Taming Large Bounds in Synthesis from Bounded-Liveness Specifications [⋆]

Philippe Heim[0000−0002−5433−8133] and Rayna Dimitrova

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{philippe.heim, dimitrova}@cispa.de

Abstract Automatic synthesis from temporal logic specifications is an attractive alternative to manual system design, due to its ability to generate correct-by-construction implementations from high-level specifications. Due to the high complexity of the synthesis problem, significant research efforts have been directed at developing practically efficient approaches for restricted specification language fragments. In this paper we focus on the `Safety LTL` fragment of Linear Temporal Logic (LTL) syntactically *extended with bounded temporal operators*. We propose a new synthesis approach with the primary motivation to solve efficiently the synthesis problem for specifications with bounded temporal operators, in particular those with large bounds. The experimental evaluation of our method shows that for this type of specifications it outperforms state-of-art synthesis tools, demonstrating that it is a promising approach to efficiently treating quantitative timing constraints in safety specifications.

1 Introduction

Reactive synthesis [8] has the goal of automatically generating an implementation from a formal specification that describes the desired behavior of a reactive system. The system requirements are typically specified using temporal logics such as Linear Temporal Logic (LTL). Temporal logics are expressive, high-level specification languages capable of describing rich properties, such as, for example, robotic missions [16]. Specifications of reactive systems often include requirements of the form “something good eventually happens”. These can be expressed in LTL via the temporal operators $\mathcal{U}$ (“until”) and $\Diamond$ (“eventually”). “Eventually” is an abstraction for the existence of some unknown time point in the future of a system execution when some property holds true. While this abstraction is useful for avoiding over-specification, there are many situations in which there are practical bounds on the time within which a requirement must be met. In such cases, it is vital that the synthesis procedure checks if the timing requirements are realizable, and synthesizes an implementation that adheres to these bounds.

As a simple example, consider a specification of the desired behavior of a controller for the front door of an office building. Our specification states that the

[⋆] Philippe Heim carried out this work as PhD candidate at Saarland University, Germany.

door must always be locked at night, and unlocked otherwise. It also stipulates that in the event of a fire the door should eventually open. Formulated like this, the specification is realizable. However, in case of a fire during night the synthesized implementation will only open the door at the start of the day. Clearly, this is not the behavior we intended! We can specify the actual desired behavior in LTL by using the temporal operator $\bigcirc$ ("next"), which allows us to state that a property should hold at the next time step. However, we would need to use nested $\bigcirc$ operators in order to express the required time bounds. This can quickly become inconvenient, especially if we need to specify various different time bounds, some of them large. This modeling inconvenience and the increase of specification size are easily avoided by adding bounded versions of the temporal operators as syntactic sugar, without increasing expressiveness.

Due to their practical significance, fragments of LTL in which the formulas (in negation normal form) include only bounded versions of the $\mathcal{U}$ and $\diamondsuit$ operators have attracted considerable attention. The most prominent such fragment is `Safety LTL` the until-free fragment of LTL in negated normal form. Since `Safety LTL` is a syntactic fragment of LTL, it can express bounded liveness properties only via nested next operators. Another notable example is the logic Extended Bounded Response LTL ($\mathrm{LTL}_{\mathsf{EBR}}$) [9], which is a fragment of LTL that includes bounded temporal operators as well as unbounded universal temporal operators (i.e., "globally" and "release"). While every $\mathrm{LTL}_{\mathsf{EBR}}$ formula can be expressed in `Safety LTL`, one significant advantage of $\mathrm{LTL}_{\mathsf{EBR}}$ is that the bounds of the temporal operators are represented in binary, which allows for exponentially more succinct formulas. However, in the course of the synthesis procedure presented in [9] these bounds are expanded into nested "next" operators. Keeping bounds symbolic is identified in [9] as an interesting direction for future developments. Indeed, in many practically relevant cases large bounds are unavoidable due to requirements on the same system across different time-scales.

In this paper we address this challenge by proposing a synthesis procedure for an extension of `Safety LTL` with bounded operators. We develop dedicated techniques for handling the temporal bounds symbolically and efficiently.

Contribution. We propose a synthesis method for specifications expressed in a fragment of LTL which is a syntactic extension of `Safety LTL` with bounded temporal operators. The distinguishing characteristic of our method is a reduction to a dedicated game model, called *countdown-timer games* in which the temporal operators' bounds are treated symbolically via the introduction of *timers*. Further features of the translation are techniques for on-the-fly pruning of edges in the constructed game and reduction of the number of introduced timers. We present an abstraction-based method for solving the resulting games. We have developed a prototype implementation of our approach, and the experimental evaluation demonstrates that it is indeed capable of handling efficiently safety specifications with large bounds. We demonstrate that on a set of benchmarks featuring bounded temporal operators with large bounds, our technique outperforms state-of-the-art tools for $\mathrm{LTL}_{\mathsf{EBR}}$ and LTL synthesis.

Related Work. *The synthesis problem for `Safety LTL`* has attracted significant interest due to its algorithmic simplicity compared to general LTL synthesis [25]. For instance, the symbolic approach presented in [25] is shown to outperform the state-of-the-art LTL synthesis tools at the time. For $\text{LTL}_{\texttt{EBR}}$, [9] proposes a synthesis algorithm based on a fully symbolic translation to deterministic safety automata. A key difference between our approach and the above techniques is that our countdown-timer game construction does not expand upfront the bounded temporal operators, but treats them symbolically instead. Furthermore, the authors of [25] point out that for large `Safety LTL` formulas the construction of the deterministic safety automaton presents a performance bottleneck. Our safety game constriction makes use of pruning in order to alleviate this problem by eliminating on-the-fly parts of the game graph that need not be explored.

Parameterized temporal logics, such as PLTL [1] enable the specification of parametric lower and upper bounds on the satisfaction time of the "globally" operator and the wait time of "eventually". In the logic PROMPT-LTL [17], only eventualities are parameterized by upper bounds. The bounds of the temporal operators in these logics are unknown parameters, while in the case that we consider, the bounds are given integer constants. The goal of our work is to develop a synthesis method that treats constant bounds efficiently.

In the *real-time setting*, temporal logics that allow for limiting the time scope of temporal operators have been extensively studied. Notable logics are Metric Temporal Logic (MTL) [15], and its fragment Metric Interval Temporal Logic (MITL) [2]. Compared to the untimed setting, synthesis from real-time logic specifications poses additional challenges. Controller synthesis is undecidable for MTL [4], for MITL [5,11], and even for the safety fragment of MTL [5]. Decidability is regained by fixing the resources (clocks and guards) of the controller [5,12]. The key challenge stems from the fact that synthesis requires deterministic automata, and it is not generally possible to construct deterministic timed automata for MITL. To circumvent this problem, the assumption of bounded variability is commonly made. Under this assumption, [20] proposes a synthesis algorithm for bounded response properties, and a translation from MTL to deterministic timed automata is presented in [23]. With respect to tool support, sound but incomplete synthesis methods for fragments of MTL have been proposed in [6] and [18], and implemented in toolchains that employ UPPAAL-TIGA [3] for timed games solving. A tool for MTL controller synthesis via translation to alternating timed automata was presented in [14]. In the case when the real-time synthesis problem is given as a timed game and the specification is a state-based winning condition, the problem of computing a control strategy is decidable [21]. Efficient on-the-fly algorithms for timed games have been developed [7], and successfully implemented in UPPAAL-TIGA [3] and UPPAAL-STRATEGO [10].Since we are interested in discrete-time systems, we circumvent the additional challenges present in the dense-time setting by remaining the realm of discrete time and focusing on efficiently treating quantitative timing constraints there.

2 Preliminaries

Reactive Synthesis Let $\mathcal{I}$ be a finite set of uncontrollable environment *input Boolean propositions* and $\mathcal{O}$ be a finite set of controllable *output Boolean propositions*. A *reactive system* is a tuple (C, c_0, γ) where C is a set of *control states*, $c_0 \in C$ the *initial control state*, and $\gamma : C \times 2^{\mathcal{I}} \to C \times 2^{\mathcal{O}}$ is the *transition function*. A *specification* is a language $\mathcal{L} \subseteq \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^{\omega}$ of infinite words over $\mathcal{I} \cup \mathcal{O}$.

A system (C, c_0, γ) *realizes* a specification $\mathcal{L}$ if for all infinite sequences of environment inputs $i \in \left(2^{\mathcal{I}}\right)^{\omega}$ it yields an output sequence $o \in \left(2^{\mathcal{O}}\right)^{\omega}$ defined by $(c_{t+1}, o_t) = \gamma(c_t, i_t)$ for $t \in \mathbb{N}$, such that $i \cup o \in \mathcal{L}$. *Reactive synthesis* is the problem of finding a realizing implementation for a given specification.

