

# Assume-Guarantee Synthesis for Prompt Linear Temporal Logic

Nathanaël Fijalkow<sup>1,2</sup>, Bastien Maubert<sup>3</sup>, Aniello Murano<sup>3</sup> and Moshe Vardi<sup>4</sup>

<sup>1</sup> CNRS & LaBRI, Bordeaux, France

<sup>2</sup> The Alan Turing Institute of data science, London, United Kingdom

<sup>3</sup> Università degli Studi di Napoli “Federico II”, Naples, Italy

<sup>4</sup> Rice University, Houston, U.S.A

## Abstract

Prompt-LTL extends Linear Temporal Logic with a bounded version of the “eventually” operator to express temporal requirements such as bounding waiting times. We study assume-guarantee synthesis for prompt-LTL: the goal is to construct a system such that for all environments satisfying a first prompt-LTL formula (the assumption) the system composed with this environment satisfies a second prompt-LTL formula (the guarantee). This problem has been open for a decade. We construct an algorithm for solving it and show that, like classical LTL synthesis, it is 2-EXPTIME-complete.

## 1 Introduction

Linear-time Temporal Logic (LTL) is a specification language for formal verification and program synthesis [Pnueli, 1977]. While formal verification aims at verifying the correctness of programs, program synthesis consists in synthesising correct-by-construction programs directly from their specification.

The synthesis problem for reactive systems [Pnueli and Rosner, 1989] is as follows: given a finite set of inputs  $I$  controlled by the environment, a finite set of outputs  $O$  controlled by the system, and a specification  $\varphi$  given as an LTL formula over alphabet  $I \times O$ , synthesise a reactive system  $S$  so that for every environment  $E$ , the closed system  $S||E$  satisfies  $\varphi$ .

The classical solution [Pnueli and Rosner, 1989; Vardi, 1995] goes in three steps. The first step is to construct a non-deterministic Büchi automaton  $\mathcal{A}$  equivalent to  $\varphi$ . The second step is to determinise  $\mathcal{A}$  into a parity automaton  $\mathcal{B}$ . The third step is to construct and solve a parity game  $\mathcal{G}$  based on  $\mathcal{B}$  where Adam picks inputs and Eve picks outputs, such that solving this game is equivalent to the synthesis problem: a winning strategy for Eve yields a system  $S$  satisfying the specification. In particular this implies that if there exists a system satisfying the specification then there exists a finite one, given by a finite state Mealy machine.

One shortcoming of LTL is that it cannot bound the waiting time of eventualities. For instance the common requirement that every request ( $r$ ) should be eventually granted ( $g$ ), expressed as  $\mathbf{G}(r \implies \mathbf{F}g)$  in LTL, is satisfied even when the waiting time between the moment a request is made and

the moment it is granted can grow arbitrarily long. There are various approaches to tackle this shortcoming [Alur *et al.*, 2001]. In this work we are interested in the logic PROMPT-LTL [Kupferman *et al.*, 2009] which extends LTL with a “bounded eventually” operator  $\mathbf{F}^{\leq B}$ . This operator expresses an eventuality that should be satisfied within the next  $B$  steps, where  $B$  is a variable existentially quantified. A formula is satisfied in a model if there exists a value  $b$  for  $B$  that makes the formula true. For instance, the PROMPT-LTL formula  $\mathbf{G}(r \implies \mathbf{F}^{\leq B}g)$  expresses that there exists a bound on the waiting time.

Assume-guarantee originates as a modular approach to program verification that allows decomposition of proof obligations (see *e.g.* [Alur and Henzinger, 1999]). Informally, an assume-guarantee specification consists of two specifications  $\varphi$  and  $\psi$ . A system  $S$  satisfies this specification if whenever it is used in a context that satisfies the assumption  $\varphi$ , the guarantee  $\psi$  holds on the system or, said differently, for all environments  $E$  satisfying  $\varphi$ , the composed system  $S||E$  satisfies  $\psi$ .

Assume-guarantee synthesis is the problem of synthesising a system  $S$  satisfying an assume-guarantee specification. We refer to [Maoz and Sa’ar, 2012] for a discussion of practical applications of assume-guarantee synthesis. When considering LTL specifications, assume-guarantee verification and synthesis reduce to classical LTL model checking and synthesis: if  $\varphi$  and  $\psi$  are LTL formulas, then the assume-guarantee specification is equivalent to the LTL specification  $\varphi \implies \psi$ . But this is not true for PROMPT-LTL: when we ask that in all environments satisfying the assumption the system also satisfies the guarantee, there is an implicit quantification on the bounds with which the assumption and the guarantee are satisfied: a universal one for the assumption and an existential one for the guarantee. Because in PROMPT-LTL the only quantification on bounds is an existential one at the front of the formula, this alternation is not reflected in the formula  $\varphi \implies \psi$  and, in fact, cannot be captured by a PROMPT-LTL formula. Classical techniques to handle PROMPT-LTL such as the alternating colour technique [Kupferman *et al.*, 2009] are thus difficult to apply, and the assume-guarantee synthesis problem for PROMPT-LTL has been open for a decade.

The problem of assume-guarantee synthesis for PROMPT-LTL as defined above can be called *uniform*, as the system that one aims at synthesising does not depend on the bounds

for which the assumption or the guarantee are satisfied. It was observed in [Jacobs *et al.*, 2018] that this variant of the problem does not always admit finite implementations, as a satisfying system may require memory that depends on the bounds. Also, in the formulation of the problem they consider, the assumption talks about both inputs *and outputs*, which makes it possible to have solution systems that always falsify the assumption.

To eliminate such unsatisfactory solutions we consider assumptions that only talk about inputs. Also, to account for the fact that a system’s memory may depend on the bound for the assumption, we introduce a *non-uniform* variant of the problem, which asks whether for every bound  $b$  on the assumption, there exists a bound  $b'$  on the guarantee and a system  $S_b$  such that whenever the assumption is satisfied with bound  $b$ , the guarantee is satisfied with bound  $b'$ .

We show that this problem is 2-EXPTIME-complete, hence not costlier than LTL synthesis. When the answer is positive, our solution produces a parameterised system which depends on the bound for the assumption.

