Non-blocking Supervisory Control of Timed Automata using Forcible Events

Downloaded from: https://research.chalmers.se, 2026-04-04 04:07 UTC

Citation for the original published paper (version of record):
Rashidinejad, A., Van Der Graaf, P., Reniers, M. et al (2020). Non-blocking Supervisory Control of Timed Automata using Forcible Events. IFAC-PapersOnLine, 53(4): 356-362.
http://dx.doi.org/10.1016/j.ifacol.2021.04.035

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 53-4 (2020) 356–362

Non-blocking Supervisory Control of Timed Automata using Forcible Events

Aida Rashidinejad * **Patrick van der Graaf** * **Michel Reniers** * **Martin Fabian** **

* *Control Systems Technology, Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands (e-mail:{a.rashidinejad,m.a.reniers}@tue.nl)*

** *Department of Electrical Engineering, Chalmers University of Technology, Sweden (e-mail:fabian@chalmers.se)*

Abstract: Real-valued clocks make the state space of timed automata (TA) infinite. Conventional supervisory control synthesis techniques are only applicable to finite automata (FA). Therefore, to synthesize a supervisor for TA using conventional techniques, an abstraction of TA to FA is required. For many applications, the abstraction of real-time values results in an explosion in the finite state space. This paper presents a supervisory control synthesis technique applied directly to TA without any abstraction. The plant is given as a TA with a set of uncontrollable events and a set of forcible events that may preempt the passage of time. To obtain a non-blocking controlled system, a synthesis algorithm is proposed that delivers a supervisor (also as a TA) avoiding the blocking states. The algorithm works by (iteratively) strengthening the guards of edges labeled by controllable events and invariants of locations where time progress can be preempted by forcible events.

Copyright © 2020 The Authors. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0)

Keywords: Automata, forcible event, real-time, non-blocking, supervisory control, synthesis.

1. INTRODUCTION

Discrete-event systems (DES) are event-driven systems with a set of discrete states. Traffic systems, communication networks, and manufacturing systems are examples of DES (Cassandras and Lafortune, 2009). DES are often modeled by means of finite automata (FA). In order to provide a compact representation for complex and large DES, FA have been extended with discrete variables into extended finite automata (EFA) (Skoldstam et al., 2007). In EFA, edges are associated with events as well as guard constraints and updates of variables.

In order to ensure that a DES satisfies a desired behaviour, supervisory control theory (SCT) has been developed (Ramadge and Wonham, 1989). For a DES, SCT synthesizes a supervisor that guarantees that the closed-loop system satisfies the desired behaviour.

In DES, only the sequences of events matter, and the time of event occurrences is neglected. However, there exist many applications for which it is necessary to involve timing information. For instance, the processing time of a manufacturing system may be given by a specific time window (Wong-Toi and Hoffmann, 1991). Considering time in DES would also be very important in a network-based supervisory control setting due to the presence of time delays (Rashidinejad et al., 2018). To incorporate time in a DES, real-time discrete-event systems (RTDES) have been introduced (Khoumsi, 2002).

Based on the model of time (discrete or dense), two types of RTDES have been proposed in the literature: 1. timed discrete-event systems (TDES), and 2. timed automata (TA). TDES are DES in which there exists a digital clock, and each event is supposed to occur between a lower and an upper time bound (Brandin and Wonham, 1994). The semantics of a TDES is given as a timed transition graph (TTG). A TTG is an FA that includes a *tick* event that represents the passage of the smallest measurable time unit (Brandin and Wonham, 1994). Similar to DES, TDES have been extended with discrete variables into timed extended finite automata (TEFA) (Miremadi et al., 2015). In (Brandin and Wonham, 1994), SCT of DES has been extended for TDES. The main problem that discrete-time modeling faces is the state space explosion, especially for systems with various time scales. Moreover, specifying the time of each event occurrence by fixed lower upper bounds, restricts the applications that can be modeled using TDES.

In order to consider dense-time in DES, TA have been proposed (Alur and Dill, 1994). A TA is modeled by means of an FA (including a finite set of locations) extended with a finite set of real-valued clocks. Edges between locations of a TA are associated with both events and timing constraints (guards). A TA is considered a more natural way to model real-life applications not only because it considers real-time, but also because it allows multiple and different timing constraints for events. However, the inclusion of time makes the state space of TA infinite. Consequently, conventional supervisory control synthesis approaches are not adequate for TA. To overcome this problem in existing

⋆ This research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 674875.

2405-8963 Copyright © 2020 The Authors. This is an open access article under the CC BY-NC-ND license.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2021.04.035

works, TA are abstracted to FA for which a supervisor can be synthesized using existing synthesis methods (Wong-Toi and Hoffmann, 1991; Tripakis and Altisen, 1999; Maler et al., 1995). The abstraction of TA to FA is based on region or zone equivalences introduced by Alur and Dill (1994). Abstraction-based synthesis of timed automata has also been implemented in tools such as UPPAAL-TIGA (Behrmann et al., 2007).

