From Liveness to Promptness *

Orna Kupferman, Hebrew University
Nir Piterman, EPFL
Moshe Y. Vardi, Rice University

Abstract

Liveness temporal properties state that something "good" eventually happens, e.g., every request is eventually granted. In Linear Temporal Logic (LTL), there is no a priori bound on the "wait time" for an eventually to be fulfilled. That is, $F\theta$ asserts that θ holds eventually, but there is no bound on the time when θ will hold. This is troubling, as designers tend to interpret an eventuality $\mathbf{F}\theta$ as an abstraction of a bounded eventuality $F^{\leq k}\theta$, for an unknown k, and satisfaction of a liveness property is often not acceptable unless we can bound its wait time. We introduce here PROMPT-LTL, an extension of LTL with the *prompt-eventually* operator $\mathbf{F_p}$. A system S satisfies a PROMPT-LTL formula φ if there is some bound k on the wait time for all prompt-eventually subformulas of φ in all computations of S. We study various problems related to PROMPT-LTL, including realizability, model checking, and assume-guarantee model checking, and show that they can be solved by techniques that are quite close to the standard techniques for LTL.

1 Introduction

Since the introduction of temporal logic into computer science [14], temporal logic, in its many different flavors, has been widely accepted as an appropriate formal framework for the description of on-going behavior of reactive systems [13]. Temporal properties are traditionally classified into *safety* and *liveness* properties [2]. Intuitively, safety properties assert that nothing bad will ever happen during the execution of the system, and liveness properties assert that something good will happen eventually. Temporal properties are interpreted with respect to systems that generate infinite computations. In satisfying liveness properties, there is no bound on the "wait time", namely the time that may elapse until an eventuality is fulfilled. For example, the LTL formula $F\theta$ is satisfied at time i if θ holds at some time $j \geq i$, but $j - i$ is not a priori bounded.

In many applications, it is important to bound the wait time. This has given rise to formalisms in which the eventually operator $\mathbf{F}$ is replaced by a bounded-eventually operator $F^{\leq k}$. The operator is parameterized by some $k \geq 0$, and it bounds the wait time to k [4, 9]. Since we assume that time is discrete, the operator $\mathbf{F}^{\leq k}$ is simply a syntactic sugar for an expression in which the next operator $\mathbf{X}$ is nested. Indeed, $\mathbf{F}^{\leq k}\theta$ is just $\theta \vee \mathbf{X}(\theta \vee \mathbf{X}(\theta \vee \overset{k}{\ldots} \vee \mathbf{X}\theta))$.

A drawback of the above formalism is that the bound k needs to be known in advance, which is not the case in many applications. For example, it may depend on the system, which may not yet be known, or it may change, if the system changes. In addition, the bound may be very large, causing the state-based description of the specification (e.g., an automaton for it) to be very large too. Thus, the common practice is to use liveness properties as an abstraction of such safety properties: one writes $\mathbf{F}\theta$ instead of $\mathbf{F}^{\leq k}\theta$ for an unknown or a too large k.

For some temporal logics, the abstraction is sound, in the sense that if a system S satisfies a liveness property ψ, then there is a bound k, which depends on S, such that S also satisfies the formula obtained from ψ by replacing all occurrences of $\mathbf{F}$ in ψ by $\mathbf{F}^{\leq k}$. For example, it

*A technical report with more technical details can be found on www.cs.rice.edu/∼vardi/prompt.pdf is shown in [9] that in the case of CTL, taking k to be the number of states in S does it. Thus, if a state s satisfies $A\mathbf{F}\theta$, then it also satisfies $A\mathbf{F}^{\leq k}\theta$, for $k = |S|$, and similarly for $E\mathbf{F}\theta$. Intuitively, since θ is a state formula, a wait time that is greater than $|S|$ indicates that the wait time may also be infinite (by looping in a cycle that aught to be taken during the wait time), and may also be shortened to at most $|S|$ (by skipping such cycles).

So the abstraction of safety properties by liveness properties is sound for CTL. Is it sound also for linear temporal logic? Consider the system $\mathcal{S}$ described in Figure 1 below. While S satisfies the LTL formula $\mathbf{FG}q$, there is no $k \geq 0$ such that $\mathcal{S}$ satisfies $\mathbf{F}^{\leq k}\mathbf{G}q$. To see this, note that for each $k \geq 0$, the computation that first loops in the first state for k times and only then continues to the second state, satisfies the eventuality $\mathbf{G}q$ with wait time $k+1$.

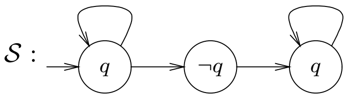

Figure 1: $\mathcal{S}$ satisfies $\mathbf{FG}q$ but does not satisfy $\mathbf{F}^{\leq k}\mathbf{G}q$, for all $k \geq 0$.

It follows that the abstraction of safety properties by liveness properties is not sound in the linear-time approach (which is more popular with users, cf. [7]). This is troubling, as designers tend to interpret eventualities as bounded eventualities, and satisfaction of a liveness property is often not acceptable unless we can bound its wait time.[1]

In this work we introduce and study an extension of LTL that addresses the above problem. In addition to the usual temporal operators of LTL, our logic, PROMPT-LTL, has a new temporal operator that is used for specifying eventualities with a bounded wait time. We term the operator *prompt eventually* and denote it by $\mathbf{F_p}$. Let us define the semantics of PROMPT-LTL formally. For a PROMPT-LTL formula ψ and a bound $k \geq 0$, let ψ^k be the LTL formula obtained from ψ by replacing all occurrences of $\mathbf{F_p}$ by $\mathbf{F}^{\leq k}$. Then, a system S satisfies ψ iff there is $k \geq 0$ such that S satisfies ψ^k.

Note that while the syntax of PROMPT-LTL is very similar to that of LTL, its semantics is defined with respect to an entire system, and not with respect to computations. For example, while each computation π in the system $\mathcal{S}$ from Figure 1 has a bound $k_\pi \geq 0$ such that $\mathbf{G}q$ is satisfied in π with wait time k_π, there is no $k \geq 0$ that bounds the wait time of all computations. It follows that, unlike linear temporal logics, we cannot characterize a PROMPT-LTL formula ψ over a set AP of atomic propositions by a set of computations $L_\psi \subseteq (2^{AP})^\omega$ such that a system S satisfies ψ iff the language of S is contained in L_ψ. On the other hand, unlike branching temporal logics, if two systems agree on their languages, then they agree also on the satisfaction of all PROMPT-LTL formulas. Thus, PROMPT-LTL intermediates between the linear and branching approaches: as in the linear approach, the specification refers to the set of computations of the system rather than its computation tree; as in the branching approach, we cannot consider these computations individually.

We study the basic problems of PROMPT-LTL. Consider a PROMPT-LTL formula ψ over AP. The set AP may be partitioned to sets I and O of input and output signals. Consider also a system S. We study the following problems: *realizability* (is there a finite-state strategy $f : (2^I)^* \to 2^O$ such that all the computations generated by f satisfy ψ?), *model checking* (does S satisfy ψ?), and *assume-guarantee model checking* (given an additional PROMPT-LTL formula φ, is it the case that for all systems S', if $S \| S'$ satisfies φ, then $S \| S'$ also satisfies ψ?). Since a system that satisfies a PROMPT-LTL formula may consist of a single regular computation, the

[1]Note that the reduction of liveness to safety as described in [3] is performed by squaring the state space rather than trying to bound the wait time of eventualities. Thus, it is not related to the discussion in this paper.

satisfiability problem for prompt-LTL can be easily reduced to LTL satisfiability (simply replace all occurrences of $\mathbf{F_p}$ by $\mathbf{F}$). For the other problems, similar reductions do not work, and we have to develop a new technique in order to solve them. Let us describe our technique briefly.

Consider a prompt-LTL formula ψ over AP. Let p be an atomic proposition not in AP. Think about p as a description of one of two colors, say green (p holds) and red (p does not hold). Each computation of the system can be partitioned to blocks such that states of the same block agree on their color. We show that a system S satisfies a PROMPT-LTL formula ψ iff there is some bound $k \geq 0$ such that we can color each computation π of S so that the induced blocks are of length k, and whenever a suffix of π has to satisfy an eventuality, the eventuality is fulfilled within two blocks. Indeed, the latter condition holds iff all eventualities have wait time at most $2k$.

The key idea behind our technique is that rather than search for a bound k for the prompt eventualities, which can be quite large, it is enough to make sure that there is a coloring in which all blocks are of a (not necessarily bounded) finite length, and then use some regularity argument in order to conclude that the size of the blocks could actually be bounded. Forcing the blocks to be of a finite length can be done by requiring the colors to alternate infinitely often. As for regularity, in the case of realizability, regularity follows from the finite-model property of tree automata. In the case of model checking and assume-guarantee model checking, regularity follows from the finiteness of the system.