We develop an automata-theoretic approach for PROMPT-LTL using a subclass of cost automata that we call prompt automata. Cost automata are a type of automata with counters introduced in the theory of regular cost functions [Colcombet, 2009; Colcombet, 2013b], leading to a wealth of results extending many results from automata theory to a quantitative setting [Colcombet, 2013a].

The key difficulty is that prompt automata cannot be determined, *i.e.* deterministic prompt automata are strictly less expressive than non-deterministic ones. Our main technical contribution is a history-determinisation procedure for prompt automata. History-determinism relaxes the notion of determinism: in a history-deterministic automaton, the non-determinism can be resolved on the fly. This property is exactly what is required for later using these automata in a game context as in the usual solution for LTL synthesis.

A benefit of our approach is that it follows the classical route for solving the synthesis problem for LTL via automata, determinisation and game solving, which proved its efficiency in recent editions of the SYNTCOMP event [Jacobs and Bloem, 2018; Jacobs *et al.*, 2019].

Assume-guarantee verification and synthesis have been extended in different directions, such as probabilistic systems [Kwiatkowska *et al.*, 2010] or specifications in quantitative extensions of LTL [Almagor *et al.*, 2017], or multi-agent synthesis [Chatterjee and Henzinger, 2007; Fisman *et al.*, 2010; Bloem *et al.*, 2015; Kupferman *et al.*, 2016; Brenguier *et al.*, 2017; Filiot *et al.*, 2018], but it has never been solved for PROMPT-LTL specifications.

Distributed synthesis for PROMPT-LTL was recently studied in [Jacobs *et al.*, 2018], where it is observed that in the asynchronous setting, to define PROMPT-LTL synthesis in a meaningful way one has to resort to assume-guarantee synthesis for PROMPT-LTL, but no solution was provided.

Besides the works already mentioned, PROMPT-LTL has also been studied in relation with logics for strategic reasoning [Aminof *et al.*, 2016; Fijalkow *et al.*, 2018]. In the latter

work, cost automata are used to solve an extension of Strategy Logic that subsumes PROMPT-LTL. However, even if this logic has in its syntax quantification on strategies and quantification on bounds, it cannot express assume-guarantee synthesis. The reason is that a syntactic constraint in the logic forbids alternation of quantification on bounds, inherent in assume-guarantee synthesis for PROMPT-LTL.

In Section 2 we define PROMPT-LTL. We define the assume-guarantee synthesis problem in Section 3 and present an example in Section 4. The first step of our solution, translating PROMPT-LTL formulas into prompt automata, is described in Section 5. The second step, history-determinisation of these automata, is developed in Section 6. The main ingredient of the third step is domination games, presented in Section 7. We wrap up and prove our main result in Section 8.

## 2 Prompt Linear Temporal Logic

We write  $[i, j]$  for the interval  $\{i, i + 1, \dots, j - 1, j\}$ , and we use parentheses to exclude extremal values, so that  $[i, j)$  is  $\{i, i + 1, \dots, j - 1\}$ . For an alphabet  $\Sigma$ , the set of finite words over  $\Sigma$  is  $\Sigma^*$  and the set of infinite words is  $\Sigma^\omega$ . If  $w = a_0 a_1 a_2 \dots \in \Sigma^\omega$  is an infinite word over  $\Sigma$  and  $i \in \mathbb{N}$ , then we let  $w_i = a_i$  and  $w_{\leq i} = a_0 \dots a_i$ .

**Definition 1.** *The syntax of PROMPT-LTL formulas over the alphabet  $\Sigma$  is given by the following grammar:*

$$\varphi ::= p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \mid \varphi \mathbf{R}\varphi \mid \mathbf{F}^{\leq B}\varphi,$$

where  $p \in \Sigma$ .

In the examples we use the classical additional operators  $\mathbf{F}$  and  $\mathbf{G}$  (“eventually” and “always”), both definable using the operators above. The size of a formula is defined classically as the size of the syntactic underlying tree denoting the formula.

PROMPT-LTL formulas are evaluated on infinite words at a position  $i \in \mathbb{N}$  on the word and with a bound  $b \in \mathbb{N}$ . The semantics is defined by induction with  $w \in \Sigma^\omega$  and  $i, b \in \mathbb{N}$ . All cases are as for LTL, except for the following one:

$$w, i, b \models \mathbf{F}^{\leq B}\varphi \quad \text{if} \quad \exists j \in [i, i + b] \text{ such that } w, j, b \models \varphi.$$

An execution  $w \in \Sigma^\omega$  satisfies a formula  $\varphi$  for the bound  $b \in \mathbb{N}$ , which we write  $w, b \models \varphi$ , if  $w, 0, b \models \varphi$ .

## 3 Assume-Guarantee Synthesis

Let  $I$  and  $O$  be finite sets of inputs and outputs respectively, which can be thought of as valuations over sets of input and output variables. We consider synchronous systems: at each time step the environment produces an input  $i \in I$  and the system reacts by producing an output  $o \in O$ . A system is a function  $S : I^+ \rightarrow O$  that maps each nonempty finite sequence of inputs to an output. An infinite word over  $I \times O$  is called an *execution*. Given an infinite sequence of inputs  $w = a_0 a_1 a_2 \dots \in I^\omega$ , we define the execution of  $S$  on  $w$  as  $S(w) = (a_0, b_0)(a_1, b_1)(a_2, b_2) \dots$  where for each  $i \geq 0$ ,  $b_i = S(w_{\leq i})$  is the output after the sequence of inputs  $w_{\leq i}$ .

An *assume-guarantee (AG) specification* is a pair of formulas  $(\varphi, \psi)$ , with  $\varphi$  over the alphabet  $I$  and  $\psi$  over the

alphabet  $I \times O$ . We study the following decision problem, which we call (non-uniform) assume-guarantee synthesis problem [Pnueli, 1985; Kupferman *et al.*, 2009]: for all  $b \in \mathbb{N}$ , there exists  $b' \in \mathbb{N}$  and a system  $S$  such that

$$w, b \models \varphi \implies S(w), b' \models \psi.$$

**INPUT:** A PROMPT-LTL AG specification  $(\varphi, \psi)$

**OUTPUT:**  $\begin{cases} \text{Yes} & \text{if there exists a solution for } (\varphi, \psi), \\ \text{No} & \text{otherwise.} \end{cases}$

Our main result in this work is stated below.

