Supervisory Control Synthesis of Timed Automata Using Forcible Events

Citation for published version (APA):
Rashidinejad, A., Reniers, M. A., & Fabian, M. (2021). Supervisory Control Synthesis of Timed Automata Using Forcible Events. *arXiv*, *2021*, Article 2102.09338. https://doi.org/10.48550/arXiv.2102.09338

DOI:
10.48550/arXiv.2102.09338

Document status and date:
Published: 18/02/2021

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Supervisory Control Synthesis of Timed Automata Using Forcible Events

Aida Rashidinejad Michel Reniers Martin Fabian

***Abstract*—Considering real-valued clocks in timed automata (TA) makes it a practical modeling framework for discrete-event systems. However, the infinite state space brings challenges to the control of TA. To synthesize a supervisor for TA using the conventional supervisory control theory, existing methods abstract TA to finite automata (FA). For many applications, the abstraction of real-time values results in an explosion in the state space of FA. This paper presents a supervisory control synthesis algorithm directly applicable to the TA without any abstraction. The plant is given as a TA with a set of uncontrollable events and a set of forcible events. Forcible events can preempt the passage of time when needed. The synthesis algorithm works by iteratively strengthening the guards of edges labeled by controllable events and invariants of locations where the progression of time can be preempted by forcible events. The synthesized supervisor, which is also a TA, is guaranteed to be controllable, maximally permissive, and results in a nonblocking and safe supervised plant.**

***Index Terms*—Automata, forcible event, real-time, maximally permissive, nonblocking, supervisory control, synthesis.**

I. Introduction

SUPERVISORY control theory (SCT) was first introduced by Ramadge-Wonham to control discrete-event systems (DES) [1]. SCT provides a synthesis method resulting in a supervisor that restricts the plant behavior towards a given set of desired behavior. Moreover, the synthesized supervisor satisfies the controllability, nonblockingness, and maximal permissivesness properties [2].

DES, such as communication networks, manufacturing and traffic systems, are typically modeled using finite automata (FA). To provide a compact representation of complex and large DES, FA have been further extended with discrete variables to extended finite automata (EFA) [3]. In EFA, transitions are labeled by events and associated with constraints on variables (guards), where variables may be updated after the occurrence of an event [3].

The dynamics of DES depend entirely on the ordering of the event occurrences, and so are independent of time [4]. However, the control of many applications needs to be able to include timing information in modeling DES. Imagine a system that needs to be controlled over a distance, due to being located in a hazardous or unreachable environment. To control such systems, the concept of networked supervisory control has been introduced [5], [6].

Networked control of systems introduces communication delays that are unavoidable and have a high impact on the system performance [7]. To consider the effects of communication delays, the DES model must include timing information of event occurrences as well as the ordering of them. For this purpose, the concepts of timed discrete-event systems (TDES), and timed automata (TA) have been introduced in [8] and [9], respectively. TDES and TA are known as real-time discrete-event systems (RTDES), which are modeled not only based on the ordering of events, but also based on timing constraints on events [10].

TDES incorporate discrete time in modeling DES. A TDES is generally a DES in which the execution of each event, called *active* event, is restricted within a lower and an upper time bound specified for the event. It is assumed that a digital clock exists in the system, and so the TDES is modeled as a FA that includes a specific event, called *tick*, indicating the passage of a unit of time. The event *tick* is generally an uncontrollable event as it spontaneously occurs in the system, and so it cannot be disabled by a supervisor. However, it is assumed that *tick* is *preemptable* by a subset of active events, called *forcible* events. Taking the nature of *tick* into account, SCT of DES, has been modified for TDES in [8]. Moreover, like DES, the model of TDES has been extended with discrete variables into timed extended finite automata (TEFA) [11].

TA incorporate dense-time in modeling DES [9]. A TA consists of a finite set of locations and a finite set of real-valued clocks [12]. To each location, a clock constraint is associated, called an *invariant*, determining the time that the system is allowed to stay in that location. Each edge between two locations is labeled by an event, the clock constraint associated to that event called the *guard*, and the set of clocks that are reset to zero, called the *reset*, by the occurrence of that event.

Compared to TDES, a TA brings a more natural modeling framework for real-life applications because 1) it considers real-time, and so it copes with the state space explosion problem introduced by discrete time; this is especially important for systems with various time scales. And 2) it easily allows events to have multiple and different timing constraints, rather than specifying the time of each event occurrence by fixed lower and upper bounds.

The control of TA is challenging due to the clock variables, making the state space of TA infinite. To overcome this problem, existing approaches abstract TA into FA, and apply

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

Aida Rashidinejad and Michel Reniers are with the Control Systems Technology Group, Department of Mechanical Engineering, Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands (e-mail: {a.rashidinejad, m.a.reniers}@tue.nl).

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

supervisory control synthesis on the abstracted result [13]–[15]. In general, the synthesis approaches can be divided into the following categories: 1) game-based (reactive) synthesis, and 2) the synthesis method proposed by Ramadge-Wonham, which is referred to as RW-based synthesis in this paper. Game-based (reactive) synthesis of TA has been investigated in [14]–[17], and it has also been implemented in tools such as UPPAAL-TIGA [15], [18]. Game-based synthesis and RW-based synthesis mainly differ in satisfying maximal permissiveness. While RW-based synthesis provides a unique maximally permissive supervisor, game-based synthesis gives a winning strategy if it exists, which is not necessarily the maximally permissive solution [19]. In this paper, we focus on RW-based synthesis as we want to achieve a maximally permissive, controllable, and nonblocking supervisor.

RW-based supervisor synthesis of TA was first investigated in [13], where the plant is first abstracted into an FA (region graph) using region-based abstraction from [9], [13]. Then, a supervisor is synthesized for the FA using existing methods. Finally, to refine the abstraction, timing information is added to the FA supervisor. For many applications, region-based abstraction results in a finite but a very large FA [20], [21].

To overcome the state-space explosion problem of region-based abstraction, some state-space minimization methods have been proposed such as zone-based abstraction [9]. These methods are mainly used for model checking and verification purposes as they do not provide sufficient information for supervisor synthesis [22].

In [20], [22], a transformation is introduced to obtain a minimal FA from a TA that is suitable for synthesis purposes. The transformation is based on two special events; *Set* and *Exp*, where *Set* represents the set and reset of a clock, and *Exp* indicates the expiration of the clock. The *SetExp*-transformation results in a minimal FA, for which a supervisor is synthesized using the concept of forcible events from TDES. Preempting time using forcible events results in a more comprehensive solution as more events can be disabled if needed. However, 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), and so the synthesis based on *SetExp*-transformation is not satisfying.

Supervisory control of TA using forcible events is also investigated in [23], in which region-based abstraction is used to abstract a TA into an FA. For the FA, a synthesis algorithm is proposed. The synthesized supervisor is transformed back into a TA using a time-refinement technique. Although this method gives the supervisor as a TA, it still suffers from the state-space explosion problem caused by the abstraction.

This paper provides a supervisory control technique for TA such that:

- no abstraction is needed to cope with the state-space explosion problem of some existing approaches,
- an algorithm is proposed that works with automata instead of languages to ease integration of an implementation in a tool set such as CIF or Supremica [24], [25],
- the RW-based synthesis is used so that the synthesized supervisor is maximally permissive, as well as controllable, and nonblocking,
- the concept of forcible events from TDES is used to provide a more comprehensive result, and
- to provide technical proofs, the notion of clock regions of timed automata is adapted in a specific way.

To the best of our knowledge, there is no work in the literature investigating TA RW-based synthesis without abstraction as we do here. Our synthesis technique is close to supervisory control synthesis for EFA. The main differences between EFA and TA are as follows: 1) an EFA deals with a set of variables belonging to a finite domain. However, a TA deals with clock variables, which belong to the infinite set of real-valued numbers, and 2) a TA includes location invariants that force the TA to leave the location before the invariant is violated. This is not the case in EFA. Dealing with real-valued clock variables and location invariants make the synthesis of TA much more complex than the synthesis of EFA. Details are discussed throughout the paper.

An earlier version of this work has been published in [26]. Compared to [26], this paper 1) provides the detailed proofs, 2) generalizes the approach for control requirements that are generally given as automata, and 3) applies the method to a well-known case study.

The rest of the paper is organized as follows. In Section II, the formal definition of TA and the relevant concepts are given. Section III presents the basic timed supervisory control (TSC) synthesis problem and the proposed solution. In Section IV, the basic TSC synthesis problem is generalized to satisfy a given set of control requirements. To verify the results, the proposed method is applied to a rail road crossing system in Section V. Finally, Section VI concludes the paper. To enhance readability, all technical lemmas and proofs are given in the appendices.

II. Preliminaries

A TA is an FA extended with a finite set of real-valued clocks. To model the timing behavior of TA, the accepting temporal conditions to switch between different modes (locations) or stay in the current one are represented by clock constraints [9], [27].

Definition 1 (Clock Constraints [27]): Given a finite set of real-valued clocks C, $x \sim n$ and $x - y \sim n$ are atomic clock constraints for any $x, y \in C$, $\sim \in \{<, =, >\}$, and $n \in \mathbb{N}$. Clock constraints are defined as follows: any atomic clock constraint is a clock constraint, and for any two clock constraints φ_1 and φ_2, also $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are clock constraints. ■

Instead of writing $x - x = 0$ with $x \in C$ as a clock constraint, we write *true*. Similarly, *false* is written instead of $x - x > 0$.

Definition 2 (Clock Valuation): Given a set of clocks C, a clock valuation $u : C \rightarrow \mathbb{R}_{\geq 0}$ assigns a real value to each clock $x \in C$. ■

Note that, initially, the valuation of each clock is $\mathbf{0}$, where $\mathbf{0}$ denotes the clock valuation where all the clock variables have value 0.

A clock valuation u satisfies a clock constraint φ, denoted $u \models \varphi$, whenever φ is *true* for the values assigned by u to each clock.

Definition 3 (Timed Automaton [9]): A timed automaton is a 7-tuple $(C, L, \Sigma, E, L_m, L_0, I)$ where

- C is a finite set of clocks with a non-negative real-value (from $\mathbb{R}_{\geq 0}$). The initial value of each clock variable is always 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 which is a clock constraint, and $r \subseteq C$ is the set of clocks to be 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 [27], guards are generally given as clock constraints, but invariants are restricted to clock constraints that are downwards closed; $x < n$ or $x \leq n$. In this work, similar to [28], both guards and invariants are allowed to be arbitrary clock constraints.

To clarify the problem and illustrate each step of the approach, the bus-pedestrian example from [29] is used throughout the paper.

Example 1 (Bus-Pedestrian): Imagine that a bus 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 realize 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 automata, representing the bus, the pedestrian, and the safe behavior of the system. The safe behavior 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.

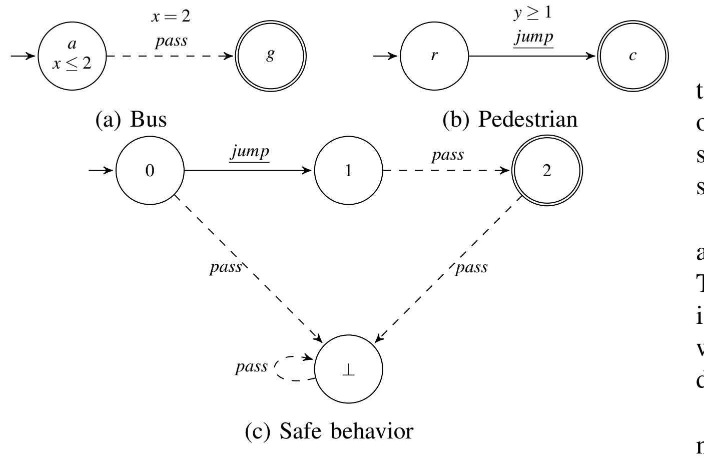


Fig. 1: Plant automata from Example 1.

For TA, we frequently use the following notations:

- the notation . is used to refer to an element of a tuple. For instance, $e.\sigma$ refers to σ from the edge $e \in E$.
- the notation $pred^{\uparrow\delta}$, for a predicate $pred$ and the increase $\delta \in \mathbb{R}_{\geq 0}$, replaces all occurrences of the variables $x \in C$ by $x + \delta$. For instance, $(x \geq 3)^{\uparrow\delta}$ gives $x + \delta \geq 3$.
- the notation $pred[r]$, for a predicate $pred$ and a reset r. The meaning of this notation is a predicate in which all occurrences of clock variables from r are replaced by zero.
- the notation $Preds(C)$, to indicate the set of all predicates over the clock variables.
- The notation P stands for the natural projection operator as defined in [4]; given a language $L \subseteq \Sigma^*$ and an event set $\Sigma' \subseteq \Sigma$: $P_{\Sigma'}(L) := \{w' \in \Sigma'^* \mid \exists w \in L, P_{\Sigma'}(w) = w'\}$.

In this paper, we only deal with *deterministic* TA.

