SUPERVISORY CONTROL OF REAL-TIME DISCRETE EVENT SYSTEMS
MODELED BY TIMED AUTOMATA WITH INVARIANTS

Ahmed Khoumsi, Lucien Ouedraogo
Department of Electrical and Computer Engineering
University of Sherbrooke, Canada
email: Ahmed.Khoumsi@USherbrooke.ca

2 Preliminaries and concepts

We present the model used to describe plants and specifica-
tions, as well as the conformance relation used to connect
them. Then, we formalize the control problem to be solved.

2.1 Timed automata, conformance relation

We assume that the plant and its specification can be mod-
eled as Timed Automata (TA) with invariants, but first we
need to define the notion of Clock and Clock constraints.

Definition 2.1 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 \( C = \{c_1, \ldots, c_n\} \) be a (finite) set of clocks.

Definition 2.2 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 \( \Phi_C \) be the (infinite) set of clock
constraints using clocks of \( C \).

Definition 2.3 A TA is syntactically defined by
\( (L, \Sigma, C, I, T, l_0) \) where: \( L \) is a finite set of locations, \( l_0 \) is
the initial location, \( C \) is a finite set of clocks, \( I : L \rightarrow 2^\Phi_C \)
associates to each location \( l \) an invariant \( I_l \in \Phi_C \), and \( \Sigma \)
is a finite set of events (alphabet). \( T \subseteq L \times \Sigma \times L \times 2^{\Phi_C} \times C \)
associates to \( (l, a, l', \Phi, \delta) \) a transition relation, and a transition of TA is defined by
\( T = \langle q; \sigma; r; G; Z \rangle \), where: \( q \) and \( r \) are origin and
destination locations, \( \sigma \) is an event, \( G \) is a finite subset of
\( \Phi_C \) called guard of \( T \), and \( Z \) is a subset of \( C \) called reset of
\( T \). \((2^X \equiv \text{the set of subsets of } X)\).

The example of Fig. 1 illustrates Def. 2.3. Each node
represents a location with its invariant, if any. Each
arrow linking \( q \) to \( r \) and labeled \( (\sigma; G; Z) \) represents a transition
\( (q; \sigma; r; G; Z) \). An empty \( G \) or \( Z \) is represented by “-”.

The semantics of a TA \( A = (L, \Sigma, C, I, T, l_0) \) is:
At time \( t_0 = 0 \), \( A \) is at \( l_0 \) with all clocks equal to 0.
A transition \( Tr = (q; \sigma; r; G; Z) \) is enabled when \( q \) is the current
location and all the clock constraints in \( G \) (if any) evaluate\( t \) to true;
otherwise, \( Tr \) is disabled. From \( q \), \( \sigma \) is executed only when
\( Tr \) is enabled; and after the execution of \( \sigma \), \( r \) is reached and
the clocks in \( Z \) are reset. During an execution of a TA, the
invariant of the current location is always satisfied.

For example, the TA of Fig. 1 is initially in \( l_0 \) and
reaches \( l_1 \) when \( \sigma \) is executed. From \( l_1 \), the TA reaches \( l_2 \)

Figure 1. Example of TA
the time of occurrence of \( P \) of length \( l \) (resp. \( l_3 \)) when \( \alpha \) (resp. \( \beta \)) is executed. From \( l_2 \) (resp. \( l_3 \)), the TA reaches \( l_0 \) when \( \phi \) (resp. \( \rho \)) is executed. Let \( \delta_{u,v} \) be the delay between \( u \) and \( v \). The guards of transitions imply: 
\[
\delta_{\sigma,\alpha} < 3, \quad \delta_{\sigma,\beta} \geq 2, \quad 1 < \delta_{\sigma,\phi} < 2, \quad \delta_{\sigma,\rho} > 3. 
\]
The invariant of \( l_1 \) implies that \( \alpha \) or \( \beta \) is executed before \( c_1 \) evaluates to 3. In \( l_2 \), \( \phi \) is executed when \( c_1 \in [1,2] \) or is never executed.

The semantics of a TA \( A \) can also be defined by the set of timed traces accepted by \( A \). A timed trace is a sequence "\((e_1, \tau_1), \ldots, (e_n, \tau_n)\)" where \( e_1, \ldots, e_n \in \Sigma \), each \( \tau_i \) is the time of occurrence of \( e_i \), and \( 0 < \tau_1 < \cdots < \tau_i < \cdots \).

We will use the following notation:

**Notation 2.1** If \( \lambda \) is a trace and \( X \) is a set of traces: 
\( \overline{X} \) is the set of **finite** prefixes of \( X \), and \( \overline{X} \) is the prefix of \( \lambda \) of length \( i \) (if \( i \) is finite).

Let us define the acceptance by \( A \) of a timed trace:

- The empty timed trace \( \lambda_0 \) is accepted by \( A \) iff: the invariant of \( \lambda_0 \) is satisfied whenever all the clocks evaluate to the same value. Intuitively, \( A \) can be “quiescent” forever from the initial time.

