Approximating Optimal Bounds in Prompt-LTL Realizability in Doubly-exponential Time*

Leander Tentrup, Alexander Weinert, and Martin Zimmermann
Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany
{tentrup, weinert, zimmermann}@react.uni-saarland.de

We consider the optimization variant of the realizability problem for Prompt Linear Temporal Logic, an extension of Linear Temporal Logic (LTL) by the prompt eventually operator whose scope is bounded by some parameter. In the realizability optimization problem, one is interested in computing the minimal such bound that allows to realize a given specification. It is known that this problem is solvable in triply-exponential time, but not whether it can be done in doubly-exponential time, i.e., whether it is just as hard as solving LTL realizability.

We take a step towards resolving this problem by showing that the optimum can be approximated within a factor of two in doubly-exponential time. Also, we report on a proof-of-concept implementation of the algorithm based on bounded LTL synthesis, which computes the smallest implementation of a given specification. In our experiments, we observe a tradeoff between the size of the implementation and the bound it realizes. We investigate this tradeoff in the general case and prove upper bounds, which reduce the search space for the algorithm, and matching lower bounds.

1 Introduction

The realizability problem for PROMPT–LTL, Linear Temporal Logic (LTL) enriched with an eventually operator of bounded scope, should be treated as an optimization problem: determine the smallest bound on the bounded eventually such that the specification is realizable with respect to that bound. The best exact algorithms for this problem have triply-exponential running times, i.e., they are exponentially slower than algorithms for the decision variant ("does there exist a bound?"), which is 2EXPTIME-complete. We take a step towards resolving the complexity of the optimization problem by presenting an approximation algorithm with doubly-exponential running time returning a bound that is at most twice the optimum.

In general, the realizability problem asks to determine the winner in an infinite-duration two-player game played between an input and an output player in rounds $n = 0, 1, 2, \ldots$: in each round n, first the input player picks a subset i_n of a fixed set I of input propositions, then the output player picks a subset o_n of a fixed set O of output propositions. The output player wins, if the sequence $(i_0 \cup o_0)(i_1 \cup o_1)(i_2 \cup o_2)\cdots$ of picks satisfies the winning condition, typically a formula φ in some logic. A strategy for the output player is a function mapping sequences $i_0 \cdots i_n \in (2^I)^*$ of inputs to an output $o_n \in 2^O$. Such a strategy is winning, if every outcome that is consistent with the strategy satisfies the winning condition. Formally, the realizability problem asks, given a formula φ, whether the output player has a winning strategy for the realizability game with winning condition φ. For winning conditions in LTL (and many extensions), finite-state strategies suffice, i.e., strategies that are implemented by finite automata with outputs.

LTL [15] is the most prominent logic for specifying reactive systems and the foundations of the LTL realizability problem are well-understood [1, 12, 14, 16, 17]. Recently, the first tools solving the

*Supported by the projects TriCS (ZI 1516/1-1) and AVACS (SFB/TR 14) of the German Research Foundation (DFG) and by the European Research Council (ERC) Grant OSARES (No. 683300)

D. Cantone and G. Delzanno (Eds.): Seventh Symposium on Games, Automata, Logics and Formal Verification (GandALF'16)
EPTCS 226, 2016, pp. 302–315, doi:10.4204/EPTCS.226.21

problem were developed [4, 5, 8, 9, 11], which show promising performance despite the prohibitive worst-case complexity. However, LTL lacks the ability to express time-bounds, e.g., the formula $\mathbf{G}(q \rightarrow \mathbf{F}\, p)$ expresses that every request q has to be responded to by a response p. However, it does *not* require a bound on the waiting times between requests and responses, i.e., it is even satisfied if the waiting times diverge. Several parameterized logics where introduced to overcome this shortcoming [2, 7, 13, 20]. Here, we focus on the smallest such logic: PROMPT–LTL, which extends LTL by the prompt eventually operator $\mathbf{F_P}$, whose semantics are defined with respect to a given bound k. For example, the formula $\mathbf{G}(q \rightarrow \mathbf{F_P}\, p)$ is satisfied with respect to k, if every request is responded to within at most k steps. In decision problems for this logic the bound is typically quantified existentially, e.g., the realizability problem asks for a given formula φ whether there exists a bound k such that the output player has a winning strategy for the realizability game where the winning condition φ is evaluated with respect to k.

Kupferman et al. showed that PROMPT–LTL has the same desirable algorithmic properties as LTL. In particular, model checking is PSPACE-complete and realizability is 2EXPTIME-complete [13]. Hence, one can add the prompt eventually operator to LTL for free. However, as already noticed by Alur et al. in their work on Parametric LTL [2] (which also contains the dual of the prompt eventually and allows for multiple bounds), one can view decision problems for parameterized logics as optimization problems: instead of asking for the existence of some bound, one searches for an optimal one. They showed that the model checking optimization problem for unipolar PLTL specifications, which includes PROMPT–LTL, can be solved in polynomial space [2]. Thus, even finding optimal bounds is not harder than solving the LTL model checking problem. However, for PROMPT–LTL realizability, or equivalently, for infinite games, the situation is different: while the decision problem is known to be 2EXPTIME-complete [13], the best algorithm for the optimization problem has triply-exponential running time [19].

1.1 Our Contributions

We show that relaxing the optimality requirement on the bound allows to recover doubly-exponential running times: an approximately optimal bound can be determined using the alternating color technique, which was introduced by Kupferman et al. to solve the decision problems for PROMPT–LTL. To this end, we present an approximation algorithm with doubly-exponential running time with an approximation ratio of two. The algorithm has to solve at most doubly-exponentially many LTL realizability problems, each solvable in doubly-exponential time. We present the algorithm for PROMPT–LTL, but it is applicable to stronger parameterized extensions of LTL like parametric LTL [2] and parametric LDL [7].

In many situations, approximating the optimal bound is sufficient, since the exact optimum depends on the granularity of the realizability problem at hand. This is even more true if the optimization problem indeed turns out to be harder than the decision variant, e.g., if it is 3EXPTIME-hard. Then, the loss in quality is made up for by significant savings in running time. On the other hand, if the optimal bound is at most exponential in the size of the formula, then it can be exactly determined in doubly-exponential time [19]: the bound can be hardwired into a non-deterministic automaton capturing the specification, which has to be determinized to solve the realizability problem. This involves an exponential blow-up, which implies that this approach only yields a doubly-exponential time algorithm, if the bound is at most exponential.

Furthermore, we report on a proof-of-concept implementation of our algorithm. To handle the solution of the LTL realizability problems, we rely on the framework of bounded synthesis [9], which searches for a minimal-size finite-state winning strategy for a given specification. The evaluation of this implementation shows that, while it suffers from a significant increase in running time compared to LTL realizability, synthesis of prompt arbiters for a small number of clients is feasible.

In our experiments, a tradeoff between size and quality (measured in the bound on the prompt eventually operators) of winning strategies becomes apparent: one can trade size of the strategy for quality and vice versa. We conclude by studying this tradeoff in depth. First, we show that fixing the size of the strategy to n (as it is done during bounded synthesis) implies an exponential upper bound (in n) on the sufficient bound k on the prompt eventually operators. This upper bound reduces the search space of our algorithm. The upper bound is then matched by a tight lower bound. Secondly, we present a family of formulas exhibiting a continuous tradeoff between size and quality with exponential extremal values, i.e., the specifications are realizable with exponential size and a linear bound or with constant size and an exponential bound and the tradeoff between these two points is continuous. Thirdly, by giving up the continuity, one can show even stronger tradeoffs: there is a family of specifications that is realizable with doubly-exponential size and bound zero or with size one and an exponential bound.