Definition 4 (Deterministic TA [9]): A timed automaton $(C, L, \Sigma, E, L_m, L_0, I)$ is deterministic if it has only one initial location $L_0 = \{l_0\}$, and for any pair of edges $e_1, e_2 \in E$, with the same source location ($e_1.l_s = e_2.l_s$) and labeled by the same event ($e_1.\sigma = e_2.\sigma$), the clock constraints are mutually exclusive ($e_1.g \wedge e_2.g = false$). ■

From now on, we only use TA with a single initial location l_0 and consequently represent them by $(C, L, \Sigma, E, L_m, l_0, I)$.

In the examples, TA are depicted graphically. The locations are represented by circles and the edges by arrows from the source location to the target location, labelled with the event, the guard and the reset. The reset of a clock $x \in r$ is denoted by $x := 0$. Invariants of locations are indicated inside the locations. Absence of an invariant in a location represents the invariant that always holds. The initial location is depicted by a dangling incoming arrow, and the marked locations by double circles.

Definition 5 (Sub-automaton of a TA): Given a TA $A = (C, L, \Sigma, E, L_m, l_0, I)$, a TA $B = (C, L', \Sigma, E', L'_m, l'_0, I')$ is a sub-automaton of A, denoted $B \subseteq A$, if

- $L' \subseteq L$,
- for all $(l_s, \sigma, g', r, l_t) \in E' : (l_s, \sigma, g, r, l_t) \in E$ for some g such that $g' \Rightarrow g$,
- $L'_m = L_m \cap L'$,
- $l'_0 = l_0$, and
- for all $l \in L'$: $I'(l) \Rightarrow I(l)$. ■

Applications are typically modeled by a network of automata, where each automaton represents a single component or subsystem; compare Figure 1. A single automaton representing the network of automata can then be generated as the synchronous product of the constituent automata.

In [9], [27], synchronous product of TA is defined under the assumption that the two TA do not share any clock variable. This assumption is relaxed here, and the synchronous product is generalized for TA with shared set of clocks. To do so, we are inspired from the synchronous product of two EFA as defined in [3].

Definition 6 (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 $G_2 = (C_2, L_2, \Sigma_2, E_2, L_{2m}, l_{20}, I_2)$, 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 $l_2 \in L_2$, $I_p(l_1, l_2) = I_1(l_1) \wedge I_2(l_2)$ and each edge in E_p is as follows:

- $\sigma \in \Sigma_1 \setminus \Sigma_2$, then for every $(l_{s1}, \sigma, g_1, r_1, l_{t1}) \in E_1$ and $l_2 \in L_2$, $((l_{s1}, l_2), \sigma, g_1, r_1, (l_{t1}, l_2)) \in E_p$
- $\sigma \in \Sigma_2 \setminus \Sigma_1$, then for every $(l_{s2}, \sigma, g_2, r_2, l_{t2}) \in E_2$ and $l_1 \in L_1$, $((l_1, l_{s2}), \sigma, g_2, r_2, (l_1, l_{t2})) \in E_p$.

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

For the bus-pdestrian example, the synchronous product of the bus, pedestrian and the safe behavior automata is shown in Figure 2.

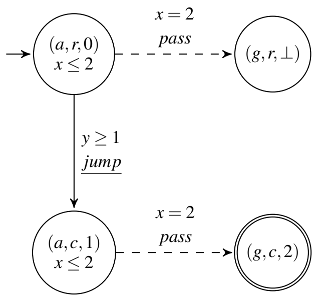


Fig. 2: Synchronous product of the TA from Example 1.

Every TA has an underlying semantic graph [9], [21].

Definition 7 (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 $X \subseteq L \times (C \to \mathbb{R}_{\geq 0})$, consisting of a location l and a clock valuation u such that $(l, u) \in X$ iff $u \models I(l)$. The initial state is $(l_0, \mathbf{0})$ if $\mathbf{0} \models I(l_0)$. Otherwise, the semantic graph is undefined. The semantic graph has the following transitions:

- 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 $u_s \models g$, and $u_s[r] \models I(l_t)$.
- time transition: from state (l, u) to state $(l, u + \Delta)$ labeled with delay $\Delta \in \mathbb{R}_{\geq 0}$ if $u + \delta \models I(l)$ 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 $(u + \delta)(x) = u(x) + \delta$ for each clock $x \in C$.

Moreover, states (l, u) in the semantic graph with $l \in L_m$ (regardless of the clock valuation u) are marked. A word w in the semantic graph of G is a finite sequence of labels; $w \in (\Sigma \cup \mathbb{R}_{\geq 0})^*$ with ε denoting the empty sequence. A state in the semantic graph of G is called reachable if it can be reached from the initial state via a word. The language of G, indicated by $L(G)$, is the set of all words in its semantic graph starting from the initial state. Note that for any $G' \subseteq G$: $L(G') \subseteq L(G)$. ■

Note that since a TA is allowed to have arbitrary clock constraints as invariants, it may be the case that $\mathbf{0} \not\models I(l_0)$. This may happen regarding modeling issues, or through synthesis, where in the latter case, synthesis actually does not result in a supervisor.

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

Definition 8 (Nonblockingness): A state in a semantic graph is nonblocking 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 nonblocking if all of the reachable states in its semantic graph are nonblocking. ■

In the rest of the paper, the plant is given as a TA G represented by $(C, L, \Sigma_G, E_G, L_m, l_0, I_G)$. It is assumed that all events are observable. However, not all of the events might be controllable. The set of events Σ_G is assumed to be partitioned into a set of uncontrollable events Σ_{uc} and a set of controllable events $\Sigma_c = \Sigma_G \setminus \Sigma_{uc}$. Uncontrollable events are events that occur spontaneously in the plant such as disturbances or sensor readings. Controllable events are signals sent to the actuators. In figures of TA, edges labelled by uncontrollable events are indicated by dashed lines, and edges labelled by controllable events are indicated by solid lines. Time passage is uncontrollable by nature. However, it may be preempted by execution of a forcible event $\sigma_f \in \Sigma_{for}$, where $\Sigma_{for} \subseteq \Sigma_G$ (forcible events are underlined in figures). Consequently, considering the semantic graph of a TA, a time transition enabled at a state is considered uncontrollable by default, unless there is also a forcible event transition enabled at that state. Then, the time transition is said to be *preemptable*. Note that a forcible event can be controllable or uncontrollable as discussed in [2]. For the bus-pedestrian example, the event *pass* is uncontrollable, and the event *jump* is controllable and forcible.

The following definition of *controllability* for TA with forcible events, is inspired from [29].

Definition 9 (Controllability of TA with Forcible Events): Given a plant G with uncontrollable events Σ_{uc}, and forcible events Σ_{for}, a TA S is *controllable* w.r.t. G if for all $w \in L(S||G)$ and $\sigma \in \Sigma_{uc} \cup \mathbb{R}_{\geq 0}$, whenever $w\sigma \in L(G)$:

1) $w\sigma \in L(S||G)$, or
2) $\sigma \in \mathbb{R}_{\geq 0}$ and $w\sigma' \in L(S||G)$ for some $\sigma' \in \Sigma_{for}$.

Property (1) above 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). ■

A supervisor S is called *proper* for a plant G whenever S is controllable w.r.t. G, and the supervised plant $S||G$ is nonblocking.

Definition 10 (Maximal Permissivenesss): A proper supervisor S is *maximally permissive* for a plant G, whenever S preserves the largest admissible behavior of G compared to any other proper supervisor S'; for any proper S': $L(S'||G) \subseteq L(S||G)$. ■

As stated in [9], the clock valuations of a TA G can be divided into a finite set of clock regions using the definition of region equivalence. Here, we introduce extended clock regions of a TA G, denoted R_G.

Definition 11 (Extended Clock Regions of TA): Consider a TA G with a set of clocks C where the the clock ceiling function, $k : C \to \mathbb{N}$ gives the largest natural number that a clock $x \in C$ is bounded to by guards or invariants. Each clock region $r_G \in R_G$ is specified by:

1) for each clock $x \in C$, a single clock constraint of one of the following forms:
   - $x = n$ for some $n \in \{0, \ldots, k(x)\}$,
   - $n - 1 < x < n$ for some $n \in \{1, 2, \ldots, k(x)\}$, or
   - $x > k(x)$
2) for any two different clocks $x, y \in C$, a single clock constraint of one of the following forms:
   - $y - x + k(x) = q$ for some $q \in \{0, \ldots, k(x) + k(y)\}$,

- $q-1 < y-x+k(x) < q$ for some $q \in \{1,\ldots,k(x)+k(y)\}$,
- $y-x+k(x) < 0$, or
- $y-x+k(x) > k(x)+k(y)$ ∎

Note that $k(x)$ does not restrict the value of the clock variable x; it only gives the largest number that x is bounded to by guards or invariants. Considering Figure 2, $k(x)=2$. However, in location $(g,r,\perp)$, the value of x can grow to any real number larger than or equal to 2.

Example 2: Figure 3 depicts the extended clock regions for a TA with two clock variables x,y, where $k(x)=2$ and $k(y)=1$. The clock regions given for the same example in [9] are indicated in black.

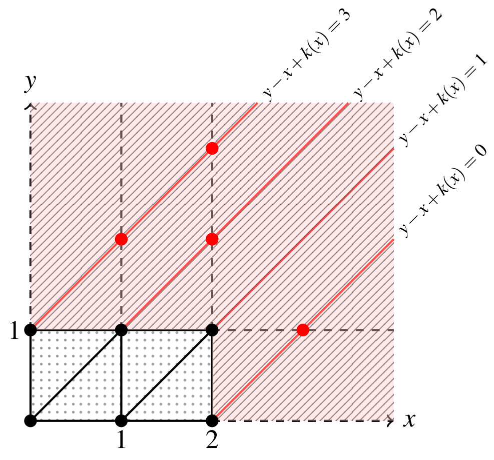


Fig. 3: Extended clock regions from Example 2.

We call a clock region *unbounded* (dashed areas/lines in Figure 3) if it is related to $x > k(x)$ for some $x \in C$. Otherwise, the region is called *bounded* (dotted areas/solid lines in Figure 3). Note that although the number of the extended clock regions is more than the number of clock regions, it is still finite because the set of clock regions is finite (see [9] for details), and the extended clock regions include all the bounded regions from the set of clock regions, and it partitions each unbounded region into a finite number of new regions. For instance, in Example 2, $0<x<1, y>1$ is an unbounded region that is partitioned into new regions as $0<x<1, y>1, y-x+k(x)<3$; $0<x<1, y>1, y-x+k(x)=3$; and $0<x<1, y>1, y-x+k(x)>3$.

Definition 12 (G-Clock Constraint): Consider a plant G with a set of clocks C, the clock ceiling function $k: C \to \mathbb{N}$, and the set of regions R_G. A clock constraint φ is called a *G-clock constraint* whenever all the atomic constraints of φ are bounded by $k(x)$ for all $x \in C$. ∎

Clearly, for any two G-clock constraints φ_1 and φ_2, $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are G-clock constraints.

Based on the extended clock regions, we are now able to discriminate the regions that satisfy a G-clock constraint. Let us consider Example 2 again. Given a G-clock constraint $\varphi = x-y>2$, there does not exist a set of clock regions satisfying φ based on the definition of clock regions in [9]. However, considering Definition 11, $y=0, x>2, y-x+k(x)<0$; $0<y<1, x>2, y-x+k(x)<0$; $y=1, x>2, y-x+k(x)<0$; and $y>1, x>2, y-x+k(x)<0$ are the extended clock regions satisfying φ. This discrimination will be the basis to prove the termination and correctness of the proposed algorithms.

Moreover, it is assumed that there exists a function Z mapping a G-clock constraint φ to the maximal set of regions from R_G such that for any region $r_G \in Z(\varphi)$, and for any valuation u represented by r_G, denoted $u \in r_G$, $u \models \varphi$. For any two G-clock constraints φ_1 and φ_2, Z necessarily satisfies the following properties:

- $Z(\varphi_1 \wedge \varphi_2) = Z(\varphi_1) \cap Z(\varphi_2)$ and $Z(\varphi_1 \vee \varphi_2) = Z(\varphi_1) \cup Z(\varphi_2)$.
- Whenever $Z(\varphi_1) = Z(\varphi_2)$, φ_1 and φ_2 represent the same G-clock constraint.

Also, for the clock constraints represented by *true* and *false*, the mapping gives R_G, and $\varnothing$, respectively.

III. Basic TSC Synthesis

A. Problem Formulation

The *Basic TSC Synthesis Problem* is defined as follows. Given a plant model G as a TA, the objective is to synthesize a timed supervisor S, also as a TA, such that

- S is controllable w.r.t. G,
- $S||G$ is nonblocking, and
- S is maximally permissive w.r.t. G.

Considering the bus-pedestrian example, a supervisor is required to avoid reaching the blocking location $(g,r,\perp)$ in Figure 2. The objective is to provide a supervisory control synthesis approach that does not need an abstraction. The synthesized supervisor should respect controllability (Definition 9), nonblockingness (Definition 8), and be maximally permissive (Definition 10).