- A **finite** timed trace \( \lambda = (e_1, \tau_1) \cdots (e_n, \tau_n) \) is accepted by \( A \) if there exists a sequence of consecutive transitions \( Tr_1 \cdots Tr_n \) of \( A \) that starts in \( \lambda_0 \) and s.t.:
  1. \( \forall i \in \{1, \ldots, n\} \); the event of \( Tr_i \) is \( e_i \); after the execution of \( \overline{Tr}_{i-1} \), \( Tr_i \) is enabled at time \( \tau_i \) and the invariant of the current location is satisfied at every time \( \tau \in [\tau_{i-1}, \tau_i] \). Note: “current location” is \( l_0 \) if \( i = 1 \), and the destination location of \( Tr_{i-1} \) if \( i > 1 \). Intuitively, \( A \) can execute \( \lambda \).

- After the execution of \( \lambda \), 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, \( A \) can become “quiescent” forever after the execution of \( \lambda \).

- Acceptance of an infinite timed trace \( \lambda \) is defined as acceptance of finite timed trace by: removing Item 2 and replacing \( n \) by \( \infty \).

**Definition 2.4** The Timed language of a TA \( A \) (\( TL_\infty(A) \)) is the set of (finite and infinite) timed traces accepted by \( A \), whereas the Timed quiescence language of \( A \) (\( TL(A) \)) is the set of finite timed traces accepted by \( A \).

**Requirement 2.1** At the time when a location is reached, its invariant must be satisfied [10].

**Requirement 2.2** When the invariant of a current location is on the point of being violated with elapse of time, a transition must be enabled [10].

**Hypothesis 2.1** TA are deterministic, that is, if two transitions \( Tr_1 \) and \( Tr_2 \) execute the same event from the same location and can be enabled at the same time, then they lead to the same location and reset the same clocks.

In the sequel, the plant and its specification are assumed modeled as TAs (\( P \) and \( S \), resp.) over an alphabet \( \Sigma \). We will use the following conformance relation \( conf_{TA} \) which is an inclusion of timed languages of TA:

**Definition 2.5** \((P, conf_{TA}, S) \equiv (TL_\infty(P) \subseteq TL_\infty(S)) \)

### 2.2 The control of a TA

Given a plant and a specification modeled by TAs \( P \) and \( S \), the aim of the control is to design a supervisor \( Sup \) that restricts the behavior of the plant so that it conforms to \( S \). The task of \( Sup \) will be to disable or to force some events in order to achieve this goal. This leads us to define the notions of (un)controllable events and (un)forbid events.

#### 2.2.1 Controllability and forbidity

The alphabet \( \Sigma \) of \( P \) is partitioned into the set of controllable events \( \Sigma_c \) and the set of uncontrollable events \( \Sigma_{uc} \), meaning that events of \( \Sigma_c \) can be disabled by a supervisor whereas events of \( \Sigma_{uc} \) can not be [1]. We also need to consider the notion of forbidity and we partition \( \Sigma \) into the set of forbible events \( \Sigma_{f} \) and the set of unforible events \( \Sigma_{uf} \).

A supervisor can force an event of \( \Sigma_f \) to occur whenever it is enabled, whereas events of \( \Sigma_{uf} \) can not be forced [5]. Overall, we have: \( \Sigma = \Sigma_c \cup \Sigma_{uc} = \Sigma_f \cup \Sigma_{uf} \).

Moreover, Requirement 2.2 needs to be strengthened for \( P \) as follows: (the underlined term has been added)

**Requirement 2.3** When the invariant of a current location is on the point of being violated with elapse of time, a transition with an uncontrollable event must be enabled.

The above requirement is justified by the fact that controllable events can be forbidden by a supervisor, and thus, we cannot rely on them for avoiding invariant violation.

**Notations 2.1** :

- \( \tau_\lambda \) is the time of the last event of a timed trace \( \lambda \).

- Acc\(\overline{\Sigma}^+\)(\( \lambda \)) is the set of events accepted in a TA \( S \) after the execution of some \( \lambda \in TL_\infty(S) \) at any time \( \nu \) such that \( \nu \sim \tau_\lambda \), where \( \tau > \tau_\lambda \) and \( \sim \in \{<,\geq,\leq,\leq,\geq,\geq\} \).

Formally: \( Acc\overline{\Sigma}^+(\lambda) = \{ \sigma, \nu \in TL_\infty(S) \mid \text{for some } \nu \text{ s.t. } \nu > \tau_\lambda \text{ and } \nu \sim \tau \} \).

- Acc\(\overline{\Sigma}^-(\lambda) \) denotes Acc\(\overline{\Sigma}^\overline{\Sigma}-(\lambda) \). That is, the set of all labels accepted in \( S \) after the execution of \( \lambda \).

In the following, \( \lambda \) denotes a timed trace of \( TL_\infty(P) \). Conceptually, a supervisor can be seen as a function \( Sup \) that, for every pair \((\lambda, \tau)\) s.t. \( \tau > \tau_\lambda \), delivers the set of actions that are disabled and the set of those that are forced in \( P \) by the control at time \( \tau \) after the the execution of \( \lambda \). Write \( Sup(P) \) for the closed loop system, consisting of the plant modeled by \( P \) and controlled by \( Sup \).

