The Complexity of Bounded Synthesis for Timed Control with Partial Observability

Hans-Jörg Peter[1,2] and Bernd Finkbeiner[2]

[1] Advanced Research Center
Atrenta Inc.
38000 Grenoble, France
[2] Department of Computer Science
Saarland University
66123 Saarbrücken, Germany

Abstract. We revisit the synthesis of timed controllers with partial observability. Bouyer et al. showed that timed control with partial observability is undecidable in general, but can be made decidable by fixing the granularity of the controller, resulting in a 2ExpTime-complete problem. We refine this result by providing a detailed complexity analysis of the impact of imposing a bound on the size of the controller, measured in the number of locations. Our results identify which types of bounds are useful (and which are useless) from an algorithmic perspective. While bounding the number of locations without fixing a granularity leaves the problem undecidable, bounding the number of locations *and* the granularity reduces the complexity to NExpTime-complete. If the controller is restricted to be a discrete automaton, the synthesis problem becomes PSpace-complete, and, for a fixed granularity of the plant, even NPTime-complete. In addition to the complexity analysis, we also present an effective synthesis algorithm for location-bounded discrete controllers, based on a symbolic fixed point computation. Synthesis of bounded controllers is useful even if the bound is not known in advance. By iteratively increasing the bound, the synthesis algorithm finds the smallest, and therefore often most useful, solutions first.

1 Introduction

The theory of timed automata has made it possible to extend the algorithms for automatic verification and controller synthesis from discrete systems to real-time systems. An open challenge is, however, to effectively synthesize real-time controllers under partial observability, i.e., in situations where there are some events in the plant that the controller cannot observe.

Since the synthesis problem of real-time controllers under partial observability is in general undecidable [4], synthesis algorithms must focus on restricted classes of controllers. Bouyer et al. studied, for example, the synthesis problem with *fixed granularity*, where the number of clocks and the precision of the guards is limited in advance [11,4]. While this restriction ensures decidability, it unfortunately

Bound \ Granularity	Unspecified	Fixed	Discrete
Unbounded	Undecidable	2ExpTime-complete	**2ExpTime-complete**
Locations	**Undecidable**	**NExpTime-complete**	**PSpace-complete / NPTime-complete**
Clocks	Undecidable	2ExpTime-complete	—

Table 1. Overview on the complexities of bounded synthesis for timed controllers with partial observability. The results written in **bold face** are established in this paper, the other results are taken from [4].

does not suffice to obtain an effective algorithm, because the synthesis problem remains intractably expensive (2ExpTime-complete). Finding restrictions on the timed controllers that lead to a significant reduction in complexity thus remained an open question.

In this paper, we undertake a systematic study of the impact of various restrictions on the complexity of the controller synthesis problem. We introduce a bound on the *size* of the controller and limit the search to only those controllers that fall below the bound. In the setting of discrete systems, this idea is known as *bounded synthesis* [30]. For plants given as timed automata, natural adaptations of the bounded synthesis approach are to search for a controller with a bounded number of locations.

We analyze the complexity of the bounded synthesis problem under different types of bounds, and under different restrictions on the granularity. The results are summarized in Table 1. Some restrictions do not help: bounding the number of locations without fixing a granularity leaves the problem undecidable. Fixing both the granularity and a bound on the number of locations, however, reduces the complexity from 2ExpTime-complete to NExpTime-complete. Most interesting is the restriction to discrete controllers, where all clocks are located in the plant and the untimed controller only reacts to discrete events. Here, the complexity reduces to PSpace, i.e., the problem is exactly as hard as standard model checking. If the granularity of the plant is fixed, the complexity reduces further to NPTime.

Related work. In his seminal work on discrete two-player games [27], Reif introduced the *knowledge-based subset construction* to transform a game with imperfect information to a game with perfect information. The construction causes an exponential blow-up. The (fully observable) timed controller synthesis problem in the framework of timed automata [1] was defined by Maler et al. by introducing two-player timed games [23,2]. The decidability of the problem was shown by demonstrating that the discrete attractor construction [31] can be adapted to a zone-based algorithm to obtain timed controllers. Henzinger and Kopke showed that the discrete attractor construction on the region graph is theoretically optimal by proving that the synthesis problem for safety properties is ExpTime-complete [16]. Controller synthesis against external specifications given

as nondeterministic timed automata was considered by D'Souza and Madhusudan [11]. They were the first who discovered that fixing the granularity of the controller leads to decidability. Bouyer et al. extended this work by introducing partial observability for the controller [4]. A more pragmatic approach was investigated by Cassez et al. by restricting the choices and the observability of the controller so that the implementation of zone-based synthesis algorithms becomes possible [7]. An extension of this work uses *alternating timed simulation relations* to efficiently control partially observable systems [9]. An alternative restriction is to only consider controllers that match a given template. We recently obtained promising experimental results with an implementation that searches for such controllers using automatic abstraction refinement [14].

The idea of a-priori fixing syntactic properties of the system that should be synthesized resembles *bounded synthesis* [30] from the (fully observable, pure discrete) LTL synthesis community. Symbolic implementations based on SMT-solving [15], antichains [13], or BDDs [12] followed. In these works, one fixes the maximal number of states that the synthesized system may have. Recently, Kupferman et al. continued this line of research by distinguishing between bounding the system and/or the environment [19]. Following a similar idea, Lustig et al. proposed synthesizing systems based on component libraries [22].

Laroussinie et al. investigated the impact of bounding the number of clocks for model checking timed automata [20]. Chen and Lu extended this work to the fully observable synthesis setting by bounding the number of clocks in the plant [10], which is in contrast to this paper, where we impose bounds on the controller. For STRIPS planning, Rintanen [28] investigated the impact of no and partial observability on finding discrete plans.

Contributions of the paper

- We provide the theoretical foundation for an extension of the bounded synthesis approach to the setting of real-time control. Our results identify which types of bounds are useful (and which are useless) from an algorithmic perspective.
- We provide matching lower and upper bounds for the complexity of the various synthesis problems, extending the complexity analysis of Bouyer et al. [4] to a complete picture of the controller synthesis problem for timed systems under partial observability. The proofs require nontrivial extensions of the techniques used in the literature that may also be of interest in other settings. For example, the proof of the NExpTime lower bound of Theorem 6 is based on an insightful connection between timed automata and the theory of problems on succinctly specified graphs.
- We demonstrate that bounded synthesis can be implemented in the setting of standard fixpoint-based verification tools for real-time systems. For this purpose, we present a construction that computes the set of discrete location-bounded controllers symbolically as a least fixed point.

Outline. We first recall the foundations of timed automata and timed controller synthesis with partial observability in Sections 2 and 3, respectively. In Section 4,

we investigate the impact of bounding the locations of the controller. Finally, Section 5 introduces discrete controllers and investigates the impact of bounded and unbounded synthesis in this setting. For each lemma and theorem newly established in this paper, we give a brief description of the proof idea in the main part of the paper. The detailed versions of the proofs as well as their underlying technical constructions can be found in the appendix.

2 Timed Automata

In this section, we recall the timed automaton model by Alur and Dill.

Definition. A *timed automaton* [1] is a tuple $A = (L, l_0, \Sigma, \Delta, X)$, where L is a finite set of (control) locations, $l_0 \in L$ is the initial location, Σ is a finite set of actions, $\Delta \subseteq L \times \Sigma \times \mathcal{C}(X) \times 2^X \times L$ is an edge relation, X is a finite set of real valued clocks, and $\mathcal{C}(X)$ is the set of clock constraints over X. A clock constraint $\varphi \in \mathcal{C}(X)$ is of the form

$$\varphi \equiv \mathbf{true} \mid x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi_1,$$

where x is a clock in X, φ_1 and φ_2 are clock constraints from $\mathcal{C}(X)$, and c is a constant in $\mathbb{Q}_{\geq 0}$ encoded in binary. A *clock valuation* $\boldsymbol{t} : X \to \mathbb{R}_{\geq 0}$ assigns a nonnegative value to each clock and can also be represented by a $|X|$-dimensional vector $\boldsymbol{t} \in \mathcal{R}$, where $\mathcal{R} = \mathbb{R}_{\geq 0}^X$ denotes the set of all clock valuations. We write $l \xrightarrow{a,\varphi,\lambda} l'$ to refer to a tuple $(l, a, \varphi, \lambda, l')$ in Δ. We say that A is *deterministic* if, for any two distinct edges $l \xrightarrow{a,\varphi,\lambda} l'$ and $l \xrightarrow{a',\varphi',\lambda'} l''$, it holds that $a = a' \Rightarrow \varphi \wedge \varphi' \equiv \mathbf{false}$. Following the setting of [2], we assume that timed automata are strongly nonzeno, i.e., there are no cycles where an infinite time-convergent sequence of transitions is possible.

