10,11,8,9], where *SetExp* is used as a black-box. In the present study, we show *how SetExp* can be realized.

In the near future, we intend to investigate the following issues:

- A software tool implementing *SetExp* has been recently realized. We intend to use this tool to implement a conformance testing method based on [10] and a supervisory control method based on [9].
- In the architectures proposed in [7,10,11,8,9], *SetExp* is applicable for *centralized* systems. It would be interesting to design a transformation applicable for distributed systems.
- The state space of SEA increases exponentially with the number of clocks, and in practice this state space does not increase significantly with the magnitudes of the constants used in timing constraints. It is interesting to quantify more rigorously how the state space increases with the magnitudes of constants. And it is also interesting to study more rigorously the complexity of *SetExp* in terms of memory and time that are necessary for the computation of SEA.
- Determination of a class of TA for which Property 6.2 is satisfied.
- For the sake of simplicity, we have considered only TA without invariants. We intend to develop an extension of *SetExp* supporting invariants.

References

[1] Alur, R., *Timed automata*, in: *Proc. 11th Intern. Conf. on Comp. Aided Verif. (CAV)* (1999), pp. 8–22.

[2] Alur, R., C. Courcoubetis and D. Dill, *Model checking for real-time systems*, in: *Proc. IEEE Sympos. on Logic in Computer Science*, 1990.

[3] Alur, R., C. Courcoubetis, N. Halbwachs, D. Dill and H. Wong-Toi, *Minimization of timed transitions systems*, in: *CONCUR* (1992), pp. 340–354.

[9] Proof in Section A.5.1

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

[5] Kang, I. and I. Lee, *State minimization for concurrent system analysis based on state space exploration*, in: *Proc. Conf. On Computer Assurance*, 1994, pp. 123–134.

[6] Kang, I. and I. Lee, *An efficient state space generation for analysis of real-time systems*, in: *Proc. Intern. Sympos. on Soft. Testing and Analysis (ISSTA'96)*, 1996.

[7] Khoumsi, A., *A method for testing the conformance of real time systems*, in: *Proc. 7th Intern. Sympos. on Formal Techn. in Real-Time and Fault Toler. Systems (FTRTFT)*, Oldenburg, Germany, 2002,
http://www.gel.usherb.ca/khoumsi/Research/Public/FTRTFT02.ps.

[8] Khoumsi, A., *Supervisory control of dense real-time discrete-event systems with partial observation*, in: *Proc. 6th Intern. Workshop on Discrete Event Systems (WODES)*, Zaragoza, Spain, 2002,
http://www.gel.usherb.ca/khoumsi/Research/Public/WODES02.ps.

[9] Khoumsi, A., *Supervisory control for the conformance of real-time discrete-event systems*, in: *Proc. 7th Intern. Workshop on Discrete Event Systems (WODES)*, Reims, France, 2004,
http://www.gel.usherb.ca/khoumsi/Research/Public/WODES04.ps.

[10] Khoumsi, A., T. Jéron and H. Marchand, *Test cases generation for nondeterministic real-time systems*, in: *Proc. Formal Approaches to TEsting of Software (FATES), LNCS 2931* (2003),
http://www.gel.usherb.ca/khoumsi/Research/Public/FATES03.ps.

[11] Khoumsi, A. 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, 2002,
http://www.gel.usherb.ca/khoumsi/Research/Public/RTCSA02-Cont.ps.

[12] Ramadge, P. and W. Wonham, *The control of discrete event systems*, Proc. IEEE **77** (1989), pp. 81–98.

[13] Tretmans, J., *Test generation with inputs, outputs and repetitive quiescence*, Software-Concepts and Tools **17** (1996).

[14] Tripakis, S. and S. Yovine, *Analysis of timed systems based on time-abstracting bisimulations*, in: *Proc. 8th Intern. Conf. on Comp. Aided Verif. (CAV)* (1995), pp. 229–242.

[15] Yannakakis, M. and D. Lee, *An efficient algorithm for minimizing real-time transition systems*, in: *Proc. 5th Conf. on Comp. Aided Verif. (CAV)* (1993), pp. 210–224.