2 Definitions

Throughout this work, fix a finite set P of atomic propositions and denote the non-negative integers by $\mathbb{N}$.

2.1 Prompt-LTL

The formulas of PROMPT–LTL are given by the grammar

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \varphi\,\mathbf{R}\,\varphi \mid \mathbf{F_P}\,\varphi,$$

where $p \in P$ represents an atomic proposition. Also, we use the standard shorthands $\mathbf{F}\,\varphi = \mathbf{tt}\,\mathbf{U}\,\varphi$ and $\mathbf{G}\,\varphi = \mathbf{ff}\,\mathbf{R}\,\varphi$ with $\mathbf{tt} = p \vee \neg p$ and $\mathbf{ff} = p \wedge \neg p$, where p is a fixed atomic proposition. Furthermore, we use $\varphi \rightarrow \psi$ as shorthand for $\neg\varphi \vee \psi$, if the antecedent φ is a (negated) atomic proposition (where we identify $\neg\neg a$ with a). We define the size $|\varphi|$ of φ to be the number of subformulas of φ.

In order to evaluate PROMPT–LTL formulas, we need to fix a bound $k \in \mathbb{N}$ to evaluate the prompt eventually operator. Hence, the satisfaction relation is defined for an ω-word $w \in \left(2^P\right)^\omega$, a position n of w, a bound k, and a PROMPT–LTL formula. The definition is standard for the classical operators and defined as follows for the prompt eventually:

- $(w,n,k) \models \mathbf{F_P}\,\varphi$ if and only if there exists a j in the range $0 \leq j \leq k$ such that $(w,n+j,k) \models \varphi$.

For the sake of brevity, we write $(w,k) \models \varphi$ instead of $(w,0,k) \models \varphi$ and say that w is a model of φ with respect to k. If $(w,k) \models \varphi$, we say that w models φ with respect to k. Note that φ is an LTL formula [15], if it does not contain the prompt eventually. In this case, we write $w \models \varphi$.

2.2 Prompt-LTL Realizability

Throughout this subsection, we fix a partition (I,O) of P. An instance of the PROMPT–LTL realizability problem over (I,O) consists of an PROMPT–LTL formula φ over $P = I \cup O$ and asks to determine the winner in the following game, played between Player I and Player O in rounds $n = 0,1,2,\ldots$: in round n, Player I picks $i_n \subseteq I$ and afterwards Player O picks $o_n \subseteq O$. The resulting play is $(i_0 \cup o_0)(i_1 \cup o_1)(i_2 \cup o_2)\cdots \in (2^P)^\omega$.

A strategy for Player O is a mapping $\sigma\colon (2^I)^+ \rightarrow 2^O$. A play as above is consistent with σ, if $o_n = \sigma(i_0 \cdots i_n)$ for every n. We say that σ realizes φ with respect to $k \in \mathbb{N}$, if every play that is consistent with σ satisfies φ with respect to k. Formally, the PROMPT–LTL realizability problem asks, given a

PROMPT–LTL formula φ, whether there is a strategy σ and a k such that σ realizes φ with respect to k. In this case, we say φ is realizable.

A memory structure $\mathscr{M} = (M, m_0, \mathrm{upd})$ consists of a finite set of states M, an initial state $m_0 \in M$, and an update function $\mathrm{upd}\colon M \times 2^I \to M$. We extend the update function to finite input sequences as usual, i.e., we define $\mathrm{upd}^*\colon (2^I)^* \to M$ inductively as $\mathrm{upd}^*(\varepsilon) = m_0$ and $\mathrm{upd}^*(wi) = \mathrm{upd}(\mathrm{upd}^*(w), i)$ for $w \in (2^I)^*$ and $i \in 2^I$. A memory structure $\mathscr{M}$ together with a next-move function $\mathrm{nxt}\colon M \times 2^I \to 2^O$ induces a strategy σ defined as $\sigma(i_0 \cdots i_n) = \mathrm{nxt}(\mathrm{upd}^*(i_0 \cdots i_{n-1}), i_n)$. We say that such a memory structure implements the strategy σ. We call any strategy σ that can be implemented by some memory structure a finite-state strategy. The size of a finite-state strategy is the size of the smallest memory structure implementing it.

The LTL realizability problem is defined by restricting the specifications φ to LTL formulas and is 2EXPTIME-complete [17]. Kupferman et al. showed that PROMPT–LTL realizability is not harder.

Theorem 1 ([13])**.** *The* PROMPT–LTL *realizability problem is* 2EXPTIME*-complete. Furthermore, if φ is realizable with respect to some k, then also with respect to some $k \in \mathcal{O}(2^{2^{|\varphi|}})$ by some finite-state strategy of size $\mathcal{O}(2^{2^{|\varphi|}})$.*

Furthermore, the doubly-exponential upper bounds on the necessary k and on the memory requirements are tight. Also, if φ is realizable with respect to some k, then also with respect to every $k' > k$.

2.3 The Alternating Color Technique

Our algorithm presented in the next section is based on an application of Kupferman et al.'s alternating color technique [13] to PROMPT–LTL realizability. We recall the technique in this subsection.

Let $p \notin P$ be a fixed fresh proposition. An ω-word $w' \in \left(2^{P \cup \{p\}}\right)^\omega$ is a p-coloring of $w \in \left(2^P\right)^\omega$ if $w'_n \cap P = w_n$, i.e., w_n and w'_n coincide on all propositions in P. We say that a position is a change point, if $n = 0$ or if the truth value of p at positions $n-1$ and n differs. A p-block is an infix $w'_m \cdots w'_n$ of w' such that m and $n+1$ are adjacent change points. Let $k \geq 1$: we say that w' is k-spaced, if the truth value of p changes infinitely often and each p-block has length at least k; we say that w' is k-bounded, if each p-block has length at most k (which implies that the truth value of p changes infinitely often).

Given a PROMPT–LTL formula φ, $\mathrm{rel}(\varphi)$ denotes the formula obtained by inductively replacing every subformula $\mathbf{F_P}\,\psi$ by

$$(p \to (p\,\mathbf{U}\,(\neg p\,\mathbf{U}\,\mathrm{rel}(\psi)))) \wedge (\neg p \to (\neg p\,\mathbf{U}\,(p\,\mathbf{U}\,\mathrm{rel}(\psi)))).$$

Intuitively, instead of requiring ψ to be satisfied within a bounded number of steps, $\mathrm{rel}(\varphi)$ requires it to be satisfied within at most one change point. The relativization $\mathrm{rel}(\varphi)$ is an LTL formula of size $\mathcal{O}(|\varphi|)$. Kupferman et al. showed that φ and $\mathrm{rel}(\varphi)$ are "equivalent" on ω-words which are bounded and spaced.

Lemma 1 ([13])**.** *Let φ be a* PROMPT–LTL *formula.*

1. *If $(w,k) \models \varphi$, then $w' \models \mathrm{rel}(\varphi)$ for every k-spaced p-coloring w' of w.*
2. *Let $k \in \mathbb{N}$. If w' is a k-bounded p-coloring of w such that $w' \models \mathrm{rel}(\varphi)$, then $(w, 2k) \models \varphi$.*

3 Approximating Optimal Bounds in Doubly-Exponential Time

Determining whether a PROMPT–LTL formula φ is realizable with respect to some k induces a natural optimization problem: determine the smallest such k. The optimum (and a strategy realizing φ with respect to the optimum) can be computed in triply-exponential time [19].