Safety LTL with Bounded Liveness Operators We consider specifications expressed using temporal logic, more concretely, in a fragment of LTL [24], which we denote by *SafeLTL*$_B$. The fragment *SafeLTL*$_B$ is a syntactic extension of `Safety LTL` [25] and defined by the following grammar:

$$\varphi, \psi := ap \mid \neg ap \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \bigcirc[n]\varphi \mid \Diamond[n]\varphi \mid \varphi\,\mathcal{W}[n]\psi \mid \varphi\,\mathcal{W}\,\psi$$

for $ap \in \mathcal{I} \cup \mathcal{O}$ and $n \in \mathbb{N}$. *SafeLTL*$_B$ extends `Safety LTL` by bounded operators with bounds encoded in binary. While all bounded operators have equivalent `Safety LTL` formulas (e.g. $\Diamond[n]\varphi \equiv \bigvee_{i \in \{0 \ldots n\}} \bigcirc^{i} \varphi$) these have exponentially larger encoding. The constants $\top$ (true), $\bot$ (false), the "globally" operator $\Box$ and "bounded until" $\mathcal{U}[n]$ can be derived as $\top := a \vee \neg a$, $\bot := a \wedge \neg a$, $\Box\varphi := \varphi\,\mathcal{W}\,\bot$, $\Box[n]\varphi := \varphi\,\mathcal{W}[n]\bot$, and $\varphi\,\mathcal{U}[n]\psi := (\varphi\,\mathcal{W}[n]\psi) \wedge \Diamond[n]\psi$, respectively.

The satisfaction of a formula $\Phi \in$ *SafeLTL*$_B$ by infinite word $w = w_0 w_1 \ldots \in \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^{\omega}$ at time point $k \in \mathbb{N}$ is denoted as $w \vDash_k \Phi$ and is defined follows:

$$\begin{array}{llll}
w \vDash_k a & :\Leftrightarrow a \in w_k & w \vDash_k \neg a & :\Leftrightarrow a \notin w_k \\
w \vDash_k \varphi \wedge \psi & :\Leftrightarrow (w \vDash_k \varphi) \wedge (w \vDash_k \psi) & w \vDash_k \varphi \vee \psi & :\Leftrightarrow (w \vDash_k \varphi) \vee (w \vDash_k \psi) \\
w \vDash_k \Diamond[n]\varphi & :\Leftrightarrow \exists i \leq n.\ w \vDash_{k+i} \varphi & w \vDash_k \bigcirc[n]\varphi & :\Leftrightarrow w \vDash_{k+n} \varphi
\end{array}$$

$$\begin{array}{ll}
w \vDash_k \varphi\,\mathcal{W}[n]\psi & :\Leftrightarrow (\forall i \leq n. w \vDash_{k+i} \varphi) \vee (\exists j \leq n. w \vDash_{k+j} \psi \wedge \forall i < j. w \vDash_{k+i} \varphi) \\
w \vDash_k \varphi\,\mathcal{W}\,\psi & :\Leftrightarrow (\forall i. w \vDash_{k+i} \varphi) \vee (\exists j. w \vDash_{k+j} \psi \wedge \forall i < j. w \vDash_{k+i} \varphi).
\end{array}$$

The language of $\Phi \in$ *SafeLTL*$_B$ is defined as $\mathcal{L}(\Phi) := \{w \in \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^{\omega} \mid w \vDash_0 \Phi\}$.

Two-Player Safety Games The synthesis problem for temporal logic specifications can be solved by translating the specification into a two-player game between the system and the environment, and then solving the game to determine the winning player. If the system wins, an implementation can be extracted.

A *game structure* is a tuple $G = (S, S_0, \mathcal{I}, \mathcal{O}, \rho)$, where S is a *set of states*, $S_0 \subseteq S$ is a set of *initial states*, $\mathcal{I}$ and $\mathcal{O}$ are sets of propositions as defined earlier, and $\rho : S \times 2^{\mathcal{I}} \times 2^{\mathcal{O}} \to S$ is a *transition function*. A game on G is played by two players, the system and the environment. In a given state $s \in S$, the environment chooses some input $i \subseteq \mathcal{I}$, then the system chooses some output $o \subseteq \mathcal{O}$, and these choices determine the next state $s' := \rho(s, i, o)$. The game then continues from s'. The resulting infinite sequence $\pi = s_0, s_1, s_2, \ldots$ of states is called a *play*. Formally, a play is a sequence $\pi = s_0, s_1, s_2, \ldots \in S^{\omega}$ such that

$s_0 \in S_0$ and for every $t \in \mathbb{N}$, $s_{t+1} = \rho(s_t, i, o)$. A *system strategy* is a function $\sigma : S^+ \times 2^{\mathcal{I}} \to 2^{\mathcal{O}}$. An *environment strategy* is a function $\pi : S^+ \to 2^{\mathcal{I}}$. Given a state $s \in S$, a system strategy σ and an environment strategy π, we denote with $\mathit{Outcome}(s, \pi, \sigma)$ the unique play $s_0, s_1, s_2, \ldots$ such that $s_0 = s$, and for all $k \in \mathbb{N}$, $s_{k+1} = \rho(s_k, i_k, \sigma((s_0, s_1 \ldots, s_k), i_k))$, where $i_k = \pi((s_0, s_1 \ldots, s_k))$.

A *safety game* is a tuple (G, UNSAFE) where $\mathit{UNSAFE} \subseteq S$ are unsafe states. The system wins the safety game if it has a strategy σ such that for all environment strategies π, $s_0 \in S_0, k \in \mathbb{N}$, it holds that $\mathit{Outcome}(s_0, \pi, \sigma)_k \notin \mathit{UNSAFE}$. Such strategy is called a *winning strategy* for the system. Intuitively, the system has to avoid the unsafe states no matter what the environment does. The environment wins if it can enforce a visit to UNSAFE, i.e., when there exist environment strategy π and $s_0 \in S_0$ such that for every system strategy σ there exists $k \in \mathbb{N}$ such that $\mathit{Outcome}(s_0, \pi, \sigma)_k \in \mathit{UNSAFE}$.

3 $\mathit{SafeLTL}_B$ Synthesis with Countdown-Timer Games

***SafeLTL*$_B$ Synthesis** We consider the realizability and synthesis problems for the fragment $\mathit{SafeLTL}_B$. We focus on the challenge of handling efficiently specifications with large bounds in the bounded temporal operators, and propose a new synthesis method towards achieving this goal. The proposed approach proceeds in two stages. In the first stage, the given $\mathit{SafeLTL}_B$ formula is transformed into a kind of safety game, in which bounds are treated symbolically. We term these games *countdown-timer games*, introduced later in this section. The second stage of our synthesis algorithm is the solving of the generated countdown-timer game in order to determine the winning player and answer the realizability question. We propose in Section 5 a method that employs symbolic representation and approximations in order to efficiently solve such games in practice.

Countdown-Timer Games Intuitively, countdown-timer games are like safety games but with additional *countdown-timers*. Countdown-timers are discrete timers that always start with an assigned duration and are decremented by one with every transition in the game. Once a timer reaches zero it times out, and the transition relation of the countdown-timer game may depend on this information for determining the successor state. A countdown-timer can be reset to the duration associated with it. In addition, countdown-timers with the same duration can swap their values, which we will later use when generating timer-games to avoid unnecessary blowup in the number of timers.

Definition 1 (Countdown-Timer Games). *A countdown-timer game structure is a tuple $G_T = (\mathcal{T}, d, L, L_0, \mathcal{I}, \mathcal{O}, \delta)$ where $\mathcal{T}$ is a finite set of* countdown timers*, $d : \mathcal{T} \to \mathbb{N}$ associates a* duration *with each timer, L is a finite set of game locations, $L_0 \subseteq L$ is the set of initial locations, $\mathcal{I}$, $\mathcal{O}$ are finite sets of uncontrollable environment input propositions and controllable system propositions, respectively, and $\delta : L \times 2^{\mathcal{I}} \times 2^{\mathcal{O}} \times 2^{\mathcal{T}} \to L \times \mathcal{E}$ is the transition relation. $\mathcal{E} := \mathcal{T} \to (\mathcal{T} \cup \{\mathit{RESET}\})$ is the* set of effects *where for all $e \in \mathcal{E}$:*

1. *for all $t \in \mathcal{T}$ either $e(t) = \mathit{RESET}$, or $e(t) \in \mathcal{T}$ and $d(e(t)) = d(t)$ and,*

2. for $t_1, t_2 \in \mathcal{T}$ with $t_1 \neq t_2$ we have $e(t_1) \neq e(t_2)$ or $e(t_1) = e(t_2) = \mathit{RESET}$.
A countdown-timer game is a pair (G_T, UNSAFE_L) where $\mathit{UNSAFE}_L \subseteq L$ is a set of unsafe locations*.*

The effects $\mathcal{E}$ capture the resets and remapping of timers that can occur upon transitions. Condition (1) states that each timer is either reset or remapped to a timer with the same duration. Condition (2) requires the remapping to be injective, i.e. no two timers are mapped to the same timer. When timers are not reset and not remapped to other timers, they are simply mapped to themselves.