In general, zone-based abstractions do not preserve sufficient information required for synthesis (Ouedraogo et al., 2010). On the other hand, region-based abstractions typically suffer from the state-space explosion problem, making industrial-sized systems intractable (Khoumsi and Nourelfath, 2002; Tripakis and Yovine, 2001). To overcome the state space explosion of region-based abstraction, a transformation is introduced by Khoumsi and Nourelfath (2002) which achieves an FA from a TA (without location invariants) using two special events *Set* and *Exp*. The event *Set* represents the set and reset of a clock, and *Exp* indicates the expiration of the clock. Although the transformation results in a minimal FA, the synthesis technique is not satisfying since it is currently unknown how to refine the synthesized supervisor (as an FA with *Set* and *Exp* events) to a TA (with these events translated into time constraints).

The objective of this paper is to provide a supervisory control synthesis technique that is applied directly on TA without involving any abstraction. The method presented in this paper is closely related to the synthesis technique for EFA (Ouedraogo et al., 2011). For an EFA with a set of forbidden locations, Ouedraogo et al. (2011) present a synthesis algorithm that iteratively strengthens the guards of edges labeled with controllable events until all the forbidden and blocking states are avoided.

The main difference between TA and EFA is that the semantics of TA is a (semantic) graph which includes time transitions besides event transitions. Compared to Ouedraogo et al. (2011), our work provides the following contributions:

- to each location is associated an invariant in the form of clock constraints, determining the time that the TA is allowed to stay in the location.
- a subset of events are forcible, i.e., they may preempt time passage at a location where they are enabled.
- we define a *non-blocking predicate* for each location that determines the condition under which a location is non-blocking. Nonblockingness of locations (see Definition 4) is determined based on the semantic graph (see Definition 3) of the TA: a state is non-blocking whenever a marked state is reachable from that state. In reaching the marked state not only execution of events (as in EFA), but also time passage is considered.
- we define a *bad-state predicate* for each location that determines the condition under which a bad state is reached in the semantic graph of the TA. In TA with forcible events, a bad state is blocking, and a state from which a bad state is reachable through uncontrollable events (as in EFA) or non-preemptable time passage.
- to avoid the bad states, a synthesis algorithm is presented which strengthens not only the guards labeled by controllable events, but also the invariant of locations where time is preemptable.

The rest of the paper is organized as follows. Formal definitions related to TA are given in Section 2. Section 3 introduces the problem investigated in this paper. The synthesis algorithm is presented in Section 4. Section 5 concludes the paper and discusses future work.

2. PRELIMINARIES

A TA is an FA extended with a set of real-valued clocks. Formally, a TA is defined as in (Alur and Dill, 1994).

Definition 1. (Timed Automaton). A timed automaton is a 7-tuple $(C, L, \Sigma, E, L_m, L_0, I)$ where

- C is a finite set of clocks $x \in \mathbb{R}_{\geq 0}$ that are defined locally and can be read by other automata (the initial value of each clock variable is assumed to be 0),
- L is a finite set of locations,
- Σ is a finite set of events,
- E is a finite set of edges with elements e of the form (l_s, σ, g, r, l_t) for which $l_s, l_t \in L$ are the source and target locations, respectively, $\sigma \in \Sigma$, g is the guard (clock constraint) which is a predicate over clock variables, and r is the clock reset (update) which is an assignment of the clock variables specifying which clocks are reset to 0,
- $L_m \subseteq L$ is the set of marked locations,
- $L_0 \subseteq L$ is the set of initial locations,
- I is a function associating an invariant to each location $l \in L$. An invariant is a clock constraint that needs to be satisfied when the system is in the location. ■

In this paper, no assumption is made on the clock constraints representing guards and invariants. Moreover, we frequently use the notation $Pred[u]$, for a predicate $Pred$ and an update u. The meaning of this notation is a predicate in which all occurrences of the variables are replaced by the right-hand sides of their updates. For instance, the predicate $(x \geq 3)[x := x + y]$ gives $x + y \geq 3$. Moreover, we frequently use the notation $P(C)$ to indicate the set of all predicates over the clock variables.

In this paper, we only deal with *deterministic* TA, defined as in (Alur and Dill, 1994).

Definition 2. (Deterministic TA). A timed automaton $G = (C, L, \Sigma, E, L_m, L_0, I)$ is deterministic if it has only one initial location and for all $l \in L$, $\sigma \in \Sigma$, and any pair of edges of the form $(l, \sigma, g_1, -, -) \in E$ and $(l, \sigma, g_2, -, -) \in E$, the clock constraints g_1 and g_2 are mutually exclusive. ■

In examples, TA are depicted graphically. The locations are represented by circles, and the edges by arrows from the source location to the target location labeled with the event, the guard, and the update. The initial location is depicted by a dangling incoming arrow, and marked locations by double circles. Every TA has an underlying semantic graph (Alur and Dill, 1994; Tripakis and Yovine, 2001).