A Proofs

In the following, by "is satisfied", we mean "evaluates to *True*".

A.1 Proof of Lemma 5.1

Let us define :

$X_1 = \{c_i - k_i < 0 : Exp(c_i; k_i) \in \mathcal{E}\}$,
$X_2 = \{c_i - k_i = c_j - k_j : Exp(c_i; k_i), Exp(c_j; k_j) \in \mathcal{E}\}$,
$X_3 = \{c_i - k_i < c_j - k_j : Exp(c_i; k_i) \in EXP(q) \setminus \mathcal{E}, Exp(c_j; k_j) \in \mathcal{E}\}$.
We have therefore : $X = \Delta C_q \cup X_1 \cup X_2 \cup X_3$.

(i) Since all the clocks progress at the same rate, the system $X_1 \cup X_2 \cup X_3$ has positive solutions *iff* with the passing of time and if no clock is reset, the following items 2 and 3 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 2 holds *when and only when* the expirations of $\mathcal{E}$ occur simultaneously.

(v) Item 3 holds *while and only while* no expiration of $EXP(q) \setminus \mathcal{E}$ has occurred.

(vi) Items 1, 4 and 5 mean that the system $X_1 \cup X_2 \cup X_3$ has positive solutions *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 6 implies that the system $X = \Delta C_q \cup (X_1 \cup X_2 \cup X_3)$ has positive solutions *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. □

A.2 Proof of Lemma 5.2

(i) Item 3 of Lemma 5.2 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 Lemma 5.2 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 1 and 2 mean that every guard of T is satisfied in q.

(iv) Item 1 of Lemma 5.2 and the above Items 1 and 3 mean that T can be executed at any time in q, without being simultaneous to any expiration. □

A.3 Proof of Lemma 5.3

(i) In Item 3 of Lemma 5.3 :
"$C_q(c_i) = (u < c_i < k')$, 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 Lemma 5.3 implies that such a guard "$< Exp(c_i; k)$" remains satisfied at the occurrence of $\mathcal{E}$.

(ii) In Item 4 of Lemma 5.3 :
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 Lemma 5.3 imply that $\mathcal{E}$ occurs before or contains $Exp(c_i; k')$.
The above a) and b) imply that every guard "$\leq Exp(c_i; k)$" of T is and *remains* satisfied at the occurrence of $\mathcal{E}$.

(iii) Item 5 of Lemma 5.3 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 $Exp(c_i; k) \in \mathcal{E}$.

(iv) Item 6 of Lemma 5.3 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 $\mathcal{E}$.

(v) In Item 7 of Lemma 5.3, 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 Lemma 5.3 means a) or b) holds.

(vi) In the above Items 1, 2, 3, 4, 5a, and 5b, all the guards of T are considered.

(vii) The above items 1, 2, 4, and 5a correspond to the guards of T that are satisfied in q. These guards remain satisfied at the occurrence of $\mathcal{E}$.

(viii) The above items 3 and 5b 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 $\mathcal{E}$.

(ix) Item 2 of Lemma 5.3 and the above Item 8 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 $\mathcal{E}$.

(x) The above items 6, 7 and 8 mean that there is no non-satisfied guard of T that will not be satisfied with the passing of time.

(xi) The above items 7, 9, and 10 and Item 1 of Lemma 5.3 mean that the simultaneous executions of T and $\mathcal{E}$ is possible in q. $\square$

A.4 Proof of Lemma 5.4

We have defined the suite $B_{i+1} = \Omega(B_i)$, for $i \geq 0$. Let Q_i, Γ_i, δ_i be such that $B_i = (Q_i, \Gamma_i, \delta_i, q^0)$. We have, by definition of the operator Ω (see Sect. 5.4):

$Q_{i+1} = Q_i \cup \Delta Q_i, \quad \delta_{i+1} = \delta_i \cup \Delta\delta_i, \quad \Gamma_{i+1} = \Gamma_i \cup \Delta\Gamma_i,$

where :