**Theorem 1.** *The assume-guarantee synthesis problem for PROMPT-LTL is 2EXPTIME-complete.*

## 4 A Motivating Example

We consider the following scenario, where the goal is to synthesise a system to run a server. There are  $N$  users and each user can request access to the server. The server may be up or down at each time step. When the server is up, it can grant access to a user, one at a time. Additionally, time is divided in sessions: sometimes a session ends, and a new session starts. We first give the specification of the server in natural language in an assume-guarantee form:

**Assumption** (over  $I$ ):

- The server is up infinitely many times, and
- there are infinitely many sessions.

**Guarantee** (over  $I \times O$ ):

- In each session each user who requests access is eventually granted access, or the session ends;
- and the server does not grant access when it is down.

Formally, the set of inputs is

$$I = \{\text{new}\} \cup (\{\text{up}, \text{down}\} \times (\{\varepsilon\} \cup \{\text{req}_i : i \in [1, N]\})).$$

Here  $\text{req}_i$  represents a request of user  $i$ ,  $\varepsilon$  means that no request is made, up and down are states of the server, and new starts a new session. The set of outputs is

$$O = \{\text{ans}_i : i \in [1, N]\} \cup \{\varepsilon\}.$$

where  $\text{ans}_i$  represents granting access to user  $i$ .

We emphasise that whenever a user is granted access to the server, this answers all of their past requests, not only the last one, and the end of a session answers all past requests.

There are two natural systems for the server. Both store the list of users who requested access and grants access to one of them whenever the server is up. The difference is in the order in which access is granted. The FIFO implementation grants access to the user who made the *earliest* unanswered request, and the LIFO implementation to the user who made the *latest* unanswered request. Both systems can be described as finite-state systems of size exponential in  $N$ .

At an intuitive level, the FIFO implementation is fairer than the LIFO one; let us see how this is reflected in formalising the specification in LTL and PROMPT-LTL.

We can write the specification above in two LTL formulas  $A$  and  $G$  and consider the LTL synthesis problem for the specification  $A \implies G$ . The formulas are

$$\begin{aligned} A &= \mathbf{GFup} \wedge \mathbf{GFnew}; \\ G &= \bigwedge_{i \in [1, N]} \mathbf{G}(\text{req}_i \implies \mathbf{F}(\text{ans}_i \vee \text{new})) \\ &\quad \wedge \bigwedge_{i \in [1, N]} \mathbf{G}(\text{ans}_i \implies \text{up}). \end{aligned}$$

Both FIFO and LIFO implementations are correct systems. Since the end of a session answers all requests and there are infinitely many sessions, the system never that never grants any access is also a solution. This is obviously not a satisfactory system, we will see that finer specifications rule it out.

Let us write the guarantee formula  $G$  in PROMPT-LTL; the first part becomes: there exists a bound  $b'$  such that in each session, each user who requests access is granted access within  $b'$  steps, or the session ends. Formally, we define the PROMPT-LTL formula

$$\begin{aligned} G' &= \bigwedge_{i \in [1, N]} \mathbf{G}(\text{req}_i \implies \mathbf{F}^{\leq B}(\text{ans}_i \vee \text{new})) \\ &\quad \wedge \bigwedge_{i \in [1, N]} \mathbf{G}(\text{ans}_i \implies \text{up}). \end{aligned}$$

Consider the PROMPT-LTL synthesis problem for the specification  $A \implies G'$ . Neither FIFO nor LIFO implementations are correct systems, and indeed there are no solutions to this problem since the server can be down for arbitrarily long consecutive time sequences, which leaves no hope of granting access in bounded time. We refer to the figure on the left hand side in Figure 1 for an illustration of such an environment.

The issue here is that we ask the system to be correct against all environments satisfying  $A$ , and there is indeed no way to ensure bounded waiting time if there are no similar timing assumptions on the environment. The assume-guarantee formulation addresses this shortcoming.

We now write the assumption formula also using PROMPT-LTL, expressing that there exists a bound  $b$  such that the server cannot be down for  $b$  consecutive steps. Formally,

$$A' = \mathbf{GF}^{\leq B}\text{up} \wedge \mathbf{GFnew}.$$

Let us look at the PROMPT-LTL synthesis problem for the assume-guarantee specification  $(A', G')$ . The FIFO implementation satisfies this specification, but the LIFO does not.

Let us first see that the LIFO implementation fails to guarantee a bound: a user can be left wanting for an arbitrary number of steps if another user keeps requesting and getting access to the server. We refer to the figure on the right hand side in Figure 1 for an illustration of this behaviour.

We now analyse the FIFO implementation. Let us consider an environment where the server is up at least once every  $b$  consecutive steps. Then inside a session every request will be granted within  $b' = b \cdot N$  steps, where  $N$  is the number of users. This is because a user only goes down in the list of unanswered requests and becomes the earliest unanswered request after at most  $N$  steps where the server is up. This example shows the essence of PROMPT-LTL assume-guarantee specification: how a timing assumption on the environment transfers to a timing guarantee on the system.

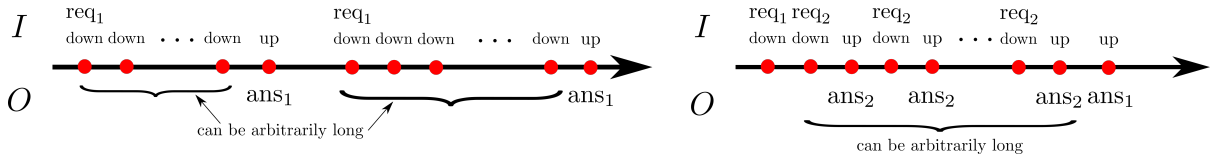


Figure 1: The figure on the left illustrates why there are no solutions to the PROMPT-LTL synthesis problem for the specification  $A \implies G'$ . The figure on the right illustrates why the LIFO implementation does not satisfy the PROMPT-LTL assume-guarantee specification  $(A', G')$ .