Definition 3. (Semantic graph). The semantic graph of a TA $G = (C, L, \Sigma, E, L_m, l_0, I)$, is a labeled graph with

a set of states $L \times (C \to \mathbb{R}_{\geq 0})$ consisting of a location and a clock valuation (i.e., a function that assigns a non-negative real value to each clock). The initial state is $(l_0, \mathbf{0})$, where $\mathbf{0}$ denotes the clock valuation where all the clock variables have value 0. The semantic graph has the following transitions:

- time transition: from state (l, u) to state $(l, u + \Delta)$ labeled with delay $\Delta \in \mathbb{R}_{\geq 0}$ if $u + \delta$ satisfies $I(l)$ for any δ such that $0 \leq \delta \leq \Delta$. Note that for a valuation u and a real value δ, $u+\delta$ denotes the clock valuation with $(u + \delta)(c) = u(c) + \delta$ for each clock c in the domain of u.
- event transition: from state (l_s, u_s) to state $(l_t, u_s[r])$ labeled by event σ if there is an edge $e = (l_s, \sigma, g, r, l_t)$ such that u_s satisfies g, and $u_s[r]$ satisfies $I(l_t)$.

Moreover, states (l, u) in the semantic graph with $l \in L_m$ (regardless of the clock valuation u) are marked. A word in the semantic graph of G is a finite sequence (with ε denoting the empty sequence) of labels (a non-negative real value representing passage of an amount of time or an event). A state in the semantic graph of G is called reachable if it can be reached from the initial state via a word $w \in (\Sigma \cup \mathbb{R}_{\geq 0})^*$. The language of G, indicated by $L(G)$, is the set of all words in its semantic graph starting from the initial state. ■

Based on the semantic graph, some relevant notions for timed automata are defined.

Definition 4. (Non-blockingness). A state in a semantic graph is non-blocking if there exists a path leading from that state to a marked state, i.e., a state (l_t, u_t) with $l_t \in L_m$. A TA is non-blocking if all of the reachable states in its semantic graph are non-blocking. ■

Applications are usually modeled by a network of automata, where each automaton represents a subsystem. A single automaton representing the network can then be composed by the *synchronous product* of the constituent automata.

Definition 5. (Synchronous product of TA). The synchronous product of two TA $G_1 = (C_1, L_1, \Sigma_1, E_1, L_{1m}, l_{10}, I_1)$ and $G_2 = (C_2, L_2, \Sigma_2, E_2, L_{2m}, l_{20}, I_2)$, under the assumption that $C_1 \cap C_2 = \varnothing$, is given by $G_1||G_2 = (C_1 \cup C_2, L_1 \times L_2, \Sigma_1 \cup \Sigma_2, E_p, L_{1m} \times L_{2m}, (l_{10}, l_{20}), I_p)$, where for each $l_1 \in L_1$ and $l_2 \in L_2$, $I(l_1, l_2) = I(l_1) \wedge I(l_2)$ and each edge in E_p is as follows:

- take $e \in \Sigma_1 \setminus \Sigma_2$, then for every $(l_{s1}, e, g_1, r_1, l_{t1}) \in E_1$ and $l_{s2} \in L_2$, $((l_{s1}, l_{s2}), e, g_1, r_1, (l_{t1}, l_{s2})) \in E_p$.
- take $e \in \Sigma_2 \setminus \Sigma_1$, then for every $(l_{s2}, e, g_2, r_2, l_{t2}) \in E_2$ and $l_{s1} \in L_1$, $((l_{s1}, l_{s2}), e, g_2, r_2, (l_{s1}, l_{t2})) \in E_p$.
- take $e \in \Sigma_1 \cap \Sigma_2$, then for every $(l_{s1}, e, g_1, r_1, l_{t1}) \in E_1$ and $(l_{s2}, e, g_2, r_2, l_{t2}) \in E_2$, $((l_{s1}, l_{s2}), e, g_1 \wedge g_2, r_1 \cup r_2, (l_{t1}, l_{t2})) \in E_p$. ■

The set of events of a TA is partitioned into disjoint sets of uncontrollable events Σ_{uc} and controllable events Σ_c. Uncontrollable events are events that occur spontaneously in the plant such as disturbances or sensor readings. In the figures, edges labeled by uncontrollable events are indicated by dashed lines. Passage of time is uncontrollable by nature. However, it may be preempted by a forcible event belonging to $\Sigma_{for} \subseteq \Sigma$. Note that, as in Brandin and Wonham (1994), both controllable and uncontrollable events can be forcible. Example of an uncontrollable forcible event is landing of a plane where air defense could force the plane to land within some time but not prevent it from landing eventually (Brandin and Wonham, 1994). Forcible events are underlined in the figures. The following definition of *controllability* for TA with forcible events, is inspired from (Brandin and Wonham, 1994).