The *(timed) states* of a timed automaton are pairs $(l, \boldsymbol{t})$ of locations and clock valuations. Timed automata have two types of transitions: *timed transitions*, where only time passes and the location remains unchanged, and *discrete transitions*, where no time passes, the current location may change and some clocks can be reset to zero. In a timed transition, denoted by $(l, \boldsymbol{t}) \xrightarrow{a} (l, \boldsymbol{t} + a \cdot \mathbf{1})$, the same nonnegative value $a \in \mathbb{R}_{\geq 0}$ is added to all clocks. A discrete transition, denoted by $(l, \boldsymbol{t}) \xrightarrow{a} (l', \boldsymbol{t}')$ for some $a \in \Sigma$, corresponds to an edge $(l, a, \varphi, \lambda, l')$ of Δ such that $\boldsymbol{t}$ satisfies the clock constraint φ, written as $\boldsymbol{t} \models \varphi$, and $\boldsymbol{t}' = \boldsymbol{t}[\lambda := 0]$ is obtained from $\boldsymbol{t}$ by setting the clocks in λ to 0.

We say that a state s is *forward reachable* if there is an $n \in \mathbb{N}$ and a finite sequence of transitions of the form $s_0 \xrightarrow{a_1} s_1 \ldots s_{n-1} \xrightarrow{a_n} s_n$ such that $s_0 = (l_0, \mathbf{0})$ is the initial state (where $\mathbf{0}$ is the zero vector), $s_n = s$, and for all $1 \leq i \leq n$, $s_i = (l_i, \boldsymbol{t}_i)$ are states and $s_{i-1} \xrightarrow{a_i} s_i$ are transitions of the automaton, respectively. We say that the sequence $a_1 a_2 \ldots a_n \in (\Sigma \cup \mathbb{R}_{\geq 0})^*$ is a *timed prefix* of A and we define $L(A)$ as the set of all timed prefixes leading to states that are forward reachable.

Granularity. The *granularity* of a timed automaton defines its timing resources [11]. Formally, a granularity is represented by a tuple $\mu = (Y, m, c_{max})$, where Y is a finite set of clocks, $m \in \mathbb{N}_{\geq 1}$, and $c_{max} \in \mathbb{Q}_{\geq 0}$. We say that a timed automaton $A = (L, l_0, \Sigma, \Delta, X)$ is μ-granular if $X = Y$ and, for each constant $c \in \mathbb{Q}_{\geq 0}$ appearing in the clock constraints of the guards of the edges in Δ, it holds that c is an integer multiple of $\frac{1}{m}$ and $c \leq c_{max}$. We call the value of a clock $x \in X$ *maximal* if it is strictly greater than c_{max}.

Composition. Timed automata can be syntactically composed into networks, in which the automata run in parallel and synchronize on shared actions. For two timed automata $A_1 = (L_1, l_0^1, \Sigma_1, \Delta_1, X_1)$ and $A_2 = (L_2, l_0^2, \Sigma_2, \Delta_2, X_2)$, the *parallel composition* $A_1 \| A_2$ is the timed automaton $(L_1 \times L_2, (l_0^1, l_0^2), \Sigma_1 \cup \Sigma_2, \Delta, X_1 \cup X_2)$, where Δ is the smallest set that contains

- $(l_1, l_2) \xrightarrow{a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2} (l_1', l_2')$,
  if $a \in \Sigma_1 \cap \Sigma_2$, $l_1 \xrightarrow{a, \varphi_1, \lambda_1} l_1' \in \Delta_1$ and $l_2 \xrightarrow{a, \varphi_2, \lambda_2} l_2' \in \Delta_2$,
- $(l_1, l_2) \xrightarrow{a, \varphi_1, \lambda_1} (l_1', l_2)$,
  if $a \in \Sigma_1 \setminus \Sigma_2$, $l_1 \xrightarrow{a, \varphi_1, \lambda_1} l_1' \in \Delta_1$, and
- $(l_1, l_2) \xrightarrow{a, \varphi_2, \lambda_2} (l_1, l_2')$,
  if $a \in \Sigma_2 \setminus \Sigma_1$, $l_2 \xrightarrow{a, \varphi_2, \lambda_2} l_2' \in \Delta_2$.

If A_1 is (X, m, c_{max})-granular and A_2 is (X', m', c'_{max})-granular, then the *combined granularity* of $A_1 \| A_2$ is $(X \cup X', m \cdot m', \max(c_{max}, c'_{max}))$.

Finite semantics. The decidability of the reachability problem of timed automata relies on the existence of the *region equivalence relation* [1] on $\mathcal{R}$ which has a finite index. In the following, we fix a μ-granular timed automaton $A = (L, l_0, \Sigma, \Delta, X)$ with $\mu = (X, m, c_{max})$. We say that two clock valuations $\boldsymbol{t}_1, \boldsymbol{t}_2 \in \mathcal{R}$ are in the same *clock region*, denoted $\boldsymbol{t}_1 \sim \boldsymbol{t}_2$, if

- the set of clocks with maximal value is the same in $\boldsymbol{t}_1$ and in $\boldsymbol{t}_2$
  (i.e., $\forall x \in X : \boldsymbol{t}_1(x) > c_{max} \Leftrightarrow \boldsymbol{t}_2(x) > c_{max}$), and
- $m \cdot \boldsymbol{t}_1$ and $m \cdot \boldsymbol{t}_2$ agree (1) on the integer parts of the clock values, (2) on the relative order of the fractional parts of the clock values, and (3) on the equality of the fractional parts of the clock values with 0. That is, for all clocks x and y with nonmaximal value, it holds that
  (1) $\lfloor m \cdot \boldsymbol{t}_1(x) \rfloor = \lfloor m \cdot \boldsymbol{t}_2(x) \rfloor$,
  (2) $fr(m \cdot \boldsymbol{t}_1(x)) \leq fr(m \cdot \boldsymbol{t}_1(y)) \Leftrightarrow fr(m \cdot \boldsymbol{t}_2(x)) \leq fr(m \cdot \boldsymbol{t}_2(y))$, and
  (3) $fr(m \cdot \boldsymbol{t}_1(x)) = 0$ iff $fr(m \cdot \boldsymbol{t}_2(x)) = 0$,
  where $fr(m \cdot \boldsymbol{t}_i(x)) = m \cdot \boldsymbol{t}_i(x) - \lfloor m \cdot \boldsymbol{t}_i(x) \rfloor$ for $i \in \{1, 2\}$.

We denote with $[\boldsymbol{t}] = \{\boldsymbol{t}' \in \mathcal{R} \mid \boldsymbol{t} \sim \boldsymbol{t}'\}$ the clock region $\boldsymbol{t}$ belongs to. We say that two states $s_1 = (l_1, \boldsymbol{t}_1)$ and $s_2 = (l_2, \boldsymbol{t}_2)$ of A are *region-equivalent*, denoted by $s_1 \sim s_2$, if their locations are the same ($l_1 = l_2$) and the clock valuations are in the same clock region ($\boldsymbol{t}_1 \sim \boldsymbol{t}_2$), and denote with $[(l, \boldsymbol{t})] = \{(l, \boldsymbol{t}') \in L \times \mathcal{R} \mid \boldsymbol{t} \sim \boldsymbol{t}'\}$ the equivalence class of region-equivalent states that $(l, \boldsymbol{t})$ belongs to.

Regions are a suitable semantics for the abstraction of timed automata because they essentially preserve the set of time-abstracted prefixes: if there is a discrete transition $s \xrightarrow{a} s'$ from a state s to a state s' of a timed automaton, then there is, for all states t with $t \sim s$, a state t' with $t' \sim s'$ such that $t \xrightarrow{a} t'$ is a discrete transition with the same label. For timed transitions, a slightly weaker property holds: if there is a timed transition $s \xrightarrow{d} s'$ from a state s to a state s', then there is, for all states t with $t \sim s$, a state t' with $t' \sim s'$ such that there is a timed transition $t \xrightarrow{d'} t'$ (but possibly with $d' \neq d$).

The *finite semantics* of a timed automaton $A = (L, l_0, \Sigma, \Delta, X)$ is the finite directed graph $[\![A]\!]_\mu = (Q, q_0, T)$ where

- the symbolic state set $Q = \{[(l, \boldsymbol{t})] \mid (l, \boldsymbol{t}) \in L \times \mathcal{R}\}$ of $[\![A]\!]_\mu$ is the set of equivalence classes of region-equivalent states of A, with
- the initial state $q_0 = [(l_0, \boldsymbol{t}_0)]$, and
- the set $T = \{(q, q') \in Q \times Q \mid \exists t \in q,\ t' \in q', a \in \Sigma \cup \mathbb{R}_{\geq 0}.\, t \xrightarrow{a} t'\}$ of transitions.

The finite semantics of a timed automaton A is also sometimes called the *region graph* of A.