## 5 Prompt Automata

In this section we show how to translate PROMPT-LTL to prompt automata. We first recall the notion of cost functions, introduce prompt automata, and then present the translation. Let us fix  $\Sigma, \Omega$  finite alphabets.

The first key conceptual idea is to reason with *cost functions* instead of languages. The usual way of reasoning semantically with a formula  $\varphi$  is to consider the language it defines by  $L(\varphi) = \{w \in \Sigma^\omega : \exists b \in \mathbb{N}, w, b \models \varphi\}$ . Instead, we will consider a more precise object. We first define  $\llbracket \varphi \rrbracket : \Sigma^\omega \rightarrow \mathbb{N} \cup \{\infty\}$  by

$$\llbracket \varphi \rrbracket(w) = \inf \{b \in \mathbb{N} : w, b \models \varphi\}.$$

The function  $\llbracket \varphi \rrbracket$  carries too much information: it gives the smallest value  $b \in \mathbb{N}$  for each word  $w$  such that  $w, b \models \varphi$ . The foundational idea of the theory of regular cost functions [Colcombet, 2009; Colcombet, 2013b] is to use approximation: instead of considering functions  $\Sigma^\omega \rightarrow \mathbb{N} \cup \{\infty\}$ , we consider such functions up to an equivalence relation  $\approx$  which informally says “two functions are equivalent if they are bounded on the same sets”. Formally, for  $f, g : \Sigma^\omega \rightarrow \mathbb{N} \cup \{\infty\}$ , we say that  $f \approx g$  if for all  $X \subseteq \Sigma^\omega$ ,  $f(X)$  is bounded if and only if  $g(X)$  is bounded, where  $f(X)$  is bounded means that  $f(X) \subseteq [0, b]$  for some  $b \in \mathbb{N}$ . The theory of regular cost functions replaces the notion of functions  $\Sigma^\omega \rightarrow \mathbb{N} \cup \{\infty\}$  by *cost functions*: a cost function is an equivalence class for the relation  $\approx$ .

From now on functions are always considered up to the equivalence relation  $\approx$ . In particular, a formula of PROMPT-LTL or an automaton defines a cost function, which is the equivalence class of the function it defines. Consequently, we say that two automata are equivalent if they define the same cost functions, although they may define different functions.

The approximation  $\approx$  is appropriate for the assume-guarantee synthesis problem for PROMPT-LTL, since it is a boundedness problem that does not specify exact values.

A finite word  $\lambda \in \{\dot{i}, \tau\}^*$  is a sequence of counter operations: the counter is initialised with value zero, so  $\text{val}(\epsilon) = 0$ , it is incremented by one using  $\dot{i}$ , so  $\text{val}(\lambda\dot{i}) = \text{val}(\lambda) + 1$ , and reset to zero using  $\tau$ , so  $\text{val}(\lambda\tau) = 0$ .

**Desert** :  $\{\dot{i}, \tau\}^\omega \rightarrow \mathbb{N} \cup \{\infty\}$  is the supremum value of the counter, defined by

$$\text{Desert}(\lambda) = \sup \{\text{val}(\lambda_{\leq i}) : i \in \mathbb{N}\}$$

A cost function  $f$  with domain  $\{0, \infty\}$  is called a language, and is equally given as  $L = f^{-1}(\{0\}) \subseteq \Sigma^\omega$ . Note that for two languages  $L, L'$  seen as functions,  $L = L'$  is equivalent to  $L \approx L'$ . We say that  $\lambda$  satisfies  $L$  if  $f(\lambda) = 0$ , equivalently

if  $\lambda \in L$ . We write  $\neg L$  for the complement of  $L$ , meaning  $\Sigma^\omega \setminus L$ . A function  $f : \Sigma^\omega \rightarrow \mathbb{N} \cup \{\infty\}$  and a bound  $b \in \mathbb{N}$  induce a language  $f[b] = \{\lambda \in \Sigma^\omega : f(\lambda) \leq b\}$ .

The parity language **Parity** uses alphabet  $[1, d]$  and is defined as the set of words in  $[1, d]^\omega$  such that the maximal priority appearing infinitely many times in  $\lambda$  is even. The special case where  $d = 2$  yields Büchi language and is written **Büchi**.

We use  $\wedge$  and  $\vee$  to combine functions over products of labels, where  $\wedge$  is the maximum and  $\vee$  the minimum. For instance, **Parity**  $\wedge$  **Desert** is a function over the set of labels  $[1, d] \times \{\dot{i}, \tau\}$  defined by

$$(\text{Parity} \wedge \text{Desert})(\lambda) = \max(\text{Parity}(\lambda(1)), \text{Desert}(\lambda(2))),$$

where  $\lambda(1)$  is the projection of  $\lambda$  on  $[1, d]^\omega$  and  $\lambda(2)$  the projection on  $\{\dot{i}, \tau\}^\omega$ . Since **Parity** is a language, if  $\lambda$  satisfies parity then  $(\text{Parity} \wedge \text{Desert})(\lambda) = \text{Desert}(\lambda)$ , and otherwise  $(\text{Parity} \wedge \text{Desert})(\lambda) = \infty$ .

Let  $f : \Omega^\omega \rightarrow \mathbb{N} \cup \{\infty\}$  a function. A (non-deterministic)  $f$ -automaton over the alphabet  $\mathcal{A}$  is a tuple  $\mathcal{A} = (Q, Q_{\text{init}}, \Delta)$  where  $Q$  is a finite set of states,  $Q_{\text{init}} \subseteq Q$  a set of initial states, and  $\Delta \subseteq Q \times \Sigma \times \Omega \times Q$  is a transition relation.

Let  $q$  a state and  $a$  a letter. We call *a-transitions* the elements of  $Q \times \{a\} \times \Omega \times Q$ , and *a-transitions* from  $q$  the elements of  $\{q\} \times \{a\} \times \Omega \times Q$ . We assume that automata are complete: for every state  $q$  and letter  $a$  there exists at least one *a-transition* from  $q$ .