Definition 6. (Controllability of TA with forcible events). Given a timed plant G with uncontrollable events Σ_u, and forcible events Σ_{for}, a supervisor S is *controllable* w.r.t. G if $\forall s \in L(S||G)$ and $\forall \sigma \in (\Sigma_u \cup \mathbb{R}_{\geq 0})$, if $s\sigma \in L(G)$ then

(1) $s\sigma \in L(S||G)$ for $\sigma \in \Sigma_u$, otherwise,
(2) $s\sigma_f \in L(S||G)$ for $\sigma_f \in \{\sigma\} \cup \Sigma_{for}$ ■

Property (1) in the above definition is the standard controllability property; S cannot disable uncontrollable events that G may generate. However, if a forcible event is enabled, this may preempt the time event, which is captured by Property (2).

3. PROBLEM STATEMENT

The problem that is solved in the rest of the paper is formalized below.

Problem Statement: For a given TA G (representing a plant) with a set of uncontrollable events and a set of forcible events, the objective is to synthesize a TA S (a supervisor) such that the supervised plant $S||G$ is controllable and non-blocking.

To clarify the problem and the proposed solution, the bus-pedestrian example from Brandin and Wonham (1994) is used.

Example 1. (Bus-pedestrian). Consider a bus that is headed directly for a pedestrian and will run over him at time $x = 2$ if he does not move. The pedestrian needs an amount of time $y = 1$ to realise his fate, after which he has the chance to jump out of the bus's path. If the pedestrian jumps before the bus passes, he is safe. Figure 1 gives the

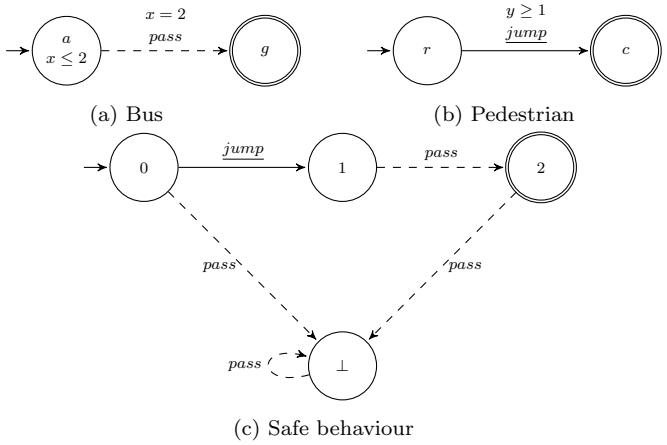

Fig. 1. Plant automata from Example 1.

automata representing the bus, pedestrian, and the safe behaviour of the system. The safe behaviour is modeled in such a way that if the pedestrian jumps before the bus passes, then the system goes to a marked state. Otherwise,

the system goes to a blocking state. The event *pass* is uncontrollable, and the event *jump* is controllable and forcible. The synchronous product of the bus, pedestrian and the safe behaviour automata is shown in Figure 2.

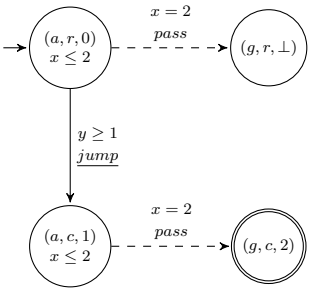

Fig. 2. Synchronous product of the TA from Example 1.

Considering Example 1, a supervisor is required that prevents the plant to reach the blocking location (location $(g, r, \perp)$ in Figure 2) while respecting controllability (Definition 6). The rest of the paper discusses how to synthesize such a supervisor.

4. SUPERVISORY CONTROL SYNTHESIS

In this section, an algorithm is proposed to synthesize a supervisor for a plant modeled as a TA (introduced in Definition 1). The supervisor is synthesized from the plant by adapting the guards and invariants such that the supervised plant is non-blocking and controllable. In other words, considering the semantic graph of a TA, the objective is to prevent the possibility of reaching the blocking states. In order to apply synthesis without any abstraction, first it is determined for which clock valuations a location is non-blocking by computing a so-called *non-blocking predicate* for each location. To take care of controllability, preventing the blocking states can be achieved only by disabling of controllable transitions. Therefore, blocking states as well as states from which a blocking state is reachable through an uncontrollable event transition or an uncontrollable time transition need to be avoided.