However, it is an open problem whether the optimization problem can be solved in doubly-exponential time, i.e., whether optimal PROMPT–LTL realizability is no harder than LTL realizability. We take a step towards resolving the problem by showing that the optimum can be approximated within a factor of two in doubly-exponential time.

The alternating color technique is applied to the PROMPT–LTL realizability problem by replacing φ by its relativization $\mathrm{rel}(\varphi)$ and by letting Player O determine the truth value of the distinguished proposition p for every position by adding it to the output propositions O. The full details are explained in [13], where the following statements are shown to prove the application of the alternating color technique to be correct. Here, ψ_k is an LTL formula of linear size in k that characterizes k-boundedness, i.e., $w' \models \psi_k$ if, and only if, w' is a k-bounded p-coloring.

Lemma 2 ([13])**.** *Let φ be a* PROMPT–LTL *formula and let $k \in \mathbb{N}$.*

1. *A strategy realizing φ with respect to k can be turned into a strategy realizing* $\mathrm{rel}(\varphi) \wedge \psi_k$.
2. *A strategy realizing* $\mathrm{rel}(\varphi) \wedge \psi_k$ *can be turned into a strategy realizing φ with respect to $2k$.*

Also, if k is not *too large*, we can check the realizability of $\mathrm{rel}(\varphi) \wedge \psi_k$ in doubly-exponential time.

Lemma 3. *The following problem is in* 2EXPTIME*: Given a* PROMPT–LTL *formula φ and a natural number k that is at most doubly-exponential in $|\varphi|$, is* $\mathrm{rel}(\varphi) \wedge \psi_k$ *realizable? Furthermore, one can compute a strategy realizing the formula (if one exists) in doubly-exponential time.*

Proof. As usual, we reduce the problem to a parity game (see [10] for background). First, we construct a deterministic parity automaton recognizing the language $\{\rho \in (2^{P \cup \{p\}})^\omega \mid \rho \models \mathrm{rel}(\varphi)\}$ and intersect it with a deterministic safety automaton that recognizes $\{\rho \in (2^{P \cup \{p\}})^\omega \mid \rho \models \psi_k\}$. It is known that the first automaton is of doubly-exponential size and has exponentially many colors (both in $|\varphi|$) while the second one is of linear size in k. Thus, the deterministic parity automaton $\mathfrak{A}$ recognizing the intersection is of doubly-exponential size in $|\varphi|$ and linear size in k and has exponentially many colors in $|\varphi|$.

Next, we split a transition of $\mathfrak{A}$ labeled by $A \subseteq P \cup \{p\}$ into two, the first one labeled by $A \cap I$ and the second one by $A \setminus I$. By declaring the original states of $\mathfrak{A}$ to be Player I states and the new intermediate states obtained by splitting the transitions to be Player O states, we obtain a parity game that is won by Player O from the initial state of $\mathfrak{A}$ if, and only if, $\mathrm{rel}(\varphi) \wedge \psi_k$ is realizable. Additionally, a winning strategy for Player O in the parity game can be turned into a strategy realizing $\mathrm{rel}(\varphi) \wedge \psi_k$. This parity game is of doubly-exponential size with exponentially many colors, both in $|\varphi|$. The winner and a winning strategy for her in such a game can be computed in doubly-exponential time [18]. □

Now, we are able to present the algorithm for approximating optimal bounds for PROMPT–LTL realizability. Given an input φ, the algorithm first checks whether φ is realizable with respect to some k. If not, then the optimum is ∞ by convention. Otherwise, Theorem 1 yields a doubly-exponential upper bound u on the optimum. Now, the algorithm determines the smallest $1 \leq k \leq u$ such that $\mathrm{rel}(\varphi) \wedge \psi_k$ is realizable and returns $2k$. The emptiness test and determining the realizability of $\mathrm{rel}(\varphi) \wedge \psi_k$ can be executed in doubly-exponential time as shown in Theorem 1 and Lemma 3. As the latter problem has to be solved at most doubly-exponentially often[1], the overall running time is doubly-exponential as well. Furthermore, due to Lemma 3 and Lemma 2.2, we even obtain a strategy realizing φ with respect to $2k$.

It remains to argue that the algorithm approximates the optimum $k_{\mathrm{opt}} \leq u$ within a factor of two: let $2k$ be the output of the approximation algorithm, i.e., k is minimal such that $\mathrm{rel}(\varphi) \wedge \psi_k$ is realizable.

[1] With binary search, this can be improved to exponentially often. However, the running time of the realizability check depends on k, which is typically small. Thus, traversing the search space $0, 1, \ldots, u$ in the natural order is more beneficial. We discuss the search strategy in more detail in Section 4.

Thus, Lemma 2.2 implies $k_{\text{opt}} \leq 2k$. Conversely, φ being realizable with respect to k_{opt} implies that $\text{rel}(\varphi) \wedge \psi_{k_{\text{opt}}}$ is realizable due to Lemma 2.1, i.e., $k \leq k_{\text{opt}}$ due to minimality of k.

Altogether, we obtain $k \leq k_{\text{opt}} \leq 2k$. Recall that the algorithm returns $2k$, i.e., φ is realizable with respect to the returned value due to monotonicity. Also, the approximation ratio $2k/2k-k_{\text{opt}}$ is bounded by $2k/2k-k_{\text{opt}} \leq 2k/2k-k = 2$, i.e., the bound found by our algorithm is at most twice the optimal bound.

Theorem 2. *The optimization problem for* PROMPT–LTL *realizability can be approximated within a factor of two in doubly-exponential time. As a byproduct, one obtains a strategy witnessing the approximatively optimal bound.*

4 Empirical Evaluation

In the previous section we have described an algorithm that, given some PROMPT–LTL specification φ, approximates the optimal bound k for which the formula can be realized. The algorithm uses LTL realizability checking as a black-box to determine the realizability of the formulas $\text{rel}(\varphi) \wedge \psi_k$, where k is a parameter from a doubly-exponential set. The search strategy heavily influences the running time of the algorithm (but not the worst-case complexity). Towards an implementation, we rely on bounded LTL synthesis [9] for checking the realizability of $\text{rel}(\varphi) \wedge \psi_k$. In addition to computing the smallest strategy that realizes $\text{rel}(\varphi) \wedge \psi_k$, bounded synthesis also allows us to search for strategies of some fixed size n. Thus, we obtain a sub-procedure that takes as input some PROMPT–LTL formula, as well as some values n and k, which checks whether or not there exists a finite-state strategy of size n that realizes φ with respect to $2k$.

To this end, it first constructs the LTL formula $\varphi' = \text{rel}(\varphi) \wedge \psi_k$ from φ, which is then given to the tool `BoSy` [6] together with the desired size n of the strategy. `BoSy` then checks φ' for realizability and returns a strategy of size n, if there exists one. In order to do so, it first translates φ' to a universal co-Büchi automaton that accepts the language of φ'. Based on this automaton, it constructs a QBF query that is satisfiable if, and only if, there exists a strategy of size n which is then solved by a combination of a QBF preprocessor and a solver. Due to Lemma 2, we know that the strategy returned by `BoSy` realizes φ with respect to $2k$ when restricted to P.