The finite semantics is reachability-preserving:

Lemma 1. *[1] For a timed automaton $A = (L, l_0, \Sigma, \Delta, X)$ there is a finite path from a state $(l, \boldsymbol{t})$ to a state $(l', \boldsymbol{t}')$ if, and only if, there is a finite path from $\big[(l, \boldsymbol{t})\big]$ to $\big[(l', \boldsymbol{t}')\big]$ in $[\![A]\!]_\mu$.*

Reachability model checking. The decidability of checking reachability properties for timed automata relies on the existence of the so called *region abstraction* that yields a *finite semantics*. Applying this abstraction on a given timed automaton gives a finite automaton whose number of states is linear in the locations and exponential in the granularity:

Lemma 2. *[1] For a μ-granular timed automaton $A = (L, l_0, \Sigma, \Delta, X)$ with $\mu = (X, m, c_{max})$, there always exists a finite automaton A' which preserves the reachability information of the states of A. Furthermore, the number of states of A' is bounded by*

$$\begin{aligned} & |L| \cdot |X|! \cdot 2^{|X|} \cdot \prod_{x \in X} O(m \cdot c_{max}) \\ = \ & |L| \cdot |X|! \cdot O(m \cdot c_{max})^{|X|}. \end{aligned}$$

For a given timed automaton A, we define $\mathsf{Reach}(A)$ as the set of all states forward reachable of A. For a set of states B, characterizing the bad states of A, we use $\mathsf{Safe}(A, B)$ as an abbreviation for $\mathsf{Reach}(A) \cap B = \emptyset$. We assume that B can be compactly represented by a Boolean predicate over the locations and clock values of A. The *model checking problem* (MC) is to decide whether $\mathsf{Safe}(A, B)$ is true. For deciding MC, the region abstraction is a theoretically optimal state space representation:

Theorem 1. *[1] For a timed automaton A and a set of bad states B, deciding* $\mathsf{Safe}(A, B)$ *is* PSPACE*-complete.*

3 Timed Control with Partial Observability

In this section, we recall some known results for timed controller synthesis, which form the starting point of our investigation.

Plants and controllers. A *partially observable plant* is a tuple $(P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, where P is a timed automaton $(L, l_0, \Sigma, \Delta, X)$, Σ_{in} and $\Sigma^{\mathsf{obs}}_{\mathsf{out}}$ are the input and observable output actions, respectively, with $\Sigma_{\mathsf{in}} \uplus \Sigma^{\mathsf{obs}}_{\mathsf{out}} \subseteq \Sigma$, and $X^{\mathsf{obs}} \subseteq X$ are the observable clocks. For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, with $P = (L_p, l^p_0, \Sigma_p, \Delta_p, X_p)$, a *controller for* $\mathcal{P}$ is a deterministic timed automaton $C = (L_c, l^c_0, \Sigma_c, \Delta_c, X_c)$ with $X_c \cap X_p = X^{\mathsf{obs}}$ and $\Sigma_c = \Sigma_{\mathsf{in}} \cup \Sigma^{\mathsf{obs}}_{\mathsf{out}}$ such that C does neither

(1) *reset plant clocks*: for each $l \xrightarrow{a,\varphi,\lambda} l' \in \Delta_c$, we require that $\lambda \cap X_p = \emptyset$;
(2) *inhibit plant actions*: for all timed prefixes $w \in L(P\|C)$ with $w.u \in L(P)$ and $u \in \Sigma^{\mathsf{obs}}_{\mathsf{out}}$, we require that $w.u \in L(P\|C)$; nor
(3) *introduce timelocks*: for all timed prefixes $w \in L(P\|C)$, we require that there is a $d \in \mathbb{R}_{\geq 0}$ and a $c \in \Sigma_{\mathsf{in}}$ such that $w.d.c \in L(P\|C)$.

We treat the case where the controller has complete information as a special case. We say that $(P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$ is *fully observable*, if

(1) P is deterministic,
(2) $\Sigma_{\mathsf{in}} \cup \Sigma^{\mathsf{obs}}_{\mathsf{out}} = \Sigma$, and
(3) $X^{\mathsf{obs}} = X$.

For a (partially or fully observable) plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$ and a set of bad states B, the *controller synthesis problem* is to synthesize a controller C such that $\mathsf{Safe}(P\|C, B)$. Recall that we require timed automata (so the controllers) to be non-zeno. This way, we rule out trivial solutions consisting of a (physically unmeaningful) zeno controller that achieves its safety objective just by executing discrete actions infinitely often in a bounded amount of time.

Controller synthesis. For the fully observable setting, Maler et al. showed that the controller synthesis problem can be reduced to solving a finite two-player safety game (which is known to be PTIME-complete [17]) on the finite semantics of the given plant. They also showed that a fully informed controller can always be expressed in the granularity of the plant:

Lemma 3. *[23] For a fully observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, where P is μ-granular, and a set of bad states B, if there is a controller C for $\mathcal{P}$, such that* $\mathsf{Safe}(P\|C, B)$*, then there is a μ-controller C' (i.e., with no own clocks) such that* $\mathsf{Safe}(P\|C', B)$*.*

Henzinger and Kopke proved that the game-theoretic synthesis algorithm based on the finite semantics is theoretically optimal.

Theorem 2. *[16] For a fully observable plant P and a set of bad states B, synthesizing a controller C for P, such that* $\mathsf{Safe}(P\|C, B)$*, is* ExpTime*-complete.*

Bouyer et al. showed that in the presence of partial observability, the general timed synthesis problem becomes undecidable, even when a bound is imposed on the number of clocks of the controller.

Theorem 3. *[4,3] For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$ and a set of bad states B, the following holds:*

(1) Synthesizing a controller C for $\mathcal{P}$, such that $\mathsf{Safe}(P\|C, B)$*, is undecidable.*
(2) For a given integer constant $k \in \mathbb{N}_{\geq 1}$, synthesizing a controller C for $\mathcal{P}$ with k clocks, such that $\mathsf{Safe}(P\|C, B)$*, is undecidable.*

However, an important result of their work is that by imposing a granularity bound on the controller, one achieves decidability.

Theorem 4. *[4,3] For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, a granularity μ, and a set of bad states B, synthesizing a μ-controller C for $\mathcal{P}$, such that* $\mathsf{Safe}(P\|C, B)$*, is* 2ExpTime*-complete.*

Inspired by the last theorem, our paper continues this line of research by investigating finer bounds on the controller.

This concludes the recalling of the results that can be found in the literature. Based on these results, we start with our investigation.

4 Location-bounded Controllers

This section starts the presentation of our new results. First, we investigate the impact of bounding only the number of locations while leaving the granularity unspecified. It turns out that this does not bring decidability.

Theorem 5. *For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, a set of bad states B, and an integer $k \in \mathbb{N}_{\geq 1}$, synthesizing a controller C for $\mathcal{P}$ with k locations, such that* $\mathsf{Safe}(P\|C, B)$*, is undecidable.*

The proof is based on a reduction from the halting problem of a given two-counter Minsky machine, which is known to be undecidable [24]. The basic idea is to let the synthesis algorithm generate a controller that simulates an accepting run of the machine, or to report that no such controller/run exists. Following the standard construction proposed in [3] (which, in turn, is an extension of the one proposed in [1]) we let the plant nondeterministically and unobservably for the controller verify that he faithfully performs the simulation. The challenge in obtaining the undecidability result of Theorem 5 is to reduce the halting problem to the existence of a controller with a *bounded* number of locations. In

the appendix, we give a novel encoding where the computation of the Minsky machine is entirely stored in the clock values. The proof thus reduces the halting problem to the existence of a controller with a *single* location.

When bounding the locations of the controller *and* its granularity, the complexity for controller synthesis drops from 2ExpTime-complete (cf. Theorem 4) to NExpTime-complete.

Theorem 6. *For a partially observable plant* $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}^{\mathsf{obs}}, X^{\mathsf{obs}})$*, a granularity* μ*, a set of bad states* B*, and a bound* $k \in \mathbb{N}$*, synthesizing a* μ*-controller* C *for* $\mathcal{P}$ *with* k *locations, such that* $\mathsf{Safe}(P \| C, B)$*, is* NExpTime*-complete.*

We note that this result does not depend on the (unary or binary) encoding of k and μ. The proof of the NExpTime lower bound is based on a novel proof technique that uses clocks to represent bits for querying an edge relation of a succinctly represented graph. The key idea is to represent exponentially many nodes via only polynomially many clocks. We provide a polynomial-time reduction from Succinct graph coloring, which is known to be NExpTime-complete [21,26,32]. In our reduction, we use an answer of the synthesis problem to decide whether there is a k-coloring of a given undirected graph that is succinctly represented (i.e., the graph's edge relation E is given by a Boolean function).

In the (possibly infinite) interaction between plant and controller, the plant nondeterministically selects a node n and queries a color c from the controller. Then, the plant selects a second node n' and queries a color c'. If n and n' are connected via E and c and c' are the same, the plant enters a bad state. Otherwise, the colors of another two nodes are queried, and so on. A selected node is communicated to the controller by letting him read the values of the clocks representing that node.

