# Automated Incremental Synthesis of Timed Automata[1]

Borzoo Bonakdarpour , Sandeep S. Kulkarni

Department of Computer Science and Engineering,
Michigan State University,
East Lansing, MI 48824, USA
Email: {borzoo, sandeep}@cse.msu.edu
http://www.cse.msu.edu/~{borzoo,sandeep}

**Abstract.** In this paper, we concentrate on incremental synthesis of timed automata for automatic addition of different types of bounded response properties. Bounded response – that *something good* will happen soon, in a certain amount of time – captures a wide range of requirements for specifying real-time and embedded systems. We show that the problem of automatic addition of a bounded response property to a given timed automaton while maintaining maximal nondeterminism is NP-hard in the size of locations of the input automaton. Furthermore, we show that by relaxing the maximality requirement, we can devise a sound and complete algorithm that adds a bounded response property to a given timed automaton, while preserving its existing universally quantified properties (e.g., MTL). This synthesis method is useful in adding properties that are later discovered as a crucial part of a system. Moreover, we show that addition of interval-bounded response, where the good thing should not happen sooner than a certain amount of time, is also NP-hard in the size of locations even without maximal nondeterminism. Finally, we show that the problems of adding bounded and unbounded response properties are both PSPACE-complete in the size of the input timed automaton.

**Keywords: Timed automata, Transformation, Synthesis, Real-time, Bounded liveness, Bounded response, Formal methods.**

## 1 Introduction

As the traditional approaches to software development turn out to be inefficient in many cases (e.g., due to maintenance, resolving bugs, etc.), *correct-by-construction* approaches to treat software development as a true form of engineering gains more attention. In this approach, a software engineer constructs a mathematical model of his/her design before any code is produced. This model is used to reason about the proposed solution, ensuring that all required functionality will be delivered.

[1] This is an extended version of a paper appeared in the proceedings of *FMICS'06: International Workshop on Formal Methods in Industrial Critical Systems*, LNCS, Springer-Verlag, 2006. This work was partially sponsored by NSF CAREER CCR-0092724, DARPA Grant OSURS01-C-1901, ONR Grant N00014-01-1-0744, NSF grant EIA-0130724, and a grant from Michigan State University.

*Automated program synthesis* is the problem of designing an algorithmic method to find a program that satisfies a mathematical model (i.e., a required set of properties) that is correct-by-construction. The synthesis problem has mainly been studied in two contexts: synthesizing programs from specification, where the entire specification is given, and synthesizing programs from existing programs along with a fully or partially available new specification. In approaches where the entire specification must be available, changes in specification, e.g., addition of a new property, requires us to begin from scratch. By contrast, in the latter approach, it is possible to *reuse* an existing program and, hence, the previous efforts made for synthesizing it. Since it may not be possible to anticipate all the necessary required properties at design time, this approach is especially useful in program maintenance, where the program needs to be modified so that it satisfies a new property of interest.

In order to *add* a new property to a program there are two ways: (1) *comprehensive redesign*, where the designer introduces new behaviors (e.g., by introducing new variables, or adding new computation paths), or (2) *local redesign*, where the designer removes behaviors that violate the property of interest, but does not add any new behaviors. While the former requires the designer to verify all other properties of the new program, the latter ensures that certain existing universally quantified properties (e.g., LTL and MTL) are preserved.

Depending upon the choice of formulation of the problem and expressiveness of specifications and programs, the class of complexity of synthesis methods varies from polynomial time to undecidability. In this paper, we focus on incremental synthesis methods that add properties typically used for specifying timing constraints. This approach is opposite to those synthesize arbitrary specifications and, hence, belong to high classes of complexity. More specifically, we study the problem of incremental addition of different types of *bounded response* properties – that something good will happen soon, in a certain amount of time – to Alur and Dill's timed automata [1], while preserving existing Metric Temporal Logic (MTL) specification [2]. A more practical application of the results presented in this paper is in aspect-oriented programming. Indeed, our synthesis methods is close in spirit to automated *weaving* of real-time aspects.

### 1.1 Related Work

In the context of untimed systems, in the pioneering work [3, 4], the authors propose methods for synthesizing the synchronization skeleton of programs from their temporal logic specification. More recently, in [5–7], the authors investigate algorithmic methods to locally redesign fault-tolerant programs using their existing fault-intolerant version and a partially available safety specification. In [8], the authors introduce a synthesis algorithm that adds UNITY properties [9] such as leads-to (which is an unbounded response property) to untimed programs.

Controller synthesis is the problem of finding an automaton (called *controller*) such that its parallel composition with a given automaton (called *plant*) satisfies a set of properties [10]. Synthesis of real-time systems has mostly been formulated in the context of timed controller synthesis. In the early work [11–13], the authors investigate the problem, where the given program is a *deterministic* timed automaton and the specification is modeled as a deterministic *internal* winning condition on the state space of the plant. The authors also assume that the controller can use *unlimited* resources (i.e., the number of new clocks and guards that compare the clocks to constants). Similarly, in [14], the authors solve the reachability problem in timed games. Deciding the existence of a winning condition with the formulation presented in [11–14] is shown to be EXPTIME-complete in [15].

In [16, 17], the authors address the problem of synthesizing timed controllers with limited resources. Similar to the aforementioned work, the plant is modeled by a deterministic timed automaton, but the specification is given by an *external* nondeterministic timed automaton that describes *undesired* behavior of the plant. With this formulation, the synthesis problem is 2EXPTIME-complete. However, if the given specification remains nondeterministic, but describes *desired* behavior of the plant the problem turns out to be undecidable.

In [18], the authors propose a synthesis method for timed games, where the game is modelled as a timed automaton, the winning condition is described by TCTL-formulae, and unlimited resources are available. In [19], the authors consider concurrent two-person games given by a timed automaton played in real-time and provide symbolic algorithms for solving them with respect to all $\omega$-regular winning conditions. In both approaches, deciding the existence of a winning strategy is EXPTIME-complete.

### 1.2 Contributions

In our work, we consider (i) the case where the entire specification of the program is not given to the synthesis algorithm; and (ii) *nondeterministic* timed automata. In fact, we study how the *level of nondeterminism* affects the complexity of synthesis methods. The main results in this paper are as follows:

- We show that adding a bounded response property while maintaining maximal nondeterminism is NP-hard in the size of the locations of the given timed automaton.
- Based on the above result and the NP-hardness of adding two bounded response properties without maximal nondeterminism [2], we focus on addition of a single bounded response property to a time automaton without maximal nondeterminism. In fact, we present a surprising result that by dropping the maximality requirement we can devise a simple *sound* and *complete* transformation algorithm that adds a bounded response property to a timed automaton while existing MTL properties. Note that since our algorithm is complete, if it fails to synthesize a solution then it informs the designer that a more comprehensive (and expensive) approach *must* be used. Moreover, since the complexity of our algorithm is comparable with that of model checking, the algorithm has the potential to provide timely insight to the designer about how the given program needs to be modified to meet the required bounded response property. Thus, in this paper, we extend the results presented in [8] to the context of timed automata.
- We show that adding *interval-bounded response*, where the good thing should not happen sooner than a certain amount of time, is also NP-hard in the size locations of the given timed automaton even without maximal nondeterminism.

[2] In [8], it is shown that adding two unbounded response properties to an untimed program is NP-hard. The same proof can be easily extended to the problem of adding two bounded response properties to a timed automaton.

– We show that the problems of adding bounded and *unbounded response* (also called *leads-to*) properties are both PSPACE-complete in the size of the input timed automaton.