The semantics of a countdown-timer game is the safety game generated by explicitly expanding the possible valuations of the timers. Intuitively, each state of the game structure is a pair $s = (l, v)$ of a location $l \in L$ and a timer valuation v. Initially, each timer t is set to its associated duration $d(t)$. The transition relation updates the values of the timers by first decrementing them and then applying the effect e of the corresponding transition in G_T. The relevant transition in G_T is determined by the location l, the input and output sets i and o, and the set of timers whose value has become 0 after the decrementation.

Definition 2 (Countdown-Timer Games Semantics). *In the context of Definition 1, let $\mathcal{V} := \{v : \mathcal{T} \to \mathbb{N} \mid \forall t \in \mathcal{T}.\ v(t) \leq d(t)\}$ be the space of all possible timer valuations. Let $G = (L \times \mathcal{V}, L_0 \times \{\lambda t.d(t)\}, \mathcal{I}, \mathcal{O}, \rho)$ be a game structure where $\rho((l, v), i, o) := \mathit{trans}(l, \mathit{step}(v), i, o)$ with*

$$\mathit{step}(v) := \lambda t.\, \max\{0, v(t) - 1\}$$

$$\mathit{trans}(l, v, i, o) := \begin{cases} \left(l', \lambda t. \begin{cases} v(e(t)) & \text{if } e(t) \in \mathcal{T} \\ d(t) & \text{if } e(t) = \mathit{RESET} \end{cases} \right), \\ \text{where } (l', e) := \delta(l, i, o, \{t \in \mathcal{T} \mid v(t) = 0\}). \end{cases}$$

The semantics of the countdown-timer game (G_T, UNSAFE_L) is the safety game $(G, \mathit{UNSAFE}_L \times \mathcal{V})$. The system (environment) wins the countdown-timer game if and only if it wins the safety game representing its semantics.

4 Countdown-Timer Game Construction

We now present the first phase of our synthesis algorithm, namely the translation of a $\mathit{SafeLTL}_B$ formula to a countdown-timer game. Our construction is based on expansion rules. For example, the formula $\diamondsuit[50]a$ is equivalent to $a \vee \bigcirc \diamondsuit[49]a$. If a is true, then the whole formula is true. Otherwise, in the next step $\diamondsuit[49]a$ has to hold. Interpreted as a state of a safety game, $\diamondsuit[50]a$ has a transition to $\top$ on $a = \top$ and to $\diamondsuit[49]a$ on $a = \bot$. This can be repeated on $\diamondsuit[49]a$ and so on. Once we reach $\diamondsuit[0]a$ we expand it to $a \vee \bigcirc \bot$, and hence, $a = \bot$ leads to $\bot$ which is the unsafe state. This construction works for safety formulas, as rejection can be decided with a finite prefix. As we show later, generating a game structure in this way has the advantage that it can be pruned using information from the formula.

However, this explicit expansion yields a sequence of formulas that is linear in the bound, and hence, exponential in the description of the formula. Instead of explicit bounds, we use countdown-timers representing multiple values. In the above example, we do not generate all the expansions $\Diamond[50]a$, ..., $\Diamond[0]a$, but instead a timer t with duration 51 to represent all expansions from 50 to 0 in the single location $a \vee \Diamond[t]a$. If t times out, $\Diamond[t]$ has reached the end of the expansion and is transformed to $\bot$. Hence, instead of having $\Diamond[50]a$, ..., $\Diamond[0]a$, $\top$ and $\bot$ as states of a safety game we only have locations $a \vee \Diamond[t]a$, $\top$ and $\bot$ in a countdown-timer game. We now describe this construction formally.

4.1 Construction of a Countdown-Timer Game from *SafeLTL$_B$*

The locations of the generated countdown-timer games are $\mathit{SafeLTL}_B$ formulas with, additionally, timers as bounds of the temporal operators. We denote the set of these formulas as $\mathit{SafeLTL}_B^t$. Given a set of timers $\mathcal{T}$, the grammar of $\mathit{SafeLTL}_B^t$ is the grammar of $\mathit{SafeLTL}_B$ but in $\Diamond[n]$, $\bigcirc[n]$, and $\mathcal{W}[n]$ we have $n \in \mathbb{N} \cup \mathcal{T}$. For $\varphi \in \mathit{SafeLTL}_B^t$, $\mathit{Timers}(\varphi) \subseteq \mathcal{T}$ denotes all timers appearing in φ.

Game Structure Let Φ be a $\mathit{SafeLTL}_B$ formula over input propositions $\mathcal{I}$ and output propositions $\mathcal{O}$. We construct a countdown-timer game structure $(\mathcal{T}, d, L, L_0, \mathcal{I}, \mathcal{O}, \delta)$ as follows. The set of timers

$$\mathcal{T} := \{t_i^d \mid \bigcirc[d], \Diamond[d-1], \text{ or } \mathcal{W}[d-1] \text{ occurs in } \Phi, 0 \leq i \leq d\}$$

consists of timers t_i^d with index i and durations $d(t_i^d) := d$ for $0 \leq i \leq d$. The duration of a timer determines the bounds of the temporal operators in Φ for which it can be used, and the indices are used for distinguishing multiple timers of the same duration (introduced at different points of the expansion).

Let $L := \mathit{PositiveBooleanCombinations}(\mathit{cl}(\Phi))$ (i.e., built from $\mathit{cl}(\Phi)$ using $\wedge, \vee$) be the set of locations, where cl is the *closure* operator defined as:

$$\begin{array}{lll}
\mathit{cl}(l) & := \{l, \top, \bot\} & l \in \{ap, \neg ap\} \\
\mathit{cl}(\varphi \; o \; \psi) & := \mathit{cl}(\varphi) \cup \mathit{cl}(\psi) & o \in \{\wedge, \vee\} \\
\mathit{cl}(\bigcirc[n]\varphi) & := \mathit{cl}(\varphi) \cup \{\bigcirc[t_i^n]\varphi \mid 0 \leq i \leq n\} & \\
\mathit{cl}(\Diamond[n]\varphi) & := \mathit{cl}(\varphi) \cup \{\Diamond[t_i^{n+1}]\varphi \mid 0 \leq i \leq n+1\} & \\
\mathit{cl}(\varphi \, \mathcal{W}[n]\psi) & := \mathit{cl}(\varphi) \cup \mathit{cl}(\psi) \cup \{\varphi \, \mathcal{W}[t_i^{n+1}]\psi \mid 0 \leq i \leq n+1\} & \\
\mathit{cl}(\varphi \, \mathcal{W} \, \psi) & := \mathit{cl}(\varphi) \cup \mathit{cl}(\psi) \cup \{\varphi \, \mathcal{W} \, \psi\}. &
\end{array}$$

Intuitively, the closure contains all possible temporal-operator sub-formulas and literals that can appear during expansion. The locations L then represent the expanded formulas, which, intuitively, correspond to the current obligations of the system. Thus, the initial location will correspond to obligation Φ. Note that $L \subseteq \mathit{SafeLTL}_B^t$. We apply simplifications to the generated formulas to ensure that L is finite. Since by definition $\mathit{cl}(\Phi)$ is finite, we can ensure that $|L| \leq 2^{|\mathit{cl}(\Phi)|}$.

In the construction of the initial location and the transition function we use two helper functions, $\mathit{introExp} : \mathit{SafeLTL}_B^t \to \mathit{SafeLTL}_B^t$, which performs expansion and introduces new timers, and $\mathit{opt} : \mathit{SafeLTL}_B^t \to L$, which performs simplifications that ensure that L is finite. We let $L_0 := \{\mathit{opt}(\mathit{introExp}(\Phi))\}$ and

$$\delta(\varphi, i, o, T) := (\mathit{opt}(\mathit{introExp}(\psi)), e) \text{ where } (e, \psi) := \mathit{squeeze}(\mathit{to}(T, \mathit{tree}(\varphi, i, o))).$$

Here, we use the additional functions $\mathit{tree} : \mathit{SafeLTL}_B^t \times 2^{\mathcal{I}} \times 2^{\mathcal{O}} \to \mathit{SafeLTL}_B^t$, which performs the input and outputs choices, $\mathit{to} : 2^{\mathcal{T}} \times \mathit{SafeLTL}_B^t \to \mathit{SafeLTL}_B^t$, which handles time-outs, and $\mathit{squeeze} : \mathit{SafeLTL}_B^t \to \mathcal{E} \times \mathit{SafeLTL}_B^t$, which determines remapping and reset of timers. Below, we describe these functions in detail.