**Controllability of TA:** A TA \( S \), over the alphabet \( \Sigma \) and s.t. \( TL_\infty(S) \subseteq TL_\infty(P) \), is controllable w.r.t. \((P, \Sigma_{uc}, \Sigma_{uf})\), if \( \forall \lambda \in TL_\infty(S) \) and \( \forall \nu > \tau_\lambda \):

\[
Acc\overline{\Sigma}^+(\lambda) \cap \Sigma_{uc} \subseteq Acc\overline{\Sigma}^\overline{\Sigma}-(\lambda) \quad \text{if: } Acc\overline{\Sigma}^+(\lambda) \neq \emptyset \\
Acc\overline{\Sigma}^+(\lambda) \cap \Sigma_{uf} \neq \emptyset \quad \text{if: } Acc\overline{\Sigma}^+(\lambda) \neq \emptyset, \quad Acc\overline{\Sigma}^+(\lambda) = \emptyset.
\]

Let then \( Conf_{\overline{\Sigma}^+} \) be the set of controllable TAs w.r.t. \((P, \Sigma_{uc}, \Sigma_{uf})\).

Intuitively, \( S \) is controllable w.r.t. \((P, \Sigma_{uc}, \Sigma_{uf})\) if \( S \) can be obtained from \( P \) without forbidding events of \( \Sigma_{uc} \) (Eq. 1) nor forcing events of \( \Sigma_{uf} \) (Eq. 2).
2.2.2 Formalization of the control problem

Objective 1 Given two TAs: the plant \( P \) and the specification \( S \), we have to synthesize a supervisor \( \text{Sup} \) s.t. \( \text{Sup}/P \) is described by a TA (also denoted \( \text{Sup}/P \)) which satisfies:

\[ \text{Sup}/P \in \text{Conf}_P \quad \text{and} \quad \text{Sup}/P \in \text{Conf}_P \]

3 Transformation of the control problem into a non-real-time form

3.1 From a TA to a SEA

Our control problem will be solved in Sect. 4 by using a transformation “\( \text{SetExp}: \text{TA} \to \text{SEA} \)”. \( \text{SetExp} \) transforms a TA into a finite state automaton, called Set-Exp-Automaton (SEA), by adding to the structure of the TA two new types of events: \( \text{Set} \) and \( \text{Exp} \) that will be used to capture the temporal aspect of the TA. [3] presents in detail a first version of events: \( \text{Set} \) and \( \text{Exp} \). \( \text{SetExp} \) presents a new version of \( \text{SetExp} \) applicable to TAs without invariant, and [10] presents a new version of \( \text{SetExp} \) that supports TA with invariants. \( \text{Set} \) and \( \text{Exp} \) have the following meaning:

- \( \text{Set}(c_i, k_1, k_2, \ldots, k_p) \) with \( k_1 < k_2 < \cdots < k_p \), means: clock \( c_i \) is reset to 0 and will expire several times, when its value is equal to \( k_1, k_2, \ldots, k_p \), respectively. A particular case is \( \text{Set}(c_i, k) \).
- \( \text{Exp}(c_i, k) \) means: \( c_i \) evaluates to \( k \) and thus expires.

To illustrate intuitively the principle of the transformation \( \text{SetExp} \), let us consider the following two expressions. Expression 1: a task \( T \) must be realized in less than two units of time. Expression 2: at the beginning of the task an alarm is programmed for occurring 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, \( \text{SetExp} \) allows to obtain the second expression from the first one. The programming of the alarm corresponds to a \( \text{Set} \) event, and the occurrence of the alarm corresponds to an \( \text{Exp} \) event.

Here are the three types of transitions of a SEA \( B = \text{SetExp}(A) \), where \( \sigma \) is an event of the alphabet of \( A \), and \( S \) (resp. \( E \)) is a set of \( \text{Set} \) (resp. \( \text{Exp} \)) events:

- **Type 1**: a transition labeled \( E \) represents the simultaneous occurrences of the \( \text{Exp} \) events of \( S \).
- **Type 2**: a transition labeled \( \sigma \) represents the occurrence of \( \sigma \); a transition labeled \( (\sigma, S) \) represents the simultaneous occurrences of \( \sigma \) and \( S \).
- **Type 3**: a transition labeled \( (E, \sigma) \) represents the simultaneous occurrences of \( E \) and \( \sigma \); and a transition labeled \( (E, \sigma, S) \) represents the simultaneous occurrences of \( E, \sigma \) and \( S \).

Let \( \text{Exp-Trans} \) denote a transition of type 1 or 3.

**Definition 3.1** “An \( \text{Exp-Trans} \) \( Tr \) is preemted” means that \( Tr \) is never executed because another transition with the same origin state as \( Tr \) is executed before \( Tr \). For a state \( q \) of \( \text{SEA} \), the set \( \text{PreemptExp}(q) \) contains the \( \text{Exp} \) event(s) forbidden in \( q \) due to the preemption of \( \text{Exp-Trans}(s) \).