A run over the finite or infinite word  $w = a_0 a_1 \dots$  is a sequence  $\rho = \delta_0 \delta_1 \dots$  of matching length consisting of consecutive transitions such that for all  $i$ ,  $\delta_i$  is an  $a_i$ -transition and  $\rho$  starts from a state in  $Q_{\text{init}}$ . We extend  $f$  to infinite runs by letting  $f(\rho) = f(\lambda)$ , where  $\lambda$  is the projection of  $\rho$  on the set of labels  $\Omega$ . An  $f$ -automaton induces the function

$$\llbracket \mathcal{A} \rrbracket : \Sigma^\omega \rightarrow \mathbb{N} \cup \{\infty\} \\ w \mapsto \inf \{f(\rho) : \rho \text{ a run over } w\},$$

If  $f$  is a language  $L$ , then the function  $\llbracket \mathcal{A} \rrbracket$  takes only values 0 and  $\infty$  so it is also a language, so considering  $L(\mathcal{A}) = \{w \in \Sigma^\omega : \llbracket \mathcal{A} \rrbracket(w) = 0\}$  we recover the usual definition of the language accepted by  $\mathcal{A}$ . In this case, an  $L$ -automaton is a non-deterministic automaton with acceptance condition  $L$ : a word is accepted if there exists a run in  $L$ , which we call an accepting run.

An automaton is deterministic if it has a unique initial state and for every state  $q$  there exists at most one (hence a unique) *a-transition* from  $q$ : then the transition relation becomes a function  $\delta : Q \times \mathcal{A} \rightarrow \Omega \times Q$ .

A **Parity**  $\wedge$  **Desert**-automaton is called a *prompt automaton*, and a **Büchi**  $\wedge$  **Desert**-automaton is called a *Büchi prompt*

*automaton*. The **Desert** function over finite words was introduced independently in [Bala, 2004] and [Kirsten, 2004]. Prompt automata are a special case of B-automata [Colcombet, 2009] that corresponds over finite words to the notion of temporal cost automata [Colcombet *et al.*, 2010].

Kuperberg and Vanden Boom showed that the classical translations from LTL to non-deterministic [Kuperberg, 2014] and alternating [Kuperberg and Vanden Boom, 2012] automata can be extended to COST-LTL and cost automata, which subsume PROMPT-LTL and prompt automata. We can easily adapt their translation to obtain the following:

**Theorem 2.** *For every formula  $\varphi$  of PROMPT-LTL of size  $n$ , there exists a Büchi prompt automaton  $\mathcal{A}$  of size  $O(2^n)$  such that  $\llbracket \mathcal{A} \rrbracket \approx \llbracket \varphi \rrbracket$ .*

Compared to the classical translation from LTL to non-deterministic automata, the main difference is that when a state contains a subformula  $\mathbf{F}^{\leq B}\psi$ , the automaton non-deterministically chooses between checking that  $\psi$  holds or postponing it to the next step and incrementing the counter.

## 6 History-Determinisation

Here we show how to construct a history-deterministic prompt automaton equivalent to a given prompt automaton.

Deterministic prompt automata are strictly less expressive than non-deterministic ones, which is a major issue in extending the classical solution to the synthesis problem from LTL to PROMPT-LTL. Indeed, recall that in this approach the second step is to turn a non-deterministic (Büchi) automaton into a (parity) deterministic one, before constructing a game equivalent to the synthesis problem for the LTL formula. This last construction would fail for non-deterministic automata. The solution comes from the notion of *history-deterministic automata*, which is a relaxation of deterministic automata that is tailored exactly to make the game construction work [Henzinger and Piterman, 2006].

Informally, a non-deterministic automaton is history-deterministic if its non-determinism can be resolved by a function  $\sigma : \Sigma^+ \rightarrow \Delta$  considering only the input read so far. In other words,  $\sigma$  reads the word letter by letter and constructs a run: for  $w \in \Sigma^+$ , the choice of the next transition is  $\sigma(w) \in \Delta$ . Hence  $\sigma$  induces a function  $\sigma^\omega : \Sigma^\omega \rightarrow \Delta^\omega$  picking a run  $\sigma^\omega(w)$  for a word  $w \in \Sigma^\omega$ .

**Definition 2.** *An  $f$ -automaton  $\mathcal{A}$  is history-deterministic if there exists a correction function  $\alpha$  such that for all  $b \in \mathbb{N}$ , there exists a function  $\sigma : \Sigma^+ \rightarrow \Delta$  such that for all  $w \in \Sigma^\omega$ , if  $\llbracket \mathcal{A} \rrbracket(w) \leq b$ , then  $f(\sigma^\omega(w)) \leq \alpha(b)$ .*

There exists a direct procedure for history-determinisation of cost automata over finite words [Colcombet and Fijalkow, 2016], yielding a single-exponential (optimal) blow-up in size. Extending this construction to infinite words is an open problem whose solution would have interesting consequences for the theory of regular cost functions. The construction relies on the use of a determinisation procedure for classical automata (for instance, Safra’s construction). Theorem 3 below is a partial answer to this problem: it extends the result given in [Colcombet and Fijalkow, 2016] to infinite words, but only for prompt automata (a strict subclass of cost automata). The

key idea in our construction is reminiscent of the alternating colour technique [Kupferman *et al.*, 2009], which Colcombet, Kuperberg, and Lombardy [Colcombet *et al.*, 2010] used in the context of temporal cost automata for showing the equivalence between the two dual models of automata.

**Theorem 3.** *For every Büchi prompt automaton with  $n$  states, there exists an equivalent history-deterministic prompt automaton with  $2^{O(n \log(n))}$  states and  $O(n)$  priorities.*

We sketch the proof of Theorem 3.