Remark: Note that for $\bigcirc[b]$ we use timers of duration b, while for $\diamondsuit[b]$ and $\mathcal{W}[b]$ we use timers of duration $b+1$. The reason for this is that for the latter we consider the last step as part of the timing as this simplifies the game structure.

Before describing the functions, we illustrate them on a simple example.

Example 1. Let $\mathcal{I} = \{r\}$, $\mathcal{O} = \{g\}$, and consider the $\mathit{SafeLTL}_B$ formula $\Phi = (\square[100]\neg g) \wedge \bigcirc[10](r \to \diamondsuit[100]g)$. Φ states that the system should not give a grant during the first 100 steps, and, if at step 10 there is a request, then a grant should be given within the following 100 steps. We show how to construct the initial location and some of the transitions in a countdown-timer game for Φ.

Initial state $\boldsymbol{\varphi_0}$ $= \mathit{opt}(\mathit{introExp}(\Phi))$

The initial state is computed from Φ by expanding the formula and introducing any necessary timers. This is done by the function *introExp*. The subformula $\square[100]\neg g$ expands to $\neg g \wedge \square[t_0^{101}]\neg g$, reflecting the semantics of the operator $\square[100]$. This introduces the timer t_0^{101} with duration 101 and index 0. The subformula $\bigcirc[10](r \to \diamondsuit[100]g)$ expands to $\bigcirc[t_0^{10}](r \to \diamondsuit[100]g)$, which introduces the timer t_0^{10} for $\bigcirc[10]$. The durations 101 and 10 of the timers correspond to the respective bounds in $\square[100]$ and $\bigcirc[10]$, and the index 0 is the smallest index of a currently unused timer of the respective duration. No timer is introduced at this step for $\diamondsuit[100]$ as it is guarded by a $\bigcirc$ operator. Thus, the initial state is the expanded formula $\varphi_0 = \neg g \wedge (\square[t_0^{101}]\neg g) \wedge \bigcirc[t_0^{10}](r \to \diamondsuit[100]g)$.

Determining transition $\boldsymbol{\delta(\varphi_0, \emptyset, \{g\}, \emptyset)}$ $= (\varphi_1, e_1)$

We apply $\mathit{tree}(\varphi_0, \emptyset, \{g\})$ which computes the effect of the input $\emptyset$ and output $\{g\}$ on the formula in the current step, and thus substitutes g with $\top$ in φ_0. This results in $\mathit{tree}(\varphi_0, \emptyset, \{g\}) = \bot$, meaning that this transition leads to location $\bot$.

Determining transition $\boldsymbol{\delta(\varphi_0, \emptyset, \emptyset, \{t_0^{10}\})}$ $= (\varphi_2, e_2)$

Again, we first compute $\mathit{tree}(\varphi_0, \emptyset, \emptyset) = (\square[t_0^{101}]\neg g) \wedge \bigcirc[t_0^{10}](r \to \diamondsuit[100]g)$, which now substitutes $\bot$ for g. To the result we apply the function *to* that handles time-outs, here $\{t_0^{10}\}$, which means that the timer t_0^{10} times out at the current step. As a result, the subformula $\bigcirc[t_0^{10}](r \to \diamondsuit[100]g)$ is replaced by $r \to \diamondsuit[100]g$, meaning that the formula $r \to \diamondsuit[100]g$ becomes part of the obligation at the next step, since the timer t_0^{10} has run out. Thus, we obtain $\mathit{to}(\{t_0^{10}\}, (\square[t_0^{101}]\neg g) \wedge \bigcirc[t_0^{10}](r \to \diamondsuit[100]g)) = (\square[t_0^{101}]\neg g) \wedge (r \to \diamondsuit[100]g)$. After that, we apply function *squeeze* that takes care of timers that might have become unused upon time-out. This is reflected in the effect e_2 that resets all timers that do not appear in the current formula. Thus, in e_2 the timer t_0^{10} that just timed out is mapped to RESET, and the timer t_0^{101} that is still present is mapped to itself. The final step is to apply function *introExp* that performs expansion on the current formula and introduces any new timers that might be needed. The subformula $\square[t_0^{101}]\neg g$ expands to $\neg g \wedge \square[t_0^{101}]\neg g$. The subformula $r \to \diamondsuit[100]g$ expands to $r \to (g \vee \diamondsuit[t_1^{101}]g)$, which introduces the timer t_1^{101} for $\diamondsuit[100]$. Note that since the formula already contains the timer t_0^{101} of duration 101, the newly introduced timer t_1^{101} has index 1. The functions *to* and *squeeze* ensure that the order between the indices of timers of the same duration represents the order in which these timers will time out. After computing $\mathit{introExp}((\Box[t_0^{101}]\neg g) \wedge (r \rightarrow \Diamond[100]g))$ we obtain $\varphi_2 = \neg g \wedge (\Box[t_0^{101}]\neg g) \wedge (r \rightarrow (g \vee \Diamond[t_1^{101}]g))$.

Construction We construct the sets of locations, timers, and transitions, by exploring the reachable parts of L from L_0. We describe several pruning mechanisms that we use in order to maintain the set of reachable locations small.

Construction Invariants. To ensure correctness and keep the game generation efficient, we maintain the following invariants for each reachable location:

1. For every reachable location φ we have (1.a) all literals and bounded operators not guarded by a "next" operator appear on the Boolean top-level, and (1.b) all bounded operators at the top-level are instantiated with a timer.
2. For every duration d, the values of the timers are ordered by index, i.e. $t_0^d < t_1^d < \dots t_j^d = \dots t_d^d = d$. The order is strict for timers whose value is not d.
3. In location φ, for any d and $i > 0$, if $t_i^d \in \mathit{Timers}(\varphi)$, then $t_{i-1}^d \in \mathit{Timers}(\varphi)$.

Invariant (1) is needed for correctness, and for ensuring that all literals that are relevant in the current step are considered, and that all relevant bounded operators are tracked by timers. Invariant (2) ensures that we never need more than the available d timers. This holds since the timers are strictly ordered when running, and once we would introduce t_{d+1}^d, t_0^d would have timed out. Furthermore, ordering the timers reduces the possible combinations of time-outs. Invariant (3) prevents having unused timers that are between used ones according to the above order, thus reducing the possible combinations of equivalent locations.

Function *tree*: *Selection of Inputs and Outputs.* The function $\mathit{tree}(\varphi, i, o)$ computes the effect of the input i and output o on the formula in the current step. With invariant (1) it suffices to consider literals on the Boolean top-level, i.e. literals that are not sub-formulas of a temporal operator. When assigning the literals in φ according to i and o, we prune and select some "obvious choices" which can immediately be decided, using the fact that we are generating a game. This pruning is an important part of our approach, as in practice it can prune a significant portion of the possible locations. Function *tree* applies recursively a set of rules. We now describe these rules in the order in which they are applied in each recursion step. Figure 1 provides a formal description.

1. With top-level disjunct c that is output literal, the system wins by making the formula $\top$. The opposite choice for the system can be safely pruned.
2. With top-level conjunct u that is input literal, the environment wins by making the formula $\bot$. The opposite choice can be safely pruned.
3. If an output proposition appears either with only positive or with only negative polarity, it suffices for the system to pick the literal with the respective polarity, as for the other choice the generated formula is subsumed.
4. If an input proposition appears either with only positive polarity or only negative polarity, it suffices to consider the case where the environment picks the negated literal, as this case is strictly more difficult to realize (i.e. one formula implies the other) and every strategy for this case works also for the other.

$$\mathit{tree}(c \vee \psi, i, o) := [\![c \in o]\!] \tag{1}$$

$$\mathit{tree}(u \wedge \psi, i, o) := \bot \tag{2}$$

$$\mathit{tree}(\psi, i, o) := \begin{cases} \mathit{tree}(\psi[c/\top]_T) & \text{if } c \in o \\ \bot & \text{if } c \notin o \end{cases} \qquad c \in \mathit{ActL}(\psi), \neg c \notin \mathit{ActL}(\psi) \tag{3}$$

$$\mathit{tree}(\psi, i, o) := \mathit{tree}(\psi[u/\bot]_T) \qquad u \in \mathit{ActL}(\psi), \neg u \notin \mathit{ActL}(\psi) \tag{4}$$

$$\mathit{tree}(\psi, i, o) := \psi[u/[\![u \in i]\!]]_T \qquad u, \neg u \in \mathit{ActL}(\psi) \tag{5}$$

$$\mathit{tree}(\psi, i, o) := \psi[c/[\![c \in o]\!]]_T \qquad c, \neg c \in \mathit{ActL}(\psi) \tag{6}$$

Figure 1: Let $u \in \mathcal{I}$ and $c \in \mathcal{O}$. For simplicity of the presentation we leave out the commutative and associative cases and negative literals. $\mathit{ActL}(\psi)$ denotes the set of literals appearing in the Boolean top-level of ψ. The formula $\psi[ap/v]_T$ is obtained from ψ by replacing ap by $v \in \{\top, \bot\}$ for all occurrences of ap at the Boolean top-level, but only there. After each replacement we simplify the formula by doing constant folding. $[\![x \in X]\!]$ is $\top$ if $x \in X$ and $\bot$ if $x \notin X$.