For the TA of Fig. 1, if we ignore the invariant of \( l_1 \), we obtain the SEA of Fig. 2. The shaded node indicates a violation of the invariant of \( l_1 \). To avoid this invariant violation, State \( N \) must be left before the occurrence of \( \text{Exp}(c_1, 3) \), that is, the two outgoing \( \text{Exp-Trans} \) of state \( N \) (dashed arrows) must be cut because preempted by \( \alpha \) or \( \beta \), and we have \( \text{PreemptExp}(N) = \{ \text{Exp}(c_1, 3) \} \).

![Figure 2. SEA obtained from the TA of Fig. 1](image)

A SEA \( B \) is syntactically defined by \((Q, \Gamma, \Pi, N, q^0)\), where \( Q \) is a set of states, \( \Gamma \) is the alphabet, \( \Pi \subseteq Q \times \Gamma \times Q \) is a set of transitions, \( \Pi \) is a function defined by \( \Pi(q) = \text{PreemptExp}(q) \), and \( q^0 \) is the initial state. In order to define the semantics of a SEA \( B = (Q, \Gamma, \Pi, N, q^0) \), its accepted language, let us define the notion of quiescent state:

**Definition 3.2** Given a SEA \( B \) and a state \( q \) of \( B \), \( q \) is a quiescent state whenever it has neither outgoing \( \text{Exp-Trans} \) nor preemted \( \text{Exp-Trans} \).

Intuitively, when \( B \) reaches a quiescent state \( q \), it can remain indefinitely without executing any event.

A trace is a (finite or infinite) sequence “\( E_1 \cdots E_i \cdots \)”, where \( E_i \in \Gamma \). Such a sequence is said “trace of \( B \)” whenever it exists states \( q_0, \ldots, q_k, \ldots \in Q \), such that \( \forall i > 0, (q_{i-1}, E_i, q_i) \in T \).

**Definition 3.3** The language of a SEA \( B \) \( (L_{\infty}(B)) \) contains the finite traces of \( B \) that terminate in a quiescent state, and the infinite traces of \( B \). The quiescent language of \( B \) \( (L(B)) \) only contains the finite traces of \( L_{\infty}(B) \).

Note that \( L_{\infty}(B) \) and \( L(B) \) implicitly respect the following Consistency condition: every \( \text{Set}(c, k) \) and its corresponding \( \text{Exp}(c, k) \) are effectively separated by time \( k \).

3.2 Control architecture of a TA

The supervisor \( \text{Sup} \) will not be directly computed on the TAs modeling the plant and the specification, but rather on a SEA computed from the two TAs. We use the control architecture represented in Fig. 3 that makes the link between \( \text{Sup} \) and the plant. It comprises the environment, the plant, the supervisor, and a Clock-Handler module which mimics the timing aspect of the plant. Formally, we have that:

- Environment interacts with the plant.
• Clock-Handler receives Set events from Sup and sends Exp events to Sup. It can be seen as a Timer module that upon the reception of a Set event, activates a timer and sends back to the supervisor the corresponding Exp event when the timer elapses.

• Sup is derived from an SEA and is tagged with the Set/Exp events coming from this SEA. It observes all the events of the plant, it disables controllable events and forces forcible events of the plant when necessary. The timing constraints that Sup has to deal with are performed via its interaction with the Clock-Handler.

The plant is said under the control of Sup if the latter disables/forces events of the plant. A contrario, the plant is said without the control of Sup if the latter disables/forces no event of the plant. Recall that Sup/P denotes the plant (modeled by the TA P) under the control of Sup.

Let SYST be the system described by the interactions: between the plant and the Environment without the control of Sup, and between Sup and the Clock-Handler. That is, SYST consists of the plant, the Clock-Handler and Sup when the latter forces and disables no event of the plant.

Conceptually, SYST can be seen as an uncontrolled system. When SYST can be modeled by a SEA PP (we will see in Sect. 4.5 (Lemma 4.3) that this is possible), then we write Sup/PP for the closed loop system consisting of SYST controlled by the supervisor Sup. That is, Sup/PP denotes the system described by the interactions: between the plant and the Environment under the control of Sup, and between Sup and the Clock-Handler. Or in other words, Sup/PP consists of the plant, the Clock-Handler and Sup when the latter forces or disables some events of the plant. We can now make the link with the control of a TA of Section 2.2 by the following lemma:

**Lemma 3.1** If Sup/PP can be modeled by a SEA, then Sup/P can be modeled by a TA. □

Lemma 3.1 states that if there exists a SEA that models the controlled SYST, then there exists a TA that models the controlled plant.

### 3.3 Conformance relation between SEAs

Let us define a conformance relation confSEA between two SEAs and make the link with the real-time conformance relation confTA introduced at the end of Sect. 2.1.

**Definition 3.4** Let P’ and S’ be two SEAs over the same alphabet: (P’ confSEA S’) ≡ (L∞(P’) ⊆ L∞(S’)). □

**Proposition 3.1** Let S be a TA: If Sup/P and Sup/PP are described by a TA and a SEA, resp., then: (Sup/PP confSEA SetExp(S)) ⇒ (Sup/P confTA S). □

Prop. 3.1 will contribute in Sect. 4 to transform the control problem into a non-real-time form.