Let  $\mathcal{A}$  a Büchi prompt automaton with  $n$  states. Following [Colcombet *et al.*, 2010], we define a clock to be an infinite word over the alphabet  $\{\varepsilon, \text{tick}\}$ , and for a clock  $c$  we let  $\mathbf{Desert}(c) = \mathbf{Desert}(\lambda)$ , where  $\lambda \in \{\mathfrak{i}, \mathfrak{r}\}^\omega$  is obtained from  $c$  by replacing each  $\varepsilon$  with  $\mathfrak{i}$  and each tick with  $\mathfrak{r}$ . The alternating colour technique is based on the following two key observations: let  $\rho$  a run over  $w$ ,

**Fact 1.** *If  $\mathbf{Desert}(\rho) \leq b$ , then the counter in  $\rho$  is reset between every tick of the clock  $c_b = \varepsilon^b \text{tick} \varepsilon^b \text{tick} \dots$*

**Fact 2.** *If there is a clock  $c$  with  $\mathbf{Desert}(c) \leq b$  such that  $\rho$  resets between every tick in  $c$ , then  $\mathbf{Desert}(\rho) \leq 2b$ .*

Intuitively, these two facts imply that **Desert** can be replaced by the property that there exists a clock  $c$  such that the counter in  $\rho$  is reset between every tick in  $c$ . The benefit of this replacement is that the latter is a regular property, hence a reduction from a quantitative property to a qualitative one.

Let  $w \in \Sigma^\omega$  and  $c \in \{\varepsilon, \text{tick}\}^\omega$  a clock, we write  $w \otimes c$  for the word over alphabet  $\Sigma \times \{\varepsilon, \text{tick}\}$  that projects on  $w$  and  $c$ . Let us define  $L_{\mathcal{A}}$  as the language of words of the form  $w \otimes c$  such that there exists a run  $\rho$  of  $\mathcal{A}$  over  $w$  satisfying **Büchi** and such that the counter is reset between every tick in  $c$ . Define also

$$f_{\mathcal{A}} : \Sigma^\omega \rightarrow \mathbb{N} \cup \{\infty\} \\ w \mapsto \inf \{ \mathbf{Desert}(c) : w \otimes c \in L_{\mathcal{A}} \}.$$

Note that  $f_{\mathcal{A}} \approx \llbracket \mathcal{A} \rrbracket$ , hence the following lemma proves the first half of Theorem 3.

**Lemma 1.** *There exists a history-deterministic prompt automaton  $\mathcal{B}$  recognising  $f_{\mathcal{A}}$  with  $2^{O(n \log(n))}$  states and  $O(n)$  priorities.*

## 7 Domination Games

An  $\Omega$ -labelled graph  $G = (V, E)$ , or simply graph when  $\Omega$  is clear from the context, is given by a set  $V$  of vertices and a set  $E \subseteq V \times \Omega \times V$  of labelled edges: if  $(v, \ell, v') \in E$ , then there is an edge from  $v$  to  $v'$  labelled by  $\ell \in \Omega$  and we say that  $v$  is the *origin vertex* of the edge  $(v, \ell, v')$ , and  $v'$  is its *destination vertex*. A *path*  $\pi$  is a (finite or infinite) sequence of consecutive edges  $(v, \ell, v')$  in  $E$ . We write  $\pi = (v_0, \ell_0, v_1)(v_1, \ell_1, v_2) \dots$  and  $\pi_i = (v_i, \ell_i, v_{i+1})$ . We also let  $\pi_{<i}$  denote the prefix of  $\pi$  of length  $i$ , meaning  $\pi_{<i} = \pi_0 \dots \pi_{i-1}$ . For a qualitative function  $W \subseteq \Sigma^\omega$ , we say that a path  $\pi$  satisfies  $W$  if  $\lambda(\pi) \in W$ , where  $\lambda(\pi)$  is the projection of  $\pi$  on the labels.

An  $\Omega$ -labelled game  $\mathcal{G} = (V, E, v_{\text{init}}, V_{\text{Eve}}, V_{\text{Adam}})$ , or game, is a finite  $\Omega$ -labelled graph  $G = (V, E)$  together with an initial vertex  $v_{\text{init}} \in V$  and two sets  $V_{\text{Eve}}$  and  $V_{\text{Adam}}$  such

that  $V = V_{\text{Eve}} \uplus V_{\text{Adam}}$ . The interaction between the two players goes as follows. A token is initially placed on the initial vertex  $v_{\text{init}}$ , and the player who controls this vertex pushes the token along an edge, reaching a new vertex; the player who controls this new vertex takes over, and this interaction goes on potentially forever, describing an infinite play.

A *strategy* is a map  $\sigma : E^* \rightarrow E$ . We say that a play  $\pi$  is consistent with a strategy  $\sigma$  for Eve (resp. for Adam) if for all  $i \geq 0$  such that  $v_i \in V_{\text{Eve}}$  (resp.  $v_i \in V_{\text{Adam}}$ ), we have  $\sigma(\pi_{<i}) = \pi_i$ .

We define strategies with memory. We let  $\mathcal{G} = (V, E, v_{\text{init}}, V_{\text{Eve}}, V_{\text{Adam}})$  be a game. A *memory structure*  $\mathcal{M} = (M, m_{\text{init}}, \mu)$  for  $\mathcal{G}$  consists of a set  $M$  of memory states, an initial memory state  $m_{\text{init}} \in M$ , and an update function  $\mu : M \times E \rightarrow M$ . A memory structure is similar to an automaton synchronised with the game: it starts from  $m_{\text{init}}$  and reads the sequence of edges produced by the game. Whenever an edge is taken, the current memory state is updated using the update function  $\mu$ . A strategy relying on a memory structure  $\mathcal{M}$ , whenever it picks the next move, considers only the current vertex and the current memory state: it is thus given by a next-move function  $\sigma : V_{\text{Eve}} \times M \rightarrow E$ .

The notion of domination games was introduced for the study of the domination between cost functions [Colcombet, 2013a]. A *domination game* is a game  $\mathcal{G}$  over the set of labels  $([1, d_1] \times \{\dot{i}, r\}) \times ([1, d_2] \times \{\dot{i}, r\})$ . A play  $\pi$  induces two projections:  $\pi_1 \in ([1, d_1] \times \{\dot{i}, r\})^\omega$  and  $\pi_2 \in ([1, d_2] \times \{\dot{i}, r\})^\omega$ . Eve wins a domination game if for all  $b \in \mathbb{N}$ , there exists  $b' \in \mathbb{N}$  and a strategy  $\sigma$  for Eve such that all plays  $\pi$  consistent with  $\sigma$  satisfy

$$(\mathbf{Parity} \wedge \mathbf{Desert})(\pi_1) \leq b \Rightarrow (\mathbf{Parity} \wedge \mathbf{Desert})(\pi_2) \leq b'.$$

Otherwise, Adam wins.