For showing the NExpTime upper bound, one can provide a nondeterministic algorithm that guesses a controller in exponential time, and then validates that the guess was correct. For a granularity $\mu = (X, m, c_{max})$, the number of distinct atomic constraints is bounded by

$$\gamma = \prod_{x \in X} O(m \cdot c_{max}) = O(m \cdot c_{max})^{|X|},$$

which is single exponential (recall that m and c_{max} are given in binary using polynomially many bits). Now, in each location admitted to C, for each atomic constraint and each event from Σ, we have to decide (1) which clocks to reset and (2) in which location to change next. Note that, because we require C to be deterministic, we only have to make this decision once for every atomic constraint. Hence, since this choice has to be repeated for every location admitted to C, the number of possible controllers is bounded by

$$\left(k \cdot 2^{|X|}\right)^{\gamma \cdot |\Sigma| \cdot k}$$

and a single controller can thus be represented using

$$\gamma \cdot |\Sigma| \cdot k \cdot \left(\lceil \log k \rceil + |X|\right)$$

(i.e., only single exponentially) many bits. The validation relies on model checking, which is, according to Theorem 1, in PSPACE $\subseteq$ NEXPTIME. Note that, from a complexity-theoretic point of view, this is the best one can do, since any deterministic algorithm would have a double exponential worst-case running time, unless NEXPTIME = EXPTIME.

Concerning the size of the representation of the smallest feasible controller, if there is one at all, the following theorem states that it is highly unlikely[3] that a small controller always exists.

Theorem 7. *For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, a granularity μ, a set of bad states B, and a bound $k \in \mathbb{N}$, if there is a μ-controller C for $\mathcal{P}$ with k locations, such that $\mathsf{Safe}(P\|C, B)$, then C cannot always be represented polynomially, unless* NEXPTIME = PSPACE.

5 Discrete Controllers

In this section, we investigate the impact of restricting the controller to be a pure discrete system communicating synchronously with an arbitrary timed plant.

Definition. For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, we say that a controller $C = (L_c, l^c_0, \Sigma_c, \Delta_c, X_c)$ is *discrete*, if $|X_c| = 1$ and for each $l \xrightarrow{a,\varphi,\lambda} l' \in \Delta_c$ it holds that $\lambda = X_c$ and either

(1) $a \in \Sigma^{\mathsf{obs}}_{\mathsf{out}}$ and $\varphi \equiv \mathbf{true}$, or
(2) $a \in \Sigma_{\mathsf{in}}$ and $\varphi \equiv x \leq 0$, assuming $X_c = \{x\}$.

Intuitively, discrete controllers only react to discrete observations of the plant. They are not allowed to measure the time between two observed events.

We want to point out that discrete controllers differ from controllers with a fixed sampling rate considered in [16,8]. Obviously, the only meaningful bound which one can impose on discrete controllers is to restrict the number of locations. In the following, we investigate the bounded and the unbounded case.

5.1 Bounded Case

Requiring that the controller should be discrete, and, additionally, bounding the number of locations of the controller, reduces the complexity of the synthesis problem from 2EXPTIME-complete (cf. Theorem 4) to PSPACE-complete. The problem is thus exactly as hard as model checking (cf. Theorem 1).

Theorem 8. *For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, a set of bad states B, and a bound $k \in \mathbb{N}$ given in unary, synthesizing a discrete controller C for $\mathcal{P}$ with k locations, such that $\mathsf{Safe}(P\|C, B)$, is* PSPACE*-complete.*

[3] as it is common belief that PSPACE $\subsetneq$ EXPTIME $\subsetneq$ NEXPTIME

The lower bound immediately follows from the PSPACE-hardness of timed model checking, which is easily seen to be a special case.

Containment in NPSPACE (which is known to coincide with PSPACE [29]) can be established through the following nondeterministic algorithm. As the synthesized controller must be discrete and since every controller should be deterministic, a number of bits polynomial in $|\Sigma_{\mathsf{in}} \cup \Sigma^{\mathsf{obs}}_{\mathsf{out}}| \cdot k$ suffices to fully describe a controller. Hence, our algorithm can just guess these bits in polynomial time and then use timed model checking as an oracle to verify the guess. In summary, our algorithm is in $\text{NPTIME}^{\text{MC}} \subseteq \text{NPSPACE} = \text{PSPACE}$.

An effective synthesis algorithm. To illustrate the practical relevance of the PSPACE upper bound, we now describe an effective deterministic algorithm for the synthesis of bounded discrete controllers. The algorithm is based on a symbolic fixed point iteration. We use (a polynomial number of) Boolean variables to represent the structure of the controller (i.e., which locations are connected via an edge with a certain action). Sets of locations are represented using Boolean functions over a set of $O(\log k)$ location variables.

Let R be the set of states of the finite semantics of the plant, S be the set of all possible controller structures, and L be the set of all locations for all possible structures. Our algorithm incrementally computes a partial function $f : R \to S \to L$ such that, for each location $l \in f(r, s)$, the combined plant/controller state (r, l) is backward reachable assuming that the controller is of structure s. In an actual implementation, one would represent f as a mapping from regions to tuples from $2^S \times 2^L$, which, in turn, can be efficiently represented using discrete symbolic data structures (such as binary decision diagrams).

Initially, f maps each bad region to **true** (representing all controllers and locations) and each other region to **false** (representing no controllers and no locations). In each step of the fixed point iteration, we backpropagate from each region r the annotated pair of controller structures/locations over all transitions leading to r. When backpropagating a pair, represented by a Boolean formula φ, over a transition t, the resulting formula φ' is obtained by computing the weakest predecessors of φ. For the source region r' of t, we update $f(r') := f(r') \vee \varphi'$.

Once the fixed point is reached, we can derive the feasible controller structures from the annotation of the initial region. For this purpose we quantify the conjunction of the annotation of the initial region with the initial controller location existentially over the location variables. The set of structures characterized by the resulting Boolean function are the *infeasible* controllers. Hence, the negation yields the *feasible* controllers.

Since, according to Lemma 2, there are only single-exponentially many regions, and since a particular region is visited at most single-exponentially often (because there are only single-exponentially many controllers), we obtain in total a single-exponential running time (note that the two exponents multiply). We can therefore conclude that, unless PTIME = PSPACE, the time-complexity of the deterministic algorithm matches the complexity established in Theorem 8.

Inspired by the argumentation for the upper bound in Theorem 8, one might ask for the complexity of the synthesis problem if we impose a polynomial bound on the finite semantics of the plant. We can show that, in this case, the synthesis problem even becomes NPTime-complete.

Lemma 4. *For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}^{\mathsf{obs}}, X^{\mathsf{obs}})$, where the number of regions of P is polynomial in the size of P, a set of bad states B, and a bound $k \in \mathbb{N}$, synthesizing a discrete controller C for P with k locations, such that $\mathsf{Safe}(P \| C, B)$, is* NPTime*-complete.*

The lower bound can be shown by a reduction from Graph coloring, which is known to be NPTime-hard. The reduction goes analogously to the one for establishing the lower bound for Theorem 6 with the difference that, here, we use polynomially many locations (and no clocks) to represent the explicitly given graph in the plant.

Before we prove containment in NPTime, let us first ascertain the following fact that immediately follows from the well-known result that reachability checking on explicitly represented graphs is NLogSpace-complete:

Lemma 5. *[18] For a given directed graph $G = (V, E)$ with nodes V and edges E, and a set of bad nodes $V' \subseteq V$, finding a lasso (i.e., a path leading to and containing some cycle in G), which avoids any nodes in V', is* NLogSpace*-complete.*

Now, the NPTime upper bound of Lemma 4 can be established by the following nondeterministic algorithm that runs in polynomial time. Analogously to establishing the upper bound for Theorem 8, we first guess a controller in polynomial time. But now, the model checking procedure runs on a region graph of only polynomial size and, according to Lemma 5, requires only logarithmic space. Thus, the problem is in $\text{NPTime}^{\text{NLogSpace}} = \text{NPTime}$.

It is straight forward to see that, according to Lemma 2, the number of regions is only exponential in the granularity, but linear in the number of locations. Also, note that no plant clocks were used in establishing the NPTime lower bound for the last lemma. Consequently, we can state the following corollary.

Corollary 1. *For a fixed granularity μ, the problem of synthesizing a bounded discrete safety controller for a partially observable μ-granular plant is* NPTime*-complete.*

5.2 Unbounded Case

It turns out that the restriction to discrete controllers does not pay off in the unbounded case. In fact, we obtain the same 2ExpTime complexity bounds (cf. Theorem 4) as for the general synthesis problem already investigated in the literature [11,4,3].