Table 1 compares the complexity of our approach and other synthesis methods in the literature. A natural question is "since direct synthesis of limited MTL to bounded response properties is PSPACE-complete, what is the advantage of our method over direct synthesis?". There are two advantages:

– Since we incrementally add properties to a given timed automaton while preserving its existing MTL specification, we do not need to have this specification at hand. This is particularly useful when the existing specification includes properties whose automated synthesis is undecidable (e.g., $\Diamond_{=\delta} q$) or lies in classes of complexity higher than PSPACE.
– The second advantage of our approach is in cases where the given timed automaton is designed manually for ensuring that it is efficient. Since in our approach, existing computations are preserved it has the potential to preserve the efficiency of the given timed automaton.

| Adding Bounded Response (This paper) | Direct Synthesis from MTL [20] | Timed Control Synthesis [16, 17] | Timed Games [11, 13, 14, 18, 19] |
|---|---|---|---|
| PSPACE-*complete* | EXPSPACE-*complete* | 2EXPTIME-*complete* | EXPTIME-*complete* |

**Table 1.** Complexity of different synthesis approaches.

**Organization of the paper.** In Section 2, we present the preliminary concepts. In Section 3, we formally state the problem of addition of an MTL property to an existing timed automaton. We describe the NP-hardness result for adding bounded response with maximal nondeterminism in Section 4. Then, in Section 5, we present a sound and complete algorithm for adding bounded response to timed automata without maximal nondeterminism. In Section 6, we present the complexity of addition of interval-bounded response and unbounded response properties. Finally, we make the concluding remarks and discuss future work in Section 7.

## 2 Preliminaries

Let $AP$ be a set of *atomic propositions*. A state is a subset of $AP$. A *timed state sequence* is an infinite sequence of pairs $(\sigma, \tau) = (\sigma_0, \tau_0), (\sigma_1, \tau_1)...$, where $\sigma_i$ ($i \in \mathbb{N}$) is a state and $\tau_i \in \mathbb{R}_{\geq 0}$, and satisfies the following constraints:

1. *Initialization*: $\tau_0 = 0$.
2. *Monotonicity*: $\tau_i \leq \tau_{i+1}$ for all $i \in \mathbb{N}$.
3. *Progress*: For all $t \in \mathbb{R}_{\geq 0}$, there exists $j$ such that $\tau_j \geq t$.

### 2.1 Metric Temporal Logic

We briefly recap the syntax and semantics of *point-based* MTL. Linear Temporal Logic (LTL) specifies the *qualitative part* of a program. MTL introduces real time by constraining temporal operators, so that one can specify the *quantitative part* as well. For instance, the constrained eventually operator $\Diamond_{[1,3]}$ is interpreted as "eventually within 1 to 3 time units both inclusive".

**Syntax.** Formulae of MTL are inductively defined by the grammar: $\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U}_I \phi_2$, where $p \in AP$ and $I \subseteq \mathbb{R}_{\geq 0}$ is an open, closed, half-open, bounded, or unbounded interval with endpoints in $\mathbb{Z}_{\geq 0}$. For simplicity, we use $\Diamond_I \phi$ and $\Box_I \phi$ instead of $true \mathcal{U}_I \phi$ and $\neg\Diamond_I \neg\phi$. We also use pseudo-arithmetic expressions to denote intervals. For instance, "$\leq 4$" means $[0, 4]$.

**Semantics.** For an MTL formula $\phi$ and a timed state sequence $(\sigma, \tau) = (\sigma_0, \tau_0), (\sigma_1, \tau_1)...$, the satisfaction relation $(\sigma_i, \tau_i) \models \phi$ is defined inductively as follows:

$(\sigma_i, \tau_i) \models p$ iff $\sigma_i \models p$ ($\sigma_i \models p$ iff $p \in \sigma_i$ and we say $\sigma_i$ is a $p$-state);
$(\sigma_i, \tau_i) \models \neg\phi$ iff $(\sigma_i, \tau_i) \not\models \phi$;
$(\sigma_i, \tau_i) \models \phi_1 \wedge \phi_2$ iff $(\sigma_i, \tau_i) \models \phi_1 \wedge (\sigma_i, \tau_i) \models \phi_2$;
$(\sigma_i, \tau_i) \models \phi_1 \mathcal{U}_I \phi_2$ iff there exists $j > i$ such that $\tau_j - \tau_i \in I$ and $(\sigma_{i'}, \tau_{i'}) \models \phi_1$ for all $i'$, where $i \leq i' < j$, and $(\sigma_j, \tau_j) \models \phi_2$.

A timed state sequence $(\sigma, \tau)$ satisfies the formula $\phi$ if $(\sigma_0, \tau_0) \models \phi$.

The formula $\phi$ defines a set $\mathcal{L}$ of timed state sequences that satisfy $\phi$. We call this set a *property*. A *specification* $\Sigma$ is the conjunction of a set of properties. In this paper, we focus on a standard class of real-time properties defined as follows. An *interval-bounded response* property is of the form $\mathcal{L}_I \equiv \Box(p \rightarrow \Diamond_{[\delta_1,\delta_2]} q)$, where $p, q \in AP$ and $\delta_1, \delta_2 \in \mathbb{Z}_{\geq 0}$, i.e., it is always the case that a $p$-state is followed by a $q$-state within $\delta_2$, but not sooner than $\delta_1$ time units. A special case of $\mathcal{L}_I$ is in which $\delta_1 = 0$ known as *bounded response* property and is of the form $\mathcal{L}_B \equiv \Box(p \rightarrow \Diamond_{\leq\delta} q)$, i.e., it is always the case that a $p$-state is followed by a $q$-state within $\delta$ time units. An *unbounded response* (or *leads-to*) property is defined as $\mathcal{L}_\infty \equiv \Box(p \rightarrow \Diamond_{[0,\infty)} q)$, i.e, it is always the case that a $p$-state is eventually followed by a $q$-state.

### 2.2 Timed Automata

A *clock constraint* over the set $X$ of clock variables is a Boolean combination of formulas of the form $x \preceq c$ or $x - y \preceq c$, where $x, y \in X$, $c \in \mathbb{Z}_{\geq 0}$, and $\preceq$ is either $<$ or $\leq$. We denote the set of all clock constraints over $X$ by $\Phi(X)$. A *clock valuation* is a function $\nu : X \rightarrow \mathbb{R}_{\geq 0}$ that assigns a real value to each clock variable. Furthermore, for $\tau \in \mathbb{R}_{\geq 0}$, $\nu + \tau = \nu(x) + \tau$ for every clock $x$. Also, for $Y \subseteq X$, $\nu[Y := 0]$ denotes the clock valuation for X which assigns 0 to each $x \in Y$ and agrees with $\nu$ over the rest of the clock variables in $X$.

**Definition 2.1.** A *timed automaton* $\mathcal{A}$ is a tuple $\langle L, L^0, \psi, X, E \rangle$, where

- $L$ is a finite set of *locations*,
- $L^0 \subseteq L$ is a set of *initial locations*,

– $\psi : L \rightarrow 2^{AP}$ is a labeling function assigning to each location the set of atomic propositions true in that location,
– $X$ is a finite set of clocks, and
– $E \subseteq (L \times 2^X \times \Phi(X) \times L)$ is a set of *switches*. A switch $\langle s_0, \lambda, \varphi, s_1 \rangle$ represents a transition from location $s_0$ to location $s_1$ under clock constraint $\varphi$ over $X$, such that it specifies when the switch is enabled. The set $\lambda \subseteq X$ gives the clocks to be reset with this switch. $\square$

The semantics of a timed automaton is as follows. A *state* of a timed automaton is a pair $(s, \nu)$, such that $s$ is a location and $\nu$ is a clock valuation for $X$ at location $s$. The labeling function for states is defined by $\psi'((s, \nu)) = \psi(s)$. Thus, if $p \in \psi(s)$, $s$ is a $p$-location (i.e., $s \models p$) and $(s, \nu)$ is a $p$-state for all $\nu$. An initial state of $\mathcal{A}$ is $(s_{init}, \nu_{init})$ where $s_{init} \in L^0$ and $\nu_{init}$ maps the value of all clocks in $X$ to 0. *Transitions* of $\mathcal{A}$ are of the form $(s_0, \nu_0) \rightarrow (s_1, \nu_1)$. They are classified into two types:

– **Delay:** for a state $(s, \nu)$ and a time increment $\tau \in \mathbb{R}_{\geq 0}$, $(s, \nu) \xrightarrow{\tau} (s, \nu + \tau)$.
– **Location switch:** for a state $(s_0, \nu)$ and a switch $(s_0, \lambda, \varphi, s_1)$ such that $\nu$ satisfies the clock constraint $\varphi$, $(s_0, \nu) \rightarrow (s_1, \nu[\lambda := 0])$.