5. If no "early decision" or "worst case-decision" can be made, we apply the environment choice, as the environment moves first in the game.
6. If no environment choices are left, we generate the branching for the system.

Function *to*: *Handling Time-out.* A consequence of invariant (2) is that only timers with index 0, i.e., of the form t_0^d, can time out since the timers are ordered. In addition, timers that do not appear inside a formula should not time out (this is enforced by *squeeze*) as we show later. Note that this does not apply to timers with duration 1 as these time out immediately. We direct impossible time-outs to $\top$ since they do not occur. Hence, $\mathit{to}(T, \varphi) := \top$ if for some $t_i^d \in T$ we have that $i \neq 0$, or $d > 1$ and $t_i^d \notin \mathit{Timers}(\varphi)$. Otherwise, $\mathit{to}(T, \varphi)$ is defined by applying the following transformations on all subformulas of φ and timing out timers $t \in T$: We transform $\Diamond[t]\psi \rightsquigarrow \bot$, $\bigcirc[t]\psi \rightsquigarrow \psi$, and $\phi\, \mathcal{W}[t]\psi \rightsquigarrow \top$. After applying *to* we do constant folding as parts of the formula may become irrelevant.

Function *squeeze*: *Determining remapping and reset of timers.* When applying the functions *tree* and *to* some timers might become unused. Hence, we have to ensure that invariant (3) holds and, as stated in the previous paragraph, reset all timers that do not appear in the formula. We define $\mathit{squeeze}(\varphi) := (e, \psi)$ as follows: For each duration d, let $t_{i_j}^d \in \mathit{Timers}(\varphi)$ with indices $i_0 < i_1 < i_2 < \dots$ be the remaining timers with sorted indices i_j. Then set $e(t_j^d) := t_{i_j}^d$ if i_j exists and $e(t_j^d) := \mathit{RESET}$ otherwise. ψ is obtained by replacing the timers $t_{i_j}^d$ by t_j^d.

Function *introExp*: *Expansion and Timer Introduction.* The function *introExp* performs the formula expansion and introduces new timers if necessary. The expansion guarantees that invariant (1) holds afterwards. When introducing new timers, invariant (2) and invariant (3) have also to be maintained. This is achieved by assigning for each bound b with associated duration d, the timer with the next unused index, i.e. $t_j^d \notin \mathit{Timers}(\varphi)$ where $t_0^d, \dots, t_{j-1}^0 \in \mathit{Timers}(\varphi)$. Let $I(d) := \max\{i \mid t_i^d \in \mathit{Timers}(\varphi)\} + 1$ be the next unused index. In addition, as timers t_i^d with $i > d$ do not exist by invariant (2), expansions generating them are redirected to $\top$. Hence, we define $\mathit{introExp}(\varphi) := \mathit{rd}(\mathit{iE}_I(\varphi))$ where $\mathit{rd}(\varphi) := \top$ if for some $i > d$ we have $t_i^d \in \mathit{Timers}(\varphi)$, and $\mathit{rd}(\varphi) = \varphi$ otherwise. The function iE_I performing the expansion is defined by

$$\begin{array}{llll}
\mathit{iE}_I(l) & := l & \mathit{iE}_I(\varphi\ o\ \psi) & := \mathit{iE}_I(\varphi)\ o\ \mathit{iE}_I(\psi) \\
\mathit{iE}_I(\Diamond[n]\varphi) & := \mathit{iE}_I(\varphi) \vee \Diamond[t^{n+1}_{I(n+1)}]\varphi & \mathit{iE}_I(\Diamond[t]\varphi) & := \mathit{iE}_I(\varphi) \vee \Diamond[t]\varphi \\
\mathit{iE}_I(\bigcirc[n]\varphi) & := \bigcirc[t^{n}_{I(n)}]\varphi & \mathit{iE}_I(\bigcirc[t]\varphi) & := \bigcirc[t]\varphi \\
\mathit{iE}_I(\varphi\,\mathcal{W}[n]\psi) & := \mathit{iE}_I(\psi) \vee \mathit{iE}_I(\varphi) \wedge & \mathit{iE}_I(\varphi\,\mathcal{W}[t]\psi) & := \mathit{iE}_I(\psi) \vee \mathit{iE}_I(\varphi) \\
 & \quad (\varphi\,\mathcal{W}[t^{n+1}_{I(n+1)}]\psi) & & \quad \wedge(\varphi\,\mathcal{W}[t]\psi) \\
\mathit{iE}_I(\varphi\,\mathcal{W}\,\psi) & := \mathit{iE}_I(\psi) \vee \mathit{iE}_I(\varphi) \wedge (\varphi\,\mathcal{W}\,\psi) & &
\end{array}$$

where $l \in \{ap, \neg ap\}$, $o \in \{\wedge, \vee\}$, $n \in \mathbb{N}$ and $t \in \mathcal{T}$.

Function *opt*: *Formula Simplification.* The function *opt* ensures that the constructed set of locations L is finite, by simplifying the formulas in order to avoid introducing infinitely many logically equivalent formulas. Since we must maintain the invariants, the simplification does not guarantee uniqueness modulo equivalence. Nevertheless, it ensures finiteness of L and performs optimizations.

Definition of *UNSAFE* and Correctness To complete the construction of the countdown-timer game, we define the set of unsafe locations as $\mathit{UNSAFE}_L = \{\bot\}$. The proof of the correctness theorem below is given in the full version [13].

Theorem 1. *Let $\Phi \in \mathit{SafeLTL}_B$ and G be the countdown-timer game structure constructed from Φ as described above. Then there exists a system realizing $\mathcal{L}(\Phi)$ if and only if the system wins in the countdown-timer game (G, UNSAFE_L).*

We augment the construction with several extensions to improve its efficiency and expand its scope. For instance, we combine explicit expansion with timer-based implicit expansion, which allows us to handle directly operators like single $\bigcirc$. We also use approximation to handle simple assumptions of the form $\Box\psi$ where ψ is fully bounded, i.e., without $\mathcal{W}$. Details can be found in the full version [13].

5 Solving Countdown-Timer Games

We now describe the second phase of our synthesis algorithm, namely the solving of the countdown-timer game generated from the $\mathit{SafeLTL}_B$ specification. In a countdown-timer game, the durations of the timers, which correspond to the bounds of the temporal operators in the specification, are encoded in binary. Hence, the set $\mathcal{V}$ of timer valuations and thus also the safety game defined in Section 3 grow exponentially in the size of the countdown-timer game. Since our goal is to efficiently solve countdown-timer games with large durations, explicitly constructing and solving the semantic safety game is not desired. We note, however, that in the worst case it is not possible to avoid this blowup. This is stated in the next theorem, the proof of which is given in the full version [13].

Theorem 2. *Solving countdown-timer games is* `EXPTIME`*-complete.*

This means that solving countdown-timer games efficiently requires an approach that manipulates sets of timer valuations symbolically, in order to avoid, if possible, explicit enumeration. We propose a symbolic algorithm for solving countdown-timer games that additionally employs an iteratively refined approximation. The method is applicable to generic symbolic representations of the set of timer valuations. We present an instantiation of the method with a representation composed of intervals of timer values and partial orders on timers.

Symbolic Game Solving The standard way to solve a safety game is to compute the set of states from which the environment can enforce reaching an unsafe state, and check if it intersects with the set of initial states. If this is the case, then the environment wins the game, and otherwise the system wins.