These states are referred to as bad states. Bad states of a TA are captured by the *bad-state predicate* associated to each location. The bad-state predicate of a location gives the clock valuations for which the location is mapped to a bad state in the semantic graph. Based on the bad-state predicate of locations, the guards of edges labeled by controllable events and invariants of locations where time is preemptable are adapted so as to guarantee that the closed-loop system avoids any bad states while respecting controllability.

Figure 3 gives an overview of the synthesis procedure. As indicated in the figure, there are two loops: 1. guard adaptation (Loop-1) considers how the supervisor can affect the controllable events, and 2. invariant adaptation (Loop-2) considers how the invariants can be modified using the concept of forcible events. In the following, first, each step of synthesis depicted in Figure 3 (non-blocking and bad-state conditions, and guard and invariant adaptation) is described in detail. Afterwards, the synthesis algorithm is presented. The bus-pedestrian example is used to illustrate each step.

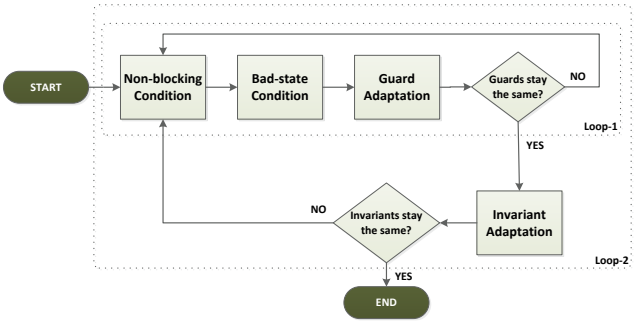

Fig. 3. An overview of the synthesis procedure.

4.1 Non-Blocking Condition

The first step is to compute the clock valuations for which a location of the plant $l \in L$ is non-blocking. For this purpose, Algorithm 1 is introduced which gives the non-blocking predicate of each location $N(l)$. Initially, $N(l)$ is set to $I(l)$ for all marked locations and to *false* for all other locations (line 2). For each location, the non-blocking predicate $N^{i+1}(l)$ is updated based on 1. the previous non-blocking predicate $N^i(l)$; 2. the condition for any outgoing edge $e = (l, \sigma, g, r, l')$ to lead to a non-blocking location; and 3. the condition to stay (for some time delay $\delta \leq \Delta$) in a non-blocking location as long as the invariant is satisfied (line 4). This will be repeated until a fix-point is reached where the non-blocking predicate stays the same for all locations (line 6).

Algorithm 1 follows the same steps as presented for the non-blocking predicate of EFA in Ouedraogo et al. (2011) with the following adjustments (indicated in red in Algorithm 1):

(1) The initial non-blocking condition for marked locations is set to the location invariant $I(l)$ instead of *true*. This is to take into account the invariants of the marked locations.
(2) In the update, the invariant of the target location is added to the second term to guarantee that the invariant of the target location is satisfied upon entrance of that location.
(3) The third term is added to take into account the time transitions in the semantic graph of the TA that may be used for reaching a non-blocking state.

Example 2 visualizes how Algorithm 1 works.

Example 2. (Non-blocking predicates for bus-pedestrian). Consider the bus-pedestrian from Example 1. The result of Algorithm 1 is given in Table 1. The conditions for locations $(g, r, \perp)$ and $(g, c, 2)$ are left out, as they are *false* and *true* respectively, for all iterations.

4.2 Bad-state Condition

Taking into account controllability (Definition 6), it is not enough to compute only the non-blocking condition to achieve a controllable supervisor. It is also needed to determine if the blocking states are reached in an uncontrollable manner, i.e., through an uncontrollable event transition

Algorithm 1 Non-blocking Predicate (NBP)

Input: $G = (C, L, \Sigma, E, L_m, l_0, I)$
Output: $N : L \to P(C)$

1: $i := 0$
2: **for** $l \in L$ **do** $N^0(l) := \begin{cases} I(l), & \text{if } l \in L_m, \\ \mathit{false}, & \text{otherwise} \end{cases}$
3: **repeat**
4: **for** $l \in L$ **do**

$$N^{i+1}(l) := N^i(l) \vee \bigvee_{l \xrightarrow{\sigma,g,r} l'} (g \wedge I(l')[r] \wedge N^i(l')[r]) \vee$$
$$\exists \Delta \in \mathbb{R}_{\geq 0}\, \forall \delta \leq \Delta\, (I(l)[C + \delta] \wedge N^i(l)[C + \delta])$$

5: $i := i + 1$
6: **until** $\forall l \in L\ N^i(l) = N^{i-1}(l)$
7: **for** $l \in L$ **do** $N(l) := N^i(l)$

Table 1. Non-blocking predicate for bus-pedestrian.