We use the well-known *railroad crossing problem* [21] as a running demonstration throughout the paper. The original problem comprises of three timed automata, but we only consider the TRAIN automaton (cf. Figure 1-a). The TRAIN automaton models the behavior of a train approaching a railroad crossing. Initially, the train is far from the gateway of the crossing. It announces approaching the gateway by resetting the clock variable $x$. The train is required to start crossing the gateway after at least 2 minutes. It passes the gateway at least 3 minutes after approaching the gateway. Finally, there is no constraint on reaching the initial location.

We now define what it means for a timed automaton $\mathcal{A}$ to satisfy an MTL specification $\Sigma$. An infinite sequence $(s_0, \nu_0, \tau_0), (s_1, \nu_1, \tau_1)...$, where $\tau_i \in \mathbb{R}_{\geq 0}$, is a *computation* of $\mathcal{A}$ iff for all $j > 0$ (1) $(s_{j-1}, \nu_{j-1}) \rightarrow (s_j, \nu_j)$ is a transition of $\mathcal{A}$, (2) the sequence $\tau_0\tau_1...$ satisfies initialization, monotonicity, and progress, and (3) $\tau_j - \tau_{j-1}$ is consistent with $\nu_j - \nu_{j-1}$. We write $\mathcal{A} \models \Sigma$ and say that timed automaton $\mathcal{A}$ *satisfies* specification $\Sigma$ iff every computation of $\mathcal{A}$ that starts from an initial state is in $\Sigma$. Thus, $\mathcal{A} \models (\Box(p \rightarrow \Diamond_{\leq\delta} q))$ iff any computation of $\mathcal{A}$ that reaches a $p$-state, reaches a $q$-state within $\delta$ time units. If $\mathcal{A} \not\models \Sigma$, we say $\mathcal{A}$ *violates* $\Sigma$.

### 2.3 Region Automata

Timed automata can be analyzed with the help of an equivalence relation of finite index on the set of states [1]. Given a timed automaton $\mathcal{A}$, for each clock $x \in X$, let $c_x$ be the largest constant in the guards of switches of $\mathcal{A}$ that involve $x$, where $c_x = 0$ if $x$ does not occur in any guard. Two clock valuations $\nu$, $\mu$ are *clock equivalent* if (1) for all $x \in X$, either $\lfloor \nu(x) \rfloor = \lfloor \mu(x) \rfloor$ or both $\nu(x), \mu(x) > c_x$, (2) the ordering of the fractional parts of the clock variables in the set $\{x \in X \mid \nu(x) < c_x\}$ is the same in $\mu$, and (3) for all $x \in \{y \in X \mid \nu(y) < c_y\}$, the clock value $\nu(x)$ is an integer if and only if $\mu(x)$ is an integer. A *clock region* $\rho$ is a clock equivalence class. Two states $(s_0, \nu_0)$ and $(s_1, \nu_1)$ are region equivalent, written $(s_0, \nu_0) \equiv (s_1, \nu_1)$, if (1) $s_0 = s_1$ and (2) $\nu_0$

and $\nu_1$ are clock equivalent. A *region* is an equivalence class with respect to $\equiv$. Also, region equivalence is a *time-abstract bisimulation* [1].

Using the region equivalence relation, we construct the *region automaton* of $\mathcal{A}$ (denoted $R(\mathcal{A})$) as follows. Vertices of $R(\mathcal{A})$ are regions. Edges of $R(\mathcal{A})$ are of the form $(s_0, \rho_0) \rightarrow (s_1, \rho_1)$ iff for some clock valuations $\nu_0 \in \rho_0$ and $\nu_1 \in \rho_1$, $(s_0, \nu_0) \rightarrow (s_1, \nu_1)$ is a transitions of $\mathcal{A}$. Figure 1-b shows the region automaton of the TRAIN automaton.

We say a region $(s_0, \rho_0)$ of region automaton $R(\mathcal{A})$ is a *deadlock region* iff for all regions $(s_1, \rho_1)$, there does not exist an edge of the form $(s_0, \rho_0) \rightarrow (s_1, \rho_1)$. The definition of a *deadlock state* is analogous. A clock region $\beta$ is a *time-successor* of a clock region $\alpha$ iff for each $\nu \in \alpha$, there exists $\tau \in \mathbb{R}_{>0}$, such that $\nu + \tau \in \beta$, and $\nu + \tau' \in \alpha \cup \beta$ for all $\tau' < \tau$. We call a region $(s, \rho)$ a *boundary region*, if for each $\nu \in \rho$ and for any $\tau \in \mathbb{R}_{>0}$, $\nu$ and $\nu + \tau$ are not equivalent. A region is *open*, if it is not a boundary region. A region $(s, \rho)$ is called *end region*, if $\nu(x) > c_x$ for all clocks $x$. For instance, in Figure 1-b, $(APPROACHING, x = 2)$ is a boundary region, $(CROSSING, 3 < x < 4)$ is an open region, and $(PASSED, x > 4)$ is an end region.

### 2.4 Measures of Complexity

We use two measures of complexity: (1) size of input timed automata, and (2) size of locations of input timed automata. We note that, the size of a region automaton is in linear order of the size of *locations* of its corresponding timed automaton [1]. Furthermore, the size of a region automaton is in exponential order of the size of *timing constraints* of the input timed automaton. It follows that the size of a region automaton is in exponential order of the size of the *input timed automaton*. Hence, when we say a problem is NP-hard in the size of the locations of the input automaton, it implies that the problem is NP-hard in the size of the corresponding region automaton as well. Moreover, when we say "a problem is in PSPACE", we mean "it is in PSPACE in the size of the input timed automaton".

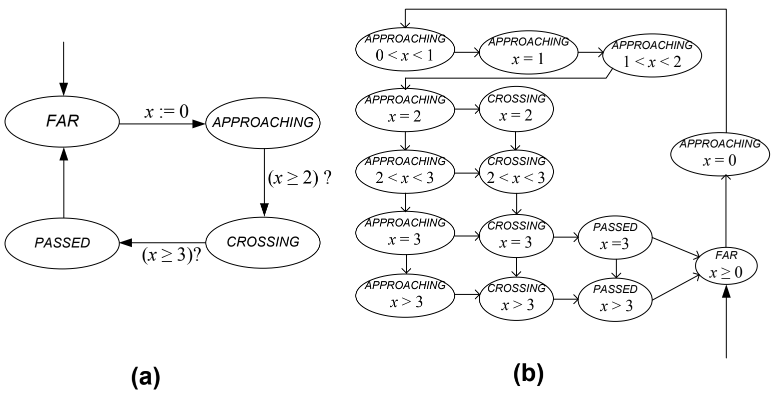


**Fig. 1.** (a) TRAIN automaton. (b) Region automaton of TRAIN automaton.

## 3 Problem Statement

Given are a timed automaton $\mathcal{A}\langle L, L^0, \psi, X, E\rangle$ and an MTL property $\mathcal{L}$ (either $\mathcal{L}_I$, $\mathcal{L}_B$, or $\mathcal{L}_\infty$). Our goal is to find a timed automaton $\mathcal{A}'\langle L', L'^0, \psi', X', E'\rangle$, such that $\mathcal{A}' \models \mathcal{L}$ and for any MTL specification $\Sigma$, if $\mathcal{A} \models \Sigma$ then $\mathcal{A}' \models \Sigma$.