Theorem 9. *For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}^{\mathsf{obs}}, X^{\mathsf{obs}})$ and a set of bad states B, synthesizing a discrete controller C for $\mathcal{P}$ with an unspecified number of locations, such that $\mathsf{Safe}(P \| C, B)$, is* 2ExpTime*-complete.*

The upper bound follows immediately from the upper bound established in Theorem 4. However, we additionally provide a deterministic algorithm that runs in double exponential time. First, we obtain a new plant automaton P' by enriching P by a fresh clock x, which is reset to 0 on every edge with an action $a \in \Sigma_{\mathsf{in}}$. On every edge of P' with an action $a \in \Sigma_{\mathsf{out}}^{\mathsf{obs}}$, we strengthen the guard with $x \leq 0$. Then, we construct the region graph of P', which, according to Lemma 2, is of single exponential size. We hide unobservable action and delay transitions by replacing them by ε-transitions. Finally, we obtain an equivalent finite game with perfect information by constructing the so-called belief space [27], which leads to a second exponential blowup. Since solving pure discrete safety games is PTime-complete [17,16], we conclude that our algorithm requires double exponential time.

The AExpSpace = 2ExpTime lower bound, which is more technically involved, is established by a reduction from the halting problem of an alternating Turing machine whose tape length is bounded exponentially in the size of the input. In our reduction, the Turing machine reaches its final state iff there exists a safe controller. Similar to a proof presented by Rintanen in the reachability planning setting [28], instead of storing the contents of the whole tape, we let the plant, unobservable for the controller, select a dedicated tape cell that should be watched. Unlike in the pure discrete setting of [28], we use polynomially many clocks (instead of Boolean variables) to represent the bits of some integer variables encoding the exponentially large index of the watched tape cell and the current position of the tape head.

Also different to [28], as our interest lies in safety controllers, we need to avoid that a controller is synthesized that never reaches the final state by infinitely looping through some other states. For this, we introduce a counter that keeps track of the number of steps executed so far. Since the maximal number of steps without visiting a state twice corresponds to the number of possible configurations, we can use this maximal number as a general bound, beyond which the plant immediately enters a bad state. Unfortunately, as this number is double exponential in the number of bits (i.e., clocks), we cannot use just another integer variable to represent the step counter (because this variable would require exponentially many bits). Instead, we let the plant force the controller to faithfully produce the correct sequence of bits of the step counter. Again, instead of remembering all bits, we let the plant nondeterministically and unobservably for the controller select a dedicated bit that should be watched, whose correct incrementation is verified.

We point out that, for establishing the lower bound, the 2ExpTime-hardness proof given in [11], where the controller is timed and can observe all clocks, does not apply to our setting of discrete controllers with partial observability.

6 Conclusion

In this paper, we have extended the bounded synthesis approach [30] to timed systems. We have established the complexity of timed control with partial observ-

ability under different types of bounds, and under different restrictions on the granularity. Our results in particular identify the synthesis of discrete controllers (over timed plants with limited observability) as a special case with significant practical relevance and, at the same time, very reasonable complexity: synthesizing discrete controllers is no harder than model checking, and can, in fact, be implemented with a symbolic fixed point iteration similar to BDD-based model checking [5,6]. Our results thus draw a much more optimistic picture for the synthesis of realistic controllers than previous work on the unbounded synthesis problem, where the introduction of real-time was shown to cause an exponential blow-up and partial observability was shown to make the problem undecidable.

The bounded synthesis approach is useful both when a reasonable bound is fixed *a priori* and when no bound is known in advance and the algorithm must, instead, *search* for the right bound. Bounded synthesis with iteratively increasing bounds is a complete method for the unbounded synthesis problem, with the significant advantage over previously studied approaches that the smallest solutions are found first.

Acknowledgment. This work was partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

References

1. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In J.-F. Lafay, editor, *Proc. 5th IFAC Conference on System Structure and Control*, pages 469–474. Elsevier, 1998.
3. Patricia Bouyer and Fabrice Chevalier. On the control of timed and hybrid systems. *EATCS Bulletin*, 89:79–96, June 2006.
4. Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit. Timed control with partial observability. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2003.
5. Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.
6. Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.
7. Franck Cassez, Alexandre David, Kim Guldstrand Larsen, Didier Lime, and Jean-François Raskin. Timed control with observation based and stuttering invariant strategies. In Namjoshi et al. [25], pages 192–206.
8. Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In Claire Tomlin and Mark R. Greenstreet, editors, *HSCC*, volume 2289 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2002.

9. Thomas Chatain, Alexandre David, and Kim G. Larsen. Playing games with timed games. In Alessandro Giua, Manuel Silva, and Janan Zaytoon, editors, *Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09)*, Zaragoza, Spain, September 2009.
10. Taolue Chen and Jian Lu. Towards the complexity of controls for timed automata with a small number of clocks. In Jun Ma, Yilong Yin, Jian Yu, and Shuigeng Zhou, editors, *FSKD (5)*, pages 134–138. IEEE Computer Society, 2008.
11. Deepak D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In Helmut Alt and Afonso Ferreira, editors, *STACS*, volume 2285 of *Lecture Notes in Computer Science*, pages 571–582. Springer, 2002.
12. Rüdiger Ehlers. Symbolic bounded synthesis. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2010.
13. Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. An antichain algorithm for ltl realizability. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2009.
14. Bernd Finkbeiner and Hans-Jörg Peter. Template-based controller synthesis for timed systems. In Cormac Flanagan and Barbara König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 392–406. Springer, 2012.
15. Bernd Finkbeiner and Sven Schewe. SMT-based synthesis of distributed systems. In *Proceedings of the 2nd Workshop on Automated Formal Methods (AFM 2007), 6 November, Atlanta, Georgia, USA*, pages 69–76. ACM Press, 2007.
16. Thomas A. Henzinger and Peter W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221(1-2):369–392, 1999.
17. Neil Immerman. Number of quantifiers is better than number of tape cells. *J. Comput. Syst. Sci.*, 22(3):384–406, 1981.
18. Neil D. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.*, 11(1):68–85, 1975.
19. Orna Kupferman, Yoad Lustig, Moshe Y. Vardi, and Mihalis Yannakakis. Temporal synthesis for bounded systems and environments. In Thomas Schwentick and Christoph Dürr, editors, *STACS*, volume 9 of *LIPIcs*, pages 615–626. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.
20. François Laroussinie, Nicolas Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2004.
21. Antoni Lozano and José L. Balcázar. The complexity of graph problems for succinctly represented graphs. In Manfred Nagl, editor, *WG*, volume 411 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 1989.
22. Yoad Lustig and Moshe Y. Vardi. Synthesis from component libraries. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2009.
23. Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In Ernst W. Mayr and Claude Puech, editors, *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995.
24. Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc, 1967.

25. Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors. *Automated Technology for Verification and Analysis, 5th International Symposium, ATVA 2007, Tokyo, Japan, October 22-25, 2007, Proceedings*, volume 4762 of *Lecture Notes in Computer Science*. Springer, 2007.
26. Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
27. John H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301, 1984.
28. Jussi Rintanen. Complexity of planning with partial observability. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *ICAPS*, pages 345–354. AAAI, 2004.
29. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
30. Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In Namjoshi et al. [25], pages 474–488.
31. Wolfgang Thomas. On the synthesis of strategies in infinite games. In *STACS*, pages 1–13, 1995.
32. Helmut Veith. Languages represented by boolean formulas. *Inf. Process. Lett.*, 63(5):251–256, 1997.

A Proof of Theorems

We first present a general technique how to use clocks of a timed automaton to represent bits of integer variables, which we will use later in the following proofs of the lemmas and theorems.

A.1 Using Clocks to Represent Bits

For an integer variable v whose value ranges over $\{0, \ldots, n-1\}$, where $n = 2^b$ and $b \in \mathbb{N}$, we introduce the clocks $x_1, \ldots, x_b$, abbreviated as $\boldsymbol{x}_b$. Assuming the value of v is represented by the bit string $\langle v_b \ldots v_1 \rangle$, we use the following *normal form* encoding: for all $1 \leq i \leq b$,

$$v_i = \begin{cases} 0 & \text{if } x_i = 0 \\ 1 & \text{if } x_i > 0 \end{cases}$$

For *comparing* two integer variables, we can use the following equivalence as an edge guard. Assuming the clocks $\boldsymbol{x}_b$ and $\boldsymbol{y}_b$ represent the bits of two variables v and w, respectively, the value represented by v equals the value represented by w iff

$$\bigwedge_{i=1}^{b} x_i = 0 \Leftrightarrow y_i = 0.$$

A *nondeterministic choice*, setting v to some arbitrary value, can be implemented using the following gadget. First, we let exactly one time unit elapse (e.g., using an auxiliary clock z). Then, we iterate over each bit $1 \leq i \leq b$ and nondeterministically reset x_i or not.