### 4 Resolution of the control problem

Let $P = (\mathcal{L}^P, \Sigma^P, T^P, I^P, 0^P)$ and $S = (\mathcal{L}^S, \Sigma^S, T^S, I^S, 0^S)$ be two TAs modeling the plant and the specification, with $\Sigma = \Sigma_f \cup \Sigma_{ic} = \Sigma_f \cup \Sigma_{nf}$.

#### 4.1 Control method

In order to ensure the conformity between the plant and the specification, we will transform the problem into Supervisory Control Problem over some SEAs derived from $P$ and $S$. Our control method consists of four steps.

- In Step 1, we construct a TA $PS$ that captures the behaviors of $P$ and of $S$ plus error locations modeling unexpected behaviors w.r.t. $P$ or $S$. This operation is necessary to be able (in Step 3) to model the plant and its specification by two SEAs over the same alphabet.
- In Step 2, we abstract away the real-time aspect, by transforming the obtained $PS$ into an SEA. For that purpose, we use the transformation $SetExp$ of [10] without its last step that processes the invariants.
- In Step 3, we extract from this SEA two SEAs $PP$ and $SS$ that model the plant and the specification. Invariants are taken into account in this step.
- Finally, based on these two SEAs, we compute a supervisor ensuring Sup/PP confSEA SS. Prop. 3.1 ensures that Sup/P confTA S.

We use $P$ and $S$ of Fig. 5 to illustrate the four steps of our method, where $\neq x$ means "any event other than $x".

**Figure 3. Control architecture**

**Figure 4. Steps of the control method**

- In Step 1, we construct a TA $PS$ that captures the behaviors of $P$ and of $S$ plus error locations modeling unexpected behaviors w.r.t. $P$ or $S$. This operation is necessary to be able (in Step 3) to model the plant and its specification by two SEAs over the same alphabet.
- In Step 2, we abstract away the real-time aspect, by transforming the obtained $PS$ into an SEA. For that purpose, we use the transformation $SetExp$ of [10] without its last step that processes the invariants.
- In Step 3, we extract from this SEA two SEAs $PP$ and $SS$ that model the plant and the specification. Invariants are taken into account in this step.
- Finally, based on these two SEAs, we compute a supervisor ensuring Sup/PP confSEA SS. Prop. 3.1 ensures that Sup/P confTA S.

We use $P$ and $S$ of Fig. 5 to illustrate the four steps of our method, where $\neq x$ means "any event other than $x".

**Figure 5. Example for illustrating the control method**

#### 4.2 Step 1: Combine $P$ and $S$ into a TA $PS$

We construct a TA $PS$ that models both the plant and the specification, in order to be able to model them by two SEAs over the same alphabet.

**Definition 4.1** The completion $Comp(A)$ of a TA $A = (\mathcal{L}, \Sigma, C, I, T, 0)$ is a TA obtained from $A$ as follows:
1. a trap TL is added to L, by trap we mean that ∀a ∈ Σ, TL has a selfloop labeled (a; −; −); and
2. ∀l ∈ L and ∀e ∈ Σ, a non-specified transition labeled (e; G; −) is added from l to TL; by non-specified we mean that G defines the domain in which e is not enabled in l of A. □

**Definition 4.2** Let A_i = (Σ^i, C_i, T_i, T_1^i, l_0^i), for i = 1, 2, be 2 TAs s.t. C_1 ∩ C_2 = ∅. A_1 ∧ A_2, the synchronized product of A_1 and A_2, is a TA defined by (L, Σ, C, T, l_0) s.t.: C = C_1 ∪ C_2, l_0 = (l_0^1, l_0^2), L = L_1 × L_2. T is s.t.: ∀(q_1; q_2) ∈ L, T((q_1; q_2)) = T_1(q_1) ∨ T_2(q_2).
The set of transitions T is defined as follows:
∀(q_1; q_2), (r_1; r_2) ∈ L, ∀σ ∈ Σ, ∀E ⊆ Φ_σ, ∀R ⊆ C:
1. ((q_1; q_2); σ; (r_1; r_2); E; R) ∈ T iff ∃E_1 ⊆ Φ_σ_1, ∃E_2 ⊆ Φ_σ_2, ∃R_1 ⊆ C_1, ∃R_2 ⊆ C_2, s.t.: R_1 = R_1 ∪ R_2, E = E_1 ∪ E_2,
2. ((q_1; σ; r_1; E_1; R_1) ∈ T_1), ((q_2; σ; r_2; E_2; R_2) ∈ T_2). □

PS is computable by: PS = Comp(P) ∩ Comp(S).

For the example of P and S in Fig. 5, we obtain Comp(P), Comp(S) and PS represented on Fig. 6. In Comp(P), the trap Error_P corresponds to impossible situations w.r.t. P. In Comp(S), the trap Error_S corresponds to undesirable situations w.r.t. S. In PS, Error_P denotes a location (Error_P, l_S) s.t. l_S ≠ Error_S, and Error_S a location (l_P, Error_S) s.t. l_P ≠ Error_P.