The following is proved in [Colcombet and Göller, 2016].

**Theorem 4.** *There exists an algorithm for deciding whether Eve wins in a domination game running in time  $O(n^d)$  where  $n$  is the number of vertices and  $d = d_A + d_B$  the total number of priorities. If Eve wins, then she has a finite memory winning strategy.*

## 8 The Complete Solution

We recall that the assume-guarantee synthesis problem takes as input two PROMPT-LTL formulas  $\varphi$  over  $I$  and  $\psi$  over  $I \times O$ , and the goal is to decide whether for all  $b \in \mathbb{B}$ , there exists  $b' \in \mathbb{N}$  and a system  $S : I^+ \rightarrow O$  such that for all  $w \in I^\omega$ ,  $\llbracket \varphi \rrbracket(w) \leq b \Rightarrow \llbracket \psi \rrbracket(S(w)) \leq b'$ .

Let  $(\varphi, \psi)$  a PROMPT-LTL assume-guarantee specification. The three steps of the procedure are:

- Build two Büchi prompt automata  $\mathcal{A}_{\text{int}}, \mathcal{B}_{\text{int}}$  recognising  $\llbracket \varphi \rrbracket$  and  $\llbracket \psi \rrbracket$  (Theorem 2).
- Build two history-deterministic automata: a prompt automaton  $\mathcal{A}$  equivalent to  $\mathcal{A}_{\text{int}}$  and a prompt automaton  $\mathcal{B}$  equivalent to  $\mathcal{B}_{\text{int}}$  (Theorem 3).
- Build a domination game  $\mathcal{G}_{\mathcal{A}, \mathcal{B}}$  and solve it (Theorem 4).

We let  $\mathcal{A} = (Q_{\mathcal{A}}, q_{\text{init}, \mathcal{A}}, \Delta_{\mathcal{A}})$  and  $\mathcal{B} = (Q_{\mathcal{B}}, q_{\text{init}, \mathcal{B}}, \Delta_{\mathcal{B}})$  be the two history-deterministic automata above, and  $d_{\mathcal{A}}$  and  $d_{\mathcal{B}}$  the number of priorities used by  $\mathcal{A}$  and  $\mathcal{B}$  respectively.

In the domination game  $\mathcal{G}_{\mathcal{A}, \mathcal{B}}$ , Adam chooses the inputs and transitions in  $\mathcal{A}$ , while Eve chooses the outputs and transitions in  $\mathcal{B}$ . A round starts in  $(p, q)$  where  $p$  is a state of  $\mathcal{A}$  and  $q$  a state of  $\mathcal{B}$ . First Adam chooses an input  $i \in I$  and a transition  $(p, i, \ell_{\mathcal{A}}, p') \in \Delta_{\mathcal{A}}$ , and then Eve chooses an output  $o \in O$  and a transition  $(q, (i, o), \ell_{\mathcal{B}}, q') \in \Delta_{\mathcal{B}}$ . The round reaches  $(p', q')$ , and its label is  $(\ell_{\mathcal{A}}, \ell_{\mathcal{B}})$ .

Formally the domination game  $\mathcal{G}_{\mathcal{A}, \mathcal{B}}$  has vertices  $V_{\text{Adam}} = Q_{\mathcal{A}} \times Q_{\mathcal{B}}$ , and  $V_{\text{Eve}} = Q_{\mathcal{A}} \times Q_{\mathcal{B}} \times I$ , and set of edges consists of moves of Adam of the form  $((p, q), (\ell_{\mathcal{A}}, \varepsilon), (p', q, i))$  with  $(p, i, \ell_{\mathcal{A}}, p') \in \Delta_{\mathcal{A}}$ , and moves of Eve of the form  $((p, q, i), (\varepsilon, \ell_{\mathcal{B}}), (p, q'))$  with  $(q, (i, o), \ell_{\mathcal{B}}, q') \in \Delta_{\mathcal{B}}$ .

A play  $\pi$  in  $\mathcal{G}_{\mathcal{A}, \mathcal{B}}$  induces a sequence of inputs  $\pi_I \in I^\omega$  and a sequence of inputs and outputs  $\pi_{I \times O} \in (I \times O)^\omega$ .

**Theorem 5.** *Eve wins in  $\mathcal{G}_{\mathcal{A}, \mathcal{B}}$  if and only if there exists a solution to the assume-guarantee synthesis problem.*

As a tool for proving Theorem 5 we construct another game. In this game  $\mathcal{G}$  Adam and Eve alternate, with Adam choosing inputs and Eve choosing outputs. The set of plays is  $(I \times O)^\omega$ , and we write  $\pi_{I \times O}$  for a play and  $\pi_I \in I^\omega$  for the induced sequence of inputs. We write  $S$  for strategies of Eve in  $\mathcal{G}$  because they indeed induce systems  $S : I^+ \rightarrow O$ . Eve wins in  $\mathcal{G}$  if for all  $b \in \mathbb{N}$ , there exists  $b' \in \mathbb{N}$  and a strategy  $S$  such that all plays consistent with  $S$  satisfy  $\llbracket \varphi \rrbracket(\pi_I) \leq b \Rightarrow \llbracket \psi \rrbracket(\pi_{I \times O}) \leq b'$ . Otherwise Adam wins.

**Fact 3.** *Eve wins in  $\mathcal{G}$  if and only if there exists a solution to the assume-guarantee synthesis problem.*

The benefit of this game reformulation is to take advantage of determinacy: since the condition is Borel, the game  $\mathcal{G}$  is determined: either Eve has a winning strategy or Adam has one. This fact, together with the following lemma, proves that the two games  $\mathcal{G}$  and  $\mathcal{G}_{\mathcal{A}, \mathcal{B}}$  are equivalent which, together with Fact 3, establishes Theorem 5.