To synthesize such a supervisor, it is needed to determine the states (l,u) in the semantic graph that should be made unreachable, referred to as *bad states*. These are the following types of states: 1) states that are blocking and should be avoided to take care of nonblockingness, and 2) states that lead to a bad state through an uncontrollable event or a time transition that cannot be preempted; these states should be avoided to respect controllability as well as nonblockingness. As the synthesis algorithm should not involve any abstraction, we need to determine the clock valuations for which a location of a TA is a bad state (in the semantic graph). For this purpose, we start by determining the clock valuations for which a location is nonblocking, referred to as the "nonblocking predicate" of a location. Based on the nonblocking predicate, a "bad state predicate" is associated to each location determining the clock valuations for which the location is mapped to a bad state in the semantic graph.

B. Nonblocking Condition

Given a plant G, Algorithm 1 associates a nonblocking predicate $N(l)$ to each location $l \in L$. Initially (line 2), $N^i(l)$ with $i=0$ is set to $I_G(l)$ if l is a marked location, and to *false* otherwise. The nonblocking predicate of each location is updated (line 4) to $N^{i+1}(l)$ based on:

① the current nonblocking predicate $N^i(l)$,

$\textcircled{2}$ the condition for any outgoing edge (l,σ,g,r,l') to lead to a nonblocking location (an event transition leading to a nonblocking state in the semantic graph), and

$\textcircled{3}$ the condition to stay (for some time delay $\delta \leq \Delta$) in a nonblocking location as long as the invariant is satisfied (represented by a time transition leading to a nonblocking state in the semantic graph).

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

Algorithm 1 Nonblocking Predicate (*NBP*)

```
Input: $G = (C, L, \Sigma_G, E_G, L_m, l_0, I_G)$
Output: $N : L \to Preds(C)$
1: $i := 0$
2: for $l \in L$ do $N^0(l) := \begin{cases} I_G(l), & \text{if } l \in L_m, \\ false, & \text{otherwise} \end{cases}$
3: repeat
4:     for $l \in L$ do
           $N^{i+1}(l) := \overbrace{N^i(l)}^{\textcircled{1}} \vee \overbrace{\bigvee_{l \xrightarrow{\sigma,g,r} l'} (g \wedge I_G(l')[r] \wedge N^i(l')[r])}^{\textcircled{2}} \vee$
           $\overbrace{\exists\Delta\, N^i(l)^{\uparrow\Delta} \wedge \forall\delta \leq \Delta\, I_G(l)^{\uparrow\delta}}^{\textcircled{3}}$
5:     $i := i + 1$
6: until $\forall l \in L\; N^i(l) = N^{i-1}(l)$
7: for $l \in L$ do $N(l) := N^i(l)$
```

Algorithm 1 follows the same steps as presented for the nonblocking predicate of EFA in [30] with the following adjustments (indicated in red in Algorithm 1):

1) The initial nonblocking condition for marked locations is set to the location invariant $I_G(l)$ instead of *true*. This is to take into account the invariants of the marked locations.
2) In the update (line 4), the invariant of the target location is added to the second term to guarantee that the invariant of the target location is satisfied upon entering 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 nonblocking state.

Property 1 (NBP Termination): Given a plant G with a set of locations L and a set of regions R_G; Algorithm 1 terminates.

Proof: See Appendix B-A. ■

Property 2 (NBP and Nonblocking States): Given a plant G and $NBP(G)$: for any (l,u) in (the semantic graph of) G, (l,u) is a nonblocking state iff $u \models N(l)$, where $N = NBP(G)$.

Proof: See Appendix B-B. ■

Example 3 (Nonblocking Predicate for Bus-Pedestrian): Consider the bus-pedestrian from Example 1. The result of Algorithm 1 is given in Table I. The conditions for locations $(g,r,\perp)$ and $(g,c,2)$ are left out, as they are *false* and *true* respectively, for all iterations. The condition $x \leq 2 \wedge (y \geq 1 \vee x - y \leq 1)$ is equivalent to $x \leq 2 \wedge x - y \leq 1$.

TABLE I: Nonblocking 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	$x \leq 2 \wedge (y \geq 1 \vee x - y \leq 1)$	$x \leq 2$
4	$x \leq 2 \wedge x - y \leq 1$	$x \leq 2$

C. Bad State Condition

Given a plant G, and the nonblocking predicate computed by Algorithm 1, Algorithm 2 associates a bad state predicate $B(l)$ to each location $l \in L$.

Initially, $B^i(l)$ with $i = 0$ is set to the logical negation of $N(l)$ for each location $l \in L$ (line 2) because these characterize the blocking states. Then, the bad state predicate of each location is updated to $B^{j+1}(l)$ (line 4) based on

$\textcircled{4}$ the previous bad state predicate $B^j(l)$,

$\textcircled{5}$ the condition of any outgoing edge (l,σ,g,r,l') labeled by an uncontrollable event $\sigma \in \Sigma_{uc}$ to lead to a bad state (an uncontrollable event transition leading to a bad state in the semantic graph), and

$\textcircled{6}$ 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 $\delta' \leq \delta$ (an uncontrollable time transition leading to a bad state in the semantic graph).

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

Algorithm 2 Bad State Predicate (*BSP*)

```
Input: $G = (C, L, \Sigma_G, E_G, L_m, l_0, I_G), NBP(G)$
Output: $B : L \to Preds(C)$
1: $j := 0$
2: for $l \in L$ do $B^0(l) := \neg N(l)$
3: repeat
4:     for $l \in L$ do
           $B^{j+1}(l) := \overbrace{B^j(l)}^{\textcircled{4}} \vee \overbrace{\bigvee_{\substack{l \xrightarrow{\sigma,g,r} l' \\ \sigma \in \Sigma_{uc}}} (g \wedge I_G(l')[r] \wedge B^j(l')[r])}^{\textcircled{5}} \vee$
           $\overbrace{\exists\Delta\, B^j(l)^{\uparrow\Delta} \wedge \forall\delta \leq \Delta \Big(I_G(l)^{\uparrow\delta} \wedge}^{\textcircled{6}}$
           $\forall\delta' \leq \delta \quad \neg \bigvee_{\substack{l \xrightarrow{\sigma_f,g,r} l' \\ \sigma_f \in \Sigma_{for}}} (g^{\uparrow\delta'} \wedge I_G(l')^{\uparrow\delta'}[r] \wedge \neg B^j(l')^{\uparrow\delta'}[r])\Big)$
5:     $j := j + 1$
6: until $\forall l \in L\; B^j(l) = B^{j-1}(l)$
7: for $l \in L$ do $B(l) := B^j(l)$
```

The differences (indicated in red) between Algorithm 2 and the bad state condition of EFA presented by [30] are as follows; 1. The invariant of the target location is considered to determine if the uncontrollable transition should exist in the semantic graph. 2. The third term takes into account the non-preemptable time transitions leading to a bad state.

Property 3 (BSP Termination): Given a plant G with the set of locations L, set of regions R_G, and $NBP(G)$; Algorithm 2 terminates.

Proof: See Appendix B-C. ■

Property 4 (BSP and Bad States): Given a plant G and $NBP(G)$: for any (l,u) in (the semantic graph of) G, (l,u) is a bad state iff $u \models B(l)$, where $B = BSP(G, NBP(G))$.

Proof: See Appendix B-D. ■

Example 4 (Bad State Predicate 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 II. The bad state predicates for $(g,r,\perp)$ and $(g,c,2)$ are *true* and *false*, respectively.

TABLE II: Bad state predicate for bus-pedestrian.

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

D. Synthesis

Figure 4 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.

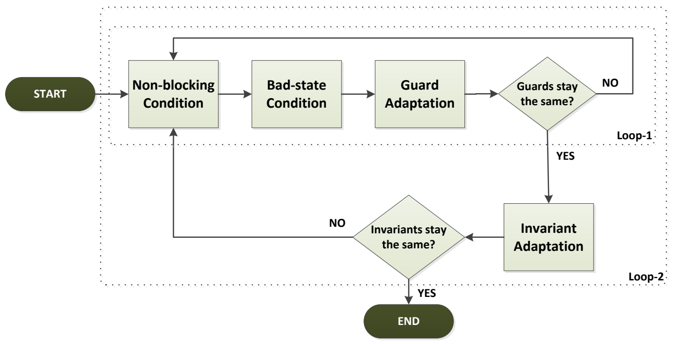


Fig. 4: An overview of the synthesis procedure [26].

1) Guard adaptation: Consider Figure 4, in Loop-1 the guards are adapted to obtain a supervisor that prevents the bad states. For this purpose, the guard of each edge (l,σ,g,r,l') labeled by a controllable event $\sigma \in \Sigma_c$ is adjusted to become $(l,\sigma,g \wedge \neg B(l')[r],r,l')$.

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

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

3) Synthesis Algorithm: Algorithm 3 is the synthesis algorithm. For a TA G with a set of uncontrollable events Σ_{uc}, and a set of forcible events Σ_{for}, it results in $S = (C,L,\Sigma_G,E_S,L_m,l_0,I_S)$. 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 and labeled by a forcible event. The algorithm starts with $S = G$. As indicated in Figure 4, in the inner loop (lines 7-13), the guards of edges labeled by controllable events are adapted until a fix-point is reached. In the outer loop (lines 5-19), the invariants of locations where there exist an edge labeled by a forcible event are adapted until a fix-point is reached. Otherwise, the synthesis goes back to Loop-1 (guard adaptation). Note that if the invariant of a location l is adapted, and in some later iteration the guard of an edge labeled by the forcible event becomes false, then the invariant should be set back to its original $I_G(l)$. This is captured in line 17.

Algorithm 3 Timed supervisory control synthesis (*TSCS*)

```
Input:  $G = (C,L,\Sigma_G,E_G,L_m,l_0,I_G)$, $\Sigma_{uc}$, $\Sigma_c$, $\Sigma_{for}$
Output: $S = (C,L,\Sigma_G,E_S,L_m,l_0,I_S)$
 1: $S := G$
 2: $n := 0$
 3: for $e \in E_S$, $e = (l,\sigma,g,r,l')$ do $e.g^0 := e.g$
 4: for $l \in L$ do $I_S^0(l) := I_G(l)$
 5: repeat                                   ▷ Loop-2: Invariant Adaptation
 6:     $m := 0$
 7:     repeat                               ▷ Loop-1: Guard Adaptation
 8:         $N^{n,m} := NBP(S)$
 9:         $B^{n,m} := BSP(S, N^{n,m})$
10:         for $e \in E_S$ such that $e.\sigma \in \Sigma_c$ do
11:             $e.g^{m+1} := e.g^m \wedge \neg B^{n,m}(l')[r]$
12:         $m := m+1$
13:     until $\forall e \in E_S$ $e.g^m = e.g^{m-1}$
14:     for $e \in E_S$ do $e.g := e.g^m$
15:     for $l \in L$ do
16:         if $F_S(l) \neq \varnothing$ then $I_S^{n+1}(l) := I_S^n(l) \wedge \neg B^{n,m}(l)$
17:         else $I_S^{n+1}(l) := I_G(l)$
18:     $n := n+1$
19: until $\forall l \in L$ $I_S^n(l) = I_S^{n-1}(l)$
20: for $l \in L$ do $I_S(l) := I_S^n(l)$
```

Given a plant G, in case that $u_0 \models B(l_0)$, with B as the result of Algorithm 2 for $TSCS(G)$ and $NBP(TSCS(G))$, then $TSCS(G)$ is undefined. In the rest of the paper, it is assumed that $u_0 \not\models B(l_0)$ for any given plant G.

Property 5 (TSCS Termination): Given a plant G; Algorithm 3 terminates.

Proof: See Appendix B-E. ■

Property 6 ($TSCS(G)$ is a TA): Given a plant G, $S = TSCS(G)$ is a TA.

Proof: See Appendix B-F. ■

Property 7 ($TSCS(G)$ is a subautomaton of G): Given a plant G: $TSCS(G) \subseteq G$.

Proof: See Appendix B-G. ■

According to Property 7, $TSCS(G)||G = TSCS(G)$.

Property 8 (Algorithm Correctness): Given a plant G and the supervisor $S = TSCS(G)$: for any reachable state (l,u) (in the semantic graph) of S: $u \not\models B(l)$, where $B = BSP(S, NBP(S))$.

Proof: See Appendix B-H. ■

The following theorems summarize the main results of the paper.

Theorem 1 (Controllability): Given a plant G with uncontrollable events Σ_{uc} and forcible events Σ_{for}, and the supervisor $S = TSCS(G)$: S is controllable w.r.t. G.

Proof: See Appendix B-I. ■

Theorem 2 (Nonblockingness): Given a plant G and the supervisor $S = TSCS(G)$: the supervised plant $S||G$ is nonblocking.

Proof: See Appendix B-J. ■

Theorem 3 (Maximal Permissiveness): Given a plant G and the supervisor $S = TSCS(G)$: S is maximally permissive for G.

Proof: See Appendix B-K. ■

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 $B^{1,0} = B^{0,0}$, $e.g^1 = e.g^0$, and the inner loop stops. 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 $B^{1,1} = B^{1,0}$, $I_S^1(l_0) = I_S^0(l_0)$ and the outer loop also terminates. The synthesized supervisor is depicted in Figure 5.