We evaluate our implementation on a family of arbiters. Each arbiter manages some number r of resources. Player I poses requests q_i for some resource $1 \leq i \leq r$, while Player O has to grant them by playing p_i for $1 \leq i \leq r$. Moreover, Player O can only grant a single resource at a time. In addition to the usual requirement that each request has to be answered eventually, we require that a request for one of the first r_p resources is answered promptly, for $0 \leq r_p \leq r$. Thus, for some parameters r and r_p, we construct the PROMPT–LTL specification

$$\varphi_{r,r_p} := \bigwedge_{1 \leq i \leq r_p} \mathbf{G}\,(q_i \rightarrow \mathbf{F_P}\, p_i) \wedge \bigwedge_{r_p < i \leq r} \mathbf{G}\,(q_i \rightarrow \mathbf{F}\, p_i) \wedge \bigwedge_{i \neq j} \mathbf{G}\,(\neg p_i \vee \neg p_j).$$

Note that, for each $r \in \mathbb{N}$, the specification $\varphi_{r,0}$ is an LTL formula.

For our experiments we used machines equipped with Intel Xeon-Haswell processors running at 3.6 GHz with 32 GB of memory. The complete dataset we report on in this evaluation is available at `https://arxiv.org/abs/1511.09450`.

We first compare the running time of LTL synthesis with the running time of our implementation on the PROMPT–LTL formulas in order to quantify the slowdown incurred by performing PROMPT–LTL synthesis instead of LTL synthesis. Since, as previously explained, a naive search strategy that simply performs bounded synthesis on $\text{rel}(\varphi) \wedge \psi_k$ for increasing k is infeasible, we instead search for a realizing

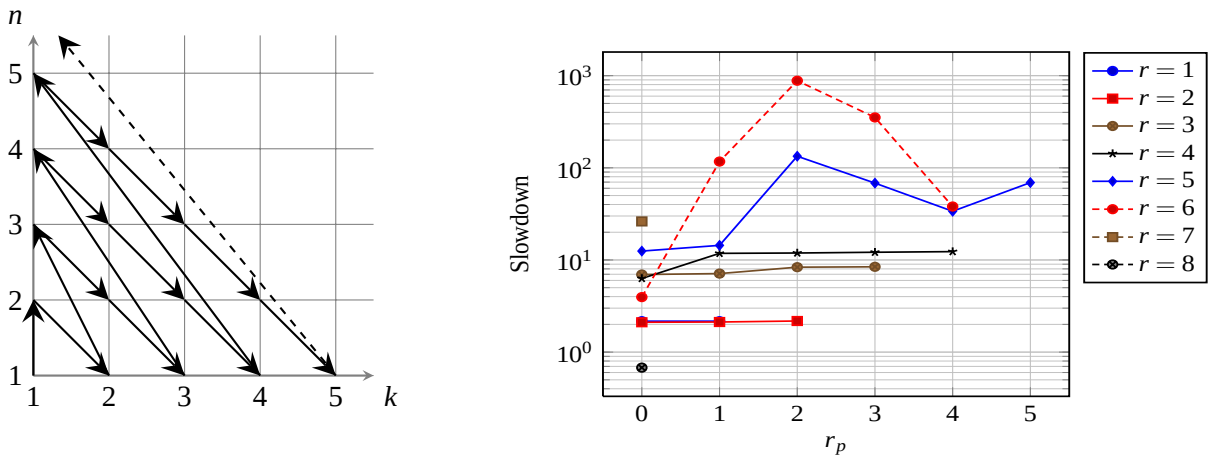

Figure 1: The search strategy for some realizing implementation on the left-hand side and the slowdown of PROMPT–LTL synthesis on the right-hand side.

implementation along the diagonals of the search space, as shown in the left-hand side of Figure 1. We run our implementation with this search strategy on φ_{r,r_p} for each $r \in [1;10]$ and each $r_p \in [0;r]$ and compared the running time to that of BoSy on $\varphi_{r,0}$.[2]

The results are shown in the right-hand side of Figure 1. For each comparison, the number of resources r is denoted by the line-color, while the number of prioritized resources is displayed on the x-axis. The slowdown is shown on the y-axis, which is logarithmically scaled. Note that there does not exist a data point for each pair (r,r_p) with $1 \leq r \leq 10$ and $0 \leq r_p \leq r$, since, for $r \geq 9$, BoSy timed out after 100 minutes and, for all other values not shown, our implementation timed out after 100 minutes.

We see that, when given an LTL formula $\varphi_{r,0}$, in general our implementation is slower than BoSy by a factor on the order of magnitude of 10^1. This results from our tool calling BoSy multiple times even for LTL formulas, as, in order to find a strategy of size n that realizes φ, our implementation first searches for strategies of sizes $n'-b$ that realize $\varphi_{r,0}$ with respect to $1+b$ for $n' < n$ and $0 \leq b < n'$ (cf. the search strategy shown in Figure 1). For $\varphi_{8,0}$, however, our implementation finds a realizing strategy after 1 299 seconds, while BoSy takes 1 914 seconds for the same task. This discrepancy is likely due to differences in the generated automaton that lead to different QBF formulas and result in different solving times.

When asked to realize φ_{r,r_p} with $r_p > 0$, however, prioritizing around half of the available resources incurs the greatest penalty in terms of running time. Recall that each $\mathbf{F_P}\,\psi$ in φ_{r,r_p} is first rewritten to $(p \rightarrow (p\,\mathbf{U}\,(\neg p\,\mathbf{U}\,\mathrm{rel}(\psi)))) \wedge (\neg p \rightarrow (\neg p\,\mathbf{U}\,(p\,\mathbf{U}\,\mathrm{rel}(\psi))))$ before being given to BoSy, while the traditional $\mathbf{F}\,\psi$ operator is a shorthand for the significantly smaller formula $\mathbf{tt}\,\mathbf{U}\,\psi$. Thus, for increasing r_p, the automaton and consequently the formula given to the QBF solver becomes larger. We noticed that determining that no realization of φ_{r,r_p} with some parameters n and k exists was faster for increasing r_p, in particular for $r = 5$ and $r = 6$. Hence, the search terminates earlier despite an increased number of solved QBF queries, resulting in an overall smaller slowdown.

After having evaluated the running time of our tool against that of that of the underlying bounded synthesis tool, we now evaluate the feasibility of our approach for the search for a strategy of a given size realizing a formula with respect to some given bound. In other words, we are given some φ_{r,r_p}, some size

[2] Note that it is not possible to run BoSy on φ_{r,r_p} for $r_p > 0$, as BoSy performs LTL synthesis, while φ_{r,r_p} is a PROMPT–LTL formula for $r_p > 0$.

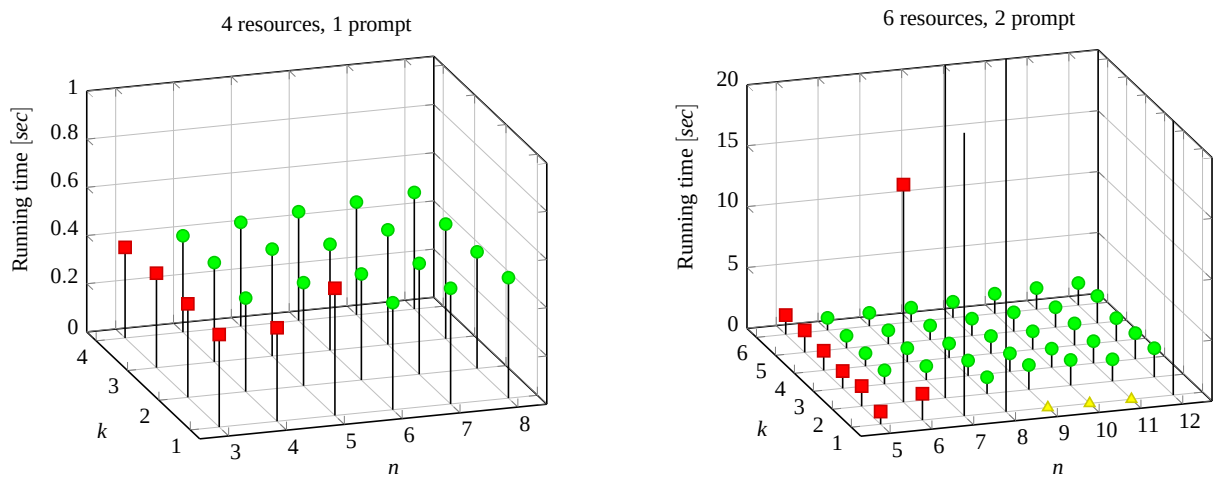

Figure 2: Running times and results for $\varphi_{4,1}$ on the left-hand side and $\varphi_{6,2}$ on the right-hand side. Green circles denote realizable parameters, while red squares denote unrealizable parameters. Yellow triangles denote that the tool encountered a time-out after 20 minutes.

n and a bound k and want to decide whether or not a strategy of size at most n exists that realizes φ_{r,r_p} with respect to $2k$.