The complexities that follow from our algorithms are encouraging: reasoning about PROMPT-LTL is not harder than reasoning about LTL: realizability is 2EXPTIME-complete, and model checking and assume-guarantee model checking are PSPACE-complete. For LTL, many heuristics have been studied and applied. Some of them are immediately applicable for PROMPT-LTL (c.f., optimal translations of formulas to automata), and some should be extended to the prompt setting (e.g., bad-cycle detection algorithms). We also study some theoretical aspects of PROMPT-LTL, such as the ability to translate PROMPT-LTL formulas to branching-temporal logics (a translation to the μ-calculus is always possible, but may involve a significant blow up), and the ability to determine whether a PROMPT-LTL formula has an equivalent LTL formula (PSPACE-complete).

2 Prompt Linear Temporal Logic

The logic PROMPT-LTL extends LTL [14] by a *prompt-eventually* operator $\mathbf{F_p}$. The syntax of PROMPT-LTL formulas (in negation normal form) is given by the grammar below, for a set AP of atomic propositions: $\varphi ::= AP \mid \neg AP \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{F_p}\varphi \mid \varphi\mathbf{U}\varphi \mid \varphi\mathbf{R}\varphi$. The semantics of a PROMPT-LTL formula is defined with respect to an infinite word $w = w_0, w_1, \ldots$ over the alphabet 2^{AP}, a position $i \geq 0$ in w, and a bound $k \geq 0$. We use $(w, k, i) \models \varphi$ to indicate that φ holds in location i of w with bound k. The relation $\models$ is defined by induction on the structure of φ as follows.

- For propositions, Boolean connectives, and LTL temporal operators, the definition is independent of k and coincides with the one for LTL.[2]
- $(w, i, k) \models \mathbf{F_p}\varphi$ iff there exists j such that $i \leq j \leq i + k$ and $(w, j, k) \models \varphi$.

We use $\mathbf{F}\theta$ and $\mathbf{G}\theta$ to abbreviate *true*$\mathbf{U}\theta$ and *false*$\mathbf{R}\theta$, respectively. Note that the negation of $\mathbf{F_p}$ is not expressible in PROMPT-LTL, thus the logic is not closed under negation. Given a PROMPT-LTL formula φ, let $live(\varphi)$ be the LTL formula obtained from φ by replacing every prompt-eventually operator $\mathbf{F_p}$ by a standard eventually operator $\mathbf{F}$.

[2]Recall that in LTL we have that $w, i \models \theta R\psi$ iff for all $j \geq i$, if $\pi, j \not\models \psi$, then for some k, $i \leq k < j$, we have $\pi, k \models \theta$.

A *(labeled) transition system* is $\mathcal{S} = \langle AP, S, \rho, s_0, L\rangle$, where AP is a finite set of atomic propositions, S is a finite set of states, $\rho \subseteq S \times S$ is a total transition relation, $s_0 \in S_0$ is an initial state, and $L : S \to 2^{AP}$ maps each state s to the set of propositions that hold in s. When $\rho(s, s')$, we say that s' is a *successor* of s, and s is a *predecessor* of s'. A *computation* of $\mathcal{S}$ is an infinite sequence of states $\pi = s_0, s_1, \ldots \in S^\omega$ such that for all $i \geq 0$, we have $\rho(s_i, s_{i+1})$. The computation π induces the *trace* $L(\pi) = L(s_0) \cdot L(s_1) \cdots$.

Given an system $\mathcal{S}$ and a PROMPT-LTL formula φ over AP, we say that $\mathcal{S}$ satisfies φ, denoted $\mathcal{S} \models \varphi$, if there exists some $k \geq 0$ such that for all traces w of $\mathcal{S}$, we have $(w, 0, k) \models \varphi$. We then say that $\mathcal{S}$ *satisfies* φ *with bound* k. Note that when $\mathcal{S} \not\models \varphi$, then for every $k \geq 0$, there exists an trace w such that $(w, 0, k) \not\models \varphi$.

In [1], Alur et al. study an extension of LTL in which the temporal operators $\mathbf{F}$ and $\mathbf{G}$ are replaced by the operators $\mathbf{F}_{\leq x}, \mathbf{F}_{>y}, \mathbf{G}_{\leq x}$, and $\mathbf{G}_{>y}$, for variables x and y (the same variable may be used in different operators, but, to ensure decidability, the same variable cannot participate in both a lower and an upper bound). Given a system $\mathcal{S}$ and a formula in their logic, one can ask whether there is an assignment to the variables for which the system satisfies the formula, with the expected interpretation of the bounded operators.[3] Our logic can be viewed as a special case of the logic studied in [1], in which only eventualities are parameterized, and only with upper bounds. The algorithms suggested by Alur et al. are rather involved. By giving up the operators $\mathbf{F}_{>y}, \mathbf{G}_{\leq x}$, and $\mathbf{G}_{>y}$, whose usefulness is debatable, we get a much simpler model-checking algorithm, which is also similar to the classical LTL model-checking algorithm. We are also able to a solve the realizability and the assume-guarantee model checking problems.

The Alternating-Color Technique We now describe the key idea of our technique for reasoning about PROMPT-LTL formulas. Let p be an atomic proposition not in AP. We think about p as a description of one of two colors, say green (p holds) and red (p does not hold). Each computation of the system can be partitioned to blocks such that states of the same block agree on their color. Our technique is based on the idea that bounding the wait time of prompt eventualities can be reduced to forcing all blocks to be of a bounded length, and forcing all eventualities to be fulfilled within two blocks, We now make this intuition formal.

Consider a word $w = \sigma_0, \sigma_1, \ldots \in (2^{AP})^\omega$. Let p be a proposition not in AP. A *p-coloring* of w is a word $w' = \sigma'_0, \sigma'_1, \ldots \in (2^{AP \cup \{p\}})^\omega$ such that w' agrees with w on the propositions in AP; i.e., for all $i \geq 0$, we have $\sigma'_i \cap AP = \sigma_i$. We refer to the assignment to p as the *color* of location i and say that i is green if $p \in \sigma'_i$ and is red if $p \notin \sigma'_i$. We say that *p changes at i* if either $i = 0$ or the colors of $i-1$ and i are different (that is, $p \in \sigma'_{i-1}$ iff $p \notin \sigma'_i$). We then call i a *p-change point*. A subword $\sigma'_i, \ldots, \sigma'_{i'}$ is a *p-block* if all positions in the subwords have the same color, and i and $i'+1$ are p-change points. We then say that i and $i'+1$ are adjacent p-change points. For $k \geq 0$, we say that w' is *k-spaced*, *k-bounded*, and *k-tight* (with respect to p) if w' has infinitely many blocks, and all the blocks are of length at least k, at most k, and exactly k, respectively.

Consider the formula $alt_p = \mathbf{GF}p \wedge \mathbf{GF}\neg p$. It requires that the proposition p alternates forever. Given a PROMPT-LTL formula φ, let $rel_p(\varphi)$ denote the formula obtained from φ by (recursively) replacing each subformula of the form $\mathbf{F_p}\psi$ by the LTL formula $(p \to (p\mathbf{U}(\neg p\mathbf{U}\psi))) \wedge (\neg p \to (\neg p\mathbf{U}(p\mathbf{U}\psi)))$. Note that the definition is recursive, thus $rel_p(\varphi)$ may be exponentially larger than φ. The number of subformulas of $rel_p(\varphi)$, however, is linear in the number of subformulas of φ, and it is this number that plays a role in the complexity analysis (equivalently, the size of the DAG-presentation of $c(\varphi)$ is linear in the size of the DAG presentation of φ). For a PROMPT-LTL formula φ, we define $c(\varphi) = alt_p \wedge rel_p(\varphi)$. Thus, $c(\varphi)$ forces the computation

[3] The work in [1] studies many more aspects of the logic, like the problem of deciding whether the formula is satisfied with *all* assignments, the problem of finding an optimal assignment, and other decidability issues.

to be partitioned into infinitely many blocks, and requires each prompt eventuality to be satisfied in the current or next block or in the position immediately after the next block (within two blocks, for short),

Lemma 2.1 *Consider a* PROMPT-LTL *formula φ, a word w, and a bound $k \geq 0$.*

1. *If $(w, 0, k) \models \varphi$, then for every k-spaced p-coloring w' of w, we have $(w', 0) \models c(\varphi)$.*
2. *If there is a k-bounded p-coloring w' of w such that $(w', 0) \models c(\varphi)$, then $(w, 0, 2k) \models \varphi$.*

Proof: Consider the first claim. Since φ does not use the proposition p, then clearly $(w', 0, k) \models \varphi$. Annotate every location in w' by the subformulas of φ that hold in this location. Every location annotated by $\mathbf{F_p}\psi$ satisfies either $p\mathbf{U}(\neg p\mathbf{U}\psi)$ or $\neg p\mathbf{U}(p\mathbf{U}\psi)$. Indeed, w' is k-spaced, and $(w, i, k) \models \mathbf{F_p}\psi$ if there exists $j \leq k$ such that $(w, i + j, k) \models \psi$. Hence, $(w', 0) \models c(\varphi)$.