Since we require that $\mathcal{A}' \models \Sigma$, if $L'$ contains locations that are not in $L$, then $\mathcal{A}'$ includes computations that are not in $\Sigma$ and as a result, $\mathcal{A}'$ may violate $\Sigma$. Hence, we require that $L' \subseteq L$ and $L'^0 \subseteq L^0$. Moreover, if $E'$ contains switches that are present in $E$, but are guarded by weaker timing constraints, or $E'$ contains switches that are not present in $E$ at all then $\mathcal{A}'$ includes computations that are not in $\Sigma$. Hence, we require that $E'$ contains a switch $\langle s_0, \lambda, \varphi', s_1\rangle$, only if there exists $\langle s_0, \lambda, \varphi, s_1\rangle$ in $E$, such that $\varphi'$ is stronger than $\varphi$. Furthermore, extending the state space of $\mathcal{A}$ by introducing new clock variables under the above circumstances is legitimate. Finally, we require $\psi'$ to be equivalent to $\psi$. Thus, the synthesis problem is as follows:

**Problem Statement 3.1.** Given $\mathcal{A}\langle L, L^0, \psi, X, E\rangle$ and an MTL property $\mathcal{L}$, identify $\mathcal{A}'\langle L', L'^0, \psi', X', E'\rangle$ such that

$(C1)$ $L' \subseteq L, L'^0 \subseteq L^0$
$(C2)$ $\psi' = \psi$
$(C3)$ $X \subseteq X'$
$(C4)$ $\forall \langle s_0, \lambda, \varphi', s_1\rangle \in E' \ : \ (\exists \ \langle s_0, \lambda, \varphi, s_1\rangle \in E : (\varphi' \Rightarrow \varphi))$
$(C5)$ $\mathcal{A}' \models \mathcal{L}$
$(C6)$ For any MTL specification $\Sigma$: $((\mathcal{A} \models \Sigma) \ \Rightarrow \ (\mathcal{A}' \models \Sigma))$ $\square$

Notice that constraint $(C6)$ implicitly implies that the synthesized program is not allowed to have deadlock states. This constraint is known as the *non-blocking condition* in the literature of controller synthesis. Furthermore, constraint $(C6)$ is similar to *language inclusion condition* in controller synthesis where the set of uncontrollable transitions is empty. Note that, based on Problem Statement 3.1, since we allow synthesis methods to remove states and transitions of a timed automaton, such methods are appropriate to preserve universally quantified properties only. In fact, constraints of Problem Statement 3.1 do not suffice to preserve existential properties of a program (e.g., TCTL).

**Soundness and completeness.** We say that an algorithm for the synthesis problem is *sound* iff its output meets the constraints of the Problem Statement 3.1. We say that an algorithm for the synthesis problem is *complete* iff it finds a solution to the Problem Statement 3.1 iff there exists one.

**Comparison to controller synthesis and game theory.** Our formulation of the synthesis problem is in spirit close to both controller synthesis and game theory where the winning condition is expressed as MTL formulae. In fact, in both problems , the objective is how to *restrict* the program actions at each state through synthesizing a controller or a wining strategy. Notice that the conditions $(C1)$ and $(C2)$ precisely express this notion of restriction. Furthermore, the condition $(C6)$ precisely expresses the notion of *language inclusion*, where the synthesized program is supposed to exhibit a subset of behaviors of the input program. As mentioned in Section 1, the main advantage of our synthesis methods over controller synthesis and game theory is our algorithms are tailored for the properties typically used in specifying real-time requirements and, hence, synthesizes programs more efficiently. Moreover, our synthesis algorithms accept nondeterministic input programs.

## 4 Adding Bounded Response Properties with Maximal Nondeterminism

In this section, we show that the synthesis problem in Problem Statement 3.1 for adding a bounded response property while maintaining maximal nondeterminism is NP-hard in the size of locations of the input timed automaton. We show this result by a reduction from the Vertex Splitting Problem [22] in directed acyclic graphs (DAG).

Given a timed automaton $\mathcal{A}$ and property $\mathcal{L}_B \equiv \Box(p \rightarrow \Diamond_{\leq\delta} q)$, we say that the synthesized timed automaton $\mathcal{A}'$ is *maximally nondeterministic* iff $\mathcal{A}'$ meets all the constraints of Problem Statement 3.1 and its set of transitions is maximal. Maintaining maximal nondeterminism is desirable in the sense that it increases the potential for future successful incremental synthesis. Indeed, in our framework, maximal nondeterminism is similar to the concept of *weakest controller* in the literature of controller synthesis.

**The DAG Vertex Splitting Problem (DVSP).** Let $G\langle V, A\rangle$ be a weighted DAG and $v_s, v_t$ be arbitrary source and target vertices in $G$. Let $G/Y$ denote the DAG when each vertex $v \in Y$ is split into vertices $v^{in}$ and $v^{out}$ such that all arcs $(v, u) \in A$, where $u \in V$, are replaced by arcs of the form $(v^{out}, u)$ and all arcs $(w, v) \in A$, where $w \in V$, are replaced by arcs of the form $(w, v^{in})$. In other words, the outgoing arcs of $v$ now leave vertex $v^{out}$ while the incoming arcs of $v$ now enter $v^{in}$, and there is no arc between $v^{in}$ and $v^{out}$. The DAG vertex splitting problem is to find a vertex set $Y$, where $Y \subseteq V$ and $|Y| \leq i$ (for some positive integer $i$), such that the length of the longest path of $G/Y$ from $v_s$ to $v_t$ is bounded by a prespecified value $d$. In [22], the authors show that DVSP is NP-hard.

We now show that the problem of adding a bounded response property while maintaining maximal nondeterminism is NP-hard.

**Instance.** A timed automaton $\mathcal{A}\langle L, L^0, \psi, X, E\rangle$, a bounded response property $\mathcal{L}_B \equiv \Box(p \rightarrow \Diamond_{\leq\delta} q)$, and a positive integer $k$, where $|E| \geq k$.

**Maximally Nondeterministic Bounded Response Addition Problem (MNBRAP).** Does there exist a timed automaton $\mathcal{A}'\langle L', L'^0, \psi', X', E'\rangle$, such that $|E'| \geq k$ and $\mathcal{A}'$ meets the constraints of the Problem Statement 3.1?

**Theorem 4.1:** MNBRAP is NP-hard in the size of locations of the input timed automaton.

**Proof.** We reduce DVSP to MNBRAP. The reduction maps a weighted DAG $G\langle V, A\rangle$ and integers $d$ and $i$ to a timed automaton $\mathcal{A}$ and integers $\delta$ and $k$, respectively.

**Mapping.** Let $G\langle V, A\rangle$ be any instance of DVSP whose longest path is to be bounded by $d$. Let $l(a)$ be the length of an arc $a \in A$. We construct a timed automaton $\mathcal{A}$ as follows (cf. Figure 2). Each vertex $v \in V$ is mapped to a pair of locations $v^{in}$ and $v^{out}$ in $\mathcal{A}$. The set of initial locations of $\mathcal{A}$ is the singleton $L^0 = \{v_s^{in}\}$, where $v_s$ is the source vertex in $G$. Switches of $\mathcal{A}$ consist of two types of switches as follows:

- We include switches of the form $v^{in} \xrightarrow{(x=0)?} v^{out}$ for all $v$ in $V$. The clock constraint $(x = 0)$ is used to force computations of $\mathcal{A}$ not to wait at location $v^{in}$.

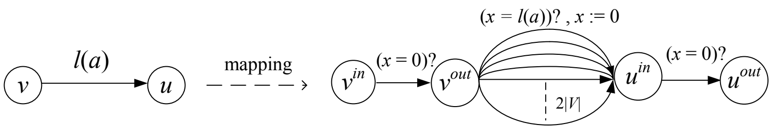


**Fig. 2.** Mapping DVSP to MNBRAP.

- We add $2|V|$ number of parallel switches of the form $v^{out} \xrightarrow{(x=l(a))?,\ x:=0} u^{in}$, for all arcs $a = (v, u) \in A$ of length $l(a)$.

Let the set of clock variables of $\mathcal{A}$ be the singleton $X = \{x\}$. Finally, let $v_s^{in} \models p$, $v_t^{out} \models q$, $k = |E| - i$, and $\delta = d$. Other locations may satisfy arbitrary atomic propositions except $p$ and $q$.