In order to satisfy the requirement that every request is eventually granted, at least $r-1$ states are required. Also, the smallest possible strategy is a round-robin strategy, which simply grants each resource in order. This strategy realizes the formula with respect to the bound r. These two propositions yield upper bounds on n and k for a given r. Hence, for each $r \in [1,10]$ and each $r_p \in [1,r]$ we search for implementations of φ_{r,r_p} of size $n \in [r-1,2r]$ and with respect to the bound $k \in [1,r]$ on the block-size.

We show the results for $\varphi_{4,1}$ and $\varphi_{6,2}$ in Figure 2. Green circles, red squares, and yellow triangles denote realizable parameter combinations, unrealizable ones, and those for which our implementation timed out after 20 minutes, respectively. Note that in the benchmark of $\varphi_{6,2}$ there are four invocations that ran longer than 20 seconds and are thus not shown in the diagram. The searches for strategies of size 7 that realize $\varphi_{6,2}$ with respect to the bounds 2 and 4, respectively, as well as the search for a strategy of size 8 that realizes $\varphi_{6,2}$ with respect to the bound 2 were eventually unsuccessful after 23 seconds, 833 seconds, and 1 009 seconds, respectively. There exists, however, a strategy of size 12 that realizes $\varphi_{6,2}$ with respect to 2, which was found after 72 seconds.

Note that both evaluations shown in Figure 2 exhibit a tradeoff. There exist strategies that realize $\varphi_{6,2}$ with respect to the bounds 6, 4, and 2. These strategies have size 6, 8, and 12, respectively. We show the minimal strategies $\sigma_{6,3}$ and $\sigma_{12,1}$ realizing $\varphi_{6,2}$ with respect to the bounds 6 and 2, respectively, in Figure 3. The strategy $\sigma_{6,3}$ proceeds in a round-robin fashion using only 6 states while $\sigma_{12,1}$ grants p_1 every second step using 12 states to ensure that all requests are eventually granted.

We also see that, in general, unsuccessful searches for a strategy with given size and bound take longer than successful searches for larger strategies or for strategies with a larger bound. Intuitively, this is due to the fact that the resulting QBF formula is satisfiable if, and only if, there exists a strategy with the given parameters and that refuting all possible strategies of size n is, in general, harder than showing that such a strategy exists. Hence, it is of interest to to investigate the border between the realizable and the unrealizable parameters. We do so in the next section.

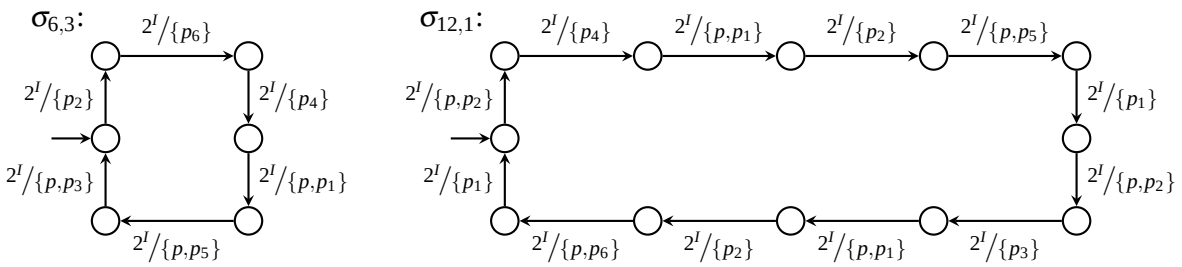

Figure 3: Two strategies $\sigma_{6,3}$ and $\sigma_{12,1}$ realizing $\varphi_{6,2}$ with respect to the bounds 6 and 2, respectively. A transition of the form $m \xrightarrow{i/o} m'$ denotes that upon reading $i \in 2^I$ in state m, Player O outputs $o \in 2^O$ and updates her memory to m' (cf. Subsection 2.2).

5 Trading Memory for Quality and Vice Versa

We have seen in the previous section that there exist PROMPT–LTL formulas φ that exhibit a tradeoff, i.e., for some $k < k'$, the minimal strategy realizing φ with respect to $2k$ may be larger than the minimal strategy realizing φ with respect to $2k'$. In this section, we investigate the Pareto frontier of this tradeoff, i.e., those positions in the search space shown in the previous section, at which it is not possible to decrease either the size of the strategy or the bound it realizes without increasing the other value. To this end, we define the set of realizable parameters $\mathscr{R}(\varphi) \subseteq \mathbb{N} \times \mathbb{N}$ of φ such that $(n,k) \in \mathscr{R}(\varphi)$ if and only if there exists a strategy σ with $|\sigma| = n$ that satisfies φ with respect to k. Note that $\mathscr{R}(\varphi)$ is upwards-closed, i.e., if $(n,k) \in \mathscr{R}(\varphi)$, then also $(n+1,k) \in \mathscr{R}(\varphi)$ and $(n,k+1) \in \mathscr{R}(\varphi)$.

A Pareto position of a formula φ is a pair of realizable parameters $(n,k) \in \mathscr{R}(\varphi)$ such that it is not possible to realize the bound k with a strategy of size $n-1$, and no strategy of size n realizes a smaller bound than k. Formally, a pair of realizable parameters $(n,k) \in \mathscr{R}(\varphi)$ is a Pareto position if both $(n-1,k) \notin \mathscr{R}(\varphi)$ and $(n,k-1) \notin \mathscr{R}(\varphi)$. When considering the set $\mathscr{R}(\varphi)$ geometrically, the Pareto positions of φ are the corner points of the area defined by $\mathscr{R}(\varphi)$, as shown in Figure 4.

By a simple geometrical argument over the space $\mathbb{N} \times \mathbb{N}$ that combines Theorem 1 with the upwards-closure of $\mathscr{R}(\varphi)$ we obtain a doubly-exponential bound in $|\varphi|$ on the number of Pareto positions of φ.

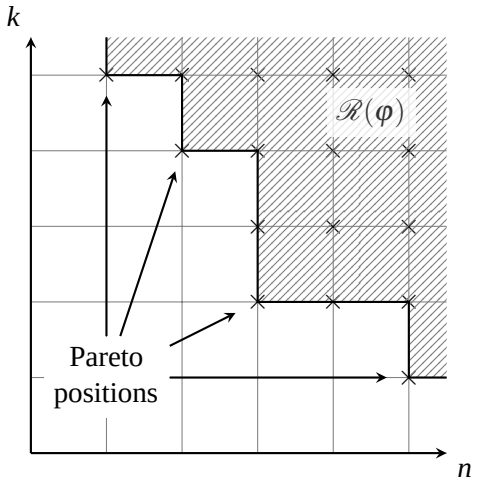

Figure 4: The geometrical interpretation of $\mathscr{R}(\varphi)$ and the Pareto positions of φ.