The *incrementation* of v is modeled by the following gadget, which resembles a chain of half adders. We introduce a location (i, c), for each bit $i \in \{1, \ldots, b\}$ and a carry flag $c \in \{0, 1\}$. After the gadget is entered, we assume that the representation for the value of v using the clocks $\boldsymbol{x}_b$ is in normal form. First, we let exactly one time unit pass by. Then, the actual incrementation starts at location $(1, 1)$. At each location (i, c), we have two edges:

(1) with guard $x_i > 1 \wedge c = 1 \ \vee \ x_i = 1 \wedge c = 0$, which sets $x_i := 0$, and
(2) with guard $x_i = 1 \wedge c = 1 \ \vee \ x_i > 1 \wedge c = 0$, which leaves x_i unchanged.

If $i < b$, the target locations of edges (1) and (2) are $(i + 1, c)$ and $(i + 1, 0)$, respectively.
If $i = b$, (1) and (2) exit the gadget.

The *decrementation* of v is modeled by a similar gadget, which exploits the fact that

$$v + n - 1 \equiv v - 1 \pmod{n}.$$

Again, we introduce a location (i, c), for each bit $i \in \{1, \ldots, b\}$ and a carry flag $c \in \{0, 1\}$. After the gadget is entered, we assume that the representation for the

value of v using the clocks $\boldsymbol{x}_b$ is in normal form. First, we let exactly one time unit pass by. Then, the actual decrementation starts at location $(1, 0)$. At each location (i, c), we have two edges:

(1) with guard $x_i > 1 \wedge c = 0 \ \vee \ x_i = 1 \wedge c = 1$, which sets $x_i := 0$, and
(2) with guard $x_i = 1 \wedge c = 0 \ \vee \ x_i > 1 \wedge c = 1$, which leaves x_i unchanged.

If $i < b$, the target locations of edges (1) and (2) are $(i + 1, 1)$ and $(i + 1, c)$, respectively.
If $i = b$, (1) and (2) exit the gadget.

Clearly, immediately after exiting the increment or decrement gadget, v is in normal form encoding. Further, note that after globally letting time elapse to increment or decrement v, it is no problem to restore the normal form of the other variables again.

A.2 Proofs

Before we come to the proof of Theorem 5, we first give a proof for the following lemma.

Lemma 6. *For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, a set of bad states B, and an integer $k \in \mathbb{N}_{\geq 1}$, synthesizing a controller C for $\mathcal{P}$ with k clocks, such that $\mathsf{Safe}(P \| C, B)$, is undecidable.*

Proof. We show undecidability by a reduction from the halting problem of a given two-counter Minsky machine, which is known to be undecidable [24]. The basic idea is to let the synthesis algorithm generate a controller that simulates an accepting run of the machine, or to report that no such controller/run exists. Following the construction proposed in [3], which, in turn, is an extension of the one proposed in [1], we let the plant nondeterministically and unobservably for the controller verify that he faithfully performs the simulation. We refer to [3] for details on the modeling of the verification gadgets. Note that the goal state of the machine is reached after finitely many steps $m \in \mathbb{N}_{\geq 1}$ for a configuration with some maximal counter value bounded by m. Hence, the synthesis algorithm must somehow determine m (or report that no such m exists) and fix a sufficiently large number of locations and fine granularity $(\{z\}, 4m, 1)$ to accommodate the necessary information to keep track of the machine's configurations arising during its execution. We let the controller and the plant communicate via the actions a, b, c, and d. W.l.o.g., we assume that the states of the given machine have a unique index and their number is less than m.

As usual for such proofs, a configuration is encoded as a sequence of actions $d^s a^{c_1} b^{c_2} c^{c_3}$ representing the current machine state with index s, the current values of the two machine counters c_1 and c_2, and the value of a step counter c_3. Here, c_3 is necessary to force the controller to reach the goal state within a finite amount of steps. After the i^{th} step of the machine, we let the controller produce the current configuration sequence within the time interval $[i, i + 1)$. Here, we force the controller that the delay between corresponding actions in

two successive configurations is exactly one time unit. This can be achieved by a plant component that nondeterministically chooses an action, waits exactly one time unit, and then verifies whether the controller immediately produces that action. In the first configuration, we let the controller choose an appropriate value for $c_3 > 0$. After each step, the plant checks that the controller decrements c_3 by one. If c_3 becomes 0, we let the plant go into a bad state.

Now, assuming that the goal state of the given machine is reachable, let us fix some m. It is easy to see that the number of distinct configurations of the machine along with a certain step count is bounded by m^4. Hence, a feasible controller could have $O(m^8)$ locations, i.e., $O(m^2)$ locations to represent and produce the current configuration of the machine as well as the step count. If the machine is in a certain configuration after a certain number of steps, the corresponding part of the controller's control structure is of the following form:

$$l_d \underbrace{\xrightarrow{d,z=\frac{1}{4m},z:=0} \cdots \xrightarrow{d,z=\frac{1}{4m},z:=0}}_{s \text{ times}} l'_d \underbrace{\xrightarrow{\epsilon,z=\frac{1}{4m},z:=0} \cdots \xrightarrow{\epsilon,z=\frac{1}{4m},z:=0}}_{m-s \text{ times}} l''_d,$$

for representing the current machine state s;

$$l_a \underbrace{\xrightarrow{a,z=\frac{1}{4m},z:=0} \cdots \xrightarrow{a,z=\frac{1}{4m},z:=0}}_{c_1 \text{ times}} l'_a \underbrace{\xrightarrow{\epsilon,z=\frac{1}{4m},z:=0} \cdots \xrightarrow{\epsilon,z=\frac{1}{4m},z:=0}}_{m-c_1 \text{ times}} l''_a,$$

for representing the current value of the machine counter c_1;

$$l_b \underbrace{\xrightarrow{b,z=\frac{1}{4m},z:=0} \cdots \xrightarrow{b,z=\frac{1}{4m},z:=0}}_{c_2 \text{ times}} l'_b \underbrace{\xrightarrow{\epsilon,z=\frac{1}{4m},z:=0} \cdots \xrightarrow{\epsilon,z=\frac{1}{4m},z:=0}}_{m-c_2 \text{ times}} l''_b,$$

for representing the current value of the machine counter c_2;

$$l_c \underbrace{\xrightarrow{c,z=\frac{1}{4m},z:=0} \cdots \xrightarrow{c,z=\frac{1}{4m},z:=0}}_{c_3 \text{ times}} l'_c \underbrace{\xrightarrow{\epsilon,z=\frac{1}{4m},z:=0} \cdots \xrightarrow{\epsilon,z=\frac{1}{4m},z:=0}}_{m-c_3 \text{ times}} l''_c,$$

for representing the current step count c_3.

Thus, there is a controller iff there is an accepting run, and furthermore, if there is a controller at all then there is one that can be represented using a single clock.

Theorem 5. For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma^{\mathsf{obs}}_{\mathsf{out}}, X^{\mathsf{obs}})$, a set of bad states B, and an integer $k \in \mathbb{N}_{\geq 1}$, synthesizing a controller C for $\mathcal{P}$ with k locations, such that $\mathsf{Safe}(P \| C, B)$, is undecidable.

Proof. To show undecidability, we give a similar reduction from the halting problem of a given two-counter Minsky machine as the one used in the proof of Lemma 6. But now, the synthesized controller generating an accepting run (if there is one at all) has only one location and uses its clocks to represent all information necessary to keep track of the machine's current configuration. Such

a one-location controller is a translation of the one-clock controller from above, where the location-based control structure is simulated by a pure clock-based control structure. In the following, we explain this translation.

Assuming, w.l.o.g., the one-clock controller has 2^b locations, for some $b \in \mathbb{N}$, we introduce b clocks in the one-location controller $x_1, \ldots, x_b$. Also, we assume that each location has a unique index between 0 and $2^b - 1$. The one-clock controller is in location with index l iff $\boldsymbol{x} = l$, where

$$\boldsymbol{x} = l \quad :\Longleftrightarrow \quad \bigwedge_{i=1}^{b} x_i \leq \frac{1}{4m} \Leftrightarrow l_i = 0$$

and l_i refers to the i^{th} bit of l. Since each step in the one-clock controller takes exactly $\frac{1}{4m}$ time units, we also have an auxiliary clock z in the one-location controller that is reset on every discrete step. For each edge between two locations l and l' in the one-clock controller, we introduce a corresponding (self-looping) edge in the one-location controller with guard $z = \frac{1}{4m} \wedge \boldsymbol{x} = l$ that resets all clocks in $\boldsymbol{x}$ whose corresponding bit in l' is zero.

Thus, if there is a controller at all then there is one that can be represented using a single location.

Theorem 6. For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}^{\mathsf{obs}}, X^{\mathsf{obs}})$, a granularity μ, a set of bad states B, and a bound $k \in \mathbb{N}$, synthesizing a μ-controller C for $\mathcal{P}$ with k locations, such that $\mathsf{Safe}(P \| C, B)$, is NExpTime-complete.