**Definition 4.3** An error-location is a location of PS that contains Error_P or Error_S. A border-error-location is an error-location of PS reachable through a single transition from a non-error-location of PS. □

In PS of Fig. 6, we have not represented error-locations that are not border-error-locations, because:

**Lemma 4.1** Error-locations that are not border-error-locations are irrelevant for our control. □

### 4.3 Step 2: SetExp : PS ⇆ PSEA
We abstract away the real-time aspect, by transforming the obtained TA PS into an SEA PSEA. For that purpose, we use the transformation SetExp of [10] without its last step that processes the invariants. Hence, we obtain a SEA PSEA that may contain states violating invariants of P or S. And to each state q of PSEA, are associated two boolean variables V_p[q] and V_s[q] that indicate whether q violates an invariant of P or S, respectively. Moreover, for every state q of PSEA, PreemptExp(q) is set to empty.

**Definition 4.4** To every location (l_P, l_S) of PS correspond one or several states of PSEA. An error-state is a state of PSEA that corresponds to an error-location of PS. E_P denotes an error-state of PSEA corresponding to a location (Error_P, l_S) s.t. l_S ≠ Error_S. E_S denotes an error-state of PSEA corresponding to a location (l_P, Error_S) s.t. l_P ≠ Error_P. E_P,S denotes an error-state of PSEA that corresponds to the location (Error_P, Error_S). A border-error-state is an error-state of PSEA reachable through a single transition from a non-error-state of PSEA. □

For the PS of Fig. 6, we obtain the PSEA of Fig. 7, where V_p (resp. V_s) indicates states violating invariants of P (resp. S). PSEA is represented in two parts: in Part 1, border-error-states E_P are not represented; in Part 2, we represent the transitions whose destination is a border-error-state E_P. In Fig. 8, we have not represented error-states that are not border-error-states, because:

**Lemma 4.2** 1 Error-states that are not border-error-states are irrelevant for our control. □

1. Note the analogy with Lemma 4.1
4.4 Step 3: Extract PP and SS from PSSEA

We extract from PSSEA two SEAs PP and SS that model SYST and a specification of its desired behavior, resp. Invariants are taken into account in this step. To simplify the method of time preemption, we assume:

Hypothesis 4.1 Location invariants use only clock constraints in the form “ci < k”. □

4.4.1 Extraction of PP from PSSEA

- Step 3.1: Erase the invariants of S, because they have no influence on the computation of PP.
- Step 3.2: Remove the error-states E_P and E_P,S, because they are not accepted by P.
- Step 3.3: Preempt every elapse of time violating an invariant in P. This is done by:
  - cutting every transition T of type 1 that leads to the violation of an invariant,
  - cutting every transition of type 3 that has the same origin state and the same expirations as T,
  - inserting the label of T (i.e., the preempted expirations) into PreemptExp(Org(T)), where Org(T) denotes the origin state of T.

For PSSEA of Fig. 7, we obtain PP of Fig. 8.

- Checking Requirement 2.1 on PP: With the SEA model, Req. 2.1 can be formulated as follows: Invariants must not be violated in the destination states of transitions of type 2 or 3. Req. 2.1 is respected in PP of Fig. 8, because PP contains no invariant violation.

- Checking Requirement 2.3 on PP: With the SEA model, Req. 2.3 can be formulated as follows: In every state where a transition (of type 1) is preempted, we state 4 in order to avoid an invariant violation. Therefore, at least one of its two outgoing transitions (labeled α and β) must be uncontrollable.

4.4.2 Extraction of SS from PSSEA

- Step 3.a: Erase the invariants of P, because they have no influence on the computation of SS.
- Step 3.b: Remove the error-states E_S and E_P,S, because they are not accepted by S.
- Step 3.c: Remove every state violating an invariant of S. (Some expirations may have to be preempted.)
- Step 3.d: Remove states not reachable by PP, in order to have L(PS) ⊆ L(PP).

For PSSEA of Fig. 7, we obtain SS of Fig. 8.

Lemma 4.3 We can consider that: the SEA PP models SYST, and the SEA SS models the most permissive desired behavior of SYST. □

4.5 Step 4: Compute Sup from PP and SS

Notations 4.1:

- Σ and Γ are the alphabets of P and PP, resp.
- S (resp. E) is a set of Set (resp. Exp) event(s).
- For M ⊆ Σ: M contains every λ ∈ Γ of the form σ or (σ, S), and M contains every λ ∈ Γ of the form (E, σ) or (E, S), σ ∈ M.
- For M ⊆ Γ, [M]Exp is obtained from M by keeping only Exp events.
- For a SEA B over Γ and μ ∈ L(Exp(B)), Acc_B(μ) is the set of labels accepted in B after the execution of μ. Formally: Acc_B(μ) = {σ ∈ Γ : μ ∈ L(Exp(B))}. □

Controllability of SEA: We define a SEA B, where the alphabet Γ and s.t. L(Exp(B)) ⊆ L(Exp(PP)), to be controllable w.r.t. PP if ∀μ ∈ L(Exp(B)):

If Acc_B(μ) ∩ Σ_f ∩ Σ uc ∩ Exp ∩ Σ uc ∩ Exp = ∅ (3)
If Acc_B(μ) ∩ Σ_f ∩ Σ uc ∩ Exp = ∅ (4)