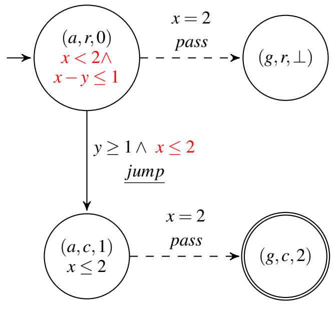


Fig. 5: Supervisor for bus-pedestrian from Example 1.

Remark 1: Invariant adaptation can highly affect the synthesis result. Consider Example 5, Algorithm 3 does not result in a supervisor without invariant adaptation. However, if the TA has no forcible event, time transitions are always uncontrollable and the synthesis procedure can be adjusted as follows:

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

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

where the last part of ⑥ is removed, and

2) the algorithm ends after the inner loop indicated in Figure 4 since guard adaptation is the only modification that can be applied through synthesis.

IV. Requirement Automata

To generalize the method to a wider class of applications, we solve the TSC synthesis problem for a given set of control requirements. It is assumed that an allowed behavior of G is denoted by the timed automaton $R = (C_R, Q, \Sigma_R, E_R, Q_m, q_0, I_R)$, where $\Sigma_R \subseteq \Sigma_G$ and $C_R \cap C = \emptyset$. Since most control requirements are defined to provide safety of a plant, we call a supervised plant $SP = S||G$ *safe* if it satisfies the control requirement R.

Definition 13 (Safety): Given a plant G and a control requirement R, a TA S with event set Σ_S is safe w.r.t. G and R if $P_{\Sigma_{SP} \cap \Sigma_R}(L(S||G)) \subseteq P_{\Sigma_{SP} \cap \Sigma_R}(L(R))$ with $\Sigma_{SP} = \Sigma_S \cup \Sigma_G$. ■

Requirement automata can be considered in synthesis by being transferred into the plant using synchronous product. However, if a requirement automaton is not controllable (Definition 9), then it is necessary to let the supervisor know about the uncontrollable events that are disabled by a given requirement. To take care of this issue, a requirement automaton R is made complete. *Completion* was first introduced in [31] for DES, where the requirement automaton R is made complete as $R^{\perp}$ in terms of uncontrollable events. By applying the synthesis on $G||R^{\perp}$, all original controllability problems in $G||R$ are translated to blocking issues. To solve the blocking issues, synthesis still takes the controllability definition into account. Inspired from [31], we present the completion of a TA.

Definition 14 (TA Completion): Given a TA $R = (C_R, Q, \Sigma, E_R, Q_m, q_0, I_R)$, the complete automaton $R^{\perp}$ is defined as $R^{\perp} = (C_R, Q \cup \{q_d\}, \Sigma, E_R^{\perp}, Q_m, q_0, I_R)$, where $q_d \notin Q$, $I_R(q_d) = true$ and $I_R(q) = I(q)$ for all $q \in Q$, and for every $q_s \in Q$, $\sigma \in \Sigma_{uc}$:

$$E_R^{\perp} = E_R \cup \{(q_s, \sigma, g^{\perp}, \{\}, q_d) \mid (q_s, \sigma, g, r, q_t) \in E_R\},$$

where $g^{\perp} = \neg\left(\bigvee_{e \in E_R, e.q_s = q_s, e.\sigma = \sigma} e.g \wedge I_R(e.q_t)[e.r]\right)$. ■

To synthesize a supervisor, Algorithm 3 is applied on $G||R^{\perp}$. The obtained supervisor is already guaranteed to be controllable, maximally permissive, and it results in a nonblocking supervised plant. Theorem 4 shows that the supervised plant is safe as well.

Theorem 4 (Safety): Given a plant G, a set of control requirements R, and the supervisor $S = TSCS(G||R^{\perp})$: S is safe for G w.r.t. R.

Proof: See Appendix B-L. ■

In general, there can be a set of control requirements $\{R_1, R_2, \ldots, R_n\}$ given for a plant. In that case, the allowed behavior of G, is determined by the synchronous product of all requirement automata; $R = R_1||R_2||\ldots||R_n$. Since completion distributes over synchronous product, $R^{\perp}$ can be computed either as $R_1^{\perp}||R_2^{\perp}||\ldots||R_n^{\perp}$, or $(R_1||R_2||\ldots||R_n)^{\perp}$.

V. Case Study

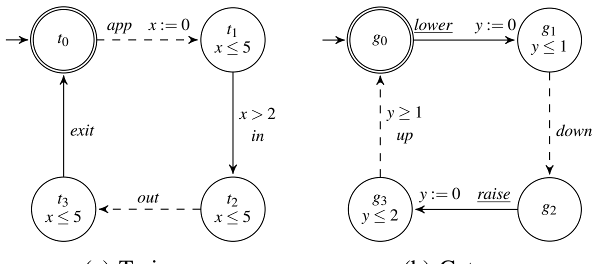
In this section, we consider the verification example from [9], [28] and modify it for synthesis. The TA representing the train and gate are depicted in Figure 6. The system in [9], [28] also involves an automatic controller, depicted in Figure 7, to open and close the gate in a railroad crossing.

The control requirements for the train-gate-controller system are as follows [9]:

- Safety requirement: whenever the train is inside the gate, the gate should be closed.
- Liveness requirement: the gate is never closed for more than 10 time units.

In [9], [28], the system is assessed to be safe by analysing the timing constraints: they say that with the (random) gate-controller, that is part of the system, the event *lower* always proceeds the event *in*, so the system is always safe. We do not consider such a controller to already be given as a part of the system. We synthesize a supervisor that is correct-by-construction, and more importantly this supervisor guarantees controllability, nonblockingness, and maximal permissiveness.

The models of train and gate are taken directly from [9], [28]. The events *app* and *out* for the train, and the events *down* and *up* for the gate are assumed to be uncontrollable. Moreover, the events *raise* and *lower* of the gate are assumed to be forcible.



(a) Train (b) Gate

Fig. 6: Train-gate system.

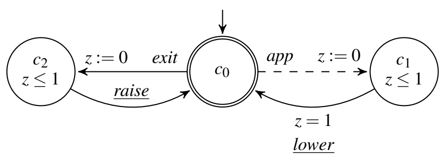


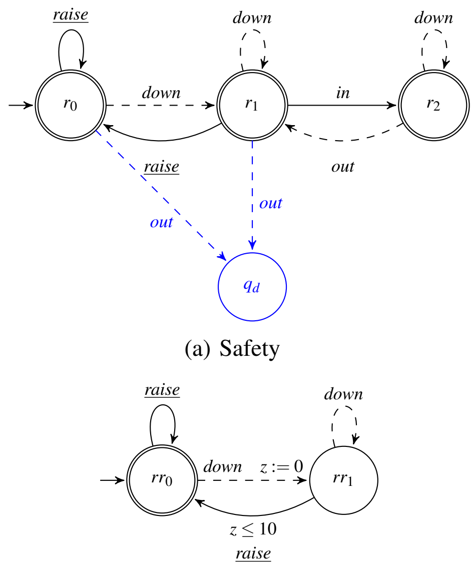
Fig. 7: Gate-controller from [9], [28].

The safety requirement is represented by the TA in Figure 8a, where the blue location and edges are added to make the TA complete. The liveness requirement is represented by the TA in Figure 8b. The liveness requirement does not need completion as the uncontrollable event *down* is enabled at both states of the automaton.

The supervisor synthesized by Algorithm 3 for the train-gate and control requirements is given in Figure 9. In this figure, the synchronous product of the train-gate and control requirements is indicated in black and the adaptations made by the supervisor in red.

VI. Conclusion and Future Work

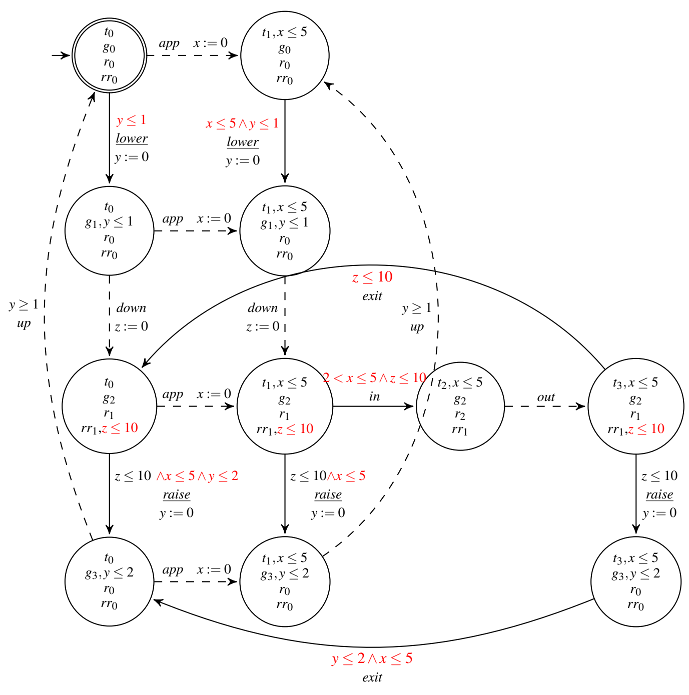
In this paper, we propose a synthesis algorithm for timed automata (TA) with a set of forcible events. The algorithm is directly applicable on TA without abstracting them to finite state automata. The objective is to avoid blocking states. To take care of controllability, not only the blocking states but also the states from which a blocking state is reachable in an uncontrollable manner (referred to as bad states) should be avoided. The bad states are determined using nonblocking and bad state predicates associated to each location. 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 exist an edge



(a) Safety

(b) Liveness

Fig. 8: Requirements for train-gate system.



t_0 g_0 r_0 rr_0
app $x := 0$
$t_1, x \leq 5$ g_0 r_0 rr_0
$y \leq 1$ lower $y := 0$
$x \leq 5 \wedge y \leq 1$ lower $y := 0$
t_0 $g_1, y \leq 1$ r_0 rr_0
app $x := 0$
$t_1, x \leq 5$ $g_1, y \leq 1$ r_0 rr_0
$y \geq 1$ up
down $z := 0$
down $z := 0$
$z \leq 10$ exit
$y \geq 1$ up
t_0 g_2 r_1 $rr_1, z \leq 10$
app $x := 0$
$t_1, x \leq 5$ g_2 r_1 $rr_1, z \leq 10$
$2 < x \leq 5 \wedge z \leq 10$ in
$t_2, x \leq 5$ g_2 r_2 rr_1
out
$t_3, x \leq 5$ g_2 r_1 $rr_1, z \leq 10$
$z \leq 10 \wedge x \leq 5 \wedge y \leq 2$ raise $y := 0$
$z \leq 10 \wedge x \leq 5$ raise $y := 0$
$z \leq 10$ raise $y := 0$
t_0 $g_3, y \leq 2$ r_0 rr_0
app $x := 0$
$t_1, x \leq 5$ $g_3, y \leq 2$ r_0 rr_0
$t_3, x \leq 5$ $g_3, y \leq 2$ r_0 rr_0
$y \leq 2 \wedge x \leq 5$ exit

Fig. 9: Synthesized supervisor for train-gate and control requirements. Edges with guards equal to *false* and locations reached by them have been removed.

labeled by a forcible event. Based on the notion of extended clock regions, it is proven that the synthesized supervisor satisfies nonblockingness, controllability, and maximal permissiveness. To generalize, we solve the problem for a given set of control (safety) requirements modeled as TA. We guarantee that the synthesized supervisor satisfies controllability, nonblockingness, maximal permissiveness, and safety. Finally, the results are verified by applying the method on a case study. Networked supervisory control of timed automata will be studied in future research. Moreover, implementation of the proposed approach in available tool sets will be investigated.

ACKNOWLEDGMENT

The authors would like to thank Patrick van der Graaf for his initial efforts on this subject.

APPENDIX A
TECHNICAL LEMMAS

Lemma 1 (G-Clock Constraint): φ is a G-clock constraint iff for any pair of clock valuations u_1, u_2, represented by the same clock region $r_G \in R_G$: $u_1 \models \varphi \Longleftrightarrow u_2 \models \varphi$.

Proof: The proof is trivial. ■

Lemma 2 (Clock Valuations): For any pair of clock valuations $u_1, u_2 \in r_G$ for some $r_G \in R_G$, if $u_1 + \Delta_1 \in r_\Delta$ for some $r_\Delta \in R_G$ and $\Delta_1 \in \mathbb{R}_{\geq 0}$: there exists $\Delta_2 \in \mathbb{R}_{\geq 0}$ such that $u_2 + \Delta_2 \in r_\Delta$.

Proof: As illustrated by Figure 3, from two valuations from the same region in each case any move to another region (by passage of time) from one of these valuations is easily mimicked from the other valuation (possibly for a different amount of time passage). ■

Remark 2: In the coming lemmas and proofs, we frequently use "this term represents a G-clock constraint". The meaning of this is that although the term may not necessarily satisfy G-clock constraints as given by Definition 12, there is G-clock constraint that is logically equivalent with it (which means that for any valuation the term and its G-clock constraint representation have the same value).

Lemma 3 (Negation of G-Clock Constraint): For any G-clock constraint φ, the negation $\neg\varphi$ also represents a G-clock constraint.

Proof: This is proved by induction on the structure of G-clock constraints.