For a game (G, UNSAFE) with $G = (S, S_0, \mathcal{I}, \mathcal{O}, \rho)$, the set of states from which the environment can enforce reaching *UNSAFE* is called *environment attractor* and is defined as $\mathit{AttrE}_G(\mathit{UNSAFE}) = \{s \in S \mid \exists \pi : \text{env. strategy}. \forall \sigma : \text{sys. strategy}. \exists k \in \mathbb{N}.\ \mathit{Outcome}(s, \pi, \sigma)_k \in \mathit{UNSAFE}\}$. The environment wins the safety game if and only if $\mathit{AttrE}_G(\mathit{UNSAFE}) \cap S_0 \neq \emptyset$.

We solve the countdown-timer game by computing a symbolic representation of the attractor of the environment player to the unsafe locations. We assume a symbolic representation *Rep* of the space of timer valuations $2^{\mathcal{V}}$. For each $R \in \mathit{Rep}$ we denote with $[\![R]\!] \subseteq \mathcal{V}$ the subset of $\mathcal{V}$ represented by R. We represent subsets of the state space $L \times \mathcal{V}$ of the semantic safety game using functions from $L \to \mathit{Rep}$ where $U \in (L \to \mathit{Rep})$ represents $\{(l, v) \mid v \in [\![U(l)]\!]\}$.

The symbolic enforceable predecessor for the environment $\mathit{CPreE}_{symb} : (L \to \mathit{Rep}) \to (L \to \mathit{Rep})$ is defined as follows. For $U \in (L \to \mathit{Rep})$, we let

$$\mathit{CPreE}_{symb}(U) := \lambda l. \bigcup_{i \subseteq \mathcal{I}} \bigcap_{o \subseteq \mathcal{O}} \bigcup_{T \subseteq \mathcal{T}} \mathit{symTrans}(\delta(l, i, o, T), T, U), \text{ where}$$

$$\mathit{symTrans}((l', e), T, U) := \mathit{inc}(\mathit{effTO}(T, \mathit{remap}(e, \mathit{effReset}(e, U(l')))))$$

is the symbolic backward application of transition $\delta(l, i, o, T)$ to the target set $[\![U(l')]\!]$. The operations that *symTrans* requires, from last to first, are as follows.

- $\mathit{inc} : \mathit{Rep} \to \mathit{Rep}$ performs the backward increment of the timers, formally, $[\![\mathit{inc}(R)]\!] = \{\lambda t.\ v(t) + 1 \in \mathcal{V} \mid v \in [\![R]\!]\}$.
- $\mathit{effTO} : 2^{\mathcal{T}} \times \mathit{Rep} \to \mathit{Rep}$ models the effect of time-outs: $[\![\mathit{effTO}(T, R)]\!] = \{v \in [\![R]\!] \mid \forall t \in \mathcal{T}.(t \in T \to v(t) = 0) \wedge (t \notin T \to v(t) \in [1, d(t)])\}$.
- $\mathit{remap} : \mathcal{E} \times \mathit{Rep} \to \mathit{Rep}$ models the effect of remapping: $[\![\mathit{remap}(e, R)]\!] = \{v \in \mathcal{V} \mid \exists v' \in [\![R]\!]. \forall t \in \mathcal{T} \text{ s.t. } e^{-1}(t) \text{ is defined. } v(t) = v'(e^{-1}(t))\}$.
- $\mathit{effReset} : \mathcal{E} \times \mathit{Rep} \to \mathit{Rep}$ models the effect of timer resets: $[\![\mathit{effReset}(e, R)]\!] = \{v \in [\![R]\!] \mid \forall t \in \mathcal{T}.e(t) = \mathit{RESET} \to v(t) = d(t)\}$. Note that $e^{-1}(t)$, the timer mapped to t by effect e is unique, since the effect is injective for values different from *RESET*, and can thus be inverted if defined.

We also require that we can preform set operations $\cup$, $\cap$, and equality checking between elements of *Rep*, in order to perform the computation.

We employ the symbolic enforceable predecessor operator CPreE_{symb} to compute a symbolic representation of the environment attractor AttrE_{symb} as follows. We set $\mathit{AttrE}^0_{symb} := (\lambda l.\text{ if } l \in \mathit{UNSAFE}_L \text{ then } \mathcal{V} \text{ esle } \emptyset)$, and then for $n \in \mathbb{N}$ we let $\mathit{AttrE}^{n+1}_{symb} := \mathit{AttrE}^n_{symb} \cup \mathit{CPreE}_{symb}(\mathit{AttrE}^n_{symb})$.

Proposition 1. *If (G_T, UNSAFE_L) is a countdown-timer game with $G_T = (\mathcal{T}, d, L, L_0, \mathcal{I}, \mathcal{O}, \delta)$ and the safety game $(G, \mathit{UNSAFE}_L \times \mathcal{V})$ with $G = (L \times \mathcal{V}, L_0 \times \{\lambda t.d(t)\}, \mathcal{I}, \mathcal{O}, \rho)$ is its semantics, then for the symbolic attractor computed above it holds $[\![\mathit{AttrE}_{symb}(l)]\!] = \{v \in \mathcal{V} \mid (l, v) \in \mathit{AttrE}_G\}$ for every $l \in L$.*

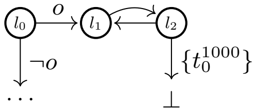
Approximation of Timer Valuations As the symbolically represented state-space described above might still lead to exploring a large number of sets, we perform an over- and under-approximation of the attractor of explored states.

We use a *threshold* $k \in \mathbb{N}$ to control the precision of the abstraction. Intuitively, when approximating for $t \in \mathcal{T}$ we would like to treat exactly timer values at the "border", i.e. timer values in $[0, k]$ and $[d(t) - k, d(t)]$, since these matter for timeouts and resets. Our approximations $\mathit{over} : \mathit{Rep} \to \mathit{Rep}$ and $\mathit{under} : \mathit{Rep} \to \mathit{Rep}$ treat the intermediate values $[k, d(t) - k]$ like a single value-block. The over-approximation $\mathit{over}(R)$ adds all intermediate values if one value from R is inside $[k, d(t) - k]$ and the under-approximation $\mathit{under}(R)$ removes all intermediate values if one value from R is not inside. Formally:

$$\begin{aligned}
&\mathit{approx}_k(t, I) := (I \cap [k, d(t) - k] \neq \emptyset) \wedge ([k, d(t) - k] \not\subseteq I)\\
&[\![\mathit{over}(R)]\!] \quad := \left\{ \lambda t. \begin{cases} v(t) \cup [k, d(t) - k] & \text{if } \mathit{approx}_k(t, v(t)) \\ v(t) & \text{otherwise} \end{cases} \;\middle|\; v \in [\![R]\!] \right\}\\
&[\![\mathit{under}(R)]\!] \quad := \left\{ \lambda t. \begin{cases} v(t) \setminus [k, d(t) - k] & \text{if } \mathit{approx}_k(t, v(t)) \\ v(t) & \text{otherwise} \end{cases} \;\middle|\; v \in [\![R]\!] \right\}
\end{aligned}$$

The attractor computation is now done as follows: We start with $k := 1$. For the current k we compute the environment attractor once using under- and once using over-approximation at each symbolic state in the computation. If the environment wins in the under-approximation, it wins the concrete game. If the system wins in the over-approximation, it wins the concrete game. If neither holds, we set $k := 2 \cdot k$ and repeat. This always terminates since for $k > d(t)/2$ the approximations become exact, and hence, one player wins for sure.

Example 2. Consider a countdown-timer game, some transitions of which are depicted in Fig. 2a. From the depicted transitions, only the transition from l_2 to



(a) Countdown-timer game, $\mathit{UNSAFE}_L = \{\bot\}$.

	0	1	2	3	4	...	7
l_1	$\emptyset$	$\emptyset$	$\{1\}$	$\{1\}$	$\{1\}, [3, 997]$	...	$\{1\}, [3, 997], \{999\}$
l_2	$\emptyset$	$\{0\}$	$\{0\}$	$\{0\}, \{2\}$	$\{0\}, \{2\}$	...	$\{0\}, [2, 998], \{1000\}$

(b) Sets during approximate attractor computation.

Figure 2: Example demonstrating the effect of approximation of timer valuations.

$\bot$ has a non-empty time-out set, $\{t_0^{1000}\}$. Since the timer t_0^{1000} has duration 1000, computing $AttrE_{symb}$ for the locations l_1 and l_2 precisely would require 1000 iterations. Employing over-approximation with threshold $k = 3$, on the other hand, reaches a fixed point in 7 iterations, as shown in Fig. 2b. This is helpful in cases like the one in the game in Fig. 2a, where the choice of transition in location l_0 is controlled by the system (via the output o). Here, the overapproximation allows the solving algorithm to quickly determine that the choice of transition to l_1 is loosing, while the system can win via the alternative transition.