**Reduction.** We need to show that vertex $v \in Y$ in $G$ must be split if and only if the switch $v^{in} \xrightarrow{(x=0)?} v^{out}$ must be removed from $\mathcal{A}$. We distinguish two cases:

- *DVSP* $\longrightarrow$ *MNBRAP*: Suppose the answer to DVSP is the set $Y$, where $|Y| \leq i$. Hence, after splitting all $v \in Y$ the length of the longest path of $G$ is at most $d$. Now, we show that we can synthesize a timed automaton $\mathcal{A}'$ from the mapped timed automaton $\mathcal{A}\langle L, \{v_s^{in}\}, \psi, \{x\}, E\rangle$ as an answer to MNBRAP. It is easy to see that if we remove switches of the form $v^{in} \xrightarrow{(x=0)?} v^{out}$ (for all $v \in Y$) from $E$ to obtain $E'$, the maximum delay between locations $v_s^{in}$ and $v_t^{out}$ in $\mathcal{A}'$ becomes at most $\delta$. Recall that, $\delta = d$ and $k = |E| - i$. Therefore, $\mathcal{A}' \models \mathcal{L}_B$ and $|E'| \geq |E| - i = k$. Other constraints of the Problem Statement 3.1 are immediately met by construction of $\mathcal{A}'$.
- *MNBRAP* $\longrightarrow$ *DVSP*: Suppose the answer to MNBRAP is $\mathcal{A}'\langle L', L'^0, \psi', \{x\}, E'\rangle$, where $|E'| \geq k$ and the maximum delay to reach $v_t^{out}$ from $v_s^{in}$ is at most $\delta$. Note that, $L'^0 = \{v_s^{in}\}$. Since the number of switches removed from $E$ is at most $|E| - k$, $k = |E| - i$, and $i \leq |V|$, we could not have removed switches of the form $v^{out} \xrightarrow{(x=l(a))?,\ x:=0} u^{in}$. This is because there are $2|V|$ of such switches and, hence, their removal would not change the maximum delay. Thus, we should have removed switches of the form $v^{in} \xrightarrow{(x=0)?} v^{out}$ from $E$ to bound the maximum delay. Indeed, these switches identify the set $Y$ of vertices that should be split in $G$, i.e, $Y = \{v \mid (v \in V) \wedge ((v^{in}, v^{out}) \in (E - E'))\}$. It is easy to see that by removing the set $Y$ from $V$ the length of the longest path of $G$ becomes at most $d$. $\Box$

Although we defined maximality in terms of transitions of a timed automaton, one may define it in terms of reachable locations or behaviors of a timed automaton. However, various definitions do not change the NP-hardness result. In fact, many of the edge and vertex deletion problems are known to be NP-hard [22,23]. In particular, in case of maximal reachable locations, one can easily reduce the *vertex deletion problem* [22] to our synthesis decision problem. Moreover, in case of maximal number of behaviors, one can develop a reduction from the $k^{\text{th}}$ *shortest path problem* [24].

## 5 Adding Bounded Response Properties without Maximal Nondeterminism

In this section, we show that by relaxing the maximality constraint, we can solve the Problem Statement 3.1 in polynomial time in the size of locations of the input timed automaton. A possible approach to add a bounded response property to a timed automaton is as follows. First, we construct an auxiliary timed automaton $\mathcal{A}_2$ accepting all behaviors of the given bounded response property. Then, we construct the product of $\mathcal{A}_2$ and the given timed automaton $\mathcal{A}_1$ (denoted $\mathcal{A}_1 \otimes \mathcal{A}_2$). Although this approach is semantically correct, it does not meet the constrains of the Problem Statement 3.1. In particular, construction of the product alone may introduce deadlock states to $\mathcal{A}_1 \otimes \mathcal{A}_2$. As a result, some of the infinite computations of $\mathcal{A}_1$ become finite in $\mathcal{A}_1 \otimes \mathcal{A}_2$ and, hence, existing MTL properties are not preserved, which in turn violates the constraint $(C6)$ of the problem statement. Thus, we need a more "behavior-aware" approach.

Since our synthesis algorithm constructs and manipulates a specific weighted directed graph introduced by Courcoubetis and Yannakakis as a solution to the maximum delay problem in timed automata [25], we review this problem in Subsection 5.1. In Subsection 5.2, we describe our synthesis algorithm.

### 5.1 The Maximum Delay Problem in Timed Automata

The maximum delay problem is as follows. Given a timed automaton $\mathcal{A}$, a source location and clock valuation, what is the latest time that a target location can appear along a computation of $\mathcal{A}$? We first construct the region automaton $R(\mathcal{A})\langle S, T\rangle$, where $S$ is the set of regions and $T$ is the set of edges. Then, we transform the region automaton to an ordinary weighted directed graph (called MaxDelay digraph). Let the subroutine ConstructMaxDelayGraph do this transformation as follows. It takes a region automaton $R(\mathcal{A})\langle S, T\rangle$, a set $X$ of source regions, and a set $Y$ of target regions, where $X, Y \subseteq S$, as input, and constructs a MaxDelay digraph $G\langle V, A\rangle$. Vertices of $G$ consist of the regions in $R(\mathcal{A})$ with the addition of a source vertex $v_s$ and a target vertex $v_t$.
*Notation*: We denote the weight of an arc $(v_0, v_1)$ by $Weight(v_0, v_1)$. Let $f$ denote a function that maps each region in $R(\mathcal{A})$ to its corresponding vertex in $G$, i.e., $f(r)$ is a vertex that represents region $r$ in $G$. Also, let $f^{-1}$ denote the inverse of $f$, i.e., $f^{-1}(v)$ is the region of $R(\mathcal{A})$ that corresponds to vertex $v$ in $G$. Likewise, let $F$ be a function that maps a set of regions in $R(\mathcal{A})$ to the corresponding set of vertices in $G$ and $F^{-1}$ be its inverse. Finally, for a boundary region $r$ with respect to clock variable $x$, we denote the value of $x$ by $r.x$ (equal to some constant in $\mathbb{Z}_{\geq 0}$).

Arcs of $G$ consist of the following:

- Arcs of weight 0 from $v_s$ to all vertices in $F(X)$, and from all vertices in $F(Y)$ to $v_t$.
- Arcs of weight 0 from $v_0$ to $v_1$, if $f^{-1}(v_0) \to f^{-1}(v_1)$ is a location switch in $R(\mathcal{A})$.
- Arcs of weight $c' - c$, where $c, c' \in \mathbb{Z}_{\geq 0}$ and $c' > c$, from $v_0$ to $v_1$, if $f^{-1}(v_0)$ and $f^{-1}(v_1)$ are both boundary regions with respect to clock variable $x_i$, such that $f^{-1}(v_0).x_i = c$, $f^{-1}(v_1).x_i = c'$, and there exists a path in $R(\mathcal{A})$ from $f^{-1}(v_0)$ to $f^{-1}(v_1)$, which does not reset $x_i$.

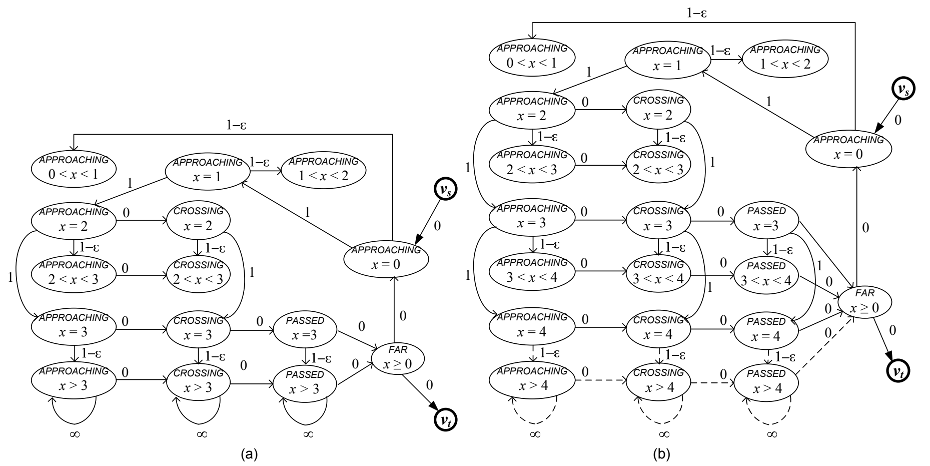


**Fig. 3.** (a) MaxDelay digraph of TRAIN automaton. (b) MaxDelay digraph with respect to $\delta = 4$

- Arcs of weight $c' - c - \epsilon$, where $c, c' \in \mathbb{Z}_{\geq 0}$, $c' > c$, and $0 < \epsilon \ll 1$, from $v_0$ to $v_1$ , if (1) $f^{-1}(v_0)$ is a boundary region with respect to clock variable $x_i$, (2) $f^{-1}(v_1)$ is an open region whose time-successor $f^{-1}(v_2)$ is a boundary region with respect to clock variable $x_i$, (3) $f^{-1}(v_0) \rightarrow f^{-1}(v_1)$ represents a delay transition in $R(\mathcal{A})$, and (4) $f^{-1}(v_0).x_i = c$ and $f^{-1}(v_2).x_i = c'$.
- Self-loop arcs of weight $\infty$ at vertex $v$, if $f^{-1}(v)$ is an end region.