Base cases:

- for the atomic G-clock constraints $x < n$ and $x - y < n$, their negations are $x \geq n$ and $x - y \geq n$, respectively;
- for the atomic G-clock constraints $x = n$ and $x - y = n$, their negations are $x > n \vee x < n$ and $x - y > n \vee x - y < n$, respectively;
- for the atomic G-clock constraints $x > n$ and $x - y > n$, their negations are $x \leq n$ and $x - y \leq n$, respectively.

Induction step: Consider the G-clock constraint $\varphi = \varphi_1 \Diamond \varphi_2$ for some G-clock constraints φ_1 and φ_2, and $\Diamond \in \{\wedge, \vee\}$, where the statement holds for φ_1 and φ_2, i.e., $\neg\varphi_1$ and $\neg\varphi_2$ also represent G-clock constraints. If $\Diamond = \vee$, then $\neg(\varphi_1 \Diamond \varphi_2) = \neg\varphi_1 \wedge \neg\varphi_2$. Also, if $\Diamond = \wedge$, then $\neg(\varphi_1 \Diamond \varphi_2) = \neg\varphi_1 \vee \neg\varphi_2$. Both $\neg\varphi_1$ and $\neg\varphi_2$ represent G-clock constraints as assumed, and the combination of any two G-clock constraints by $\wedge$ and $\vee$ is also a G-clock constraint. So, $\neg(\varphi_1 \Diamond \varphi_2)$ represents a G-clock constraint.

Conclusion: By the principle of induction, the claim of Lemma 3 holds for any G-clock constraint φ. ■

Lemma 4 (Reset Update of G-Clock Constraint): For any G-clock constraint φ and any reset r, $\varphi[r]$ also represents a G-clock constraint.

Proof: This is proved by induction on the structure of G-clock constraints.

Bases cases:

- for the atomic G-clock constraints $x < n$ and $x - y < n$, $\varphi[r]$ represents the G-clock constraint φ if $x, y \notin r$. If $x, y \in r$, $\varphi[r]$ represents the G-clock constraint *true* if $n \neq 0$ and *false* if $n = 0$. For $x - y < n$, $\varphi[r]$ represents the G-clock constraint $y > n$ if only $x \in r$, and $x < n$ if only $y \in r$.
- for the atomic G-clock constraints $x = n$ and $x - y = n$, $\varphi[r]$ represents the G-clock constraint φ if $x, y \notin r$. If $x, y \in r$, $\varphi[r]$ represents the G-clock constraint *true* if $n = 0$ and *false* if $n \neq 0$. For $x - y = n$, $\varphi[r]$ represents the G-clock constraint $y = n$ if only $x \in r$, and $x = n$ if only $y \in r$.
- for the atomic G-clock constraints $x > n$ and $x - y > n$, $\varphi[r]$ represents the G-clock constraint φ if $x, y \notin r$. If $x, y \in r$, $\varphi[r]$ represents the G-clock constraint *false*. For $x - y > n$, $\varphi[r]$ represents the G-clock constraint $y < n$ if only $x \in r$, and $x > n$ if only $y \in r$.

Induction step: Consider the G-clock constraint $\varphi = \varphi_1 \Diamond \varphi_2$ for some G-clock constraints φ_1 and φ_2, and $\Diamond \in \{\wedge, \vee\}$, where the statement holds for φ_1 and φ_2, i.e., $\varphi_1[r]$ and $\varphi_2[r]$ also represent G-clock constraints. $(\varphi_1 \Diamond \varphi_2)[r] = \varphi_1[r] \Diamond \varphi_2[r]$ because the reset update does not change anything else than replacing all clock variables of r by zero. So, in $(\varphi_1 \Diamond \varphi_2)[r]$, the clock variables from r in both φ_1 and φ_2 are replaced by zero which can equivalently be represented by $\varphi_1[r] \Diamond \varphi_2[r]$. Since the combination of any two G-clock constraints by $\wedge$ and $\vee$ is also a G-clock constraint, $(\varphi_1 \Diamond \varphi_2)[r]$ represents a G-clock constraint.

Conclusion: By the principle of induction, the claim of Lemma 4 holds for any G-clock constraint φ. ■

Lemma 5 (Δ-Time Invariance for NBP): Given G-clock constraints φ_1 and φ_2, $\exists\Delta\ \varphi_1^{\uparrow\Delta} \wedge\ \forall\delta \leq \Delta\ \varphi_2^{\uparrow\delta}$ represents a G-clock constraint.

Proof: Let us indicate $\exists\Delta\ \varphi_1^{\uparrow\Delta} \wedge\ \forall\delta \leq \Delta\ \varphi_2^{\uparrow\delta}$ by Φ. Take a clock valuation u_1 represented by a clock region of G, say $r_G \in R_G$, such that $u_1 \models \Phi$. According to Lemma 1, it suffices to prove that for any region r_G and any two clock valuations u_1 and u_2 represented by r_G, $u_1 \models \Phi$ iff $u_2 \models \Phi$. Because of symmetry considerations it suffices to prove that $u_1 \models \Phi$ implies $u_2 \models \Phi$. Let us assume $u_1 \models \Phi$. Then there exists some Δ_1 such that $u_1 \models \varphi_1^{\uparrow\Delta_1}$ and $u_1 \models \forall\delta \leq \Delta_1\ \varphi_2^{\uparrow\delta}$. It is proved that there always exists a Δ_2 for which $u_2 \models \varphi_1^{\uparrow\Delta_2}$ and $u_2 \models \forall\delta \leq \Delta_2\ \varphi_2^{\uparrow\delta}$. Let us say $u_1^{\uparrow\Delta_1} \in r_\Delta$ for some $r_\Delta \in R_G$. Then, based on Lemma 2, there exists a $\Delta_2 \in \mathbb{R}_{\geq 0}$ such that $u_2^{\uparrow\Delta_2} \in r_\Delta$. Since $u_1^{\uparrow\Delta_1} \models \varphi_1$ and $u_1^{\uparrow\Delta_1}, u_2^{\uparrow\Delta_2} \in r_\Delta$, by Lemma 1: $u_2^{\uparrow\Delta_2} \models \varphi_1$. It suffices to prove that for all $\delta \leq \Delta_2$: $u_2 \models \varphi_2^{\uparrow\delta}$.

Take $\delta_2 \leq \Delta_2$, and assume that $u_2^{\uparrow\delta_2} \in r_\delta$, where r_δ can be r_G, r_Δ, or any region in between. Based on Lemma 2, there exists a δ_1 such that $u_1^{\uparrow\delta_1} \in r_\delta$. We prove that $\delta_1 \leq \Delta_1$ by contradiction. Assume $\delta_1 > \Delta_1$. Then, r_δ already passed r_Δ, and this contradicts the fact that r_δ is either r_G, r_Δ, or any other region in between. So, $\delta_1 \leq \Delta_1$, and $u_2^{\uparrow\delta_2} \models \varphi_2$ because $u_1^{\uparrow\delta_1} \models \varphi_2$, and $u_1^{\uparrow\delta_1}, u_2^{\uparrow\delta_2} \in r_\delta$. ■

Lemma 6 (Nonblocking Predicate): Given a plant G, $N^i(l)$ computed by Algorithm 1 in each iteration i and for each location $l \in L$ represents a G-clock constraint.

Proof: We do the proof by induction on the number of iterations i.

Base case: $i = 0$. Then, $N^0(l)$ is either $I_G(l)$ or *false*, and in each case, this is a G-clock constraint by definition.

Induction step: Assume that the statement holds for i, i.e., $N^i(l)$ is a G-clock constraint for all $l \in L$. It suffices to prove that the statement holds for $i+1$, i.e., $N^{i+1}(l)$ is a G-clock constraint for all $l \in L$. Consider Algorithm 1-line 4, $N^{i+1} = \textcircled{1} \vee \textcircled{2} \vee \textcircled{3}$. It suffices to prove that each of $\textcircled{1}$, $\textcircled{2}$, and $\textcircled{3}$ is a G-clock constraint because then, the disjunction of them is also a G-clock constraint. $\textcircled{1}$ is a G-clock constraint as assumed. $\textcircled{2}$ is a G-clock constraint because g and $I_G(l')$ are G-clock constraints by definition, and $N^i(l')[r]$ is a G-clock constraint since $N^i(l')$ is a G-clock constraint as assumed, and the reset update represents a G-clock constraint according to Lemma 4. Then, the conjunction of g, $I_G(l')$, and $N^i(l')[r]$ gives a G-clock constraint by definition. Finally, the big disjuction in $\textcircled{2}$ is over a finite number of G-clock constraints as the number of edges is finite. $\textcircled{3}$ is a G-clock constraint because $I_G(l)$ is a G-clock constraint by definition, and $N^i(l')$ is a G-clock constraint as assumed. So, based on Lemma 5, $\textcircled{3}$ represents a G-clock constraint.

Conclusion: By the principle of induction, the claim of Lemma 6 holds for any iteration i and location $l \in L$. ■

Lemma 7 (Δ-Time Invariance for BSP): Given G-clock constraints φ_1, φ_2, and φ_3, $\exists\Delta\, \varphi_1^{\uparrow\Delta} \wedge \forall\delta \leq \Delta \left(\varphi_2^{\uparrow\delta} \wedge \forall\delta' \leq \delta\, \varphi_3^{\uparrow\delta'}\right)$ also represents a G-clock constraint.

Proof: Let us indicate $\exists\Delta\, \varphi_1^{\uparrow\Delta} \wedge \left(\forall\delta \leq \Delta\, \varphi_2^{\uparrow\delta} \wedge \forall\delta' \leq \delta\, \varphi_3^{\uparrow\delta'}\right)$ by Φ. Take a clock valuation u_1 represented by a clock region of G, say $r_G \in R_G$, such that $u_1 \models \Phi$. According to Lemma 1, it suffices to prove that for any region r_G and any two clock valuations $u_1, u_2 \in r_G$: $u_1 \models \Phi$ iff $u_2 \models \Phi$. Because of symmetry considerations it suffices to prove that $u_1 \models \Phi$ implies $u_2 \models \Phi$.

Consider an arbitrary region $r_G \in R_G$ and arbitrary clock valuations $u_1, u_2 \in r$. Let us assume $u_1 \models \Phi$. Then there exists some Δ_1 such that $u_1 \models \varphi_1^{\uparrow\Delta_1}$, and $u_1 \models \forall\delta \leq \Delta_1\, \varphi_2^{\uparrow\delta} \wedge \forall\delta' \leq \delta\, \varphi_3^{\uparrow\delta'}$. It is proved that there always exists a Δ_2 for which $u_2 \models \varphi_1^{\uparrow\Delta_2}$, and $u_2 \models \forall\delta \leq \Delta_2\, \varphi_2^{\uparrow\delta} \wedge \forall\delta' \leq \delta\, \varphi_3^{\uparrow\delta'}$. Let us say that $u_1^{\uparrow\Delta_1} \in r_\Delta$ for some $r_\Delta \in R_G$. Then, based on Lemma 2, there exists a $\Delta_2 \in \mathbb{R}_{\geq 0}$ such that $u_2^{\uparrow\Delta_2} \in r_\Delta$. Since $u_1^{\uparrow\Delta_1} \models \varphi_1$ and $u_1^{\uparrow\Delta_1}, u_2^{\uparrow\Delta_2} \in r_\Delta$, by Lemma 1, we have $u_2^{\uparrow\Delta_2} \models \varphi_1$.

What remains to prove is that for all $\delta \leq \Delta_2$: $u_2 \models \varphi_2^{\uparrow\delta}$ and for all $\delta' \leq \delta$: $u_2 \models \varphi_3^{\uparrow\delta'}$. Take $\delta_2 \leq \Delta_2$, and assume $u_2^{\uparrow\delta_2} \in r_\delta$, where r_δ can be r_G, r_Δ, or any region in between. Then, for any $\delta' \leq \delta_2$, $u_2^{\uparrow\delta'}$ moves to either r_G, r_δ, or any region in between. Let us take $\delta'_2 \leq \delta_2$, and assume that $u_2^{\uparrow\delta'_2}$ moves to $r_{\delta'}$. Based on Lemma 2, there exists a δ_1 and a δ'_1 such that $u_1^{\uparrow\delta_1} \in r_\delta$ and $u_1^{\uparrow\delta'_1} \in r_{\delta'_1}$. We prove that $\delta_1 \leq \Delta_1$ and $\delta'_1 \leq \delta_1$ by contradiction. 1) Assume $\delta_1 > \Delta_1$. Then, r_δ already passed r_Δ, and this contradicts the fact that r_δ is either r_G, r_Δ, or any other region in between. 2) Assume $\delta'_1 > \delta_1$. Then, $r_{\delta'}$ already passed r_δ, and this contradicts the fact that $r_{\delta'}$ is either r_G, r_δ, or any other region in between. ■

Lemma 8 (Bad State Predicate): Given a plant G and $NBP(G)$, $B^i(l)$ computed by Algorithm 2 in each iteration i and for each location $l \in L$ represents a G-clock constraint.

Proof: This is proved in a similar way to the proof of Lemma 6, where Lemma 7 is used to show that $\textcircled{6}$ represents a G-clock constraint. ■

Lemma 9 (Adapted Guards): Given a plant G, $e.g^m$ computed by Algorithm 3 in each iteration m and for each edge $e \in E_S$ represents a G-clock constraint.