Symbolic Representation using Boxes As a symbolic domain we chose an interval representation augmented with partial orders over timers $Rep := 2^{PartialOrder(\mathcal{T}) \times 2^{Rec}}$ where $Rec := \{\ i \in (\mathcal{T} \to \mathbb{N} \times \mathbb{N}) \mid \forall t \in \mathcal{T}, (a,b) = i(t).0 \leq a \leq b \leq d(t)\}$ are the intervals in the form of a hyper-cube. Intuitively, we have a set of partial-orders and for each of them we have a set of hyper-cubes. Formally:

$$\llbracket R \rrbracket := \bigcup_{(p,C) \in R} \left(\{v \in \mathcal{V} \mid \forall (t_1 \sim t_2) \in p : v(t_1) \sim v(t_2)\} \cap \bigcup_{r \in C} \lambda t.[r(t)_1, r(t)_2] \right)$$

where $r(t)_i$ is the i-th projection of $r(t)$. It remains to define the necessary operations: *inc*, *effReset*, *effTO*, and *remap* are mostly straightforward according to their definition, as they can be performed by modifying and inspecting all intervals individually or just reordering timers. Additionally, *effReset* uses the partial order to derive bounds on timers that are in relation with a timer that is reset. *effTO* refines the partial order, since on time-out T, all timers in T are smaller than $\mathcal{T} \backslash T$. Also the approximations can be performed point-wise on the intervals, as an approximate interval is again an interval.

We chose this domain since it is simple, and, at the same time, due to the use of partial orders, well suited for the type of problem we are solving. Our solving algorithm is generic and can accommodate other, more sophisticated domains.

6 Evaluation

We implemented[1] and evaluated our approach. We compare our prototype implementation to `ebr-ltl-synth` introduced in [9] which performs synthesis for LTL$_{\mathsf{EBR}}$. We also compare to the state-of-the-art LTL synthesis tool `strix` version 21.0.0 [19, 22]. In the following, we present the benchmarks we used, the experiments, and the results. We ran all experiments on an Intel Core i7-1165G7 processor with 16GB RAM and a single core available. All times are wall-clock times. A detailed description of the benchmarks is given in the full version [13].

Bounded Response Benchmarks In our first set of experiments we evaluate the tools on LTL$_{\mathsf{EBR}}$ formulas from [9], and on 23 SYNTCOMP 2021 benchmarks[2] that fall into LTL$_{\mathsf{EBR}}$ and are used for a similar comparison in [9]. Figure 3 and

[1] Available at: `https://github.com/phheim/lisynt`

[2] `https://github.com/SYNTCOMP/benchmarks`

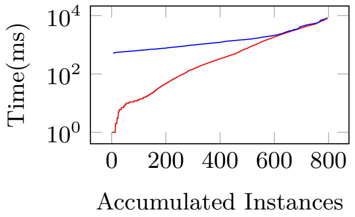


Figure 3: Execution times in milliseconds on the benchmarks [9].

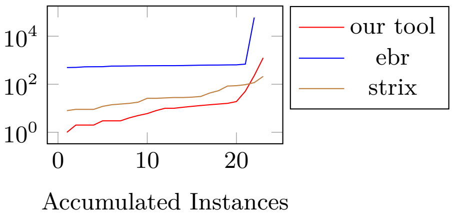


Figure 4: Execution times in milliseconds on the LTL$_{\mathsf{EBR}}$ SYNTCOMP benchmarks.

Figure 4 show the runtimes with a time-out of one minute, respectively. Unfortunately, for roughly half of the benchmarks from [9] `strix` did not accept the input formula for being too long, since the bounded operators must be expanded explicitly upon input. We therefore left `strix` out for this comparison. Figure 3 shows that on the benchmarks from [9] both our implementation and `ebr-ltl-synth` have roughly the same runtime, ignoring different startup times. Figure 4 shows that on the selected SYNTCOMP benchmarks all three tools are comparable.

These experiments evaluate our implementation on relevant benchmarks that are partially not designed in the spirit of the problem that our approach targets. The results show that our implementation is comparable to existing tools.

Adaption of Real-Time Benchmarks In our second set of experiments, we took MTL synthesis problems from [14] and adapted them to $SafeLTL_B$ formulas. The

Name	$\|L\|$	$\|\mathcal{T}\|$	τ_{Gen}	k	Win.	τ_Σ	$\tau_{\mathtt{strix}}$	Name	$\|L\|$	$\|\mathcal{T}\|$	τ_{Gen}	k	Win.	τ_Σ	$\tau_{\mathtt{strix}}$
Clean(1)	8	2	0.01	1	S	**0.01**	3.56	$Clean_H$(1)	3	2	0.02	512	E	**0.07**	1.61
Clean(2)	16	3	0.02	1	S	**0.03**	7.99	$Clean_H$(2)	3	2	0.02	512	E	**0.07**	2.63
Clean(3)	41	4	0.06	8	S	**0.33**	21.4	$Clean_H$(3)	3	2	0.02	512	E	**0.07**	4.99
Clean(4)	123	5	0.22	8	S	**1.45**	97.3	$Clean_H$(4)	3	2	0.02	512	E	**0.07**	5.64
$Clean_C$(1)	10	4	0.03	1	S	**0.05**	189	$Clean_N$(1)	23	4	0.07	1	S	**0.12**	TO
$Clean_C$(2)	22	5	0.08	16	S	**617**	TO	$Clean_N$(2)	32	4	0.10	1	S	**0.27**	TO
$Clean_C$(3)	61	6	0.32	-	-	TO	TO	$Clean_N$(3)	48	4	0.15	8	S	**7.47**	TO
$Clean_C$(4)	205	7	1.30	-	-	TO	TO	$Clean_N$(4)	75	4	0.26	8	S	**13.7**	TO
Coffee(1)	14	4	0.03	1	S	**0.04**	TO	$Coffee_C$(1)	46	6	0.16	1	S	**0.88**	F
Coffee(2)	44	5	0.12	2	S	**0.33**	TO	$Coffee_C$(2)	151	7	0.59	1	S	**5.51**	F
Coffee(3)	175	6	0.55	2	S	**3.53**	TO	$Coffee_C$(3)	613	8	2.73	1	S	**62.9**	F
Coffee(4)	418	7	1.34	2	S	**10.2**	TO	$Coffee_C$(4)	1634	9	6.82	1	S	**191**	F
conv-belt	9	3	0.01	1	S	**0.02**	F	rail(4,8)	647	7	2.53	1	S	**3.96**	TO
robo-cam	22	5	0.04	1	S	**0.19**	F	rail(8,8)	647	7	2.60	1	S	**4.03**	TO
rail(2,2)	647	6	2.60	1	S	**3.93**	TO	rail(1,1,1)	3111	7	27.8	-	-	TO	TO
rail(2,4)	647	6	2.58	1	S	**4.05**	TO	rail(2,1,1)	9179	9	89.1	1	S	**220**	TO
rail(2,8)	647	6	2.62	1	S	**3.97**	TO	rail(2,2,2)	9179	9	93.7	1	S	**225**	TO
rail(4,4)	647	7	2.67	1	S	**4.10**	TO								

Table 1: Results on the office-robot and adapted real-time benchmarks. $|L|$ and $|\mathcal{T}|$ are the numbers of locations and timers in the generated countdown-timer game. τ_{Gen} is the runtime of the game generation in seconds. k is the approximation threshold on which the solving terminated. Win. shows whether the system (S) or the environment (E) wins. τ_Σ is the total runtime including the game generation and solving, where TO means a time-out after 15 minutes. $\tau_{\mathtt{strix}}$ is the runtime of `strix`. For some benchmarks `strix` rejects the input for being too long (F) which is due to expanding the bounded operators when using `strix`.

benchmarks include a conveyor belt (conv-belt), a robot camera (robo-cam), and several parametrized instances of a multiple railroad-crossings controller (rail). We discretized the real-time bounds. The benchmarks use up to 19 propositions and 16 bounded operators, and bounds between 60 and 4000. Detailed results can be found in Table 1. `ebr-ltl-synth` was not applicable to these benchmarks as we had to use assumptions (which cannot be captured by the specifications in the $\text{LTL}_{\mathsf{EBR}}$ fragment) to model the timed environment.

These experiments show that $\mathit{SafeLTL}_B$ can express interesting requirements from the real-time domain by appropriate discretization. We did not compare directly to the tool in [14], as the underlying modeling formalism is different, and hence we adapted the benchmarks. However, a superficial comparison of our results to those in [14] shows that our tool compares well (and is in some cases better). Furthermore, on these benchmarks our tool clearly outperforms `strix`.