In order to compute the maximum delay between $X$ and $Y$, it suffices to find the longest distance between $v_s$ and $v_t$ in $G$. As an example, Figure 3-a shows the MaxDelay digraph of the TRAIN automaton.

### 5.2 The Synthesis Algorithm

In this subsection, we present a sound and complete algorithm, Add_BoundedResponse (cf. Figure 4), for solving the Problem Statement 3.1 with respect to $\mathcal{L}_B \equiv \Box(p \rightarrow \Diamond_{\leq \delta} q)$. The core of the algorithm is straightforward. It begins with an empty digraph and builds up a subgraph of the MaxDelay digraph by adding paths of length at most $\delta$ that start from the set of vertices that represents $p$-regions in $G$ to the set of vertices that represents $q$-regions. Then, it adds the rest of vertices and arcs while ensuring that no new paths from $p$-regions to $q$-regions are introduced. In order to ensure completeness, the algorithm preserves $p$-regions.

We now describe the algorithm in detail. First, in order to keep track of time elapsed since $p$ have become true, we add an extra clock variable $t$ to $\mathcal{A}$ as a timer. Moreover, the maximum value that $t$ would be compared with is $\delta$ (lines 1-2). Next, we construct the region automaton $R(\mathcal{A})\langle S, T\rangle$, where $S$ is the set of regions and $T$ is the set of edges (Line 3). Let the function $g : AP \rightarrow 2^S$ calculate the set of regions with respect to an arbitrary atomic proposition $ap$ as follows:

$$g(ap) = \{(s_1, \rho_1) \mid (s_1 \models ap) \ \wedge \\ (\exists \, (s_0, \rho_0) \mid (((s_0, \rho_0), (s_1, \rho_1)) \in T) \ : \ (s_0 \not\models ap))\}$$

Add_BoundedResponse($\mathcal{A}\langle L, L^0, \psi, X, E\rangle$ : timed automata, $\mathcal{L}_B \equiv \Box(p \rightarrow \Diamond_{\leq\delta} q)$)
{
(1) $X = X \cup \{t\}; c_t := \delta;$
(2) $\forall\langle s_0, \lambda, \varphi, s_1\rangle \mid (\langle s_0, \lambda, \varphi, s_1\rangle \in E \ \wedge\ (s_0 \not\models p \wedge s_1 \models p))\ :\ \lambda := \lambda \cup \{t\};$
(3) $R(\mathcal{A})\langle S, T\rangle :=$ ConstructRegionAutomaton($\mathcal{A}$);
**Repeat**
(4) IsQRemoved := *false*;
(5) $G\langle V, A\rangle :=$ ConstructMaxDelayGraph($R(\mathcal{A}), g(p), g(q)$); \\ Defined in Subsection 5.1
(6) $G'\langle V', A'\rangle :=$ ConstructSubgraph($G, \delta$);

(7) $R(\mathcal{A}')\langle S', T'\rangle := \{\};$
(8) $S' := F^{-1}(V');$
$T' := \{(r_0, r_1) \mid (r_0, r_1) \in T \ \wedge\ (f(r_0), f(r_1)) \in A'\} \cup$
$\{(r_1, r_2) \mid (r_1, r_2) \in T \ \wedge\ (f(r_1), f(r_2)) \notin A' \ \wedge$
(9) $\exists r_0 : Weight(f(r_0), f(r_1)) = 1 - \epsilon\};$
(10) **while** $(\exists r_0 \mid r_0 \in S'\ :\ (\forall r_1 \mid r_1 \in S'\ :\ (r_0, r_1) \notin T'))$
(11) $S' := S' - \{r_0\}; T' := T' - \{(r, r_0), (r_0, r) \mid r \in S'\};$
(12) **if** $r_0 \in g(q)$ **then**
(13) IsQRemoved $:= true;$
(14) $S := S - \{r_0\}; T := T - \{(r, r_0), (r_0, r) \mid r \in S\};$ **break**;
**until** (IsQRemoved = *false*);
**if** $\{(s, \rho) \mid (s, \rho) \in S' \wedge s \in L^0 \wedge (\forall x, \nu \mid (\nu \in \rho \wedge x \in X) : \nu(x) = 0)\} = \{\}$ **then**
(15) declare failure; **exit**;
(16) $\mathcal{A}'$ := ConstructTimedAutomata($R(\mathcal{A}')$);
(17) **return** $\mathcal{A}'$;
}
ConstructSubgraph($G\langle V, A\rangle$ : MaxDelay digraph, $\delta$ : **integer**)
{
(18) $G'\langle V', A'\rangle = \{\};$
(19) for all vertices $v$ such that $(v_s, v) \in A$
(20) **if** the length the shortest path $\mathcal{P}$ from $v$ to $v_t$ is at most $\delta$ **then**
(21) $V' := V' \cup \{u \mid u$ is on $\mathcal{P}\};$
(22) $A' := A' \cup \{a \mid a$ is on $\mathcal{P}\};$
(23) $A' := A' \cup \{(u, v) \mid (u, v) \in A \ \wedge\ (u \notin V' \ \vee\ (u, v_t) \in A')\};$
(24) $V' := (V' \cup \{u \mid (\exists v : (u, v) \in A' \ \vee\ (v, u) \in A')\}) - \{v_s, v_t\};$
(25) **return** $G'\langle V', A'\rangle$;
}

**Fig. 4.** The synthesis algorithm for adding bounded response.

We now reduce our problem to the problem of bounding the length of longest path in ordinary weighted digraphs. Towards this end, we first generate the MaxDelay digraph $G\langle V, A\rangle$ (Line 5), as described in Subsection 5.1. Then, we invoke (Line 6) the subroutine ConstructSubgraph (lines 18-25) which takes a MaxDelay digraph $G$ and an integer $\delta$ as input. It generates a subgraph $G'$ whose longest path from $v_s$ to $v_t$ is bounded by $\delta$. Recall that $v_s$ and $v_t$ are additional source and target vertices connected to $F(g(p))$ and $F(g(q))$, respectively. We now begin with an empty digraph and add a certain number of paths in polynomial order of $|S|$. To this end, first, we include the shortest path from each vertex in $F(g(p))$ to $v_t$, provided its length is at most $\delta$ (lines 19-22). Then, we add the rest of the vertices and arcs to $G'$ (lines 23-24) while ensuring that no new paths are added from $v_s$ to $v_t$.

After invoking ConstructSubgraph, we transform $G'$ back to a region automaton $R(\mathcal{A}')$ (lines 7-9). Next, due to pruning some vertices and arcs in ConstructSubgraph, we remove deadlock regions from $R(\mathcal{A}')$ (lines 10-11). However, in order to ensure that this removal does not break the completeness of our algorithm, we should consider the case where a $q$-region becomes a deadlock region (lines 12-14). In case the removal of deadlock regions leaves no initial regions, the algorithm declares failure and terminates (Lines 15). Otherwise, it constructs the timed automaton $\mathcal{A}'$ out of $R(\mathcal{A}')$ and terminates successfully (lines 16-17).

**Level of nondeterminism.** In order to increase the level of nondeterminism, we can include additional paths whose length is at most $\delta$. However, every time we add a path, we need to test that this path does not create new paths of length greater than $\delta$ or cycles containing an edge of nonzero weight. To this end, we can use one of the algorithms in the literature of graph theory (e.g., [26]) to find and add $k$ shortest paths in an ordinary weighted digraph.

Let us now consider the TRAIN automaton presented in Section 2 (cf. Figure 1-a). Our goal is to bound the delay of revisiting the initial location within at most 4 minutes. To this end, we add the property $\mathcal{L}_B \equiv \Box(APPROACHING \rightarrow \Diamond_{\leq 4} FAR)$ to the TRAIN automaton. Since $\delta = 4$, we have $c_x = 4$ when generating the region automaton. Next, we construct the MaxDelay digraph (cf. Figure 3-b). In Figure 3-b, the dotted arcs contribute in violating $\mathcal{L}_B$, but the solid arcs do not. It is easy to observe by adding 12 shortest paths, we includes all computations that satisfy $\mathcal{L}_B$. Figure 5-a shows the synthesized region automaton and Figure 5-b shows the the final timed automaton.