Consider the second claim. Let w' be a k-bounded p-coloring of w such that $(w', 0) \models c(\varphi)$, Annotate every location in w' by the subformulas of $c(\varphi)$ that hold in this location. Consider a location i annotated by $p\mathbf{U}(\neg p\mathbf{U}\psi)$ or $\neg p\mathbf{U}(p\mathbf{U}\psi)$. Since w' is k-bounded, it follows that for some $j \leq i + 2k$, the location j is annotated by ψ. Therefore, location i satisfies $\mathbf{F_p}\psi$. Hence, $(w, 0, 2k) \models \varphi$. □

The alternating-color technique sets the basis to reasoning about a PROMPT-LTL formula φ by reasoning about the LTL formula $c(\varphi)$. The formula $c(\varphi)$, however, does not require the blocks in the colored computation to be of a bounded length. Indeed, the conjunct alt_p only forces the colors to be finite, and it does not prevent, say, a p-coloring in which each block is longer than its predecessor block, and which is not k-bounded, for all $k \geq 0$. Thus, the challenge of forcing the p-coloring to be k-bounded for some k remains, and we have to address it in each of the decision procedures described in the following sections.

3 Realizability

Given an LTL formula ψ over the sets I and O of input and output signals, the *realizability problem* for ψ is to decide whether there is a *strategy* $f : (2^I)^* \to 2^O$ such that all the computations generated by f satisfy ψ [16]. Formally, a computation $w \in (2^{I \cup O})^\omega$ is generated by f if $w = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2), \ldots$ and for all $j \geq 0$, we have $o_j = f(i_0 \cdot i_1 \cdots i_j)$. Thus, the interaction is initiated by the environment that generates i_0, and the first state in the computation is labeled $i_0 \cup f(i_0)$. Then, the environment generates i_1, and the second state in the computation is $i_1 \cup f(i_0 \cdot i_1)$, and so on. It is known that if some strategy that realizes ψ exists, then there also exists a *regular strategy* (i.e, a strategy generated by a finite-state *transducer*) that realizes ψ [6]. Formally, a transducer is $\mathcal{D} = \langle I, O, Q, \eta, q_0, L\rangle$, where I and O are the sets of input and output signals, Q is a finite set of states, $\eta : Q \times 2^I \to Q$ is a deterministic transition function, $q_0 \in Q$ is an initial state, and $L : Q \to 2^O$ maps each state to a set of output signals. The transducer $\mathcal{D}$ generates f in the sense that for every $\tau \in (2^I)^*$, we have $f(\tau) = L(\eta(\tau))$, with the usual extension of η to words over 2^I.

We first show that PROMPT-LTL realizability of a formula φ cannot be simply reduced to the realizability of $live(\varphi)$. Thus, we describe a formula φ such that $live(\varphi)$ is realizable, but for every strategy f that realizes φ and for every candidate bound $k \geq 0$, there is a computation w generated by f such that $(w, 0, k) \not\models \varphi$. Let $I = \{i\}$ and $O = \{o\}$. We define

$$\varphi = o \ \wedge \ (\mathbf{G}(i \to o)) \ \wedge \ ((\mathbf{X}\neg o)\mathbf{R}i) \ \wedge \ (\mathbf{F_p}\mathbf{G}o).$$

Thus, a computation satisfies φ if o holds in the present and whenever i holds, whenever i does not hold in some position, then o does not hold in this position or in an earlier one, and the computation prompt-eventually reaches a position from which o holds everywhere. It is not hard to see that $live(\varphi)$ is realizable. Indeed, the strategy that sets o to *true* everywhere except in the first time that i is *false* realizes $live(\varphi)$. On the other hand, φ is not realizable. To see this, note that the position in which the input i is set to *false* can be delayed arbitrarily by the environment, forcing a delay also in the fulfilment of the $\mathbf{G}o$ eventuality. Thus, for every candidate bound $k \geq 0$, the input sequence in which i is *false* at the $(k+1)$-th position cannot be extended to a computation that satisfies $\mathbf{F_p G}o$ with bound k.

The good news is that while realizability of φ cannot be reduced to the realizability of $live(\varphi)$, it can be reduced to the realizability of $c(\varphi)$. Intuitively, it follows from the fact that in a regular strategy, the fact that all blocks are of a finite length does imply that they are also of a bounded length. Formally, we have the following.

Theorem 3.1 *A* PROMPT-LTL *formula φ over input signals I and output signals O is realizable iff the LTL formula $c(\varphi)$ over input signals I and output signals $O \cup \{p\}$ is realizable.*

Proof: Suppose that φ is realizable. Then there exists a strategy $f : (2^I)^* \to 2^O$ and a bound $k \geq 0$ such that all the computations w of f satisfy $(w, 0, k) \models \varphi$. We extend f to a strategy $f' : (2^I)^* \to 2^{O \cup \{p\}}$ that realizes $c(\varphi)$. Intuitively, we add to the computations of f a p-coloring that is $2k$-tight. Formally, for $\tau \in (2^I)^*$, we define $f'(\tau) = f(\tau) \cup \{p\}$ if $|\tau| \bmod 2k$ is between 0 and $k-1$ and $f'(\tau) = f(\tau)$ if $|\tau| \bmod 2k$ is between k and $2k-1$. Consider a computation w induced by f'. Note that w is k-tight and it satisfies φ. Therefore, by Lemma 2.1, we conclude that $w \models c(\varphi)$.

Assume now that $c(\varphi)$ is realizable. Let $f : 2^I \to 2^{O \cup \{p\}}$ be a regular strategy that realizes it. We show that the strategy $f' : 2^I \to 2^O$ obtained from f by projecting it on I (that is, for all $\tau \in (2^I)^*$, we have $f'(\tau) = f(\tau) \cap I$) realizes φ. Let n be the number of states in the transducer that generates f. We show that all the computations generated by f' satisfy φ with bound $2n+2$. Consider a computation w of f'. We claim that w is $(n+1)$-bounded. To see this, assume by way of contradiction that w has adjacent p-change points i and j such that $j - i > n + 1$. Let $\mathcal{D} = \langle 2^I, 2^O, Q, \eta, q_0, L \rangle$ be the transducer that generates f, and let $q_0, q_1, q_2, \ldots$ be the run of $\mathcal{D}$ that corresponds to w. Since $\mathcal{D}$ has n states, there exists a state q and locations i' and j' such that $i \leq i' < j' \leq j$ and $q_{i'} = q_{j'}$. Thus, some state repeats along the p-block that starts at i and ends at $j-1$. Then, the run $q_0, q_1, \ldots, q_{i'-1}, (q_{i'}, \ldots, q_{j'-1})^\omega$ is also a run of $\mathcal{D}$. This run, however, generates a computation of f that does not satisfy alt_p, contradicting the fact that f realizes $c(\varphi)$. So, every computation w of f' is $(n+1)$-bounded, and it satisfies $c(\varphi)$. Therefore, by Lemma 2.1, we conclude that $(w, 0, 2n+2) \models \varphi$. $\square$

Since LTL realizability is 2EXPTIME-complete and every LTL formula is also a PROMPT-LTL formula, we can conclude:

Theorem 3.2 *The problem of prompt realizability is 2EXPTIME-complete in the size of the formula.*

As demonstrated above, the alternating-color technique is very powerful in the case of realizability. Indeed, the challenge of forcing the p-coloring to be k-bounded for some k is taken care of by the regularity of the strategy. We now proceed to the model-checking problem, where a reduction to $c(\varphi)$ is not sufficiently strong.

4 Model Checking

In this section we describe an algorithm for solving the model-checking problem for PROMPT-LTL. An alternative algorithm is described for the richer parameterized linear temporal logic in [1]. Our algorithm is much simpler, and it deviates from the standard LTL model-checking algorithm only slightly. In addition, as we show in Section 6, the idea behind our algorithm can be applied also in order to solve assume-guarantee model checking, which is not known to be the case with the algorithm in [1]. Our algorithm is based on the automata-theoretic approach to LTL model-checking, and we first need some definitions.