Proof: We do the proof by induction on the number of iterations m.

Base case: $m = 0$. Then, $e.g^m = e.g$ for any $e \in E_S$ which is a G-clock constraint by definition.

Induction step: Assume that the statement holds for m, i.e., $e.g^m$ is a G-clock constraint for all $e \in E_S$. It suffices to prove that the statement holds for $m+1$, i.e., $e.g^{m+1}$ represents a G-clock constraint for all $e \in E_S$. Consider Algorithm 3-line 11, $e.g^{m+1} = e.g^m \wedge \neg B^{n,m}(l')[r]$. Now, $e.g^m$ is a G-clock constraint as assumed. $B^{n,m}(l')$ is a G-clock constraint because according to the proof of Lemma 8, in each iteration, the bad state predicate of each location is a G-clock constraint. Based on Lemma 4, $B^{n,m}(l')[r]$ represents a G-clock constraint, and so due to Lemma 3, $\neg B^{n,m}(l')[r]$ represents a G-clock constraint. Then, the conjunction of $e.g^m$, and $\neg B^{n,m}(l')[r]$ gives a G-clock constraint by definition.

Conclusion: By the principle of induction, the claim of Lemma 9 holds for any iteration m and edge $e \in E_S$. ■

Lemma 10 (Adapted Invariants): Given a plant G, $I_S^n(l)$ computed by Algorithm 3 in each iteration n and for each location $l \in L$ represents a G-clock constraint.

Proof: This is proved in a similar way to the proof of Lemma 9. ■

Remark 3: As Algorithm 3 terminates (See Appendix B-E), in the coming proofs, for a given a plant G and $TSCS(G)$, it is assumed that the outer loop (Loop-2) terminates in $n = N$ iterations, and for each $0 \leq n \leq N$, the inner loop (Loop-1 inside Loop-2) terminates in $m = M_n$ iterations. S^{N,M_N} denotes the result of the final iteration, so that $S = S^{N,M_N}$ is the output of $TSCS(G)$.

Lemma 11 (Synthesis Intermediate Results): Given a plant G, the result of $TSCS(G)$ at any iteration n, m ($n \leq N, m \leq M_n$) is a TA.

Proof: Initially, S is set to G, which is a TA. Then, at each iteration over n, m, only some of the guards and invariants may change. According to Lemma 9 and Lemma 10, the adapted guards and invariants are always clock constraints. So, based on Definition 3, the result of $TSCS(G)$ at any iteration n, m is a TA. ■

Remark 4: As Lemma 11 holds, in the coming proofs, the result of Algorithm 3 at iteration n, m ($n \leq N, m \leq M_n$) is assumed to be the TA $S^{n,m}$, represented by the automaton $(C, L, \Sigma_G, E_S^m, L_m, l_0, I_S^n)$, where E_S^m is the set of edges, and I_S^n gives the invariants of $S^{n,m}$.

Appendix B
Proofs of Properties and Theorems

A. *Proof of Property 1*

Based on Lemma 6, in each iteration of the algorithm, say i, and for any location $l \in L$: $N^i(l)$ represents a G-clock constraint. At line 4, $N^i(l)$ is adapted to the G-clock constraint $N^{i+1}(l) = N^i(l) \vee (\textcircled{2} \vee \textcircled{3})$ for all $l \in L$. Both $\textcircled{2}$ and $\textcircled{3}$ represent G-clock constraint as proved in Lemma 6. So, $Z(N^{i+1}(l)) = Z(N^i(l)) \cup Z(\textcircled{2} \vee \textcircled{3})$ where $Z(\textcircled{2} \vee \textcircled{3}) \in \mathscr{P}(R_G)$. Then, if $Z(\textcircled{2} \vee \textcircled{3}) \subseteq Z(N^i(l))$: $Z(N^{i+1}(l)) = Z(N^i(l))$. So, $N^{i+1}(l) = N^i(l)$, and the algorithm terminates (line 6). Otherwise, at least a region $r_G \in R_G$ is added to $Z(N^i(l))$ so that $Z(N^{i+1}(l)) = Z(N^i(l)) \cup \{r_G\}$. Since L and R_G are both finite, this can occur only finitely many times.

B. *Proof of Property 2*

This property is proved in two parts:

1) take an arbitrary nonblocking state (l,u), and assume that from (l,u), a marked state can be reached in j transitions. Since the algorithm terminates and in each iteration (line 4), the nonblocking condition for a given location l is never strengthened, it always holds that $N^i(l) \Rightarrow N(l)$. So, to conclude that $u \models N(l)$, we prove that $u \models N^i(l)$ for some i by induction on j:

Base case: assume that from (l,u), a marked state is reached in 0 transitions. In other words, $l \in L_m$, and so $N^0(l) = I_G(l)$ by definition. Then, for $i = 0$, $u \models N^i(l)$ since the semantic graph only contains states (l,u) for which the clock valuation satisfies the invariant of the location; $u \models I_G(l)$.

Induction step: assume that from (l,u), a marked state is reached in $j+1$ transitions. Also, assume that (l,u) leads to a state, say (l',u'), in one transition (this means that from (l',u'), a marked state is reached in j transitions), where the statement holds for (l',u') i.e., $u' \models N^i(l')$ for some i (by induction assumption). We prove that $u \models N^{i+1}(l)$.

If (l,u) moves to (l',u') by an event transition, say $\sigma \in \Sigma$, where this transition is related to an edge, (l,σ,g,r,l'). Then, based on Definition 7, $u \models g$, and $u[r] \models I_G(l')$. Also, $u[r] \models N^i(l')$ since $u' \models N^i(l')$ as assumed and $u' = u[r]$. So, $u \models N^{i+1}(l)$ since $u \models \textcircled{2}$. If (l,u) moves to (l',u') by a time transition, say Δ. Then, $u + \Delta \models N^i(l)$ because based on Definition 7, $l' = l$, $u' = u + \Delta$, and $u' \models N^i(l')$ as assumed. Also, for all $\delta \leq \Delta$: $u + \delta \models I_G(l)$ by definition. So, $u \models N^{i+1}(l)$ since $u \models \textcircled{3}$.

Conclusion: for any state (l,u) in the semantic graph of G that is nonblocking: $u \models N(l)$.

2) take an arbitrary (l,u) for which $u \models N(l)$. Since the algorithm terminates, and in each iteration the nonblocking condition for a given location l is never strengthened, there is always some i such that $u \models N^i(l)$. We prove by induction on i that from (l,u), a marked state is reached:

Base case: assume $u \models N^0(l)$. Then, $N^0(l)$ cannot be *false*, and so from (l,u), a marked state is reached (in 0 transitions) as $l \in L_m$.

Induction step: assume $u \models N^{i+1}(l)$, and the statement holds for i , i.e., for any (l',u') with $u' \models N^i(l')$: from (l',u'), a marked state is reached (induction assumption).

Considering the nonblocking predicate computation (line4), $u \models N^{i+1}(l)$ either because already $u \models N^i(l)$, or because $u \models \textcircled{2}$ or $u \models \textcircled{3}$.

In case $u \models N^i(l)$. Then, from (l,u), a marked state is reached based on the induction assumption.

In case $u \models \textcircled{2}$, then there exists at least one edge (l,σ,g,l',r) such that $u \models g \wedge I_G(l')[r] \wedge N^i(l')[r]$. Since $u \models N^i(l')[r]$ and $u' = u[r]$, $u' \models N^i(l')$. So, based on the induction assumption, from (l',u'), a marked state is reached. Also, since $u \models g \wedge I_G(l')[r]$, due to Definition 7, there is an event transition leading from (l,u) to (l',u'). So, from (l,u), a marked state is reached.

In case $u \models \textcircled{3}$, then there exists Δ such that $u + \Delta \models N^i(l)$, and for all $\delta \leq \Delta$: $u + \delta \models I_G(l)[r]$. Since $u + \Delta \models N^i(l)$ with $u' = u + \Delta$, based on induction assumption, from (l',u'), a marked state is reached. Also, due to Definition 7, there is a time transition from (l,u) to (l',u') as for all $\delta \leq \Delta$: $u + \delta \models I_G(l)[r]$. So, from (l,u), a marked state is reached.

Conclusion: from any state (l,u) in the semantic graph of G with $u \models N(l)$, a marked state is reached.

C. *Proof of Property 3*

This property is proved in a similar way to the proof of Property 1, where Lemma 7 is used to show that $\textcircled{6}$ represents a G-clock constraint.

D. *Proof of Property 4*

This property is proved in two parts:

1) take an arbitrary bad state (l,u), and assume that from (l,u) a blocking state can be reached in j (uncontrollable) transitions. Since the algorithm terminates, and in each iteration (line 4), the bad state condition for a given location l is never strengthened, it always holds that $B^i(l) \Rightarrow B(l)$. So, to conclude that $u \models B(l)$, we prove that $u \models B^i(l)$ for some i by induction on j:

Base case: from (l,u), a blocking state is reached in 0 transitions. Then, due to Property 2 $u \not\models N(l)$, and so for $i = 0$, $u \models B^i(l)$ by definition.

Induction step: from (l,u), a blocking state can be reached in $j+1$ (uncontrollable) transitions. Assume that in one (uncontrollable) transition, (l,u) moves to a state, say (l',u'), where the statement holds for (l',u') i.e., $u' \models B^i(l')$ for some i (by the induction assumption). We prove that $u \models B^{i+1}(l)$.

If (l,u) moves to (l',u') by an uncontrollable event transition that is related to an edge (l,σ,g,r,l'). Then, based on Definition 7, $u \models g$, and $u[r] \models I_G(l')$. Also, $u[r] \models B^i(l')$ since $u' \models B^i(l')$ as assumed and $u' = u[r]$. So, $u \models B^{i+1}(l)$ since $u \models \textcircled{5}$.

If (l,u) moves to (l',u') by a time transition, say Δ, that is not preemptable. Then, $u \models B^{i+1}(l)$ since $u \models \textcircled{6}$ for the following reasons: 1) $u + \Delta \models B^i(l)$ because based on

Definition 7, $l' = l$, $u' = u + \Delta$, and $u' \models B^i(l')$ as assumed, 2) for all $\delta \leq \Delta$: $u + \delta \models I_G(l)$ by definition, and 3) since Δ is not a preemptable time transition, there is no forcible event enabled at (l,u) so that the condition on forcible events always holds (is *true*).

Conclusion: for any bad state (l,u) in (the semantic graph of) G: $u \models B(l)$.

Assume that $u \models B^i(l)$. We prove by induction on i that from (l,u), a blocking state is reached within i uncontrollable transitions:

Base case: $u \models B^0(l)$. Then, $u \not\models N(l)$ by definition. So, based on Property 2, (l,u) is not a marked state, and any transition enabled at (l,u) does not lead to a nonblocking state. So, from (l,u), a blocking state is reached in 0 uncontrollable transitions.

Induction step: assume $u \models B^{i+1}(l)$, where the statement holds for i, i.e., for any (l',u') with $u' \models B^i(l')$: from (l',u'), a blocking state is by the induction assumption reached within i uncontrollable transitions.

Considering the bad state predicate computation, $u \models B^{i+1}(l)$ because already $u \models B^i(l)$, or because $u \models \textcircled{5}$ or $u \models \textcircled{6}$.

If $u \models B^i(l)$, then, (l,u) is a bad state based on the induction assumption.

If $u \models \textcircled{5}$, then there exists at least one edge (l, σ, g, l', r), labeled by an uncontrollable event, such that $u \models g \wedge I_G(l')[r] \wedge B^i(l')$. Since $u \models B^i(l')[r]$ and $u'[r] = u$, $u' \models B^i(l')$. So, based on the induction assumption, (l',u') is a bad state.

Also, since $u \models g \wedge I_G(l')[r]$, due to Definition 7, there is an uncontrollable event transition from (l,u) to (l',u'). So, (l,u) is a bad state.

If $u \models \textcircled{6}$, then there exists Δ such that $u + \Delta \models B^i(l)$, and for all $\delta \leq \Delta$: $u + \delta \models I_G(l)[r]$ (note that there is no forcible event that can preempt time). Since $u + \Delta \models B^i(l)$ with $u' = u + \Delta$, based on the induction assumption, (l',u') is a bad state.

Also, since $u + \delta \models I_G(l)[r]$ for all $\delta \leq \Delta$, due to Definition 7, there is a time transition from (l,u) to (l',u') that is not preemptable. So, (l,u) is a bad state.

Conclusion: for any state (l,u) in (the semantic graph of) G such that $u \models B(l)$: (l,u) is a bad state.