**Theorem 5.1:** The algorithm Add_BoundedResponse is sound and complete.

**Proof.** We show that the timed automaton synthesized by Add_BoundedResponse meets the constraints of Problem Statement 3.1 with respect to $\mathcal{L}_B$:

- *Constraint* $C1$: It is easy to observe that the algorithm Add_BoundedResponse only removes states of $\mathcal{A}$. Hence, $L' \subseteq L$ and $L'^0 \subseteq L^0$. Note that, pruning regions only changes the guards of the associated switches and it does not affect reconstruction of $\mathcal{A}'$ such that $L' \subseteq L$.
- *Constraint* $C2$: We only add an extra clock variable $t$. Hence, $X \subseteq X'$. Note that, since the length of a path in MaxDelay digraph is equal to the time elapsed along regions, our algorithm works correctly even if $t$ is reset in between a $p$-state and a $q$-state (e.g., a computation that goes from a $p$-state to a $(\neg p)$-state, then again to a $p$-state, and finally to a $q$-state).

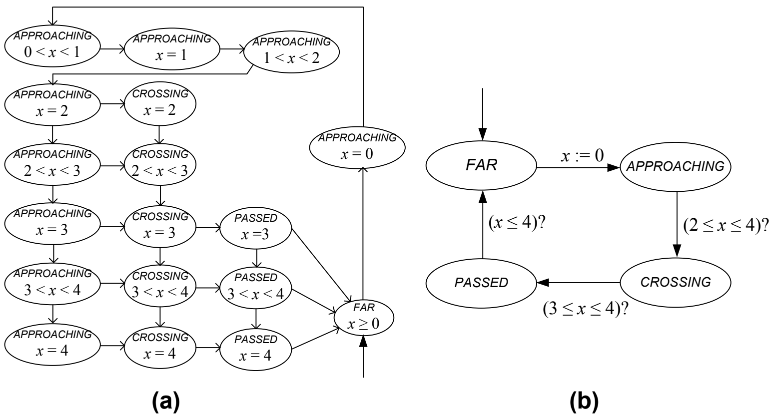


**Fig. 5.** (a) Synthesized region automaton (b) Synthesized TRAIN automaton.

- *Constraint* $C3$: The algorithm does not touch the labels of locations and, hence, $\psi' = \psi$.
- *Constraint* $C4$: The subroutine ConstructSubgraph may only remove regions or edges from a region automaton. This removal either removes a switch from the original timed automaton completely or makes some regions unreachable, which in turn strengthens the guard of one or more switches. Hence, the set of switches of $\mathcal{A}'$ meets the constraint $C4$.
- *Constraint* $C5$: The subroutine ConstructSubgraph ensures that the maximum delay of any computation that starts from a region in $g(p)$ and reaches a region in $g(q)$ is finite and bounded by the required response time in $\mathcal{L}_B$. Hence, we are assured that the synthesized timed automaton satisfies $\mathcal{L}_B$.
- *Constraint* $C6$: First, since the algorithm removes deadlock regions from $R(\mathcal{A}')$ all computations of $\mathcal{A}'$ are infinite. Moreover, from constraints $C1...C4$, it follows that the algorithm does not introduce new computations to $\mathcal{A}'$. Thus, the set of computations of $\mathcal{A}'$ is a subset of the set of computations of $\mathcal{A}$. Furthermore, as mentioned in Section 2, an MTL formula $\Sigma$ defines a set of timed state sequences. Note that, an automaton $\mathcal{A}$ satisfies specification $\Sigma$ iff all computations of $\mathcal{A}$ are in $\Sigma$. Hence, a subset of computations of $\mathcal{A}$ satisfies $\Sigma$ as well. In the context of the algorithm Add_BoundedResponse, although it excludes some of the computations, since it ensures that all computations are infinite (by removing deadlock regions), it continues to satisfy its old MTL specification. A possible confusion is that "the given automaton (before synthesis) does not satisfy the bounded response property $\mathcal{L}_B$, but it does satisfy $\mathcal{L}_B$ after synthesis". Note, however, that "an automaton does not satisfy $\mathcal{L}_B$" cannot be expressed as "the automaton satisfies $\mathcal{L}'$", where $\mathcal{L}'$ is an MTL property. Also, if a given automaton satisfies $\neg\mathcal{L}_B$ then no computation of the automaton satisfies $\mathcal{L}_B$ and, hence, it is not possible to synthesize an automaton that satisfies $\mathcal{L}_B$. In such a case, the algorithm Add_BoundedResponse declares failure. Hence, for all MTL specifications $\Sigma$, if $\mathcal{A} \models \Sigma$ then $\mathcal{A}' \models \Sigma$ as well.
  Note, however, that the same problem cannot be defined by branching-time temporal logics (e.g., TCTL), as "an automaton does not satisfy $\mathcal{L}$" can be expressed as "the automaton satisfies $\mathcal{L}'$", where $\mathcal{L}'$ is a TCTL property.

This completes the proof of soundness. In order to prove the completeness, we show that any initial location removed from the synthesized automaton must be removed. Observe that if there exists a vertex $v \in F(g(p))$ from where there does not exist a path to $v_t$ where the delay is at most $\delta$, $f^{-1}(v)$ becomes a deadlock region and it should be removed. It follows that such regions must be removed in any timed automaton that satisfies the constraints of Problem Statement 3.1. Furthermore, if a $q$-region $r_0$ becomes a deadlock region, it is possible that all the regions along a path that starts from a region in $g(p)$ and ends at $r_0$ become deadlock regions. Thus, we need to find another path from that region in $g(p)$ to a region in $g(q)$ other than $r_0$. Hence, we remove $r_0$ from the set of regions of the original region automaton $R(\mathcal{A})$ and start over. Furthermore, if removal of a region causes another region to become a deadlock region then that region must be removed for satisfying the constraint $C5$. Continuing thus, if an initial region becomes a deadlocked region then it *must* be removed. Our algorithm declares failure when all initial locations are removed. Based on the above discussion, in this case, any

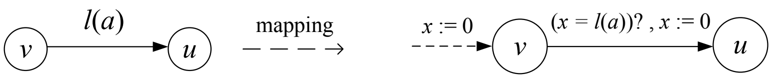


**Fig. 6.** Mapping the longest path problem to addition of interval-bounded response.

timed automaton that satisfies the constraints of Problem Statement 3.1 cannot contain any of the initial locations from $L^0$. Since this is a contradiction, it follows that when Add_BoundedResponse declares failure, no solution exists for the given instance. Therefore, Add_BoundedResponse is complete. $\square$

**Theorem 5.2:** The problem of adding a bounded response property to a timed automaton is in PSPACE.

**Proof.** The core of the algorithm is reachability analysis for a timed automaton. Deciding reachability of a location in timed automata is in $P$ in the size of the region automaton [25]. Moreover, our synthesis algorithm involves finding shortest paths and (possibly) $k$ shortest paths in an ordinary weighted digraph. Eppstein [26] proposes an algorithm that finds the $k$ shortest paths (allowing cycles) in time $O(m + n \log n + k)$, where $n$ is the number of vertices and $m$ is the number of arcs of a given digraph. Note that, we require that $k$ must be in polynomial order of the number of locations of the input timed automaton. Hence, one can implement a synthesis algorithm which runs in polynomial time in the qualitative part (locations), and polynomial space in the quantitative part (timing constraints) of the input automaton. $\square$

## 6 Adding Interval-Bounded and Unbounded Response Properties

We first consider automatic addition of an interval-bounded response property $\mathcal{L}_I \equiv \Box(p \rightarrow \Diamond_{[\delta_1,\delta_2]} q)$ to a timed automaton, where $\delta_1 > 0$. As an intuition, let us use the algorithm Add_BoundedResponse to add $\mathcal{L}_I$. Since the required response time has a lower bound, the subroutine ConstructSubgraph has to enumerate and ignore all the paths whose lengths are less than $\delta_1$. Obviously, this enumeration cannot be done in polynomial time in the size of region automata.

**Theorem 6.1:** The problem of adding an interval-bounded response property to a timed automaton is NP-hard in the size of the locations of the input timed automaton.