Proof. Containment in NExpTime follows from the following nondeterministic algorithm. Let us fix $\mu = (X, m, c_{max})$. Observe that the number of distinct atomic constraints is bounded by

$$\gamma = \prod_{x \in X} O(m \cdot c_{max}) = O(m \cdot c_{max})^{|X|},$$

which is single exponential (recall that m and c_{max} are given in binary using polynomially many bits). Now, in each location admitted to C, for each atomic constraint and each event from Σ, we have to decide (1) which clocks to reset and (2) in which location to change next. Note that, because we require C to be deterministic, we only have to make this decision once for every atomic constraint. Hence, the number of possible controllers is bounded by

$$\left(k \cdot 2^{|X|}\right)^{\gamma \cdot |\Sigma| \cdot k}$$

and a single controller can thus be represented using

$$\gamma \cdot |\Sigma| \cdot k \cdot \left(\lceil \log k \rceil + |X|\right)$$

(i.e., only single exponentially) many bits. A closer look on the last formula reveals that we would still need exponentially many bits even if we would assume a unary (instead of a binary) encoding of the constants. This is because the single

exponential blow-up in γ comes from both the number of clocks and the encoding of the constants. In fact, the number of clocks, the constants in the granularity, and k (encoded in binary) independently induce at most a single exponential blow-up in the number of bits.

Now, our algorithm first guesses a deterministic μ-granular timed automaton C with k locations and actions $\Sigma_{\mathsf{in}} \cup \Sigma_{\mathsf{out}}^{\mathsf{obs}}$. Then, it verifies C by checking whether $\mathsf{Safe}(P \| C, B')$, where $B' \supseteq B$ is an enriched safety property that also forbids infeasible controllers (i.e., controllers which reset plant clocks, inhibit plant actions, or introduce timelocks). Since, according to Theorem 1, model checking can be done in PSPACE, we conclude that our algorithm is in $\text{NEXPTIME}^{\text{MC}} = \text{NEXPTIME}^{\text{PSPACE}} = \text{NEXPTIME}$. Note that, even though C can be exponentially large, the PSPACE upper bound for model checking $P \| C$ still holds: In the worst case, C is of the same size as its region graph, which is exponential. Then, the combined region graph of $P \| C$ is still of at most exponential size, since the exponential blow-up in P due to the succinctness of its clocks and constraints multiplies with the exponentially large control structure of C.

For proving NEXPTIME-hardness, we provide a polynomial reduction from SUCCINCT GRAPH COLORING, which is known to be NEXPTIME-complete [21,26,32]. Roughly speaking, this proof is an adaptation of the NPTIME-hardness proof for Lemma 4. The difference is that we assume here that the given graph is succinctly represented (i.e., its edge relation is given as a compact Boolean function).

For a given undirected graph $G = (V, E)$, our synthesis procedure should synthesize a controller with k locations iff there exists a k-coloring for G. W.l.o.g., we assume $|V| = 2^b$, for some $b \in \mathbb{N}$, and, for two nodes $n_1, n_2 \in V$, we assume that $E(n_1, n_2) = \mathbf{true}$ iff n_1 and n_2 are adjacent. Moreover, we assume that E is succinctly represented as a Boolean function with $2b$ inputs and one output, only using *AND*, *OR* and *NOT* gates. In our reduction, we will use b plant clocks for representing the bits of a node from V, as explained in A.1. For each color $1 \leq i \leq k$, we introduce a controller action c_i.

The (possibly infinite) interaction between plant and controller works as follows. The plant nondeterministically selects a node n and queries a color c from the controller. Then, the plant selects a second node n' and queries a color c'. If $E(n, n') = \mathbf{true}$ and $c = c'$, then the plant enters a bad state. Otherwise, the colors of another two nodes are queried, and so on. Technically, this node querying gadget represents n and n' using two node variables with b bits each. The gadget for nondeterministic choice is explained in A.1, and the translation of E into a corresponding isomorphic clock constraint is straight forward. For instance, suppose $b = 2$ and

$$E(n, n') \equiv n[0] \wedge \big(\neg n'[1] \vee n[1]\big),$$

assuming that $n[i]$ refers to the i^{th} bit of n, $0 \leq i < 2$, then we can translate this Boolean function into the clock constraint

$$(x_0 > 0) \wedge \big((y_1 = 0) \vee (x_1 > 0)\big),$$

assuming that x_0, x_1, and y_1 are the clocks representing the bits $n[0]$, $n[1]$, and $n'[1]$, respectively. We point out that, at the expense of introducing (a polynomial amount of) locations, this technique also works when we restrict ourselves to convex clock constraints.

We communicate a selected node n to the controller by letting him read the values of the clocks representing n. Whenever the selection of a node is done, the plant sends a dedicated action *query* to the controller. To ensure that the controller only uses one color per location, we introduce a gadget that runs in parallel and checks that the choice of the color stays stable between two subsequent *query* actions. To avoid that the controller uses clock constraints in his guards when sending a color, we always reset all observable clocks when the plant sends *query* (recall that, on executing a discrete transition in a timed automata, the clocks are reset *after* their values are read). Note that we can safely ignore the case where the controller wastes two locations for the same color, because, if such a controller can be synthesized, then also a $(k-1)$-coloring for G is possible.

Finally, it remains to force the controller not to cheat by sending inconsistent color assignments for the same node. For this purpose, we add a gadget that first nondeterministically and unobservably for the controller selects a node whose color should be watched. When that watched node is queried first, the plant memorizes the color sent by the controller. Then, whenever the watched node is queried again, the plant checks if the proposed color is the same as the memorized one.

We hide all steps from the controller that occur in the verification gadgets; only the node querying is visible for the controller. Note that the plant can be represented assuming a granularity $(X_p, 1, 1)$, where X_p contains all observable and hidden clocks needed to represent the node variables, and an auxiliary clock z for expressing urgent constraints. For the controller, we fix the granularity $\mu = (X_c, 1, 1)$, where X_c contains z and all observable clocks needed to read the queried node variable (i.e., the controller does not need any clocks of its own).

Theorem 7. For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}^{\mathsf{obs}}, X^{\mathsf{obs}})$, a granularity μ, a set of bad states B, and a bound $k \in \mathbb{N}$, if there is a μ-controller C for $\mathcal{P}$ with k locations, such that $\mathsf{Safe}(P \| C, B)$, then C cannot always be represented polynomially, unless NExpTime = PSpace.

Proof. We assume that C can always be represented polynomially in the size of the problem instance (i.e., C can be represented using polynomially many bits). Then, we can always guess a correct C (if there is one at all) in polynomially many steps and represent C using polynomial space. Thus, C can be guessed in NPSpace, which is known to coincide with PSpace [29]. Validating the guess can be done using model checking, which is, according to Theorem 1, in PSpace. Hence, the overall complexity is in $\text{PSpace}^{\text{PSpace}} = \text{PSpace}$. However, according to Theorem 6, synthesizing C is NExpTime-complete. Thus, NExpTime = PSpace (under the assumption that C can always be represented polynomially).

Theorem 8. For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}^{\mathsf{obs}}, X^{\mathsf{obs}})$, a set of bad states B, and a bound $k \in \mathbb{N}$ given in unary, synthesizing a discrete controller C for $\mathcal{P}$ with k locations, such that $\mathsf{Safe}(P \| C, B)$, is PSPACE-complete.

Proof. PSPACE-hardness follows immediately from Theorem 1 since the standard model checking problem for timed automata is just a special case: we assume that $\mathcal{P}$ is fully observable and $\Sigma_{\mathsf{in}} = \emptyset$.

Containment in NPSPACE, which is known to coincide with PSPACE [29], is established through the following nondeterministic algorithm. Observe that in each of the k locations admitted to a deterministic controller, there can be at most one edge per action $a \in \Sigma = \Sigma_{\mathsf{in}} \cup \Sigma_{\mathsf{out}}^{\mathsf{obs}}$ to some other location. Hence, the number of possible controllers is bounded by $k^{|\Sigma| \cdot k}$ and a single controller can be represented using only a polynomial number of bits. Now, our algorithm proceeds as follows: First, it guesses a controller C in polynomial time. Then, it verifies the guess by model checking $C \| P$, whose number of regions is still single exponential. Note that unobservable actions only limit the choices for C, while neither observable nor unobservable clocks do affect C at all. In summary, our algorithm is in $\text{NPTIME}^{\text{MC}} \subseteq \text{NPSPACE} = \text{PSPACE}$.

Lemma 4. For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}^{\mathsf{obs}}, X^{\mathsf{obs}})$, where the number of regions of P is polynomial in the size of P, a set of bad states B, and a bound $k \in \mathbb{N}$, synthesizing a discrete controller C for P with k locations, such that $\mathsf{Safe}(P \| C, B)$, is NPTIME-complete.