A *nondeterministic Büchi word automaton* (NBW for short) is $\mathcal{A} = \langle \Sigma, S, \delta, s_0, \alpha \rangle$, where Σ is a finite alphabet, S is a finite set of states, $\delta : S \times \Sigma \to 2^S$ is a transition function, $s_0 \in S$ is an initial state, and $\alpha \subseteq S$ is a *Büchi* acceptance condition. A *run* of $\mathcal{A}$ on a word $w = w_0 \cdot w_1 \cdots$ is an infinite sequence of states $s_0, s_1, \ldots$ such that s_0 is the initial state and for all $j \geq 0$, we have $s_{j+1} \in \delta(s_j, w_j)$. For a run $r = s_0, s_1, \ldots$, let $\inf(r) = \{s \in S \mid s = s_i \text{ for infinitely many } i\text{'s}\}$ be the set of all states occurring infinitely often in the run. A run is *accepting* if $\inf(r) \cap \alpha \neq \emptyset$. That is, the run visits infinitely many states from α. A word w is *accepted* by $\mathcal{A}$ if there exists some accepting run of $\mathcal{A}$ over w. The *language* of $\mathcal{A}$, is the set of words accepted by $\mathcal{A}$.

Theorem 4.1 [20] *For every LTL formula φ over AP there exists an NBW $\mathcal{A}_\varphi$ over the alphabet 2^{AP} such that $\mathcal{A}_\varphi$ accepts exactly all words that satisfy φ. The number of states of $\mathcal{A}_\varphi$ is at most exponential in the number of subformulas of φ.*

In order to check whether a system $\mathcal{S}$ satisfies an LTL formula φ, one takes the product of $\mathcal{S}$ with the NBW $\mathcal{A}_{\neg\varphi}$ and tests the product for non-emptiness [19]. Indeed, a path in this product witnesses a computation of $\mathcal{S}$ that does not satisfy φ. As discussed in Section 1, in the case of PROMPT-LTL we cannot translate formulas to languages. Moreover, we also cannot simply apply the alternating-color technique: even if we check the nonemptiness of the product of the system (an augmentation of it in which the proposition p behaves nondeterministically, thus all p-colorings are possible) with the automaton for $alt_p \wedge \neg rel_p(\varphi)$, a path in this product only implies that for some bound $k \geq 0$, the formula φ is not satisfied in $\mathcal{S}$ with bound k. For proving that $\mathcal{S}$ does not satisfy φ we have to prove something stronger, namely, that φ is not satisfied in $\mathcal{S}$ with bound k, for *all* bounds $k \geq 0$. For that, we do take the product of the system with the automaton for $alt_p \wedge \neg rel_p(\varphi)$, but add a twist to the nonemptiness check: we search for a path in the product in which each p-block contains at least one state that repeats. Such a state indicates that for all bounds $k \geq 0$, the p-block can be pumped to a p-block of length greater than k, implying that φ cannot be satisfied in $\mathcal{S}$ with bound k. We now formalize this intuition.

A *colored Büchi graph* is a tuple $G = \langle \{p\}, V, E, v_0, L, \alpha \rangle$, where p is a proposition, V is a set of vertices, $E \subseteq V \times V$ is a set of edges, $v_0 \in V$ is an initial vertex, $L : V \to 2^{\{p\}}$ describes the color of each vertex, and $\alpha \subseteq V$ is a set of accepting states. A path $\pi = v_0, v_1, v_2, \ldots$ of G is *pumpable* if all its p-blocks have at least one state that repeats. Formally, for all $i \geq 0$, if i and i' are adjacent p-change points, then there are positions j and j' such that $i \leq j < j' < i'$ and $v_j = v_{j'}$. Also, π is *fair* if it visits α infinitely often. The *pumpable nonemptiness* problem is to decide, given G, whether is has a pumpable fair path.

Let $\overline{c}(\varphi) = alt_p \wedge \neg rel_p(\varphi)$. That is, we relativize the satisfaction of $\mathbf{F_p}$ to the new proposition p, negate the resulting formula, and require the proposition p to alternate forever. Let $\mathcal{A}_{\overline{c}(\varphi)} = \langle 2^{AP \cup \{p\}}, Q, \delta, q_0, \alpha \rangle$ be the NBW for $\overline{c}(\varphi)$ per Theorem 4.1. Consider a system $\mathcal{S} = \langle AP, S, \rho, s_0, L \rangle$. We now define the product of $\mathcal{S}$ with $\mathcal{A}_{\overline{c}(\varphi)}$ by means of a colored Büchi graph. Note that $\mathcal{S}$ does not refer to the proposition p, and we duplicate its state space in order to have in the product all possible p-colorings of computations in $\mathcal{S}$. Thus, the product is $\mathcal{P} = \langle \{p\}, S \times \{\{p\}, \emptyset\} \times Q, M, \langle s_0, \{p\}, q_0 \rangle, L, S \times \{\{p\}, \emptyset\} \times \alpha \rangle$, where $M(\langle s, c, q \rangle, \langle s', c', q' \rangle)$ iff $\rho(s, s')$ and $q' \in \delta(q, L(s) \cup c)$, and $L(\langle s, c, q \rangle) = c$.

It is not hard to see that a path $\pi = \langle s_0, c_0, q_0 \rangle, \langle s_1, c_1, q_1 \rangle, \langle s_2, c_2, q_2 \rangle, \ldots$ in $\mathcal{P}$ corresponds to a computation $s_0, s_1, s_2, \ldots$ of $\mathcal{S}$, a p-coloring $L(s_0) \cup c_0, L(s_1) \cup c_1, L(s_2) \cup c_2, \ldots$ of the trace that the computation induces, and a run $q_0, q_1, q_2, \ldots$ of $\mathcal{A}_{\overline{c}(\varphi)}$ on this p-coloring.

Theorem 4.2 *The system $\mathcal{S}$ does not satisfy φ iff the product of $\mathcal{S}$ and $\mathcal{A}_{\overline{c}(\varphi)}$ is pumpable nonempty.*

Proof: Assume first that $\mathcal{S} \not\models \varphi$. Then, for every bound $k \geq 0$, there exists a computation π_k of $\mathcal{S}$ such that $(\pi_k, 0, 2k) \not\models \varphi$. Let k be larger than $|S| \cdot |Q|$ and let π_k be as above. Since $(\pi_k, 0, 2k) \not\models \varphi$, then, by Lemma 2.1, for all k-bounded p-coloring π'_k of π_k, we have $(\pi'_k, 0) \not\models c(\varphi)$. Consider the k-tight p-coloring π'_k of π_k that starts with a green block. By the above, $(\pi'_k, 0) \not\models c(\varphi)$. Also, clearly, $(\pi'_k, 0) \models alt_p$. Thus, $(\pi'_k, 0) \models \overline{c}(\varphi)$. In addition, since $k > |S| \cdot |Q|$, every path in the product $\mathcal{P}$ that corresponds to a k-tight p-coloring of π_k is pumpable. Hence, the product of π'_k with an accepting run of $\mathcal{A}_{\overline{c}(\varphi)}$ is a pumpable fair path in $\mathcal{P}$.

Assume now that $\mathcal{P}$ contains a pumpable fair path $\pi = \langle s_0, c_0, q_0 \rangle, \langle s_1, c_1, q_1 \rangle, \langle s_2, c_2, q_2 \rangle, \ldots$. We claim that for every $k \geq 0$, we can pump the computation $s_0, s_1, s_2, \ldots$ of $\mathcal{S}$ to a computation that does not satisfy φ with bound k. To see this, note that for each k, we can pump the path π to a fair path π_k such that the p-coloring of the trace that corresponds to π_k is k-spaced and satisfies $\neg rel_p(\varphi)$. Hence, by Lemma 2.1, it does not satisfy φ with bound k. $\square$

In Section 5, we study the problem of deciding whether a colored Büchi graph is pumpable-nonempty, and prove that it is in NLOGSPACE and can also be solved in linear time. This, together with Theorems 4.1 and 4.2, imply the upper bound in the following theorem. The lower bound follows from the known lower bound for LTL.

Theorem 4.3 *The model-checking problem for* PROMPT-LTL *is PSPACE-complete and can be solve in time exponential in the length of the formula and linear in the size of the system.*

Note that while the pumpable nonemptiness problem to which PROMPT-LTL model-checking is reduced is a variant of the nonemptiness problem to which LTL model checking is reduced, the construction of the product is almost the same. In particular, the extensive work on optimal compilation of LTL formulas to NBW (see survey in [18]), is applicable to our solution too.

Remark 4.4 The model-checking algorithm of the parametric linear temporal logic of [1] is based on the observation that if a PROMPT-LTL formula φ is satisfied in a system $\mathcal{S}$, then it is satisfied with bound k, for some k that is exponential in φ and polynomial in $\mathcal{S}$. One cannot hope to improve this bound. Indeed, for every $n \geq 1$, we can define a PROMPT-LTL formula ψ_n of size linear in n such that a systems satisfies ψ_n iff in all its computations, the atomic proposition q corresponds to an n-bit counter, and the value of the counter promptly eventually reaches $2^n - 1$. Clearly, ψ_n is promptly satisfied, but the minimal bound k with which ψ_n is satisfied with bound k (in some system) is exponential in n. $\square$

5 Algorithms for Colored Büchi Graphs

In Section 4 we reduced model-checking for PROMPT-LTL to a pumpable nonemptiness problem for colored Büchi graphs. In this section we solve this problems, and provide space and time bounds.