**Proof.** The proof is a simple reduction from the *longest path problem* to an instance of the problem, where $\mathcal{L}_I \equiv \Box(p \rightarrow \Diamond_{[\delta_1,\infty)} q)$. Figure 6 illustrates the mapping of a digraph $G$ to a timed automaton $\mathcal{A}$. It is easy to see that if $G$ has a path of length at least $\delta_1$ from a source vertex $v_s$ to a target vertex $v_t$ then $\mathcal{A}$ can be transformed to a timed automaton $\mathcal{A}'$ whose delay from $v_s$ to $v_t$ is at least $\delta_1$ time units and vice versa. $\square$

Next, we discuss the problem of addition of unbounded response (also called leads-to) properties.

**Theorem 6.2:** The problem of addition of an unbounded response property to a timed automaton is PSPACE-complete in the size of the input timed automaton.

**Proof.** Since this problem is an instance of adding bounded response, membership to PSPACE follows from Theorem 5.2 immediately. We now show that the problem is PSPACE-hard. To this end, we reduce the *reachability problem* in timed automata [25] (whether a location $s_1$ is reachable from another location $s_0$) to an instance of our problem.

**Mapping.** Let the timed automaton $\mathcal{A}$ be any instance of the reachability problem. We map $\mathcal{A}$ to an instance of our problem as follows. Let $\mathcal{A}^*$ be an automaton identical to $\mathcal{A}$ with the following modifications. Let $s_0 \models p$ and $s_1 \models q$. Other locations of $\mathcal{A}^*$ may satisfy arbitrary atomic propositions except $p$ and $q$. Let $s_0$ be the only initial location of $\mathcal{A}^*$. We also add a self-loop at $s_1$.
**Reduction.** If $s_1$ is reachable from $s_0$ in $\mathcal{A}$ then there exists a computation in $\mathcal{A}^*$ that starts from $s_0$ and ends at $s_1$. A timed automaton $\mathcal{A}'$ constructed from this computation plus the self-loop at $s_1$ satisfies $\mathcal{L}_\infty$ and meets the constraints of Problem Statements 3.1. Now, we show the other direction. Let us assume that the answer to the decision problem is affirmative and we can synthesize a timed automaton $\mathcal{A}'$ from $\mathcal{A}^*$ such that $\mathcal{A}' \models \mathcal{L}_\infty$. Then $\mathcal{A}'$ should contain both $s_0$ and $s_1$. This means that $s_1$ is reachable from $s_0$. Otherwise, $\mathcal{A}'$ would not satisfy $\mathcal{L}_\infty$. □

Since an unbounded response property is an instance of bounded and interval-bounded response properties, problems of adding those properties are also PSPACE-hard.

**Corollary 6.3:** The problem of adding a bounded response property to a timed automaton is PSPACE-complete in the size of the input timed automaton. □

**Remark 6.4.** The time complexity of adding an unbounded response property to a timed automaton with maximal nondeterminism in terms of transitions remains open in this paper. However, we refer the reader to [8], where the authors introduce a synthesis algorithm for adding leads-to properties to an untimed program, while maintaining maximal nondeterminism in terms of reachable states of the given program.

We summarize the complexity of problems of addition of different types of response properties in Table 2.

| Bounded Response | | Unbounded Response | | Interval-Bounded Response |
|---|---|---|---|---|
| **Maximal** (Sec. 4) | **NonMaximal** (Sec. 5) | **Maximal** (Sec. 6) | **NonMaximal** (Sec. 6) | (Sec. 6) |
| *NP-hard* | *P* | see Rem. 6.4 | *P* | *NP-hard* |

**Table 2.** Complexity of adding response properties in the size of the locations.

## 7 Conclusion and Future Work

In this paper, we focused on automated incremental synthesis of timed automata by adding various types of bounded response properties, while preserving its existing Metric Temporal Logic (MTL) specification. Unlike specification-based methods, in our approach, we start with an existing program rather than specification and, hence, the previous efforts made for synthesizing the input program are reused.

First, we showed the problem of addition of a bounded response property to a timed automaton while maintaining maximal nondeterminism is NP-hard in the size of locations of the input automaton. Then, we presented a simple sound and complete transformation algorithm that adds a bounded response property to a timed automaton (without maximal nondeterminism), such that the automaton continues to satisfy its existing

MTL specification. The complexity of the algorithm is polynomial in the size of region automata. Furthermore, we showed that the problem of addition of interval-bounded response properties is also NP-hard. Moreover, we showed that adding bounded and unbounded response properties are PSPACE-complete in the size of the input timed automaton.

Detailed region automata are not an efficient finite representation of timed automata in terms of space complexity. On other hand, zone automata [27] are more efficient finite representation of timed automata used in model checking techniques. Since our goal was to evaluate complexity classes for adding bounded response, we focused on region automata. However, an interesting improvement step is modifying Add_BoundedResponse, so that it manipulates a zone automaton rather than a detailed region automaton.

In many hard real-time systems (e.g., mission-critical systems) meeting deadlines in the presence of faults is a necessity. As future work, we plan to study the problem of automatic addition of fault-tolerance to existing fault-intolerant real-time systems. More specifically, we plan to extend the theory of automated addition of fault-tolerance to untimed programs [5–7] to the context of real-time programs. In particular, we plan to study how time-bounded recovery can be achieved in the presence of faults using the results presented in this paper.

**Acknowledgment.** The authors would like to thank Edith Elkind at Princeton University for her ideas on the NP-hardness result in Section 4.

## References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. R. Alur and T.A. Henzinger. Real-Time Logics: Complexity and Expressiveness. *Information and Computation*, 10(1):35–77, May 1993.
3. E.A. Emerson and E.M. Clarke. Using branching time temporal logic to synthesis synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.
4. Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, 1984.
5. S. S. Kulkarni and A. Arora. Automating the addition of fault-tolerance. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 82–93, 2000.
6. S. S. Kulkarni, A. Arora, and A. Chippada. Polynomial time synthesis of Byzantine agreement. In *20th Symposium on Reliable Distributed Systems*, pages 130–140, 2001.
7. S. S. Kulkarni and A. Ebnenasir. Automated synthesis of multitolerance. In *International Conference on Dependable Systems and Networks*, pages 209–219, 2004.
8. A. Ebnenasir, S. S. Kulkarni, and B. Bonakdarpour. Revising UNITY programs: Possibilities and limitations. In *9th International Conference on Principles of Distributed Systems*, 2005. To appear.
9. K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.
10. W. M. Wonham P. J. G. Ramadge. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.
11. H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. In *30th Conf. Decision and Control*, pages 1527–1528, Brighton, UK, 1991.

12. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. *12th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 229–242, 1995.
13. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. *IFAC Symposium on System Structure and Control*, pages 469–474, 1998.
14. E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, pages 19–30, 1999.
15. T. A. Henzinger and P. W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221(1-2):369–392, 1999.
16. D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *19th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 571–582, 2002.
17. P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In *Computer Aided Verification (CAV)*, pages 180–192, 2003.
18. M. Faella, S. LaTorre, and A. Murano. Dense real-time games. In *Logic in Computer Science (LICS)*, pages 167–176, 2002.
19. L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *14th International Conference on Concurrency Theory (CONCUR)*, 2003.
20. R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
21. N. G. Leveson and J. L. Stolzy. Analyzing safety and fault tolerance using time petri nets. In *International Joint Conference on Theory and Practice of Software Development (TAPSOFT) on Formal Methods and Software*, pages 339–355, 1985.
22. D. Paik, S.M. Reddy, and S. Sahni. Deleting vertices to bound path length. *IEEE Transation on Computers*, 43(9):1091–1096, 1994.
23. M. Yannakakis. Node- and edge-deletion NP-complete problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing*, pages 253–264. ACM press, 1978.
24. M.R. Garey and D.S. Johnson. *Computers and Intractability:A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.
25. C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Computer-Aided Verificaion*, LNCS 575:399–409, 1991.
26. D. Eppstein. Finding the k shortest paths. *SIAM Journal of Computing*, 28(2):652–673, 1999.
27. R. Alur, C. Courcoubetis, N. Halbwachs, D. L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *International Conference on Concurrency Theory (CONCUR)*, pages 340–354, 1992.