Intuitively, B controllable w.r.t. PP means that after an execution accepted by PP and B: (in the following, σ uc ∈ Σ uc and σ_f ∈ Σ_f)

1. If B accepts a σ_f (or (σ_f, S)) then: B accepts every σ uc or (σ uc, S) accepted by PP.
2. If B accepts no σ_f (or (σ_f, S)) then:
   - B accepts every σ uc and (σ uc, S) accepted by PP, and also every (E, σ uc) and (E, σ uc, S) accepted by PP, because B can preempt a (E, σ uc) or (E, σ uc, S) only by the execution of a (σ_f) or (σ_f, S).
   - If B does not accept a L = E accepted by PP, then B accepts a L (σ_f) or (L, σ_f, S) accepted by PP, because B can preempt a L = E only by the execution of (σ_f) or (σ_f, S) or (L, σ_f) or (L, σ_f, S).
3. If B forbids a $(E, \sigma_{uc})$ or $(E, \sigma_{uc}, S)$ accepted by PP, then B forbids every transition with the same expiration(s), because $(E, \sigma_{uc})$ or $(E, \sigma_{uc}, S)$ can be forbidden only by preemption of the same expiration(s).

Let $Cont_{\text{SEA}}^{PP}$ be the set of controllable SEAs w.r.t. $PP$. $Cont_{\text{SEA}}^{PP}(SS) = \{ B \in Cont_{\text{SEA}}^{PP} : \exists L_{\infty}(B) \subseteq L_{\infty}(SS) \}$, and $SupL_{\text{SEA}}^{PP}(SS) = \bigcup_{B \in Cont_{\text{SEA}}^{PP}(SS)} L_{\infty}(B)$.

**Lemma 4.4** Let $B_1, B_2 \in Cont_{\text{SEA}}^{PP}(SS)$, B be a SEA s.t. $L_{\infty}(B) = L_{\infty}(B_1) \cup L_{\infty}(B_2)$, and $B'$ be a SEA s.t. $L_{\infty}(B') = SupL_{\text{SEA}}^{PP}(SS)$. We have: $B, B' \in Cont_{\text{SEA}}^{PP}(SS)$. □

From Lemma 4.4, we assert the existence of a unique minimal SEA $B$ s.t. $L_{\infty}(B) = SupL_{\text{SEA}}^{PP}(SS)$. Such SEA is denoted $SupC_{\text{SEA}}^{PP}(SS)$ and called “supremal controllable SEA w.r.t. PP and SS”.

**Lemma 4.5** $SupC_{\text{SEA}}^{PP}(SS)$ conf_{\text{SEA}} SS. □

**Proposition 4.1** Objective 1 is reached if $Sup/PP$ is described by $SupC_{\text{SEA}}^{PP}(SS)$. □

Prop. 4.1 is a consequence of Lemma 4.5 and Prop. 3.1. It allows to replace Objective 1 by:

**Objective 2** To synthesize a supervisor s.t. $Sup/PP$ is described by $SupC_{\text{SEA}}^{PP}(SS)$. □

The control problem that was expressed in Objective 1 by the use of TA, has been transformed into a non-real-time form in Objective 2 by the use of SEA. Objective 2 implies Objective 1, but the converse is not true.

**Notations 4.2**

- $B_i$ denotes a SEA $(Q_i, \Gamma, T_i, \Pi_i, q_{i0})$.
- $T_i(q, \mu)$ is the state reached from a state $q \in Q_i$ by the execution of a trace $\mu \in L_{\infty}(B_i)$.
- $T_i(q, \mu)$ means that $T_i(q, \mu)$ is defined; otherwise we write $T_i(q, \mu) = \bot$.
- $K_i(q)$ is the result of the intersection of Eq. 5 ($= \emptyset$ when Eq. 5 satisfied) where $q = T_i(q_0, \mu)$.
- $\text{Orig}(t)$ denotes the origin state of a transition $t$.
- $[t]_{\text{Exp}}$ is obtained from the label of a transition $t$ by keeping (all and only) its Exp events.

$SupC_{\text{SEA}}^{PP}(SS)$ can be computed from PP and SS by using a fixpoint method as follows:

\[ B_{-1} := PP, B_0 := SS = (Q_0, \Gamma, T_0, \Pi_0, q_{00}), i = 0. \]

\[
\text{While } B_{i-1} \neq B_i \text{ Do:}
\]

\[ Q := \{ q \in Q_i : \exists \mu \in L_{\infty}(B_i), q = T_i(q_0, \mu), \text{ Eq}(3) \text{ or } (4) \text{ unsatisfied} \}; \]

\[ Q' := \{ q \in Q_i : \exists \mu \in L_{\infty}(B_i), q = T_i(q_0, \mu), K_i(q) \neq \emptyset \}; \]

\[ T := \{ t \in T_i : \text{Orig}(t) \in Q', [t]_{\text{Exp}} \in K_i(q) \}; \]

\[ q_{i+1}^0 := q_i^0; \]

\[ Q_{i+1} := \{ q_i : Q \}; \]

\[ T_{i+1} := T_i \cap (Q_{i+1} \times \Gamma \times Q_{i+1}) \setminus T; \]

\[ \forall q_i \in Q_{i+1}; \]

\[ \Pi_{i+1}(q_i) := \Pi_i(q_i) \cup \{ \epsilon : T_i(q_i^0, \epsilon), T_i(q_i^0, \epsilon) \}; \]

\[ i := i + 1; \]

\[ \text{EndWhile} \]

\[ SupC_{\text{SEA}}^{PP}(SS) := B_1 = (Q, \Gamma, T, \Pi, q_0^0), \]

Sup obtained from $SupC_{\text{SEA}}^{PP}(SS)$ by associating to each state the forbidden and the forced events.