**Lemma 2.**

- If Eve wins in  $\mathcal{G}$  then she wins in  $\mathcal{G}_{\mathcal{A}, \mathcal{B}}$ ;
- If Adam wins in  $\mathcal{G}$  then he wins in  $\mathcal{G}_{\mathcal{A}, \mathcal{B}}$ .

Theorem 5 together with Theorem 4 establish the decidability of the assume-guarantee synthesis problem for PROMPT-LTL announced in Theorem 1. Let us have a more precise complexity analysis. We start with two PROMPT-LTL formula  $\varphi$  and  $\psi$  of size  $n$ , construct two Büchi prompt automata  $\mathcal{A}_{\text{int}}$  and  $\mathcal{B}_{\text{int}}$  of size  $s = 2^{O(n)}$ , turn them into history-deterministic prompt automata  $\mathcal{A}$  and  $\mathcal{B}$  of size  $2^{2^{O(n)}}$  with  $2^{O(n)}$  priorities. This induces a domination game with  $2^{2^{O(n)}}$  vertices and  $2^{O(n)}$  priorities. The algorithm for solving this domination game runs in time  $2^{2^{O(n^2)}}$ . Thus the whole procedure runs in doubly-exponential time. The lower bound is inherited from LTL synthesis.

We leave as an open question whether the uniform assume-guarantee synthesis problem is decidable, where there exists a system independent of  $b$ . These two problems (uniform and non-uniform) are not equivalent, which is related to the fact that prompt automata can be made history-deterministic in a non-uniform way, but not in a uniform way.

## References

- [Almagor *et al.*, 2017] Shaull Almagor, Orna Kupferman, Jan Oliver Ringert, and Yaron Velner. Quantitative assume guarantee synthesis. In *CAV*, pages 353–374, 2017.
- [Alur and Henzinger, 1999] Rajeev Alur and Thomas A Henzinger. Reactive modules. *Formal methods in system design*, 15(1):7–48, 1999.
- [Alur *et al.*, 2001] Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for “model measuring”. *ACM Transactions on Computational Logic (ToCL)*, 2(3):388–407, 2001.
- [Aminof *et al.*, 2016] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. Prompt alternating-time epistemic logics. *KR*, 16:258–267, 2016.
- [Bala, 2004] Sebastian Bala. Regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms. In *STACS*, pages 596–607, 2004.
- [Bloem *et al.*, 2015] Roderick Bloem, Rüdiger Ehlers, and Robert Könighofer. Cooperative reactive synthesis. In *ATVA*, pages 394–410, 2015.
- [Brenquier *et al.*, 2017] Romain Brenquier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83, 2017.
- [Chatterjee and Henzinger, 2007] Krishnendu Chatterjee and Thomas A Henzinger. Assume-guarantee synthesis. In *TACAS*, pages 261–275, 2007.
- [Colcombet and Fijalkow, 2016] Thomas Colcombet and Nathanaël Fijalkow. The bridge between regular cost functions and omega-regular languages. In *ICALP*, pages 126:1–126:13, 2016.
- [Colcombet and Göller, 2016] Thomas Colcombet and Stefan Göller. Games with bound guess actions. In *LICS*, pages 257–266, 2016.
- [Colcombet *et al.*, 2010] Thomas Colcombet, Denis Kuperberg, and Sylvain Lombardy. Regular temporal cost functions. In *ICALP*, pages 563–574, 2010.
- [Colcombet, 2009] Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *ICALP*, pages 139–150, 2009.
- [Colcombet, 2013a] Thomas Colcombet. Fonctions régulières de coût. Habilitation Thesis, 2013.
- [Colcombet, 2013b] Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013.
- [Fijalkow *et al.*, 2018] Nathanaël Fijalkow, Bastien Maubert, Aniello Murano, and Sasha Rubin. Quantifying bounds in strategy logic. In *CSL*, 2018.
- [Filiot *et al.*, 2018] Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Rational synthesis under imperfect information. In *LICS*, pages 422–431, 2018.
- [Fisman *et al.*, 2010] Dana Fisman, Orna Kupferman, and Yoav Lustig. Rational synthesis. In *TACAS*, pages 190–204, 2010.
- [Henzinger and Piterman, 2006] Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *CSL*, pages 395–410, 2006.
- [Jacobs and Bloem, 2018] Swen Jacobs and Roderick Bloem. The 5th reactive synthesis competition (SYNTCOMP). <http://www.syntcomp.org/syntcomp-2018-results>, 2018.
- [Jacobs *et al.*, 2018] Swen Jacobs, Leander Tentrup, and Martin Zimmermann. Distributed synthesis for parameterized temporal logics. *Information and Computation*, 262:311–328, 2018.
- [Jacobs *et al.*, 2019] Swen Jacobs, Guillermo A. Pérez, and Roderick Bloem. The 6th reactive synthesis competition (SYNTCOMP). <http://www.syntcomp.org/syntcomp-2019-results/>, 2019.
- [Kirsten, 2004] Daniel Kirsten. Desert automata and the finite substitution problem. In *STACS*, pages 305–316, 2004.
- [Kuperberg and Vanden Boom, 2012] Denis Kuperberg and Michael Vanden Boom. On the expressive power of cost logics over infinite words. In *ICALP*, pages 287–298, 2012.
- [Kuperberg, 2014] Denis Kuperberg. Linear temporal logic for regular cost functions. *Logical Methods in Computer Science*, 10(1), 2014.
- [Kupferman *et al.*, 2009] Orna Kupferman, Nir Piterman, and Moshe Y Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009.
- [Kupferman *et al.*, 2016] Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Annals of Mathematics and Artificial Intelligence*, 78(1):3–20, 2016.
- [Kwiatkowska *et al.*, 2010] Marta Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Assume-guarantee verification for probabilistic systems. In *TACAS*, pages 23–37, 2010.
- [Maoz and Sa’ar, 2012] Shahar Maoz and Yaniv Sa’ar. Assume-guarantee scenarios: Semantics and synthesis. In *MoDELS*, pages 335–351, 2012.
- [Pnueli and Rosner, 1989] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.
- [Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.
- [Pnueli, 1985] Amir Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and models of concurrent systems*, pages 123–144. Springer, 1985.
- [Vardi, 1995] Moshe Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In *CAV*, pages 267–278, 1995.