Theorem 5.1 *The pumpable nonemptiness problem for colored Büchi graphs is NLOGSPACE-complete and can be solved in linear time.*

Proof: Let $G = \langle \{p\}, V, E, v_0, L, \alpha \rangle$. We start with an algorithm in NLOGSPACE. It is not hard to see that it is enough to search for pumpable fair paths of the form uw^ω where $u, w \in V^+$. In addition, we can assume that $|u|$ is a p-change point, that is, the color of the last vertex in u is different from the color of the first vertex in w, and in addition that the first p-block in w visits α.

It is well known that we can check whether a vertex v is reachable from a vertex v' in NLOGSPACE. We guess a successor v'' of v, if $v'' = v'$ the answer is yes, otherwise we check whether v' is reachable from v''. The algorithm requires logarithmic space in order to store the vertices v, v' and v''.

In order to find a pumpable fair path we have to iterate the search of paths described above[4]. We say that a vertex v' is *block-reachable* from v if there exists a path from v to v' such that all vertices on the path agree on their color. Block-reachability can be established by an algorithm similar to the above where the search is restricted to vertices that agree with v and v' on their color. We say that vertex v' is *pump-block-reachable* from v if v' is block-reachable from v and in addition some vertex repeats on the path from v to v'. We can establish that v' is pump-block-reachable from v by an algorithm similar to the above. We guess a vertex v'' that agrees with v and v' on their color, ensure that v'' is block-reachable from v, that v'' is block-reachable from itself, and that v' is block-reachable from v''. A simple modification of the above can check that v' is pump-block-reachable from v by a path that visits α.

Using the pump-block-reachable check described above we do the following. We guess a vertex v_1 that is the first vertex in w. We check that v_1 is reachable from v_0 with a sequence of pump-block-reachable steps. That is, to make one step from node v we guess a node v' that does not agree with v on its color. We guess a predecessor v'' of v' that does agree with v on its color and check that v'' is pump-block-reachable from v. Then we continue the search from v'. Once we have established that v_1 is reachable from v_0, we guess a vertex v_2, make sure that some predecessor v_1' of v_2 is pump-block-reachable from v_1 with a path that visits α. Finally, we check that v_1 is reachable from v_2 by a sequence of pump-block-reachable steps (as before).

Since the reachability problem in directed graphs is in NLOGSPACE, our algorithm can be implemented in NLOGSPACE.

We now move to the time complexity. For standard Büchi nonemptiness, one looks for a reachable nontrivial strongly connected component that intersects α. In the colored case, we should further check that each p-block in the path can be pumped. We do this by making sure that every green p-block contains at least one vertex that belongs to a nontrivial strongly connected component in the graph of the green vertices, and similarly for the red p-blocks.

Consider the graph $G_g = \langle V_g, E_g \rangle$ obtained from G by restricting attention to green vertices. Thus, $V_g = \{v \in V \mid L(v) = \{p\}\}$ and $E_g = E \cap (V_g \times V_g)$. The graph $G_r = \langle V_r, E_r \rangle$ is defined similarly. We can find the maximal strongly connected components (MSCC) of G_g and G_r in linear time [17] (note we are interested also in MSCCs that are not reachable from v_0 in G_g and G_r). Let $S_g \subseteq V_g$ and $S_r \subseteq V_r$ denote the union of all non-trivial MSCCs in G_g and G_r, respectively.

Let $\mathit{back}_g(S_g)$ be the vertices that can reach some vertex in S_g and let $\mathit{e\text{-}back}_g(S_g)$ be the edges that are used to reach these vertices. Formally, we define $\mathit{back}_0^g(S_g) = S_g$ and $\mathit{back}_{i+1}^g(S_g) = \{v \in V_g \mid \exists v' \in \mathit{back}_i^g(S_g) \text{ and } (v, v') \in E_g\}$. Then, $\mathit{back}_g(S_g) = \bigcup_{i \geq 0} \mathit{back}_i^g(S_g)$ and $\mathit{e\text{-}back}_g(S_g) = (E_g \cap (S_g \times S_g)) \cup \bigcup_{i \geq 0} \{(v, v') \in E_g \mid v \in \mathit{back}_i^g(S_g) \text{ and } v' \in \mathit{back}_{i+1}^g(S_g)\}$. In a similar way, we define $\mathit{forward}_g(S_g)$ to be the set of vertices that are reachable from some vertex in S_g and define $\mathit{e\text{-}forward}_g(S_g)$ to be the edges that are used to reach these vertices. The sets back_r, $\mathit{e\text{-}back}_r$, $\mathit{forward}_r$, and $\mathit{e\text{-}forward}_r$ are defined similarly. Another type of edges we need are edges between p-blocks. Let $E_{g \to r} = E \cap (\mathit{forward}_g(S_g) \times \mathit{back}_r(S_r))$ be the set of edges along which the color changes from green to red, and let $E_{r \to g} = E \cap (\mathit{forward}_r(S_r) \times \mathit{back}_g(S_g))$ be the set of edges along which the color changes from red to green.

[4]This is similar to the proof that emptiness of Büchi graphs is solvable in NLOGSPACE. We guess a vertex $v \in \alpha$, show that it is reachable from v_0 and that it is reachable from itself.

Consider now the subgraph $G' = \langle V', E' \rangle$ of G, where $V' = \mathit{back}_g(S_g) \cup \mathit{forward}_g(S_g) \cup \mathit{back}_r(S_r) \cup \mathit{forward}_r(S_r)$, and

$$E' = E_{g \to r} \cup E_{r \to g} \cup \mathit{e\text{-}forward}_g(S_g) \cup \mathit{e\text{-}forward}_r(S_r) \cup \mathit{e\text{-}back}_g(S_g) \cup \mathit{e\text{-}back}_r(S_r).$$

Intuitively, the graph G' contains exactly all the pumpable computations of G. Indeed, along each p-block, there must exists a vertex that belongs to an MSCC of the graph of the corresponding color. We prove that G is pumpable nonempty iff G' has some non-trivial MSCC that contains a vertex from α.

Suppose that there is some non-trivial MSCC in G' with a vertex v from α. Let $\pi = v_0, v_1, \ldots,$ be a path from v_0 that visits v infinitely often. We show that we can find a pumpable path π' from v_0 that visits v infinitely often. Consider the path π. Every edge in π is in one of the six sets of edges that comprise E'. Partition π to the p-blocks that comprise it $\pi_0, \pi_1, \ldots$, where $\pi_i = v_{j_i}, v_{j_i+1}, \ldots, v_{j_{i+1}-1}$. We construct by induction a path π' in which every block is pumpable. Consider the blocks π_0 and π_1. Suppose that π_0 is green and π_1 is red (the dual case is similar). It follows that $(v_{j_1-1}, v_{j_1}) \in E_{g \to r}$ and that $v_{j_1-1} \in \mathit{forward}_g(S_g)$. Hence, there has to be a path in G_g between v_0 and v_{j_1-1} that passes through some MSCC in G_g. Let π'_1 be the path that goes from v_0 to v_{j_1-1} through this MSCC and passes at least some vertex in this MSCC twice. Consider the blocks $\pi_{i-1}, \pi_i, \pi_{i+1}$. Assume that π_{i-1} is green, π_i is red, and π_{i+1} is green. Then $(v_{j_i-1}, v_{j_i}) \in E_{g \to r}$ and $(v_{j_{i+1}-1}, v_{j_{i+1}}) \in E_{r \to g}$. It follows that $v_{j_i} \in \mathit{back}_r(S_r)$ and $v_{j_{i+1}-1} \in \mathit{forward}_r(S_r)$. Then π_i visits some vertex v in S_r, and we set π'_i to be a path between v_{j_i} and $v_{j_{i+1}-1}$ that visits v twice. The case where the path π has only finitely many p-blocks can be handled similarly.

Consider a pumpable fair path π in G. It is simple to see that all the edges on π are present also in G'. It follows that in G' there is some reachable MSCC that visits α.

We analyze the time it takes to construct G' and to check whether it has a non-trivial MSCC that intersects α. Clearly, the MSCC decomposition of G_g and G_r can be done in linear time. The search for back_g and $\mathit{forward}_g$ is done by a DFS on G_g and its inverse. During the search, the edges in $\mathit{e\text{-}back}_g$ and $\mathit{e\text{-}forward}_g$ can be marked. The case of back_r and $\mathit{forward}_r$ is similar. This stage can be completed in linear time as well. Finally, the MSCC decomposition of G' is completed again in linear time, thus the overall running time is linear. □

We note than our algorithm is based in MSCC-decomposition. It is an open question whether a linear-time algorithm based on nested depth-first-search can be found (see discussion of these types of algorithms in [18]).