Lemma 4. *Let φ be a* PROMPT–LTL *formula. There exist at most $\mathscr{O}(2^{2^{|\varphi|}})$ Pareto positions of φ.*

Proof. If φ is not realizable with respect to any bound k, then we have $\mathscr{R}(\varphi) = \emptyset$ and thus, the statement holds true. Thus, assume φ is realizable with respect to some k. Due to Theorem 1, we obtain that φ is realizable with respect to some $k' \in \mathscr{O}(2^{2^{|\varphi|}})$, which is witnessed by a strategy of size $n' \in \mathscr{O}(2^{2^{|\varphi|}})$, i.e., $(n',k') \in \mathscr{R}(\varphi)$.

Clearly, there are at most k' Pareto positions (n,k) with $k \leq k'$, since otherwise, upwards-closure of $\mathscr{R}(\varphi)$ would be violated. For the same reason, there are at most n' Pareto positions (n,k) with $n \leq n'$.

Finally, there can exist no Pareto positions (n,k) with either $n \geq n'$ or $k \geq k'$, again due to upwards-closure of $\mathscr{R}(\varphi)$. Thus, there exist at most $n'+k' \in \mathscr{O}(2^{2^{|\varphi|}})$ Pareto positions of φ. □

Having shown that the number of Pareto positions has an upper bound, we now investigate the Pareto frontier in general. We show that fixing one parameter yields exponential and doubly-exponential upper bounds on the other parameter, respectively. For a fixed n, this upper bound on k is obtained by a reduction to the model checking problem for PROMPT–LTL. For a fixed k, however, we obtain the upper bound on n by turning φ into a parity game of doubly-exponential size and solving this game.

Lemma 5. *Let φ be a* PROMPT–LTL *formula.*

1. *Let σ be a strategy that realizes φ with $|\sigma| = n$. Then $(n,k) \in \mathscr{R}(\varphi)$ for some $k \in \mathscr{O}(n \cdot 2^{|\varphi|})$.*
2. *Let φ be realizable w.r.t. k. Then $(n,k) \in \mathscr{R}(\varphi)$ for some $n \in \mathscr{O}(2^{|\varphi|^2 \cdot (2(k+1))^{2|\varphi|}})$*

Proof. 1.) Fixing a strategy σ of size n simplifies the realizability problem to the problem of model checking PROMPT–LTL. The upper bound of k in the model checking problem for PLTL, which includes PROMPT–LTL, is known to be linear in n and exponential in $|\varphi|$ [2].

2.) Given a bound k, we can translate φ to a parity game $\mathscr{P}$ of size $\mathscr{O}(2^{|\varphi|^2 \cdot (2(k+1))^{2|\varphi|}})$ that is winning for player 0 if, and only if, φ is realizable with bound k [19]. As a positional winning strategy for player 0 in $\mathscr{P}$ can be translated into a realizing strategy σ for φ with respect to k, with $|\sigma| \in \mathscr{O}(|\mathscr{P}|)$. This proves our upper bound on the size of a realizing strategy. □

The previous two lemmas each presented upper bounds on the number of Pareto positions. These bounds permit us to restrict the search space when looking for a realizing strategy: Instead of fixing some n or k and checking doubly-exponentially many possibilities for the respective other parameter, we only need to consider exponentially many possible values for it.

We now turn our attention to the respective lower bounds, i.e., we provide a family of formulas φ_b that exhibit such a Pareto frontier. More precisely, for each φ_b, there exists a family of strategies $\sigma_{b,j}$ such that $\sigma_{b,j}$ is of size exponential in j and realizes φ_b with respect to some k that is exponential in $b-j$. Each of these $\sigma_{b,j}$ is minimal for its respective bound.

Intuitively, φ_b describes a game in which Player O decides at the beginning how much memory she wants to use by playing some number j. Player I then plays some number in $[0;2^j)$, which Player O has to repeat afterwards, thus requiring her to use exponential memory in j. Afterwards, Player I implements a binary counter using $b-j$ bits. The game ends once Player I has counted up to $2^{b-j}-1$. Moreover, φ_b requires that this end is reached promptly, i.e., the bound k is in $\mathscr{O}(2^{b-j})$, while every strategy realizing φ_b with respect to that bound k has at least size 2^j.

Theorem 3. *For each $b \in \mathbb{N}$ there exists a* PROMPT–LTL *formula φ_b with $|\varphi_b| \in \mathscr{O}(b)$ such that for each $0 \leq j \leq b$, there exists an $n \in \mathscr{O}(2^j)$ and a $k \in \mathscr{O}(2^{b-j})$, such that (n,k) is a Pareto position.*

Proof. We construct an LTL formula φ_b that specifies the following game $\mathscr{G}_b$ for a given b with $P = I \cup O$, where $I = \{i, \#_i\}$ and $O = \{o, \#_o\}$.

The game begins with Player O playing some number $0 \leq j \leq b$ in unary encoding, i.e., she plays j times her proposition o and ends this encoding by playing $\#_o$. After this first $\#_o$, Player I plays the binary encoding of some number $0 \leq n < 2^j$ using j positions, and finishes with a $\#_i$. After Player I has issued his $\#_i$, Player O must repeat his sequence and finish with $\#_o$. When Player O has finished repeating Player I's sequence, Player I must implement a binary counter with $b-j$ bits, starting with the binary encoding of 0. Two consecutive values of the counter must be delimited by $\#_i$, and after encoding

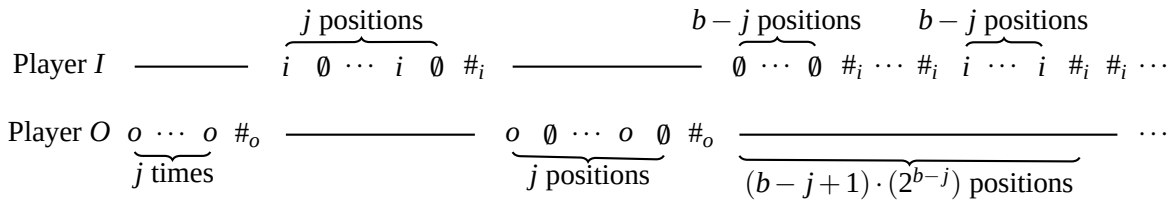

Figure 5: A play of the game $\mathscr{G}_b$. Sequences of $\emptyset$ are denoted by black lines for readability.

$2^{b-j}-1$, Player I must play $\#_i\#_i$. During the respective other player's turn, both players always have to play the empty set. If either player does not conform to the rules of this game, she loses. A play of this game is illustrated in Figure 5.

Towards a formal definition of φ_b, fix some j with $0 \leq j \leq b$. We construct formulas $\varphi^{\text{Pick}}_{b,j}$, $\varphi^{I}_{b,j}$ and $\varphi^{O}_{b,j}$ that encode the fact that Player O starts by announcing the number j in unary encoding, and assumptions about the behavior of both players in $\mathscr{G}_b$, respectively, if Player O starts by announcing j.