$\Delta Q_i = \{(q \xrightarrow{\ell}) : q \in Q_i, \ell \in (2^{EXP(q)} \times (OUT(L_q) \cup \{\sharp\})) \setminus \{(\emptyset, \sharp)\}, (q \xrightarrow{\ell})!\}$

$\Delta\delta_i = \{q \xrightarrow{\ell} r : q \in Q_i, (q \xrightarrow{\ell})!, r = (q \xrightarrow{\ell}) \in Q_{i+1}\}$

$\Delta\Gamma_i = \Gamma \cup \{\ell : \exists (q \xrightarrow{\ell} r) \in \delta_{i+1}\}$.

We will also use the notation $|U|$ for the *cardinal* (i.e., number of elements) of a set U. In order to prove Lemma 5.4, let us prove the following five lemmas :

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

Lemma A.2 $\forall p \geq 0 : ((Q_p = Q_{p+1}) \Rightarrow (B_{p+2} = B_{p+1}))$.

Lemma A.3 $\forall p \geq 0 : ((Q_p = Q_{p+1}) \Rightarrow (\forall n > p \ : B_n = B_{p+1}))$.

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

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

A.4.1 Proof of Lemma A.1

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

A.4.2 Proof of Lemma A.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 :
$\Delta Q_p = \{(q \xrightarrow{\ell}) : q \in Q_p, \ell \in (2^{EXP(q)} \times (OUT(L_q) \cup \{\sharp\})) \setminus \{(\emptyset, \sharp)\}, (q \xrightarrow{\ell})!\}$,
$\Delta Q_{p+1} = \{(q \xrightarrow{\ell}) : q \in Q_{p+1}, \ell \in (2^{EXP(q)} \times (OUT(L_q) \cup \{\sharp\})) \setminus \{(\emptyset, \sharp)\}, (q \xrightarrow{\ell})!\}$,

(iii) Item 2 implies that if $Q_p = Q_{p+1}$, then $\Delta Q_p = \Delta Q_{p+1}$.

(iv) Items 1 and 3 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 : $\Delta\delta_p = \{q \xrightarrow{\ell} r : q \in Q_p, (q \xrightarrow{\ell})!, r = (q \xrightarrow{\ell}) \in Q_{p+1}\}$, and $\Delta\delta_{p+1} = \{q \xrightarrow{\ell} r : q \in Q_{p+1}, (q \xrightarrow{\ell})!, r = (q \xrightarrow{\ell}) \in Q_{p+2}\}$.

(vii) Items 4 and 6 imply that if $Q_p = Q_{p+1}$, then $\Delta\delta_p = \Delta\delta_{p+1}$.

(viii) Items 5 and 7 imply that if $Q_p = Q_{p+1}$, then $\delta_{p+2} = \delta_{p+1}$.

(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 : $\Delta\Gamma_p = \Gamma \cup \{\ell : \exists(q \xrightarrow{\ell} r) \in \delta_{p+1}\}$, and
$\Delta\Gamma_{p+1} = \Gamma \cup \{\ell : \exists(q \xrightarrow{\ell} r) \in \delta_{p+2}\}$.

(xi) Items 8 and 10 imply that if $Q_p = Q_{p+1}$, then $\Delta\Gamma_p = \Delta\Gamma_{p+1}$.

(xii) Items 9 and 11 imply that if $Q_p = Q_{p+1}$, then $\Gamma_{p+2} = \Gamma_{p+1}$.

(xiii) Items 4, 8 and 12 imply that if $Q_p = Q_{p+1}$, then : $B_{p+2} = B_{p+1}$. □

A.4.3 Proof of Lemma A.3

(i) Let us consider a $p \geq 0$ such that $Q_p = Q_{p+1}$. Lemma A.2 implies that $B_{p+2} = B_{p+1}$.

(ii) Let us us consider a $n \geq p+2$ such that $B_n = B_{n-1}$. Therefore, $Q_n = Q_{n-1}$.
From Lemma A.2, we deduce that $B_{n+1} = B_n$.