In Section 6 we reduce assume-guarantee model-checking for PROMPT-LTL to a pumpable nonemptiness problem for colored Büchi graphs with two colors. We now turn to consider such graphs.

A *colored Büchi graph of degree two* is a tuple $G = \langle \{p, q\}, V, E, v_0, L, \alpha \rangle$. It is similar to a colored Büchi graph, only that now there are two sets of colors, described by p and q. Accordingly, $L : V \to 2^{\{p,q\}}$. Also, α is a generalized Büchi condition of index 2, thus $\alpha = \{\alpha_1, \alpha_2\}$. A path $\pi = v_0, v_1, v_2, \ldots$ of G is *pumpable* if we can pump all its q-blocks without pumping its p-blocks. Formally, for all $i \geq 0$, if i and i' are adjacent q-change points, then there are positions j, j', and j'' such that $i \leq j < j' < j'' < i'$, $v_j = v_{j''}$ and $p \in L(v_j)$ iff $p \notin L(v_{j'})$. Also, π is *fair* if it visits both α_1 and α_2 infinitely often. The *pumpable nonemptiness* problem is to decide, given G, whether it has a pumpable fair path.

Theorem 5.2 *The pumpable nonemptiness problem for colored Büchi graphs of degree two is NLOGSPACE-complete and can be solved in linear time.*

Proof: As before, we are searching for a pumpable fair path of the form uw^ω where $u, w \in V^+$ and that $|u|$ is a p-change point.

The search for a q-block that connects v to v' and contains a pumpable section is partitioned as follows. We guess v'' that agrees with v on the assignment to q and search for a path from v to v' while maintaining the same q assignment. Then we search for a path from v'' to itself. This path has to maintain the same assignment to q, however, has to change the assignment to p at least once. Finally, we search for a path from v'' to v' that maintains the same assignment to q.

Using this basic reachability algorithm we do the following. We guess a vertex v_1 that is the first vertex in w. We make sure that v_1 is reachable from v_0 with a sequence of such moves. We make sure that v_1 is reachable from itself with a sequence of such moves that visit both α_1 and α_2. The entire algorithm can be implemented in NLOGSPACE.

We now describe a linear-time algorithm for solving the problem. Assume that v_0 has no incoming edges. Consider the graph $G_q = \langle V_q, E_q \rangle$ where V_q is the subset of vertices labeled by q, i.e. $V_q = \{v \in V \mid q \in L(v)\}$ and $E_q = E \cap (V_q \times V_q)$. The graph $G_{\overline{q}} = \langle V_{\overline{q}}, E_{\overline{q}} \rangle$ is defined similarly for vertices not labeled by q. We can analyze the maximal strongly connected components (MSCC) of G_q and $G_{\overline{q}}$ in linear time [17]. We restrict our attention to MSCCs that contain both vertices labeled by p and vertices not labeled by p. Let $S_q \subseteq V_q$ denote the union of all non-trivial MSCCs M in G_q such that there exist $v, v' \in M$ such that $p \in L(v)$ and $p \notin L(v')$. Define $S_{\overline{q}} \subseteq V_{\overline{q}}$ similarly.

For $\beta \in \{q, \overline{q}\}$, the sets $\textit{back}_\beta(S_\beta)$, $\textit{e-back}_\beta(S_\beta)$, $\textit{forward}_\beta(S_\beta)$, $\textit{e-forward}_\beta(S_\beta)$ are defined as before.

Consider now the subgraph $G' = \langle \{p\}, V', E', v_0, L, \alpha \rangle$ of G where $V' = \textit{back}_q(S_q) \cup \textit{forward}_q(S_q) \cup \textit{back}_{\overline{q}}(S_{\overline{q}}) \cup \textit{forward}_{\overline{q}}(S_{\overline{q}})$ and E' is as follows.

$$E' = E_{q \to \overline{q}} \cup E_{\overline{q} \to q} \cup \textit{e-forward}_q(S_q) \cup \textit{e-forward}_{\overline{q}}(S_{\overline{q}}) \cup \textit{e-back}_q(S_q) \cup \textit{e-back}_{\overline{q}}(S_{\overline{q}})$$

where $E_{q \to \overline{q}}$ and $E_{\overline{q} \to q}$ are defined as follows.

$$\begin{array}{rcl} E_{q \to \overline{q}} & = & E \cap (\textit{forward}_q(S_q) \times \textit{back}_{\overline{q}}(S_{\overline{q}})) \\ E_{\overline{q} \to q} & = & E \cap (\textit{forward}_{\overline{q}}(S_{\overline{q}}) \times \textit{back}_q(S_q)) \end{array}$$

If in G' some non-trivial MSCC contains a vertices from α and from α_2 then we conclude that G is pumpable nonempty.

Claim 5.3 *The graph G is pumpable nonempty iff G' has some non-trivial MSCC with a vertices from α_1 and from α_2.*

Proof: Suppose that there is some non-trivial MSCC in G' with a vertices v_1 and v_2 from α_1 and α_2, respectively. Let $\pi = v_0, v_1, \ldots,$ be a path from v_0 that visits v_1 and v_2 infinitely often. We show that we can find a pumpable path π' from v_0 that visits v infinitely often. Consider the path π. Every edge in π is in one of the six sets of edges that comprise E'. Partition π to the q-blocks that comprise it $\pi_0, \pi_1, \ldots$, where $\pi_i = v_{j_i}, v_{j_i+1}, \ldots, v_{j_{i+1}-1}$. We build by induction a path π' in which every block is pumpable. Consider the blocks π_0 and π_1. Suppose that π_0 is labeled by q and π_1 is not labeled by q. It follows that $(v_{j_1-1}, v_{j_1}) \in E_{q \to \overline{q}}$ and that $v_{j_1-1} \in \textit{forward}_q(S_q)$. Hence, there has to be a path in G_q between v_0 and v_{j_1-1} that visits some vertex v in S_q. By definition of S_q, the vertex v belongs to an MSCC M that contains vertices labeled by p and vertices not labeled by p. Let π'_1 be the path that goes from v_0 to v_{j_1-1}, visits v twice, and between the two visits to v passes vertices labeled by p and vertices not labeled by p. Consider the blocks $\pi_{i-1}, \pi_i, \pi_{i+1}$. Assume that π_{i-1} is labeled by q, π_i is not labeled by q, and π_{i+1} is labeled by q. Then $(v_{j_i-1}, v_{j_i}) \in E_{q \to \overline{q}}$ and $(v_{j_{i+1}-1}, v_{j_{i+1}}) \in E_{\overline{q} \to q}$. It follows that $v_{j_i} \in \textit{back}_{\overline{q}}(S_{\overline{q}})$ and $v_{j_{i+1}-1} \in \textit{forward}_{\overline{q}}(S_{\overline{q}})$. Then π_i visits some vertex v in $S_{\overline{q}}$, and we set π'_i

to be a path between v_{j_i} and $v_{j_{i+1}-1}$ that visits v twice, and in addition between the two visits to v passes vertices labeled by p and vertices not labeled by p. The case where the path π has only finitely many q-blocks can be handled similarly.

Consider a pumpable fair path π in G. It is simple to see that all the edges on π are present also in G'. It follows that in G' there is some reachable MSCC that visits both α_1 and α_2. $\square$

As before, all parts of the algorithm can be executed in linear time. $\square$

Remark 5.4 The algorithms described above are explicit. A symbolic PROMPT-LTL model checking algorithm follows from the translation of PROMPT-LTL to the μ-calculus described later in Theorem 7.3. The translation, however, involves a significant blow up. A symbolic algorithm that performs well on the colored Büchi graphs is left open. For standard Büchi graphs, algorithms can be classified as ones that are based on a nested fixed point that calculates the set of states that can reach α infinitely often [8], and ones that calculate symbolically the MSCC of the graph [5]. We believe that algorithms of the second type can be extended to colored graphs. $\square$

6 Assume-Guarantee Model Checking

For two systems $\mathcal{S} = \langle AP, S, \rho, s_0, L\rangle$ and $\mathcal{S}' = \langle AP, S', \rho', s_0', L'\rangle$, the parallel composition of $\mathcal{S}$ with $\mathcal{S}'$, denoted $\mathcal{S}\|\mathcal{S}'$, is the system that contains all the joint behaviors of $\mathcal{S}$ and $\mathcal{S}'$. Formally, $\mathcal{S}\|\mathcal{S}' = \langle AP, S'', \rho'', s_0'', L''\rangle$, where $S'' \subseteq S \times S'$ contains exactly all pairs that agree on their label, that is $\langle s, s'\rangle \in S''$ iff $L(s) = L'(s')$. Then, $s_0'' = \langle s_0, s_0'\rangle$ and $\rho''(\langle s, s'\rangle, \langle t, t'\rangle)$ iff $\rho(s,t)$ and $\rho'(s', t')$. Finally, $L''(\langle s, s'\rangle) = L(s)$.