Office Robot Benchmarks Our last set of experiments considers benchmarks we created ourselves. They consists of a number of specifications describing tasks for a robot in an office building with four rooms. The benchmarks are parametrized by the number of rooms that have to be serviced. They use up to 11 propositions and 14 bounded temporal operators, and bounds between 10 and 21600. Detailed results can be found in Table 1. `ebr-ltl-synth` was either not applicable due to use of assumptions (4 benchmarks) or timed out (25 benchmarks).

The results show that $\mathit{SafeLTL}_B$ can express meaningful synthesis tasks, and that our approach is viable for solving them. Furthermore, they show that our method indeed fulfills its purpose: for specifications requiring large bounds in the temporal operators our method clearly outperforms the state-of-the-art tools.

Overall Analysis Table 1 shows that the countdown-timer game generation is very efficient compared to the solving. As we expect to be able to improve the solving by more sophisticated symbolic techniques, we expect the countdown-timer game based approach to be viable for even more complex properties. In most cases the solver terminated with a low approximation threshold, which shows the usefulness of approximation. In our experience, without approximation solving the benchmarks with large bounds becomes infeasible with our current technique.

7 Conclusion

We presented a new synthesis approach for specifications expressed in an extension of `Safety LTL` with bounded temporal operators. A distinguishing feature of our method is that it is specifically targeted at efficiently solving the synthesis problem for specifications with bounded temporal operators, in particular those with large bounds. Our evaluation results show that our technique performs very well on a range of benchmarks featuring such timing requirements. The key to this success is a novel translation to a safety game with symbolically represented bounds, whose efficiency is due to the use of effective pruning techniques. We observe that our method for solving the generated game is viable, as shown by the evaluation. However, it has potential for further improvement by employing more performant symbolic representations and abstraction techniques.

Data-Availability Statement

The datasets generated during and/or analysed during the current study are available in the Zenodo repository,
https://doi.org/10.5281/zenodo.7505914.

References

1. Alur, R., Etessami, K., Torre, S.L., Peled, D.A.: Parametric temporal logic for "model measuring". ACM Trans. Comput. Log. **2**(3), 388–407 (2001). https://doi.org/10.1145/377978.377990, https://doi.org/10.1145/377978.377990
2. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM **43**(1), 116–146 (1996). https://doi.org/10.1145/227595.227602, https://doi.org/10.1145/227595.227602
3. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: Uppaal-tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4590, pp. 121–125. Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_14, https://doi.org/10.1007/978-3-540-73368-3_14
4. Bouyer, P., Bozzelli, L., Chevalier, F.: Controller synthesis for MTL specifications. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4137, pp. 450–464. Springer (2006). https://doi.org/10.1007/11817949_30, https://doi.org/10.1007/11817949_30
5. Brihaye, T., Estiévenart, M., Geeraerts, G., Ho, H., Monmege, B., Sznajder, N.: Real-time synthesis is hard! In: Fränzle, M., Markey, N. (eds.) Formal Modeling and Analysis of Timed Systems - 14th International Conference, FORMATS 2016, Quebec, QC, Canada, August 24-26, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9884, pp. 105–120. Springer (2016). https://doi.org/10.1007/978-3-319-44878-7_7, https://doi.org/10.1007/978-3-319-44878-7_7
6. Bulychev, P.E., David, A., Larsen, K.G., Li, G.: Efficient controller synthesis for a fragment of $\mathrm{mtl}_{0,\infty}$. Acta Informatica **51**(3-4), 165–192 (2014). https://doi.org/10.1007/s00236-013-0189-z, https://doi.org/10.1007/s00236-013-0189-z
7. Cassez, F.: Efficient on-the-fly algorithms for partially observable timed games. In: Raskin, J., Thiagarajan, P.S. (eds.) Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4763, pp. 5–24. Springer (2007). https://doi.org/10.1007/978-3-540-75454-1_3, https://doi.org/10.1007/978-3-540-75454-1_3
8. Church, A.: Logic, arithmetic and automata. In: International congress of mathematicians. pp. 23–35 (1962)
9. Cimatti, A., Geatti, L., Gigante, N., Montanari, A., Tonetta, S.: Reactive synthesis from extended bounded response LTL specifications. In: 2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020. pp. 83–92. IEEE (2020). https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_15, https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_15

10. David, A., Jensen, P.G., Larsen, K.G., Mikucionis, M., Taankvist, J.H.: Uppaal stratego. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9035, pp. 206–211. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_16, https://doi.org/10.1007/978-3-662-46681-0_16
11. Doyen, L., Geeraerts, G., Raskin, J., Reichert, J.: Realizability of real-time logics. In: Ouaknine, J., Vaandrager, F.W. (eds.) Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5813, pp. 133–148. Springer (2009). https://doi.org/10.1007/978-3-642-04368-0_12, https://doi.org/10.1007/978-3-642-04368-0_12
12. D'Souza, D., Madhusudan, P.: Timed control synthesis for external specifications. In: Alt, H., Ferreira, A. (eds.) STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2285, pp. 571–582. Springer (2002). https://doi.org/10.1007/3-540-45841-7_47, https://doi.org/10.1007/3-540-45841-7_47
13. Heim, P., Dimitrova, R.: Taming large bounds in synthesis from bounded-liveness specifications (full version) (2023). https://doi.org/10.48550/ARXIV.2301.10032, https://arxiv.org/abs/2301.10032
14. Hofmann, T., Schupp, S.: Tacos: A tool for MTL controller synthesis. In: Calinescu, R., Pasareanu, C.S. (eds.) Software Engineering and Formal Methods - 19th International Conference, SEFM 2021, Virtual Event, December 6-10, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13085, pp. 372–379. Springer (2021). https://doi.org/10.1007/978-3-030-92124-8_21, https://doi.org/10.1007/978-3-030-92124-8_21
15. Koymans, R.: Specifying real-time properties with metric temporal logic. Real Time Syst. **2**(4), 255–299 (1990). https://doi.org/10.1007/BF01995674, https://doi.org/10.1007/BF01995674
16. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE Trans. Robotics **25**(6), 1370–1381 (2009). https://doi.org/10.1109/TRO.2009.2030225, https://doi.org/10.1109/TRO.2009.2030225
17. Kupferman, O., Piterman, N., Vardi, M.Y.: From liveness to promptness. Formal Methods Syst. Des. **34**(2), 83–103 (2009). https://doi.org/10.1007/s10703-009-0067-z, https://doi.org/10.1007/s10703-009-0067-z
18. Li, G., Jensen, P.G., Larsen, K.G., Legay, A., Poulsen, D.B.: Practical controller synthesis for mtl$_{0,\ \infty}$. In: Erdogmus, H., Havelund, K. (eds.) Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017. pp. 102–111. ACM (2017). https://doi.org/10.1145/3092282.3092303, https://doi.org/10.1145/3092282.3092303
19. Luttenberger, M., Meyer, P.J., Sickert, S.: Practical synthesis of reactive systems from LTL specifications via parity games. Acta Informatica **57**(1-2), 3–36 (2020). https://doi.org/10.1007/s00236-019-00349-3, https://doi.org/10.1007/s00236-019-00349-3
20. Maler, O., Nickovic, D., Pnueli, A.: On synthesizing controllers from bounded-response properties. In: Damm, W., Hermanns, H. (eds.) Computer Aided Ver-

ification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4590, pp. 95–107. Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_12, https://doi.org/10.1007/978-3-540-73368-3_12

21. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: Mayr, E.W., Puech, C. (eds.) STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings. Lecture Notes in Computer Science, vol. 900, pp. 229–242. Springer (1995). https://doi.org/10.1007/3-540-59042-0_76, https://doi.org/10.1007/3-540-59042-0_76
22. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 578–586. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_31, https://doi.org/10.1007/978-3-319-96145-3_31
23. Nickovic, D., Piterman, N.: From mtl to deterministic timed automata. In: Chatterjee, K., Henzinger, T.A. (eds.) Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6246, pp. 152–167. Springer (2010). https://doi.org/10.1007/978-3-642-15297-9_13, https://doi.org/10.1007/978-3-642-15297-9_13
24. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32, https://doi.org/10.1109/SFCS.1977.32
25. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: A symbolic approach to safety LTL synthesis. In: Strichman, O., Tzoref-Brill, R. (eds.) Hardware and Software: Verification and Testing - 13th International Haifa Verification Conference, HVC 2017, Haifa, Israel, November 13-15, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10629, pp. 147–162. Springer (2017). https://doi.org/10.1007/978-3-319-70389-3_10, https://doi.org/10.1007/978-3-319-70389-3_10