	N	
i	**Loc** $(a, r, 0)$	**Loc** $(a, c, 1)$
0	*false*	*false*
1	*false*	$x = 2$
2	$x = 2 \wedge y \geq 1$	$x \leq 2$
3	$x \leq 2 \wedge y \geq 0$	$x \leq 2$
4	$x \leq 2$	$x \leq 2$

or through a non-preemptable time transition. This issue is captured by the bad-state predicate. For each location $l \in L$, Algorithm 2 gives the bad-state predicate $B(l)$. Initially, $B(l)$ is set to the logical negation of $N(l)$ for each location $l \in L$ (line 2) because these characterize the blocking states. Then for each location (line 4), the bad-state predicate $B^{j+1}(l)$ is updated based on

(1) the previous bad-state predicate $B^j(l)$;
(2) the condition of any outgoing edge $e = (l, \sigma, g, r, l')$ labeled by an uncontrollable event $\sigma \in \Sigma_{uc}$ to lead to a bad state; and
(3) the condition of staying in a bad state for some time delay $\delta \leq \Delta$ as long as the invariant is satisfied for all the clock variables and while there is no forcible event able to preempt time for any $\delta' \leq \delta$.

This is repeated until a fix-point is reached where the bad-state predicate stays the same for all locations (line 6).

The differences (indicated in red) between Algorithm 2 and the bad-state condition of EFA presented by Ouedraogo et al. (2011) are as follows; 1. The invariant of the target location is considered to determine if the uncontrollable transition exists in the semantic graph of the TA. 2. The third term takes into account the non-preemptable time transitions leading to a bad state.

Example 3. (Bad-state predicates for bus-pedestrian). By applying Algorithm 2 on the bus-pedestrian example, the bad-state predicate of locations $(a, r, 0)$ and $(a, c, 1)$ are obtained as in Table 2. The bad-state predicates for $(g, r, \perp)$ and $(g, c, 2)$ are *true* and *false*, respectively.

Algorithm 2 Bad-State Predicate (BSP)

Input: $G = (C, L, \Sigma, E, L_m, l_0, I), N(l)$
Output: $B : L \to P(C)$

1: $j := 0$
2: **for** $l \in L$ **do** $B^0(l) := \neg N(l)$
3: **repeat**
4: **for** $l \in L$ **do**

$$B^{j+1}(l) := B^j(l) \vee \bigvee_{\substack{l \xrightarrow{\sigma,g,r} l' \\ \sigma \in \Sigma_u}} (g \wedge I(l')[r] \wedge B^j(l')[r]) \vee$$
$$\exists \Delta \in \mathbb{R}_{\geq 0}\, \forall \delta \leq \Delta \Big(\big(B^j(l)[C + \delta] \wedge I(l)[C + \delta]\big) \wedge$$
$$\forall \delta' \leq \delta \neg \bigvee_{\substack{l \xrightarrow{\sigma_f,g,r} l' \\ \sigma_f \in \Sigma_{for}}} \Big(g[C + \delta'] \wedge I(l')[C + \delta'][r] \wedge$$
$$\neg B^j(l')[C + \delta'][r] \Big) \Big)$$

5: $j := j + 1$
6: **until** $\forall l \in L\ B^j(l) = B^{j-1}(l)$
7: **for** $l \in L$ **do** $B(l) := B^j(l)$

Table 2. Bad-state predicate for bus-pedestrian.

	B	
j	**Loc** $(a, r, 0)$	**Loc** $(a, c, 1)$
0	$x > 2$	$x > 2$
1	$x \geq 2$	$x > 2$
2	$x \geq 2$	$x > 2$

4.3 Guard Adaptation

Considering Figure 3, in Loop-1, the guards are adapted to obtain a supervisor that prevents the blocking states. For this purpose, the guard of each edge $e = (l, \sigma, g, r, l')$ labeled by a controllable event $\sigma \in \Sigma_c$ is adjusted to become $e = (l, \sigma, g \wedge \neg B(l')[r], r, l')$.

4.4 Invariant Adaptation

So far, forcible events have not been taken into account. The effect of forcible events preempting time events is taken into account in the invariant adaptation (Loop-2). Example 4 clarifies how the invariant adaptation affects the result of the synthesis.

Example 4. Consider the plant given in Figure 4 for which $\Sigma_{uc} = \{a\}$ and $\Sigma_{for} = \{b\}$. The non-blocking and bad-state predicates are $N(0) = x < 1, N(1) = \mathit{false}, N(2) = \mathit{true}$, $B(0) = x \geq 1, B(1) = \mathit{true}, B(2) = \mathit{false}$. Assume that the synthesis is limited to the guard adaptation. Then, the result would be an empty supervisor since there is no way to prevent the blocking state. However, since the edge $(0, b, x < 1, -, 2)$ is enabled at location 0, time is preemptable there, and $I(0) = x \leq 2$ can be changed to $x < 1$ to satisfy non-blockingness.