E. *Proof of Property 5*

Inside each iteration over n (loop-2), the iteration over m (loop-1) terminates because the computation of both $N^{n,m}$ and $B^{n,m}$ terminate due to property 1 and property 3, respectively. Also, whenever all the guards stay the same (line 13). Due to Lemma 9, in each iteration m and for any edge $e \in E_S$, $e.g^m$ represents a clock constraint which is adapted to the clock constraint $e.g^{m+1} = e.g^m \wedge \neg B^{n,m}(l')[r]$ at line 11. Based on the properties stated for Z, $Z(e.g^{m+1}) = Z(e.g^m) \cap Z(\neg B^{n,m}(l')[r])$, and so $Z(e.g^{m+1}) \subseteq Z(e.g^m)$ for all $e \in E_S$. In case that $Z(e.g^{m+1}) = Z(e.g^m)$ for all $e \in E_S$, the iteration over m terminates because $e.g^{m+1} = e.g^m$ for any $e \in E_S$. Otherwise, in each iteration, at least one region $r_G \in R_G$ is excluded from $Z(e.g^m)$ for some $e \in E_S$, i.e., $Z(e.g^{m+1}) = Z(e.g^m) \setminus \{r\}$ such that $r \notin Z(e.g^m)$, and so loop-1 can iterate only finitely often as E_S and R_G are both finite. The iteration over n (loop-2) terminates whenever all location invariants stay the same (line 19). Based on Lemma 10, in each iteration of the algorithm n, and for any location $l \in L$: $I_S^n(l)$ represents a clock constraint which is adapted to $I_S^{n+1} = I_S^n(l) \wedge \neg B^{n,m}(l)$ at line 16. Then, for the same reason stated for termination of loop-1, loop-2 also terminates.

F. *Proof of Property 6*

The proof follows immediately from Lemma 11.

G. *Proof of Property 7*

According to Lemma 11, for any n,m ($n \leq N, m \leq M_n$): $S^{n,m}$ is a TA. Also, according to Property 6, S is a TA. To conclude that $S \subseteq G$, we prove that for all $n \leq N$, for all $m \leq M_n$: $S^{n,m} \subseteq G$ using nested induction on n and m. Then, in particular $S^{N,M_N} = TSCS(G) \subseteq G$, which is to be proven. Induction on n:

Base case: $n = 0$, and we prove that for all $m \leq M_0$: $S^{0,m} \subseteq G$ by induction on m:

- **Base case:** $S^{0,0} = G$, and so $S^{0,0} \subseteq G$.
- **Induction step:** assume $S^{0,m} \subseteq G$. Then, $S^{0,m+1}$ differs from $S^{0,m}$ only in terms of guards. So, considering Definition 5 and the construction of S in Algorithm 3, it only suffices to prove that for all $(l_s, \sigma, g_S^{m+1}, r, l_t) \in E_G^{m+1}$: $(l_s, \sigma, g_G, r, l_t) \in E_G$ for some g_G such that $g_S^{m+1} \Rightarrow g_G$. Take arbitrary edge $(l_s, \sigma, g_S^{m+1}, r, l_t) \in E_G^{m+1}$. Then, considering line 11, $(l_s, \sigma, g_S^m, r, l_t) \in E_S^m$ such that either $g_S^m = g_S^{m+1}$, or it is strengthened, and so $g_S^{m+1} \Rightarrow g_S^m$. Also, since $S^{0,m} \subseteq G$, then for $(l_s, \sigma, g_S^m, r, l_t) \in E_S^m$: $(l_s, \sigma, g_G, r, l_t) \in E_G$ for some g_G such that $g_S^m \Rightarrow g_G$. Thereto, for all $(l_s, \sigma, g_S^{m+1}, r, l_t) \in E_G^{m+1}$: $(l_s, \sigma, g_G, r, l_t) \in E_G$ for some g_G such that $g_S^{m+1} \Rightarrow g_G$.
- **Conclusion:** for all $m \leq M_0$: $S^{0,m} \subseteq G$.

Induction step: assume $S^{n,m} \subseteq G$ for all $m \leq M_n$. We prove that $S^{n+1,m} \subseteq G$ for all $m \leq M_{n+1}$ using induction on m:

- **Base case:** $S^{n,0} \subseteq G$ by assumption. $S^{n+1,0}$ differs from $S^{n,0}$ only in terms of invariants. So, considering Definition 5, it suffices to prove that for all $l \in L$: $I_S^{n+1}(l) \Rightarrow I_G(l)$. Take arbitrary $l \in L$. Then, $I_S^n(l) \Rightarrow I_G(l)$ since $S^{n,0} \subseteq G$. Considering line 16, at iteration $n+1$, either the invariant stays the same, or it is strengthened such that $I_S^{n+1}(l) \Rightarrow I_S^n(l)$. So, $I_S^{n+1}(l) \Rightarrow I_G(l)$ as $I_S^n(l) \Rightarrow I_G(l)$.
- **Induction step:** assume $S^{n+1,m} \subseteq G$ for all $m \leq M_{n+1}$. Then, $S^{n+1,m+1}$ differs from $S^{n+1,m}$ only in terms of guards. Then, $S^{n+1,m+1} \subseteq G$ for the same reason stated in the previous induction step on m.
- **Conclusion:** for all $m \leq M_n$: $S^{n,m} \subseteq G$.

Conclusion: for all $n \leq N$, for all $m \leq M_n$: $S^{n,m} \subseteq G$.

H. *Proof of Property 8*

Take arbitrary state (l,u) that is reachable in (the semantic graph of) S. We prove that $u \not\models B(l)$ by using induction on the length of the path from (l_0,u_0) to (l,u).

Base case: $(l,u) = (l_0,u_0)$. Then, we already have assumed that $u_0 \not\models B(l_0)$.

Induction step: Assume that (l,u) is reached from a (reachable) state, say (l',u') (by an event or time transition), where the statement holds for (l',u') (by induction assumption), i.e., $u' \not\models B(l')$. We prove that the statement holds for (l,u), i.e., $u \not\models B(l)$ for different cases of transitions from (l',u') to (l,u).

(l,u) is reached from (l',u') by $\sigma \in \Sigma_c$, and assume that this transition is related to an edge $(l',\sigma,g^{M_N},r,l) \in E_S^{N,M_N}$. According to Definition 7, $u' \models g^{M_N}$. Also, based on line 11, g^{M_N} has been adapted in the last iteration such that $u' \models \neg B(l)[r]$, which is equivalent to $u'[r] \models \neg B(l)$. Again according to Definition 7, $u = u'[r]$, and so $u \models \neg B(l)$, which is equivalent to $u \not\models B(l)$.

(l,u) is reached from (l',u') by a time transition, say Δ, where $F_S(l) \neq \varnothing$. According to Definition 7, $u'+\Delta \models I^N(l)$. Based on line 16, $I_S^N(l)$ has been adapted in the last iteration such that $u'+\Delta \not\models B(l')$. Again according to Definition 7, $l'=l$, and $u = u'+\Delta$. So, $u \not\models B(l)$

(l,u) is reached from (l',u') by $\sigma \in \Sigma_{uc}$, and assume that this transition is related to an edge $(l',\sigma,g^{M_N},r,l) \in E_S^{N,M_N}$. Then, $u' \models g^{M_N} \wedge I_S^N(l)[r]$ by Definition 7. By contradiction, assume that $u \models B(l)$. Then, $u' \models B(l)[r]$ because $u' = u[r]$. So, $u' \models B(l')$ as $u' \models \textcircled{5}$ in the bad state predicate computation of l'. $u' \models B(l')$ contradicts the induction assumption, and consequently it must be the case that $u \not\models B(l)$ as required.

(l,u) is reached from (l',u') by a time transition, say Δ, where $F_S(l) = \varnothing$. Then, Also, $u'+\delta \models I_S^N(l)$ for all $\delta \leq \Delta$ by Definition 7. By contradiction, assume that $u \models B(l)$. Then $u'+\Delta \models B(l)[r]$ because $u' = u+\Delta$. Since $F_S(l)=\varnothing$, the condition on δ' in the bad state predicate computation of l' always gives true. As a result, $u' \models B(l')$ as $u' \models \textcircled{6}$ in the bad state predicate computation. $u' \models B(l')$ contradicts the induction assumption, and consequently it must be the case that $u \not\models B(l)$ as required.

Conclusion: for any reachable state (l,u) (in the semantic graph) of S: $u \not\models B(l)$.

I. Proof of Theorem 1

We need to prove that for any $w \in L(S||G)$ and $\sigma \in \Sigma_{uc} \cup \mathbb{R}_{\geq 0}$, whenever $w\sigma \in L(G)$, then $w\sigma \in L(S||G)$, or $\sigma \in \mathbb{R}_{\geq 0}$ and $w\sigma' \in L(S||G)$ for some $\sigma' \in \Sigma_{for}$. Consider arbitrary $w \in L(S||G)$ and $\sigma \in \Sigma_{uc} \cup \mathbb{R}_{\geq 0}$, and assume that $w\sigma \in L(G)$. Now assume that $\sigma \notin \mathbb{R}_{\geq 0}$ or $w\sigma' \notin L(S||G)$ for all $\sigma' \in \Sigma_{for}$. It suffices to prove that $w\sigma \in L(S||G)$. Since $S \subseteq G$ (based on Property 7), it suffices to prove $w\sigma \in L(S)$.

To conclude that $w\sigma \in L(S)$, we prove that for all $n \leq N$, for all $m \leq M_n$: $w\sigma \in L(S^{n,m})$ using nested induction on n and m. Induction on n:

Base case: $n=0$. We prove that for all $m \leq M_0$, $w\sigma \in L(S^{0,m})$ using induction on m:

- **Base case:** $w\sigma \in L(S^{0,0})$ since $S^{0,0} = G$ and $w\sigma \in L(G)$ as assumed.
- **Induction step:** assume $w\sigma \in L(S^{0,m})$.
  $S^{0,m+1}$ may differ from $S^{0,m}$ only because the guards of some edges labeled by controllable events have been modified. Thereto, nothing changes in terms of the occurrence of an uncontrollable event or a time transition so that $w\sigma \in L(S^{0,m+1})$.
- **Conclusion:** for all $m \leq M_0$: $w\sigma \in L(S^{0,m})$.

Induction step: assume that for all $m \leq M_n$, $w\sigma \in L(S^{n,m})$. We prove that for all $m \leq M_{n+1}$, $w\sigma \in L(S^{n+1,m})$ using induction on m:

- **Base case:** we prove that $w\sigma \in L(S^{n+1,0})$.
  Since $w \in L(S||G)$, $w \in L(S)$, and so $w \in L(S^{n,m})$ for any n,m as S is the final result of the algorithm. For $w \in L(S^{n,0})$ (it holds by the induction assumption), assume that there exists some $l_s \in L$ and a clock valuation u_s such that (l_s,u_s) is reached from $(l_0,\mathbf{0})$ by w in (the semantic graph of) $S^{n,0}$. To conclude $w\sigma \in L(S^{n+1,0})$, we prove that σ occurs at (l_s,u_s) in $S^{n+1,0}$ for different cases of σ:
  $\sigma \in \Sigma_{uc}$. Based on the assumption, σ occurs at (l_s,u_s) in $S^{n,0}$. Assume that σ transition is related to an edge $e = (l_s,\sigma,g,r,l_t)$ in $S^{n,0}$. Then, according to Definition 7: $u_s \models e.g^0$ and $u_s[r] \models I_S^n(l_t)$. So, $u_s \models e.g^0$, and it suffices to prove that $u_s[r] \models I_S^{n+1}(l_t)$. We continue the proof by contradiction. Assume that $u_s[r] \not\models I_S^{n+1}(l_t)$. Then, based on line 16, $u_s[r] \models B^{n,0}(l_t)$ because already $u_s[r] \models I_S^n(l_t)$ as assumed. Considering the computation of the bad state predicate of l_s, $u_s \models B^{n,0}(l_s)$ because $u_s \models e.g^0 \wedge I_S^n(l_t)[r] \wedge B^{n,0}(l_t)[r]$.
  Since $u_s \models B^{n,0}(l_s)$, based on Property 4, (l_s,u_s) is a bad state, and this contradicts the assumption that (l_s,u_s) is reachable in S because then due to Property 8, (l_s,u_s) is not a bad state.
  $\sigma = \Delta$, and there is no $\sigma' \in \Sigma_{for}$ such that $w\sigma' \in L(S||G)$. Then, for $w\sigma \in L(S^{n,0})$, according to Definition 7: $u_s + \delta \models I_S^n(l_s)$ for all $\delta \leq \Delta$. Also, the algorithm does not change the invariant as $F_S(l_s) = \varnothing$ so that $I_S^{n+1}(l_s) = I_S^n(l_s)$. So, σ occurs at (l_s,u_s) in $S^{n+1,0}$.
- **Induction step:** assume $w\sigma \in L(S^{n+1,m})$. Then, $w\sigma \in L(S^{n+1,m+1})$ for the same reason stated in the previous induction step on m.
- **Conclusion:** for all $m \leq M_n$, $w\sigma \in L(S^{n,m})$.

Conclusion: for all $n \leq N$, for all $m \leq M_n$: $w\sigma \in L(S^{n,m})$.

J. Proof of Theorem 2

First of all $L(S) \subseteq L(G)$, and so $L(S||G) = L(S)$. So, it suffices to prove that S is nonblocking, i.e., any reachable state in (the semantic graph of) S is nonblocking.

Take arbitrary state (l,u) that is reachable in (the semantic graph of) S. According to Property 8, $u \not\models B^{N,M_N}(l)$. This, based on Property 4, means that (l,u) is not a bad state in S^{N,M_N} where $S^{N,M_N} = S$ (S^{N,M_N} is the final result of the algorithm). So, (l,u) is not a blocking state in (the semantic graph of) S as (l,u) is not a bad state.