The formula $\varphi^{\text{Pick}}_{b,j}$ is trivial to construct in linear size in j using nested $\mathbf{X}$-operators, as it just argues about a prefix of length $j+1$ of the resulting play. The formula $\varphi^{I}_{b,j}$ encodes the following assumptions about the behavior of Player I:

1. At any time, Player I picks either i or $\#_i$, or neither, but never both,
2. Player I plays $\emptyset$ until Player O plays $\#_o$ for the first time,
3. immediately after the first position where Player O plays $\#_o$, Player I does not pick $\#_i$ for j positions, but does pick it after j turns,
4. after Player I has played $\#_i$ for the first time, he plays $\emptyset$ until Player O has played $\#_o$ again,
5. immediately after Player O has played $\#_o$ for the second time, Player I plays $\emptyset$ for $b-j$ turns, followed by $\#_i$,
6. whenever Player I plays $\#_i$ after the first time he has done so, if the $b-j$ positions preceding that $\#_i$ encode some $\ell \in [0, 2^{b-j}-1)$ in binary using the proposition i, the $b-j$ positions succeeding that $\#_i$ encode $\ell+1$ and are followed directly by another $\#_i$, and
7. if and when Player I encodes $2^{b-j}-1$ at some point after he has played his first $\#_i$, this encoding is directly followed by $\#_i\#_i$.

These properties can be specified using polynomial-length LTL formulas in both b and j. In particular, the correct behavior of the $(b-j)$-bit counter can be specified using a formula of polynomial size in $(b-j)$ using standard constructions. Thus, we obtain the formula $\varphi^{I}_{b,j}$ of polynomial size in b and j.

Similarly, the formula $\varphi^{O}_{b,j}$ encodes the following guarantees that Player O has to ensure, if the assumptions regarding the play of Player I are met:

1. At any time, Player O plays either o or $\#_o$, or neither, but never both,
2. after playing $\#_o$ for the first time, Player O only plays $\emptyset$, until Player I plays a $\#_i$,
3. if Player I has played some word $w \in (\{i\}+\emptyset)^j$ directly preceding his first $\#_i$, then Player O must play w with every i replaced by an o immediately after Player I has played his first $\#_i$, and
4. after her second $\#_o$, Player O exclusively plays $\emptyset$.

Again, all these properties only argue about infixes of linear size in j and can all be specified using formulas of polynomial size in b and j. Hence, we obtain a formula $\varphi^O_{b,j}$ that specifies all these guarantees, which is again of size polynomial in b and j.

Using the formulas $\varphi^{\text{Pick}}_{b,j}$, $\varphi^I_{b,j}$, and $\varphi^O_{b,j}$ we then define $\varphi_b = \bigvee_{0\leq j\leq b} \varphi^{\text{Pick}}_{b,j} \wedge (\varphi^I_{b,j} \rightarrow \varphi^O_{b,j} \wedge \mathbf{F_P}(\#_i \wedge \mathbf{X}\#_i))$, which formally denotes the requirement that Player O starts by playing some j in unary encoding, and, if Player I satisfies the assumptions about his behavior in this situation, then Player O fulfills the requirements to her part of the play, and that Player I promptly plays $\#_i\#_i$, i.e., promptly finishes counting.

We now show that for each j in $[0;b]$, there exist an $n \in \mathcal{O}(2^j)$ and a $k \in \mathcal{O}(2^{b-j})$ such that (n,k) is a Pareto position. To this end, fix some j with $0 \leq j \leq b$. Clearly, Player O has a strategy $\sigma_{b,j}$ with $|\sigma_{b,j}| \in \mathcal{O}(2^j)$ that realizes φ with respect to some $k \in \mathcal{O}(2^{b-j})$. Intuitively, Player O first uses $j+1$ memory states as a unary counter up to j. She plays o until this counter reaches j, which happens after $\mathcal{O}(j)$ steps. Once the counter has reached j, Player O plays $\#_o$ and stores the number encoded by Player I using $\mathcal{O}(2^j)$ memory states, which again takes $\mathcal{O}(j)$ steps. After Player I has played $\#_i$, Player O repeats the encoding of the number played by Player I, again using $\mathcal{O}(2^j)$ memory states and $\mathcal{O}(j)$ steps. Afterwards, Player O plays a single $\#_o$ followed by $\emptyset$ ad infinitum. Player I then implements a binary counter with $b-j$ bits and has to play $\#_i\#_i$ after that counter has reached its maximal value. This occurs after $\mathcal{O}((b-j)\cdot 2^{b-j})$ steps. Hence, this strategy requires $\mathcal{O}(j+2^j+2^j) = \mathcal{O}(2^j)$ memory states and realizes φ_b with respect to some $k \in \mathcal{O}(3j+(b-j)\cdot 2^{b-j}) = \mathcal{O}(2^{b-j})$.

It remains to show that $\mathcal{O}(2^j)$ is a lower bound on the size of any strategy that realizes some bound $k \in \mathcal{O}(2^{b-j})$ and that $\mathcal{O}(2^{b-j})$ is a lower bound on the parameter k with respect to which a strategy with size in $\mathcal{O}(2^j)$ can realize φ_b. First, assume that there exists a strategy $\sigma'_{b,j}$ with $|\sigma'_{b,j}| \in o(2^j)$ that realizes φ_b with respect to some $k \in \mathcal{O}(2^{b-j})$. Then, there must exist two numbers $0 \leq \ell < \ell' < 2^j$ such that $\sigma'_{b,j}$ ends up in the same state after the two plays $o^j\#_o\text{BIN}_j(\ell)\#_i$ and $o^j\#_o\text{BIN}_j(\ell')\#_i$, where $\text{BIN}_j(\ell)$ denotes the encoding of ℓ in binary using j bits (encoded by i). Hence, Player O cannot differentiate between ℓ and ℓ' and does not ensure her guarantees in one of the two cases. Thus, $\sigma'_{b,j}$ does not realize φ_b.

Moreover, it is clear that, due to the strict structure of the game, Player O cannot force the occurrence of $\#_i\#_i$ in $o(2^{b-j})$ steps using a memory structure of size $\mathcal{O}(2^j)$. The only way for her to force Player I to play $\#_i\#_i$ after less than $\mathcal{O}(2^{b-j})$ steps is to play some number $j' > j$ at the beginning of the game. Doing so, however, would give Player I j' bits to encode some number at the beginning of the second part of the game, which in turn would require Player O to use $\mathcal{O}(2^{j'})$ memory states to store and repeat this number, as argued before. □

We observe that each φ_b has linearly many Pareto positions in b, where the extremal values in $\mathscr{R}(\varphi_b)$ are (n,k) for $n \in \mathcal{O}(1)$, $k \in \mathcal{O}(2^b)$ and (n',k') for $n' \in \mathcal{O}(2^b)$ and $k' \in \mathcal{O}(b)$. In order to show that the distance between n and k may even become doubly-exponential, we move from the continuous tradeoff exhibited by the previous theorem to a discrete tradeoff, i.e., for each b we provide a formula φ_b such that there are two ways to realize φ_b; Either, Player O realizes this formula with respect to some constant bound, but requires doubly-exponential memory to do so, or she realizes it with respect to some exponential bound, but can do so by using only constant memory.

These bounds are obtained by letting Player O choose between one of two games, in which Player I has to implement either a doubly- or singly-exponentially bounded counter. In the former case, this realization is formalized by an LTL formula, hence the specification is trivially realized with respect to $k = 0$. Player O does, however, require doubly-exponential memory to denote errors in Player I's implementation of the counter [16]. In the latter case, Player O does not require any memory, but the specification requires that Player I finishes counting promptly. Hence, the specification is only fulfilled with respect to an exponential bound.