The invariant of a location $l \in L$ can be changed only if there exists an edge labeled by a forcible event $f \in \Sigma_{for}$ starting from l. In this case, the invariant is adapted to prevent the blocking states as follows:

$$I(l) := I(l) \wedge \neg B(l).$$

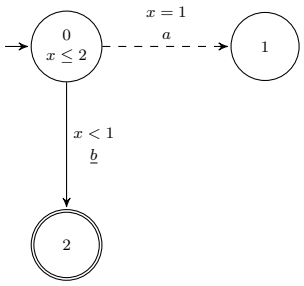

Fig. 4. Plant of Example 4.

Considering Figure 3, after the invariant adaptation, the synthesis goes back to Loop-1 (guard adaptation). Therefore, one needs to be careful that if a forcible event starting from location l becomes disabled (in some later iteration), then the invariant of l should be set back to its original value. This issue is considered in Loop-2 of the synthesis algorithm.

4.5 Synthesis Algorithm

In Algorithm 3, the synthesis algorithm is presented. For a TA G with a set of uncontrollable events Σ_{uc}, and a set of forcible events Σ_{for}, it results in a non-blocking and controllable supervisor $S = (C, L, \Sigma, E_s, L_m, l_0, I_s)$. In this algorithm, the following concepts are used:

- the notation . is used to refer to an element of a tuple. For instance, $e.\sigma$ refers to σ from e.
- the notation $F_S(l) = \{e \in E_s \mid e.l_s = l, e.\sigma \in \Sigma_{for}, e.g \text{ is satisfiable}\}$ gives the set of edges of S starting from location l_s and labeled by a forcible event.

The algorithm starts from $S = G$. As indicated in Figure 3, in the inner loop (line 6-11), the guards of edges labeled by controllable events are adapted. In the outer loop (line 5-17), the invariants of locations where there exists an edge labeled by a forcible event are adapted until a fix-point is reached. Note that if the invariant of a location is adapted, and in some later iteration, the guard of the edge labeled by the forcible event becomes unsatisfiable, the invariant should be set back to the original one (from the plant). This issue is captured in line 15. In case that the guard of the edge labeled by the forcible event becomes false at location l, then the invariant is set back to $I(l)$.

Remark 1. In Algorithm 3, the conditions for repetition depend on constraints on clock variables that do not necessarily have a finite number of possible values. Therefore, Algorithm 3 terminates only under certain assumptions on guards and location invariants of TA. The assumptions are not discussed here, as this paper presents the primary results.

Conjecture 1. Consider a TA G and the supervisor S resulting for G from Algorithm 3 (if it terminates). Then, S is controllable for G, and the supervised plant $S||G$ is non-blocking. ■

Example 5. (Supervisor synthesis for bus-pedestrian). Let us apply Algorithm 3 to the bus-pedestrian from Example 1. Initially, S is set to the plant depicted in Figure 2. First, the guard of the edge labeled by the controllable event *jump* is modified to $y \geq 1 \wedge x \leq 2$. Since $N^{1,0} = N^{0,0}$ and also $B^{1,0} = B^{0,0}$, $g^1.e = g^0.e$, and the inner loop stops.

Algorithm 3 Supervisory control synthesis of TA

Input: $G = (C, L, \Sigma, E, L_m, l_0, I)$, Σ_{uc}, Σ_c, Σ_{for}
Output: $S = (C, L, \Sigma, E_s, L_m, l_0, I_s)$

1: $S := G$
2: $m, n := 0$
3: **for** $e \in E_s$, $e = (l, \sigma, g, r, l')$ **do** $e.g^0 := e.g$
4: **for** $l \in L$ **do** $I_s^0(l) := I(l)$
5: **repeat** ▷ Loop2: Invariant Adaptation
6: **repeat** ▷ Loop1: Guard Adaptation
7: $N^{m,n} := NBP(S)$
8: $B^{m,n} := BSP(S, N^{m,n})$
9: **for** $e \in E_s$ such that $e.\sigma \in \Sigma_c$ **do** $e.g^{m+1} := e.g^m \wedge \neg B^{m,n}(l')[r]$
10: $m := m + 1$
11: **until** $\forall e \in E_s\ e.g^m = e.g^{m-1}$
12: **for** $e \in E_s$ **do** $e.g := e.g^m$
13: **for** $l \in L$ **do**
14: **if** $F_S(l) \neq \varnothing$ **then** $I_s^{n+1}(l) := I_s^n(l) \wedge \neg B^{m,n}(l)$
15: **else** $I_s^{n+1}(l) := I(l)$
16: $n := n + 1$
17: **until** $\forall l \in L\ I_s^n(l) = I_s^{n-1}(l)$
18: **for** $l \in L$ **do** $I_s(l) := I^n(l)$

Next, for $l_0 = (a, r, 0)$, the invariant is adapted to $x \leq 2 \wedge x < 2 = x < 2$. Since $N^{1,1} = N^{1,0}$ and also $B^{1,1} = B^{1,0}$, $I^1(l_0) = I^0(l_0)$ and the outer loop also terminates. The synthesized supervisor is depicted in Figure 5.