K. Proof of Theorem 3

We need to prove that for any other proper supervisor S': $L(S'||G) \subseteq L(S||G)$. Take arbitrary $w \in L(S'||G)$. We need to prove that $w \in L(S||G)$. Since $L(S) \subseteq L(G)$, it suffices to prove that $w \in L(S)$. We do the proof by induction on the structure of w:

Base case: Assume $w = \varepsilon$. Then $w \in L(S)$ by definition.

Induction step: Assume $w = v\sigma$ for some $v \in (\Sigma_G \cup \mathbb{R}_{\geq 0})^*$ and $\sigma \in \Sigma_G \cup \mathbb{R}_{\geq 0}$ where the statement holds for v, i.e., $v \in$

$L(S)$. It suffices to prove that the statement holds for $v\sigma$, i.e., $v\sigma \in L(S)$. To conclude that $v\sigma \in L(S)$, we prove that for all $n \leq N$, for all $m \leq M_n$: $v\sigma \in L(S^{n,m})$ using nested induction on n and m. Induction on n:

- **Base case:** $n = 0$. We prove that for all $m \leq M_0$: $v\sigma \in S^{0,m}$ by induction on m:
  - **Base case:** $v\sigma \in L(S^{0,0})$ because $S^{0,0} = G$, and $v\sigma \in L(G)$ as $v\sigma \in L(S'||G)$ by assumption.
  - **Induction step:** assume that $v\sigma \in L(S^{0,m})$. It suffices to prove that $v\sigma \in L(S^{0,m+1})$. $S^{0,m+1}$ differs from $S^{0,m}$ only in terms of the guards of (some) controllable edges. So, according to Definition 7, $v\sigma \in L(S^{0,m+1})$ for $\sigma \in \Sigma_{uc} \cup \mathbb{R}_{\geq 0}$ since the guards of uncontrollable edges and the invariants stay the same, and already $v\sigma \in L(S^{0,m})$.
    Let us say that $\sigma \in \Sigma_c$, and assume that for $v\sigma \in L(S^{0,m})$, there exists states (l_s, u_s) (in the semantic graph of $S^{0,m}$) reached by v from the initial state, and (l_t, u_t) reached from (l_s, u_s) by σ. Then, due to Definition 7, $u_s \models e.g^m$, and $u_s[r] \models I^0(l_t)$. To conclude that $v\sigma \in L(S^{0,m+1})$, it suffices to prove that $u_s \models e.g^{m+1}$ because the invariants stay the same. Assume that $u_s \not\models e.g^{m+1}$. Then, considering line 11, $u_s \models B^{0,m}(l_t)[r]$. Again by Definition 7, $u_s[r] = u_t$. So, $u_t \models B^{0,m}(l_t)$ as $u_s \models B^{0,m}(l_t)[r]$. Then, by Property 4, (l_t, u_t) is a bad state, and this contradicts the assumption that S' is a proper supervisor as it does not prevent all the bad states to take care nonblockingness and controllability.
  - **Conclusion:** for all $m \leq M_0$: $v\sigma \in L(S^{0,m})$.
- **Induction step:** assume that for all $m \leq M_n$: $v\sigma \in L(S^{n,m})$. We prove that for all $m \leq M_{n+1}$: $v\sigma \in L(S^{n+1,m})$ using induction on m.
  - **Base case:** We need to prove that $v\sigma \in L(S^{n+1,0})$.
    For $v\sigma \in L(S^{n,0})$ (it holds by the induction assumption), assume that there exists some $l_s \in L$ and a clock valuation u_s such that (l_s, u_s) is reached from $(l_0, \mathbf{0})$ by v in (the semantic graph of) $S^{n,0}$. To conclude $v\sigma \in L(S^{n+1,0})$, we prove that σ occurs at (l_s, u_s) in $S^{n+1,0}$ for different cases of σ:
    σ is an event transition, related to an edge (l_s, σ, g, r, l_t). Then, according to Definition 7: $u_s \models e.g^0$ and $u_s[r] \models I^n_S(l_t)$. So, $u_s \models e.g^0$, and it suffices to prove that $u_s[r] \models I^{n+1}_S(l_t)$. We continue the proof by contradiction. Assume that $u_s[r] \not\models I^{n+1}_S(l_t)$. Then, based on line 16, $u_s[r] \models B^{n,0}(l_t)$ because already $u_s[r] \models I^n_S(l_t)$ as assumed. So, $u_t \models B^{n,0}(l_t)$ as $u_s[r] = u_t$ (again by Definition 7), and based on Property 4, (l_t, u_t) is a bad state, and this contradicts the assumption that S' is a proper supervisor.
    σ is a time transition, say Δ. Then, for $v\sigma \in L(S^{n,0})$, according to Definition 7: $u_s + \delta \models I^n_S(l_s)$ for all $\delta \leq \Delta$. Also, $l_t = l_s$, and $u_t = u_s + \Delta$. It suffices to prove that $u_s + \delta \models I^{n+1}_S(l_s)$ for all $\delta \leq \Delta$. By contradiction, assume that for some $\delta \leq \Delta$, $u_s + \delta \not\models I^{n+1}_S(l_s)$. Then, based on line 16, $u_s + \delta \models B^{n,0}(l_s)$ because $u_s + \delta \models I^{n+1}_S(l_s)$. In this case, based on Property 4, $(l_t, u_s + \delta)$ is a bad state. This contradicts the assumption that S' is a proper supervisor because, as a proper supervisor, it should prevent $(l_t, u_s + \delta)$. However, $(l_t, u_s + \delta)$ can be reached through the Δ transition that occurs in S'.
  - **Induction step:** assume that $v\sigma \in L(S^{n+1,m})$. Then, $v\sigma \in L(S^{n+1,m+1})$ for the same reason states in the previous induction step on m.
  - **Conclusion:** for all $m \leq M_n$: $v\sigma \in L(S^{n,m})$.
- **Conclusion:** for all $n \leq N$, for all $m \leq M_n$: $v\sigma \in L(S^{n,m})$.

Conclusion: by the principle of induction, $w \in L(S||G)$ for all $w \in L(S'||G)$.

L. *Proof of Theorem 4*

This proof is inspired from the proof of safety in [32]. Since S and G has the same event set Σ_G and $\Sigma_R \subseteq \Sigma_G$, $\Sigma_G \cap \Sigma_R = \Sigma_R$. So, it suffices to prove that if we take any $w \in P_{\Sigma_R}(L(S||(G||R^\perp)))$: $w \in L(R)$. Take $w \in P_{\Sigma_R}(L(S||(G||R^\perp)))$, then due to the projection properties, there exists $w' \in L(S||(G||R^\perp))$ such that $P_{\Sigma_R}(w') = w$. Also, based on Property 7, $L(S) \subseteq L(G||R^\perp)$, and so $w' \in L(G||R^\perp)$. Applying the projection on Σ_R gives $P_{\Sigma_R}(w') \in L(R^\perp)$. For $w \in P_{\Sigma_R}(L(S||(G||R^\perp))) \cap L(R^\perp)$, $w \in L(R)$ since the blocking state q_d added to $G||R$ to make $G||R^\perp$ is removed by S as guaranteed by Theorem 2.

References

[1] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[2] W. M. Wonham, "Supervisory control of discrete-event systems," *Encyclopedia of systems and control*, pp. 1396–1404, 2015.

[3] M. Skoldstam, K. Akesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *2007 46th IEEE Conference on Decision and Control*. IEEE, 2007, pp. 3387–3392.

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

[5] F. Lin, "Control of networked discrete event systems: Dealing with communication delays and losses," *SIAM Journal on Control and Optimization*, vol. 52, no. 2, pp. 1276–1298, 2014.

[6] A. Rashidinejad, M. Reniers, and L. Feng, "Supervisory control of timed discrete-event systems subject to communication delays and non-FIFO observations," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 456 – 463, 2018, 14th IFAC Workshop on Discrete Event Systems WODES 2018.

[7] W. M. H. Heemels, A. R. Teel, N. Van de Wouw, and D. Nesic, "Networked control systems with communication constraints: Tradeoffs between transmission intervals, delays and performance," *IEEE Transactions on Automatic Control*, vol. 55, no. 8, pp. 1781–1796, 2010.

[8] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[9] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[10] A. Khoumsi, "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)*. IEEE, 2002, pp. 105–112.

[11] S. Miremadi, Z. Fei, K. Åkesson, and B. Lennartson, "Symbolic supervisory control of timed discrete event systems," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 584–597, 2015.

[12] A. Dubey, "A discussion on supervisory control theory in real-time discrete event systems," *ISIS*, vol. 9, p. 112, 2009.

[13] H. Wong-Toi and G. Hoffmann, "The control of dense real-time discrete event systems," in *Proceedings of the 30th IEEE Conference on Decision and Control*, 1991, pp. 1527–1528.

[14] S. Tripakis and K. Altisen, "On-the-fly controller synthesis for discrete and dense-time systems," in *International Symposium on Formal Methods*. Springer, 1999, pp. 233–252.

[15] O. Maler, A. Pnueli, and J. Sifakis, "On the synthesis of discrete controllers for timed systems," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 1995, pp. 229–242.

[16] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, “Controller synthesis for timed automata,” *IFAC Proceedings Volumes*, vol. 31, no. 18, pp. 447–452, 1998.

[17] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime, “Efficient on-the-fly algorithms for the analysis of timed games,” in *International Conference on Concurrency Theory*. Springer, 2005, pp. 66–80.

[18] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, “Uppaal-tiga: Time for playing games!” in *International Conference on Computer Aided Verification*. Springer, 2007, pp. 121–125.

[19] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi, “Supervisory control and reactive synthesis: a comparative introduction,” *Discrete Event Dynamic Systems*, vol. 27, no. 2, pp. 209–260, 2017.

[20] A. Khoumsi and M. Nourelfath, “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)*, 2002.

[21] S. Tripakis and S. Yovine, “Analysis of timed systems using time-abstracting bisimulations,” *Formal Methods in System Design*, vol. 18, no. 1, pp. 25–68, 2001.

[22] L. Ouedraogo, A. Khoumsi, and M. Nourelfath, “Setexp: a method of transformation of timed automata into finite state automata,” *Real-Time Systems*, vol. 46, no. 2, pp. 189–250, 2010.

[23] A. Rashidinejad, P. van der Graaf, and M. Reniers, “Nonblocking supervisory control synthesis of timed automata using abstractions and forcible events,” in *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, 2020, pp. 1–8.

[24] D. A. van Beek, W. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. Van De Mortel-Fronczak, and M. A. Reniers, “Cif 3: Model-based engineering of supervisory controllers,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 575–580.

[25] K. Akesson, M. Fabian, H. Flordal, and R. Malik, “Supremica-an integrated environment for verification, synthesis and simulation of discrete event systems,” in *2006 8th International Workshop on Discrete Event Systems*. IEEE, 2006, pp. 384–385.

[26] A. Rashidinejad, P. van der Graaf, M. Reniers, and M. Fabian, “Non-blocking supervisory control of timed automata using forcible events,” in *15th International Workshop on Discrete Event Systems (WODES 2020)*. IEEE, 2020, accepted. [Online]. Available: https://michelreniers.files.wordpress.com/2020/06/wodes20_0055_fi.pdf

[27] J. Bengtsson and W. Yi, *Timed Automata: Semantics, Algorithms and Tools*. Springer Berlin Heidelberg, 2004, pp. 87–124.

[28] R. Alur, “Timed automata,” in *International Conference on Computer Aided Verification*. Springer, 1999, pp. 8–22.

[29] B. A. Brandin and W. M. Wonham, “Supervisory control of timed discrete-event systems,” *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[30] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, “Nonblocking and safe control of discrete-event systems modeled as extended finite automata,” *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, 2011.

[31] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, “Compositional synthesis of maximally permissive supervisors using supervision equivalence,” *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, 2007.

[32] A. Rashidinejad, M. Reniers, and M. Fabian, “Networked supervisory control synthesis of timed discrete-event systems,” 2020, manuscript submitted for publication.

Aida Rashidinejad received the M.Sc. degree in electrical-control engineering from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2014. She is currently working towards PhD degree in mechanical engineering-control systems from Eindhoven University of Technology, Eindhoven, The Netherlands. Her current research interests include supervisory control synthesis, networked control, and cyber-physical systems.

Michel Reniers (S’17) is currently an Associate Professor in model-based engineering of supervisory control at the Department of Mechanical Engineering at TU/e. He has authored over 100 journal and conference papers. His research portfolio ranges from model-based systems engineering and model-based validation and testing to novel approaches for supervisory control synthesis. Applications of this work are mostly in the areas of cyber-physical systems.

Martin Fabian is Professor in Automation and Head of the Automation Research group at the Department of Electrical Engineering, Chalmers University of Technology. His research interests include formal methods for automation systems in a broad sense, merging the fields of Control Engineering and Computer Science. He has authored more than 200 publications, and is co-developer of the formal methods tool Supremica, which implements several state-of-the-art algorithms for supervisory control synthesis.