An *assume-guarantee specification* for a system $\mathcal{S}$ is a pair of two specifications φ_1 and φ_2. The system $\mathcal{S}$ satisfies the specification, denoted $\langle\varphi_1\rangle\mathcal{S}\langle\varphi_2\rangle$, if it is the case that for all systems $\mathcal{S}'$, if $\mathcal{S}\|\mathcal{S}'$ satisfies φ_1, then $\mathcal{S}\|\mathcal{S}'$ also satisfies ψ [15]. In the context of LTL it is not hard to see that $\langle\varphi_1\rangle\mathcal{S}\langle\varphi_2\rangle$ iff $\mathcal{S} \models \varphi_1 \to \varphi_2$. Intuitively, since the $\|$ operator amounts to taking the intersection of the languages of $\mathcal{S}$ and $\mathcal{S}'$, it is sound to restrict attention to systems $\mathcal{S}'$ that correspond to single computations of $\mathcal{S}$. In the case of PROMPT-LTL, we can also restrict attention to single computations, but we have to take the bounds into an account. Formally, we have the following.

Lemma 6.1 *Consider a system $\mathcal{S}$ and* PROMPT-LTL *formulas φ_1 and φ_2. The specification $\langle\varphi_1\rangle\mathcal{S}\langle\varphi_2\rangle$ does not hold iff there is a bound $k_1 \geq 0$ such that for every bound $k_2 \geq 0$, there is a trace w of $\mathcal{S}$ such that $(w, 0, k_1) \models \varphi_1$ but $(w, 0, k_2) \not\models \varphi_2$.*

Since refuting assume-guarantee specifications refer to two bounds, we extend the alternating-color technique to refer to two sets of colors. The atomic proposition p partitions the computation to blocks that bound k_1, and a new atomic proposition q does the same for k_2. According to Lemmas 2.1 and 6.1, refuting $\langle\varphi_1\rangle\mathcal{S}\langle\varphi_2\rangle$ amounts to finding a bound $k_1 \geq 0$ such that for all bounds $k_2 \geq 0$, there is a computation w of $\mathcal{S}$ such that w has a k_1-bounded p-coloring that satisfies $rel_p(\varphi_1)$, but w also has a k_2-spaced q-coloring that satisfies $alt_q \wedge \neg rel_q(\varphi_2)$. Indeed, such a computation satisfies φ_1 with bound k_1, and does not satisfy φ_2 with bound k_2.

The intuition above led us to the definition of colored Büchi graphs of degree two and the corresponding definition of pumpable nonemptiness. As before, the pumpable nonemptiness technique can be used for solving the assume-guarantee model-checking problem.

Let $c(\varphi_1) = alt_p \wedge rel_p(\varphi_1)$ and $\overline{c}(\varphi_2) = alt_q \wedge \neg rel_q(\varphi_2)$, and let $\mathcal{A}_{c(\varphi_1)} = \langle 2^{AP\cup\{p\}}, Q_1, \delta_1, q_0^1, \alpha_1\rangle$, and $\mathcal{A}_{\overline{c}(\varphi_2)} = \langle 2^{AP\cup\{q\}}, Q_2, \delta_2, q_0^2, \alpha_2\rangle$ be the corresponding NBWs (per Theorem 4.1). We define the product of $\mathcal{S}$ with $\mathcal{A}_{c(\varphi_1)}$ and $\mathcal{A}_{\overline{c}(\varphi_2)}$ as the colored Büchi graph of degree two.

$\mathcal{P} = \langle \{p, q\}, S \times 2^{\{p,q\}} \times Q_1 \times Q_2, M, \langle s_0, \{p, q\}, q_0^1, q_0^2 \rangle, L, \{S \times 2^{\{p,q\}} \times \alpha_1 \times Q_2, S \times 2^{\{p,q\}} \times Q_1 \times \alpha_2\} \rangle$, where $M(\langle s, c, q_1, q_2 \rangle, \langle s', c', q_1', q_2' \rangle)$ iff $\rho(s, s')$, $q_1' \in \delta_1(q_1, L(s) \cup (c \cap \{p\}))$, and $q_2' \in \delta_2(q_2, L(s) \cup (c \cap \{q\}))$. Finally, $L(\langle s, c, q_1, q_2 \rangle) = c$.

Theorem 6.2 *The specification $\langle \varphi_1 \rangle \mathcal{S} \langle \varphi_2 \rangle$ does not hold iff the product of $\mathcal{S}$ with $\mathcal{A}_{c(\varphi_1)}$ and $\mathcal{A}_{\overline{c}(\varphi_2)}$ is pumpable nonempty,*

Proof: Assume that $\langle \varphi_1 \rangle \mathcal{S} \langle \varphi_2 \rangle$ does not hold. Then, by Lemma 6.1, there is a bound $k_1 \geq 0$ such that for every bound $k_2 \geq 0$, there is a trace w_{k_1,k_2} of $\mathcal{S}$ such that $(w_{k_1,k_2}, 0, k_1) \models \varphi_1$ but $(w_{k_1,k_2}, 0, 2k_2) \not\models \varphi_2$. Let k_2 be larger than $2 \cdot |S| \cdot |Q_1| \cdot |Q_2| \cdot k_1$ and let π_{k_1,k_2} be as above. Since $(\pi_{k_1,k_2}, 0, k_1) \models \varphi_1$, then, by Lemma 2.1, for all k_1-spaced p-coloring π'_{k_1,k_2} of π_{k_1,k_2}, we have $(\pi'_{k_1,k_2}, 0) \models c(\varphi_1)$. Since $(\pi_{k_1,k_2}, 0, 2k_2) \not\models \varphi_2$, then, by Lemma 2.1, for all k_2-bounded q-coloring π''_{k_1,k_2} of π_{k_1,k_2}, we have $(\pi''_{k_1,k_2}, 0) \not\models c(\varphi_2)$. Consider the k_1-tight p-coloring and k_2-tight q-coloring π'_{k_1,k_2} of π_{k_1,k_2} that starts with p and q. By the above, $(\pi'_{k_1,k_2}, 0) \not\models c(\varphi_2)$. Also, clearly, $(\pi'_{k_1,k_2}, 0) \models alt_q$. Thus, $(\pi'_{k_1,k_2}, 0) \models \overline{c}(\varphi_2)$. In addition, since $k_2 > 2 \cdot |S| \cdot |Q_1| \cdot |Q_2| \cdot k_1$, every path in the product $\mathcal{P}$ is (p, q)-pumpable. Hence, the product of π'_{k_1,k_2} with accepting runs of $\mathcal{A}_{c(\varphi_1)}$ and of $\mathcal{A}_{\overline{c}(\varphi_2)}$ is a (p, q)-pumpable fair path in $\mathcal{P}$.

Assume now that $\mathcal{P}$ contains a (p, q)-pumpable fair path $\pi = \langle s_0, c_0, q_0^1, q_0^2 \rangle, \langle s_1, c_1, q_1^1, q_1^2 \rangle, \langle s_2, c_2, q_2^1, q_2^2 \rangle, \ldots$. Let k_1 denote the size of the maximal p-block in π (as explained in Section 5, if $\mathcal{P}$ is (p, q)-pumpable nonempty, then it has a regular (p, q)-pumpable path, thus the maximum is well defined). We claim that for every $k_2 \geq 0$, we can pump the computation $s_0, s_1, s_2, \ldots$ of $\mathcal{S}$ to a computation that satisfies φ_1 with bound $2k_1$ but does not satisfy φ_2 with bound k_2. Note that if we pump π, we get a path π' such that the p-coloring of the trace that corresponds to π' is k_1-bounded and satisfies $c(\varphi_1)$. In addition, for each k_2, we can pump that path π to a fair path π_{k_2} such that the q-coloring of the trace that corresponds to π_{k_2} is $2k_2$-spaced and satisfies $\neg rel_q(\varphi_2)$. Hence, by Lemma 2.1, it satisfies φ_1 with bound $2k_1$, and does not satisfy φ_2 with bound k_2. □

Theorems 4.1, 5.2, and 6.2 imply the upper bound in the following theorem. The lower bound follows from the known lower bound for LTL.