Theorem 4. *For each $b \in \mathbb{N}$ there exists a* PROMPT–LTL *formula φ_b with $|\varphi_b| \in \mathcal{O}(b)$ such that there exist $n \in \mathcal{O}(2^{2^b})$, $n' \in \mathcal{O}(1)$, and $k' \in \mathcal{O}(2^b)$, such that both $(n,0)$ and (n',k') are Pareto positions of φ_b.*

Proof. Let $b \in \mathbb{N}$. We give a realizable PROMPT–LTL formula φ_b that exhibits the stated tradeoff. Let ψ be a realizable LTL formula with $|\psi| \in \mathcal{O}(b)$ where each strategy realizing ψ has at least doubly-exponentially many states in b. Let ψ' be a PROMPT–LTL formula with $|\psi'| \in \mathcal{O}(b)$ that is realizable with respect to $k \in O(2^{|\psi'|})$ and constant strategy size. We construct φ to be $(o \rightarrow \mathbf{X}\,\psi) \wedge (\neg o \rightarrow \mathbf{X}\,\psi')$ where o is a fresh atomic proposition controlled by Player O. Player O decides in the first step whether she wants to satisfy the LTL formula ψ or the PROMPT–LTL formula ψ'. Given ψ and ψ', it is trivial to verify that the stated properties hold.

It remains to show that such formulas ψ and ψ' exist. It is known that a LTL formula ψ with the required properties exists [16]. Intuitively, ψ requires Player I to implement a binary counter with exponentially many bits in b, which counts up to 2^{2^b}. The task of Player O is to mark errors in Player I's implementation of the counter, for which she requires doubly-exponential memory in b.

The PROMPT–LTL formula ψ' requires Player I to implement a binary counter, similarly to the latter phase of the game $\mathcal{G}_b$ constructed in the proof of Theorem 3. After Player I has counted up to 2^b, he plays some delimiter #. Then the formula ψ' is of the form $\psi_{\text{count}} \rightarrow \mathbf{F_P}\,\#$, where ψ_{count} specifies the assumption that Player I implements the binary counter correctly and finishes with a #. Clearly, Player O can realize this formula with a strategy of size one, but she cannot enforce a realization with respect to some bound $k \in o(2^b)$. □

6 Conclusion

In this work, we presented an approximation algorithm for the PROMPT–LTL realizability problem with doubly-exponential running time with an approximation factor of two. This is an exponential improvement over the fastest known exact algorithms. The algorithm relies on repeated calls to an LTL realizability solver. We have implemented the algorithm using `BoSy` as LTL realizability solver, which implements the bounded synthesis approach. In our proof-of-concept experiments, a tradeoff between the size and the quality of a strategy becomes apparent, which we investigated: we proved upper bounds on the tradeoff, which reduces the search space of our algorithm, and proved matching lower bounds.

Although we presented our results only for PROMPT–LTL, they also hold for the more expressive logics PLTL [2] and PLDL [7], as they can be compiled into Büchi automata of exponential size and as the alternating color technique is applicable to them as well.

There are several open problems to consider in future work. Most importantly, the computational complexity of the exact optimization problem is still open. Similarly, the exact memory requirements of optimal strategies are open: triply-exponential memory is always sufficient [19], but it is open whether doubly-exponential memory suffices as well, as it does for LTL specifications. Other open problems relate to the tradeoffs: we have studied the tradeoff between size and quality of strategies. One can also consider tradeoffs between different parameters in PLTL and PLDL formulas or take the running time into account as well. The former problem is tightly related to the study of the solution space, i.e., the space of the realizable parameter valuations (see [2] for results on the model checking).

References

[1] Martín Abadi, Leslie Lamport & Pierre Wolper (1989): *Realizable and Unrealizable Specifications of Reactive Systems*. In Ausiello et al. [3], pp. 1–17, doi:10.1007/BFb0035748.

[2] Rajeev Alur, Kousha Etessami, Salvatore La Torre & Doron Peled (2001): *Parametric temporal logic for "model measuring"*. *ACM Trans. Comput. Log.* 2(3), pp. 388–407, doi:10.1145/377978.377990.

[3] Giorgio Ausiello, Mariangiola Dezani-Ciancaglini & Simona Ronchi Della Rocca, editors (1989): *ICALP 89*. *LNCS* 372, Springer.

[4] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2012): *Acacia+, a Tool for LTL Synthesis*. In P. Madhusudan & Sanjit A. Seshia, editors: *CAV*, *LNCS* 7358, Springer, pp. 652–657, doi:10.1007/978-3-642-31424-7_45.

[5] Rüdiger Ehlers (2012): *Symbolic Bounded Synthesis*. *Formal Methods in System Design* 40(2), pp. 232–262, doi:10.1007/s10703-011-0137-x.

[6] Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe & Leander Tentrup (2015): *Encodings of Reactive Synthesis*. In: *Quantify 2015*. Available at http://fmv.jku.at/quantify15/Faymonville-et-al-QUANTIFY2015.pdf.

[7] Peter Faymonville & Martin Zimmermann (2014): *Parametric Linear Dynamic Logic*. In Adriano Peron & Carla Piazza, editors: *GandALF*, *EPTCS* 161, pp. 60–73, doi:10.4204/EPTCS.161.8.

[8] Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2011): *Antichains and compositional algorithms for LTL synthesis*. *Formal Methods in System Design* 39(3), pp. 261–296, doi:10.1007/s10703-011-0115-3.

[9] Bernd Finkbeiner & Sven Schewe (2013): *Bounded synthesis*. *STTT* 15(5-6), pp. 519–539, doi:10.1007/s10009-012-0228-z.

[10] Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors (2002): *Automata, Logics, and Infinite Games: A Guide to Current Research*. *LNCS* 2500, Springer.

[11] Barbara Jobstmann & Roderick Bloem (2006): *Optimizations for LTL Synthesis*. In: *FMCAD*, IEEE Computer Society, pp. 117–124, doi:10.1109/FMCAD.2006.22.

[12] Orna Kupferman, Nir Piterman & Moshe Y. Vardi (2006): *Safraless Compositional Synthesis*. In Thomas Ball & Robert B. Jones, editors: *CAV*, *LNCS* 4144, Springer, pp. 31–44, doi:10.1007/11817963_6.

[13] Orna Kupferman, Nir Piterman & Moshe Y. Vardi (2009): *From liveness to promptness*. *Formal Methods in System Design* 34(2), pp. 83–103, doi:10.1007/s10703-009-0067-z.

[14] Orna Kupferman & Moshe Y. Vardi (2005): *Safraless Decision Procedures*. In: *FOCS*, IEEE Computer Society, pp. 531–542, doi:10.1109/SFCS.2005.66.

[15] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS 1977*, IEEE Computer Society, pp. 46–57, doi:10.1109/SFCS.1977.32.

[16] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of a Reactive Module*. In: *POPL*, pp. 179–190, doi:10.1145/75277.75293.

[17] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of an Asynchronous Reactive Module*. In Ausiello et al. [3], pp. 652–671, doi:10.1007/BFb0035790.

[18] Sven Schewe (2007): *Solving Parity Games in Big Steps*. In Vikraman Arvind & Sanjiva Prasad, editors: *FSTTCS 2007*, *LNCS* 4855, Springer, pp. 449–460, doi:10.1007/978-3-540-77050-3_37.

[19] Martin Zimmermann (2013): *Optimal bounds in parametric LTL games*. *Theor. Comput. Sci.* 493, pp. 30–45, doi:10.1016/j.tcs.2012.07.039.

[20] Martin Zimmermann (2015): *Parameterized Linear Temporal Logics Meet Costs: Still not Costlier than LTL*. In Javier Esparza & Enrico Tronci, editors: *GandALF 2015*, *EPTCS* 193, pp. 144–157, doi:10.4204/EPTCS.193.11.