Proof. Containment in NPTIME can be shown by the following nondeterministic algorithm that runs in polynomial time. Analogously to the proof of Theorem 8, we first guess a controller in polynomial time. But now, the model checking procedure runs on a region graph of only polynomial size and, according to Lemma 5, requires only logarithmic space. Thus, the algorithm is in $\text{NPTIME}^{\text{NLOGSPACE}} = \text{NPTIME}$.

The NPTIME lower bound is established through a polynomial reduction from GRAPH COLORING, which is known to be NPTIME-hard. For a given undirected graph $G = (V, E)$, our synthesis procedure should synthesize a controller with k locations iff there exists a k-coloring for G. For each node $1 \leq i \leq |V|$, we introduce a plant action n_i, and for each color $1 \leq j \leq k$, we introduce a controller action c_j. The (possibly infinite) interaction between plant and controller works as follows. The plant nondeterministically selects a node n and queries a color c from the controller. Then, the plant selects an adjacent node n' and queries a color c'. If $c = c'$, then the plant enters a bad state. Otherwise, the colors of another two nodes are queried and verified, and so on. The nondeterministic node selection is modeled by letting the plant traverse over an automaton which is isomorphic to G (i.e., for an edge $(n, n') \in E$ we introduce two edges from location n to location n' sending the action corresponding to n', and vice versa). The verification of the colors can be done using a gadget with $O(k)$ locations. We add another gadget which ensures that the controller proposes only one color at a location admitted to him. Once the controller has proposed a certain color, the gadget enters a bad state whenever the controller sends a different color

before the next node change. This gadget can be realized using $O(k)$ locations. Note that we can safely ignore the case where the controller wastes two locations for the same color, because, if such a controller can be synthesized, then also a $(k-1)$-coloring for G is possible. Finally, it remains to force the controller not to cheat by sending inconsistent color assignments for the same node. For this purpose, we add a gadget that first nondeterministically and unobservably for the controller selects a node whose color should be watched. When that watched node is queried first, the plant memorizes the color sent by the controller. Then, whenever the watched node is queried again, the plant checks if the proposed color is the same as the memorized one. This gadget can be realized using $O(|V| \cdot k)$ locations.

In the construction of the plant, we do not need any clocks. We hide all steps from the controller that occur in the verification gadgets; only the traversal through G is visible for the controller.

Theorem 9. For a partially observable plant $\mathcal{P} = (P, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}^{\mathsf{obs}}, X^{\mathsf{obs}})$ and a set of bad states B, synthesizing a discrete controller C for $\mathcal{P}$ with an unspecified number of locations, such that $\mathsf{Safe}(P \| C, B)$, is 2ExpTime-complete.

Proof. Containment in 2ExpTime follows from the usual knowledge-based subset construction [27] on the region graph. First, we obtain a new plant automaton P' by enriching P by a fresh clock x, which is reset to 0 on every edge with an action $a \in \Sigma_{\mathsf{in}}$. On every edge of P' with an action $a \in \Sigma_{\mathsf{out}}^{\mathsf{obs}}$, we strengthen the guard with $x \leq 0$. Then, we construct the region graph of P', which, according to Lemma 2, is of single exponential size. We hide unobservable action and delay transitions by replacing them by ε-transitions. Finally, we obtain an equivalent finite game with perfect information by constructing the so-called belief space, which leads to a second exponential blowup. Since solving pure discrete safety games is PTime-complete [17,16], we conclude that our algorithm requires double exponential time.

AExpSpace-hardness (which is known to coincide with 2ExpTime-hardness) can be shown by a reduction from the halting problem of an exponentially bounded alternating Turing machine $M = (Q, q_0, q_f, \Gamma, \delta)$, where Q is a finite set of states with $Q = Q_\exists \uplus Q_\forall$, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the final state, Γ is a finite set of tape symbols, and $\delta : Q \times \Gamma \to Q \times Q \times \Gamma \times \{L, R\}$ is the transition function. We assume w.l.o.g. that M runs on a tape of length $n = 2^b$, for some $b \in \mathbb{N}$, and that $|\Gamma| = 2$. Let $m \leq n$, $m \in \mathbb{N}$, be the length of the input. We let the plant verify that the controller correctly simulates M. In our reduction, M reaches q_f iff there exists a safe controller.

Similar to a proof presented by Rintanen in the planning setting [28], instead of storing the contents of the whole tape, we let the plant, unobservable for the controller, select a dedicated tape cell that should be watched. Unlike in the pure discrete setting of [28], we use b clocks (instead of b Boolean variables) to represent the bits of some integer variable ranging from 0 to $n-1$, using the encoding from A.1. We use an unobservable integer variable *watched* to store the

index of the watched cell and another integer variable *cur* to store the current position of the tape head.

The (possibly infinite) interaction between plant and controller works as follows. As an initialization step, unobservable for the controller, the plant nondeterministically chooses a value for *watched* and sets *cur* to 0. Then, the plant keeps track of the movement of the tape head by decrementing or incrementing *cur*. Here, the plant enters a bad state whenever the controller proposes to move the head to the left and it is over the first cell, or to move to the right and the head is over the last cell. If $cur \neq watched$, the plant ignores the correctness of the tape operations (writing in a cell and moving the head) determined by the controller. If $cur = watched$, the tape operations are verified and the writing of the new tape symbol is memorized. The states of M as well as the contents of the watched tape cell are represented using $O(|Q| \cdot |\Gamma|)$ plant locations.

The transitions in δ are encoded in the following way. In case M is in an existential state from $Q_\exists$, the plant prompts the controller by sending him the dedicated action *prompt*, upon which he immediately reacts with a decision what to write into the current tape cell and how to move the tape head. In case M is in a universal state from $Q_\forall$, the plant decides in which state M branches by either sending *prompt1* or *prompt2* to the controller. Recall that the verification gadget gets activated whenever the current position of the head is over the watched cell. In this case, for a $q \in Q$ and a $\gamma \in \Gamma$, if $\delta(q, \gamma) = (q_1, q_2, \gamma', d)$ and $q \in Q_\exists$, the plant sends *prompt* to the controller, upon which he can either respond (q_1, γ', d) or (q_2, γ', d). If $q \in Q_\forall$, the plant either sends *prompt1* or *prompt2* to the controller, upon which he has to respond with (q_1, γ', d) or (q_2, γ', d), respectively. If the tape head is not over the watched cell, the plant prompts the controller nevertheless. But in this case, the plant does not restrict the choices of the controller.

Each edge must be taken in an urgent manner (i.e., no time may pass by), unless not explicitly mentioned otherwise (e.g., in some locations of the gadgets for manipulating integer variables from A.1). We can easily achieve this by introducing an auxiliary clock z, which is always set to zero after an edge is taken. Now, for a location l, imposing urgency for the plant can be established by strengthening the guards of the outgoing edges by $z = 0$. Imposing urgency for the controller can be established by adding all states $(l, \boldsymbol{t})$, for which $\boldsymbol{t} \models z > 0$ to the set of bad states. Note that if l is a location where exactly one time unit should pass by, then, of course, we adjust the guards to $z = 1$ and $z > 1$, respectively.

We add a dedicated location l_f representing the goal state q_f. In l_f, there are no outgoing transitions and time can elapse forever without reaching B. Finally, to avoid that a safe controller is synthesized that never reaches l_f (by infinitely looping through the states of M), we have to force the controller to reach l_f before a certain timeout (in terms of number of steps of M) has occurred. The maximal number of steps without visiting a state twice corresponds to the number of possible configurations of M. That is, we need to count steps up to a

number bounded by

$$|\Gamma|^n = |\Gamma|^{2^b} = 2^{2^b}.$$

Obviously, as this number is double exponential in the number of bits (i.e., clocks), we cannot use just another integer variable to represent the timeout counter. Instead, similar to the incrementation gadget proposed in A.1, we introduce a gadget implementing a chain of 2^b half adders. Before M performs a step, the plant forces the controller to produce a sequence of 2^b bits representing the current value of the timeout counter (ranging from 0 to $2^{2^b} - 1$). Now, to verify that the controller increments the timeout counter correctly, similar to the watched tape cell explained above, we let the plant nondeterministically and unobservably for the controller select a dedicated bit j, $0 \leq j < 2^b$, whose correct incrementation is verified. The actual gadget implements a loop that iterates an integer variable i from 0 to $2^b - 1$. In each iteration, the controller has to provide the carry flag before bit i is incremented, the new (incremented) value of bit i, and the carry flag for bit $i + 1$. If $i \neq j$, we require that the controller provides *some* value for the carry flags and the new value of bit i. If $i = j$, we actually check that the controller provides correct values (as explained in A.1). It is easy to see that the new value of bit j only depends on its last value and the incoming carry flag. Furthermore, if $i = 0$, we always require that the first carry flag is 1. If $i = 2^b - 1$ and the next carry flag is 1, the counter overflows, which means that we have reached the timeout and let the plant enter some bad state in B.