(iii) Items 1 and 2 imply by induction that :
if $(Q_p = Q_{p+1})$ for a given $p \geq 0$, then $\forall n \geq p+2 \ : B_n = B_{n-1}$.

(iv) Item 3 is equivalent to :
if $Q_p = Q_{p+1}$ for a given $p \geq 0$, then $\forall n \geq p+2 \ : B_n = B_{p+1}$.

(v) Since $B_n = B_{p+1}$ for $n = p+1$, Item 4 can be written :
if $Q_p = Q_{p+1}$ for a given $p \geq 0$, then $\forall n \geq p+1 \ : B_n = B_{p+1}$. □

A.4.4 Proof of Lemma A.4

Let $A = (\mathcal{L}, \mathcal{A}, \mathcal{C}, \mathcal{T}, l_0)$ be the TA from which we construct $B_0, B_1, B_2, \cdots$. Recall that each state of SEA, and thus of B_i, is defined by three parts :

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 ΔClock-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 2 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 ki, 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 $p = \max_{i=1,\cdots,N_c} p_i$. It can be easily proved that U can be taken equal to p.

(iv) Item 3 and the fact that the number of clocks of 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 guards of A, then every ΔClock-Cond using $c_i - c_j$ is expressed by using (integer) values that fall within the interval $[-M_j, M_i]$.

(vi) Items 2 and 5 imply that for each pair of clocks $c_i, c_j \in \mathcal{C}$, the number of possible ΔClock-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 $\leq 2k$, it can be easily proved that V can be taken equal to 2^{2k}, where k is the greatest constant used in timing constraints.

(vii) Item 6 and the fact that the number of pairs of clocks of A is finite ($< \frac{|\mathcal{C}|^2}{2}$) imply that $2^{2k\frac{|\mathcal{C}|^2}{2}}$ ($= 2^{k|\mathcal{C}|^2}$) is a (finite) upper bound of the cardinal of Part 3.

(viii) Items 1, 4 and 7 imply that $|\mathcal{L}|p^{|\mathcal{C}|}2^{k|\mathcal{C}|^2}$ is a (finite) upper bound of the number of states of every B_i. $\square$

A.4.5 Proof of Lemma A.5

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

A.4.6 Proof of Lemma 5.4

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

(ii) Item 1 and Lemma A.2 imply $\exists p \geq 0$ such that $\forall n > p \ : B_n = B_{p+1}$. $\square$

A.5 Proof sketch of Theorem 6.1

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

(i) Step 1 of *SetExp* relabels transitions of the TA A without changing its structure. The fact to reset a clock c in a transition Tr1, in order to compare further in a transition Tr2 the value of c with k, is clearly *equivalent* to :
resetting c and programming it such that it expires after a delay k (Substep 1a), and then checking in Tr2 whether the expiration of c has occurred (Substep 1b). Therefore, A and A' 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 (Lemma 5.4) and necessitates in each iteration :
a) to determine the labels enabled in every state, and
b) to construct the states reached by these enabled transitions.
Therefore, this fix-point method generates a correct result *iff* (a) and (b) are realized correctly.

(iii) Lemmas 5.1, 5.2 and 5.3 imply that the labels enabled in every state are determined correctly.

(iv) The detailed explanations of the procedures in Sects. 5.3.1 and 5.3.2 can be considered as a proof sketch that the states reached by the enabled labels are constructed correctly.

(v) Items 2, 3 and 4 imply that the fixpoint method is correct, i.e., Step 2 of *SetExp* is correct. Therefore, B accepts all and only the possible order of events that respects the specification of A'.

(vi) Item 5 and the fact that TL_B^{SEA} (see Def. 6.1) models the behaviour of B, mean that TL_B^{SEA} contains all and only the possible order of events that respects the specification of A'.

(vii) Items 1 and 6 mean that TL_A^{TA} (Def. 2.1) is obtained from TL_B^{SEA} by removing all *Set* and *Exp* events, i.e., A and B are equivalent. □

A.5.1 Proof of Lemma 6.1

See Proof of Lemma A.4.