Theorem 6.3 *The assume-guarantee model-checking problem for* PROMPT-LTL *is PSPACE-complete and can be solve in time exponential in the length of the formulas and linear in the size of the system.*

For two formulas φ_1 and φ_2, we say that φ_1 *implies* φ_2 iff for every system $\mathcal{S}$, if $\mathcal{S}$ satisfies φ_1, then it also satisfies φ_2. In the case of LTL, φ_1 implies φ_2 iff the formula $\varphi_1 \rightarrow \varphi_2$ is valid. In the case of PROMPT-LTL, φ_1 *implies* φ_2 iff $\langle \varphi_1 \rangle \mathcal{U} \langle \varphi_2 \rangle$, where $\mathcal{U}$ is the universal system (a clique over 2^{AP} that contains all traces over AP). Indeed, since for every system $\mathcal{S}$ we have that $\mathcal{S} \| \mathcal{U} = \mathcal{S}$, then $\langle \varphi_1 \rangle \mathcal{U} \langle \varphi_2 \rangle$ does not hold iff there is a system $\mathcal{S}$ such that if $\mathcal{S}$ satisfies φ_1 but $\mathcal{S} \not\models \varphi_2$. Since $\mathcal{U}$ is exponential in AP, and the PSPACE complexity of assume-guarantee model checking originates from an algorithm that is polynomial in the formulas and only logarithmic in the system, we have the following (the lower bound follows from the PSPACE hardness of LTL implication).

Theorem 6.4 *The implication problem for* PROMPT-LTL *is PSPACE-complete.*

7 Expressiveness

In this section we study expressiveness aspects of PROMPT-LTL. We show that a PROMPT-LTL formula φ has an equivalent LTL formula iff φ and $live(\varphi)$ are equivalent, thus the problem of deciding whether φ can be translated to LTL is PSPACE-complete. Since the semantics of PROMPT-LTL is defined with respect to a system, a natural question is whether we can translate PROMPT-LTL formulas to branching temporal logics. We show that indeed, all PROMPT-LTL formulas can be translated to the μ-calculus.

All our results refer to finite-state systems. Thus, we say that two formulas φ and φ' are equivalent iff for all finite systems $\mathcal{S}$, we have that $\mathcal{S} \models \varphi$ iff $\mathcal{S} \models \varphi'$.

7.1 From PROMPT-LTL to LTL

Some PROMPT-LTL formulas φ are equivalent to the LTL formula $live(\varphi)$. For example, it is not hard to see that $\mathbf{F_p}r$ is equivalent to $\mathbf{F}r$, for an atomic proposition r. On the other hand, as demonstrated in Section 1, the PROMPT-LTL formula $\mathbf{F_p}\mathbf{G}r$ is not equivalent to the LTL formula $\mathbf{FG}r$. Is $\mathbf{F_p}\mathbf{G}q$ equivalent to another LTL formula? A negative answer follows from Lemma 7.1 below.

Lemma 7.1 *Consider a* PROMPT-LTL *formula φ. There is an LTL formula equivalent to φ iff φ is equivalent to $live(\varphi)$.*

Proof: Assume that φ has an equivalent LTL formula. Then, there is a ω-regular language $L_\varphi \subseteq (2^P)^\omega$ such that a system $\mathcal{S}$ satisfies φ iff all the traces of $\mathcal{S}$ are contained in L_φ. We prove that for every system $\mathcal{S}$, we have that $\mathcal{S} \models live(\varphi)$ iff $\mathcal{S} \models \varphi$. The direction from right to left holds always. For the other direction, assume by way of contradiction that $\mathcal{S} \models live(\varphi)$, but the traces of $\mathcal{S}$ are not contained in L_φ. Since $\mathcal{S}$ is finite state and L_φ is ω-regular, but there is an ω- regular trace w of $\mathcal{S}$ that does not belong to L_φ. Let k be such that w satisfies $live(\varphi)$ with bound k (since w is a single trace of a finite state system, such a bound k must exist). Then, w satisfies also φ, and it therefore belongs to L_φ. $\square$

Theorem 7.2 *Deciding whether a* PROMPT-LTL *formula has an equivalent LTL formula is PSPACE-complete.*

Proof: By Lemma 7.1, the problem of deciding whether a PROMPT-LTL formula φ has an equivalent LTL formula can be reduced to checking the equivalence of φ and $live(\varphi)$. Since $\varphi \rightarrow live(\varphi)$ is valid for all φ, one should only check the implication $live(\varphi) \rightarrow \varphi$, which, according to Theorem 6.4, can be done in PSPACE.

We prove hardness in PSPACE by a reduction from the satisfiability problem of LTL. Consider an LTL formula φ, and a proposition r not used in φ. It is not hard to prove that the PROMPT-LTL formula $\varphi \wedge \mathbf{F}_b\mathbf{G}p$ has an equivalent LTL formula iff φ is unsatisfiable. $\square$

7.2 From PROMPT-LTL to the μ-calculus

It is not hard to prove that the PROMPT-LTL formula $\mathbf{F_p}\mathbf{G}q$ is equivalent to the CTL formula $\mathbf{AFAG}q$. Indeed, a system satisfies both formulas iff there is a bound $k \geq 0$ such that all the computations may visit a state in which q does not hold only in the first k positions. One may wonder whether this argument can be generalized, leading to a simple translation of PROMPT-LTL formulas to CTL * formulas: given a PROMPT-LTL formula φ, translate it to a CTL* formula φ' by (recursively) replacing all subformulas of the form $\mathbf{F_p}\theta$ by $\mathbf{FA}\theta$ (and adding an external $\mathbf{A}$). To see that the reduction does not hold in general, consider the PROMPT-LTL formula $\varphi = \mathbf{F_p}(\mathbf{X}q \vee \mathbf{G}q)$. While the system $\mathcal{S}$ from Figure 1 satisfies φ (with bound 3), the system $\mathcal{S}$ does not satisfy the CTL* formula $\varphi' = \mathbf{AFA}(\mathbf{X}q \vee \mathbf{G}q)$. The question whether PROMPT-LTL can be expressed in CTL* is open. On the other hand, we now show that every PROMPT-LTL formula has an equivalent μ-calculus formula.

Theorem 7.3 *Every* PROMPT-LTL *formula has an equivalent μ-calculus formula.*

Proof: Given a PROMPT-LTL formula φ over P, let $\mathcal{A}_{\forall c(\varphi)}$ be a alternating parity tree automaton that accepts exactly all trees all of whose paths satisfy $c(\varphi)$; in fact, $\mathcal{A}_{\forall c(\varphi)}$ can be taken to be a universal co-Büchi automaton [12]. Note that $\mathcal{A}_{\forall c(\varphi)}$ is over the alphabet $2^{P \cup \{p\}}$, thus it refers also to the atomic proposition p. Let ψ be a μ-calculus formula equivalent to $\mathcal{A}_{\forall c(\varphi)}$ [10]. As in Theorem 3.1, it can be shown that over finite systems φ is equivalent to $\exists p.\psi$. Hence, by [11], φ is equivalent to some μ-calculus formula. $\square$

References

[1] R. Alur, K. Etessami, S. La Torre, and D. Peled. Parametric temporal logic for model measuring. *ACM ToCL*, 2(3):388–407, 2001.

[2] B. Alpern and F.B. Schneider. Defining liveness. *IPL*, 21:181–185, 1985.

[3] A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. In *Proc. 7th FMICS*, ENTCS 66:2, 2002.

[4] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th CAV*, LNCS 818, 182–193, 1994.

[5] R. Bloem, H.N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In *Proc. FMCAD*, LNCS 1954, 37–54, 2000.

[6] J.R. Büchi and L.HG. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

[7] C. Eisner and D. Fisman. *A Practical Introduction to PSL.* Springer, 2006.

[8] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st LICS*, 267–278, 1986.

[9] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Proc. 2nd CAV*, LNCS 531, 136–145, 1990.

[10] D. Janin and I. Walukiewicz, Automata for the modal μ-calculus and related results", In *Proc. 20th MFCS*, LNCS 969, 552–562, 1995.

[11] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus with respect to the monadic second order logic. In *Proc. 7th CONCUR*, LNCS 1119, 263–277, 1996.

[12] O. Kupferman and M.Y. Vardi Safraless Decision Procedures. In *Proc. 46th FOCS*, 531–540, 2005.

[13] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, Berlin, January 1992.

[14] A. Pnueli. The temporal logic of programs. *Proc. 18th FOCS*, 46–57, 1977.

[15] A. Pnueli. In transition from global to modular temporal reasoning about programs. *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, 123–144, 1985.

[16] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, 179–190, 1989.

[17] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.

[18] M.Y. Vardi. Automata-theoretic model checking revisited. In *Proc. 7th VMCAI*, LNCS 4349, 137–150, 2007.

[19] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, 332–344, 1986.

[20] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *I&C*, 115(1):1–37, 1994.