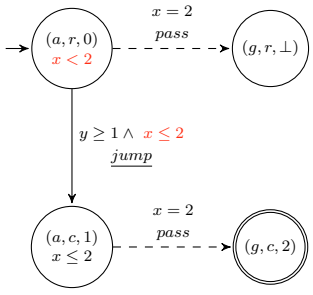

Fig. 5. Supervisor synthesized for bus-pedestrian from Example 1.

Remark 2. The synthesis procedure is easily adjustable for TA without forcible events as follows:

(1) the update of the bad-state predicate (Algorithm 2-line 4) changes to

$$B^{j+1}(l) := B^j(l) \vee \bigvee_{l \xrightarrow[\sigma \in \Sigma_u]{\sigma, g, r} l'} \left(g \wedge B^j(l')[r] \wedge I(l')[r]\right) \vee \exists \Delta\, \forall \delta \leq \Delta \left(B^j(l)[C + \delta] \wedge I(l)[C + \delta]\right)$$

since a time transition always becomes uncontrollable, and

(2) the algorithm ends after the inner loop indicated in Figure 3 since guard adaptation is the only modification that can be applied through synthesis.

5. CONCLUSION

In this paper, a synthesis algorithm has been proposed for timed automata (TA) with a set of forcible events. The algorithm is directly applied on TA without being

abstracted to finite state automata. To ensure that the supervisor makes controllable decisions, the modifications made through synthesis are as follows: 1. guard adaptation of edges labeled by controllable events, and 2. invariant adaptation of locations from which there exists an edge labeled by a forcible event. The objective is to avoid the blocking states. To take care of controllability, not only the blocking states but also the states from which a blocking state is reachable in an uncontrollable manner (known as the bad states) should be avoided. The bad states are determined using non-blocking and bad-state predicates associated to each location.

The condition for termination of the synthesis algorithm will be investigated in future research. The synthesis technique will be generalized to consider control requirements (also as TA) describing the allowed behaviour of the plant. Moreover, the approach will be implemented in available toolsets for discrete-event systems.

REFERENCES

Alur, R. and Dill, D.L. (1994). A theory of timed automata. *Theoretical computer science*, 126(2), 183–235.

Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., and Lime, D. (2007). Uppaal-tiga: Time for playing games! In *International Conference on Computer Aided Verification*, 121–125. Springer.

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

Cassandras, C.G. and Lafortune, S. (2009). *Introduction to discrete event systems*. Springer Science & Business Media.

Khoumsi, A. (2002). Supervisory control of dense real-time discrete-event systems with partial observation. In *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES'02)*, 105–112. IEEE.

Khoumsi, A. and Nourelfath, M. (2002). An efficient method for the supervisory control of dense real-time discrete event systems. In *Proceedings of the 8th International Conference on Real-Time Computing Systems (RTCSA)*.

Maler, O., Pnueli, A., and Sifakis, J. (1995). On the synthesis of discrete controllers for timed systems. In *Annual Symposium on Theoretical Aspects of Computer Science*, 229–242. Springer.

Miremadi, S., Fei, Z., Åkesson, K., and Lennartson, B. (2015). Symbolic supervisory control of timed discrete event systems. *IEEE Transactions on Control Systems Technology*, 23(2), 584–597.

Ouedraogo, L., Khoumsi, A., and Nourelfath, M. (2010). Setexp: a method of transformation of timed automata into finite state automata. *Real-Time Systems*, 46(2), 189–250.

Ouedraogo, L., Kumar, R., Malik, R., and Akesson, K. (2011). Nonblocking and safe control of discrete-event systems modeled as extended finite automata. *IEEE Transactions on Automation Science and Engineering*, 8(3), 560–569.

Ramadge, P.J. and Wonham, W.M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 81–98.

Rashidinejad, A., Reniers, M., and Feng, L. (2018). Supervisory control of timed discrete-event systems subject to communication delays and non-fifo observations. *IFAC-PapersOnLine*, 51(7), 456–463.

Skoldstam, M., Akesson, K., and Fabian, M. (2007). Modeling of discrete event systems using finite automata with variables. In *2007 46th IEEE Conference on Decision and Control*, 3387–3392. IEEE.

Tripakis, S. and Altisen, K. (1999). On-the-fly controller synthesis for discrete and dense-time systems. In *International Symposium on Formal Methods*, 233–252. Springer.

Tripakis, S. and Yovine, S. (2001). Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1), 25–68.

Wong-Toi, H. and Hoffmann, G. (1991). The control of dense real-time discrete event systems. In *Proceedings of the 30th IEEE Conference on Decision and Control*, 1527–1528.