The intuition of the above algorithm is to start with SS and to proceed iteratively as follows:

- We remove states where Eq. (3) or (4) is not satisfied.
- For each state where Eq. (5) is not satisfied, some Exp-Trans may be removed in order to satisfy Eq. (5).
- We update the sets of preempted expirations of each state.

For the sake of simplicity, we have not represented the removal of unreachable states.

**Theorem 1** The supervisor computed by the above algorithm guarantees Objective 2 (and thus Obj. 1 from Prop. 4.1). □

If we compare $PP$ and SS of Fig. 9, the aim is to: forbid $\phi$ in State 9 and preempt $Exp(c, 2)$ in state 3 (i.e., forbid the three outgoing Exp-Trans of state 3). $\alpha$ or $\beta$ must be uncontrollable (see checking of Req. 2.3 in Sect. 4.4.1).

Here are the possible results of the above algorithm:

1. If $\sigma$ is uncontrollable and $\alpha$ unforcible then $Sup = \emptyset$, because; the reach of State 3 cannot be prevented, and $\alpha$ cannot be forced before $Exp(c, 2)$ in State 3.

2. If $\sigma$ and $\phi$ are uncontrollable then $Sup = \emptyset$ because; the reach of State 9 cannot be prevented, and $\phi$ cannot be forbidden in state 9.

3. If $\sigma$ is controllable and: $\alpha$ is unforcible or $\phi$ is uncontrollable, then $Sup$ forbids $(\sigma, Set(c, 1, 2, 3))$ from the initial state.

4. If $\alpha$ is forcible and $\phi$ is controllable, then $Sup$ is represented in Fig. 9. Intuitively, the supervisor: forbids $\phi$ in state 9, and forces $\alpha$ to preempt (i.e., to occur before) $Exp(c, 2)$ in state 3.

The figure 9. Step 4: $Sup$ obtained when $\alpha$ is forcible and $\phi$ is controllable.

### 4.6 A simple concrete example

We consider a modified version of the Bus/Pedestrian system used in [5]. The two TAs of Fig. 10 model the bus and the pedestrian respectively. The bus **passes** $x$ time units after it starts **approaching** $(1 < x < 2)$. The pedestrian, who is on the **road**, **jumps** on the **curb** at least 1 time unit after she takes the decision to jump. If we combine these two
systems, we obtain the plant of Fig. 11. This TA models the two possible scenarios. First scenario: the pedestrian jumps on the curb before the passage of the bus (sequence $jump.pass$). Second (undesirable) scenario, the bus passes and then the (run down) pedestrian jumps on the curb (sequence $pass.jump$). The specification of Fig. 11 requires that the pedestrian jumps before the passage of the bus.

Figure 10. TAs modeling the bus and the pedestrian

Figure 11. The plant and the specification of the Bus/Pedestrian system

If $jump$ is controllable and forcible and $pass$ is uncontrollable, we obtain the supervisor represented on Fig. 12: $jump$ is forced to occur simultaneously to $Exp(c_1, 1)$ and $Exp(c_2, 1)$. That is, the pedestrian must $jump$ on the curb 1 time unit after she takes the decision to jump.

In this example, the supervisor is not a distinct module which forces the pedestrian to jump at a given time. The supervisor represents here the obligations of the pedestrian in order to respect the specification. In a sense, the supervisor (obligations) is integrated in the plant (pedestrian). In the general case, the separation between the plant and the supervisor is conceptual and not necessarily physical.

Figure 12. Supervisor obtained from the plant and the specification of Fig. 11

5 Contribution and future Work

We present a method for controlling real-time discrete event systems. The plant and the specification are modeled by TA with invariants. The control method is based on a transformation $SetExp$ that transforms a TA into an automaton called SEA, by adding to the structure of the TA two additional types of events: $Set$ and $Exp$ that capture the temporal aspect of the TA.

Unlike previous control methods, in practice the complexity of our method does not increase significantly with the magnitudes of the constants used in timing constraints. And in comparison to [9], the present method supports TA with invariants, that permit to define formally timing constraints of the form “an event must occur in a given interval of time”, in addition to the usual constraints of the form “an event cannot occur in a given interval of time”. Here are some interesting future work:

- The supervisor synthesized by our method is not necessarily the most permissive one that respects the specification. We will investigate some ideas for improving such permissiveness.
- We intend to extend the method to make applicable for the modular control of real-time DES.
- We intend to implement our control method and apply it to complex examples.

References