Finite LTL Synthesis with Environment Assumptions and Quality Measures
Alberto Camacho, Meghyn Bienvenu, Sheila Mcilraith

To cite this version:

HAL Id: lirmm-01892548
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01892548
Submitted on 10 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Finite LTL Synthesis with Environment Assumptions and Quality Measures

Alberto Camacho  
Department of Computer Science  
University of Toronto, Canada  
acamacho@cs.toronto.edu

Meghyn Bienvenu  
CNRS, Univ. Montpellier, Inria  
Montpellier, France  
meghyn@lirmm.fr

Sheila A. McIlraith  
Department of Computer Science  
University of Toronto, Canada  
sheila@cs.toronto.edu

Abstract

In this paper, we investigate the problem of synthesizing strategies for linear temporal logic (LTL) specifications that are interpreted over finite traces—a problem that is central to the automated construction of controllers, robot programs, and business processes. We study a natural variant of the finite LTL synthesis problem in which strategy guarantees are predicated on specified environment behavior. We further explore a quantitative extension of LTL that supports specification of quality measures, utilizing it to synthesize high-quality strategies. We propose new notions of optimality and associated algorithms that yield strategies that best satisfy specified quality measures. Our algorithms utilize an automata-game approach, positioning them well for future implementation via existing state-of-the-art techniques.

1 Introduction

The problem of automatically synthesizing digital circuits from logical specifications was first proposed by Church (1957). In 1989, Pnueli and Rosner examined the problem of synthesizing strategies for reactive systems, proposing Linear Temporal Logic (LTL) (Pnueli 1977) as the specification language. In a nutshell, LTL is used to express temporally extended properties of infinite state sequences (called traces), and the aim of LTL synthesis is to produce a winning strategy, i.e., a function that assigns values to the state variables under the control of the system at every time step, in such a way that the induced infinite trace is guaranteed to satisfy the given LTL formula, no matter how the environment sets the remaining state variables.

In 2015, De Giacomo and Vardi introduced the problem of \( LTL_\Sigma \) synthesis in which the specification is described in a variant of LTL interpreted over finite traces (De Giacomo and Vardi 2013). Finite interpretations of LTL have long been exploited to specify temporally extended goals and preferences in AI automated planning (e.g., Bacchus and Kabanza 2000; Baier, Bacchus, and McIlraith 2009). In contrast to LTL synthesis, which produces programs that run in perpetuity, \( LTL_\Sigma \) synthesis is concerned with the generation of terminating programs. Two natural and important application domains are automated synthesis of business processes, including web services; and automated synthesis of robot controllers, in cases where program termination is desired.

Despite recent work on \( LTL_\Sigma \) synthesis, there is little written on the nature and form of the \( LTL_\Sigma \) specifications and how this relates to the successful and nontrivial realization of strategies for such specifications. \( LTL_\Sigma \) synthesis is conceived as a game between the environment and an agent. The logical specification that defines the problem must not only define the desired behavior that execution of the strategy should manifest—what we might loosely think of as the objective of the strategy, but must also define the context, including any assumptions about the environment’s behavior upon which realization of the objective is predicated. As we show in this work, if assumptions about environment behavior are not appropriately taken into account, specifications can either be impossible to realize or can be realized trivially by allowing the agent to violate assumptions upon which guaranteed realization of the objective is predicated.

We further examine the problem of how to construct specifications where the realization of an objective comes with a quality measure, and where strategies provide guarantees with respect to these measures. The addition of quality measures is practically motivated. In some instances we may have an objective that can be realized in a variety of ways of differing quality (e.g., my automated travel assistant may find a myriad of ways for me to get to KR2018—some more preferable than others!). Similarly, we may have multiple objectives that are mutually unachievable and we may wish to associate a quality measure with their individual realization (e.g., I’d like my home robot to do the laundry, wash dishes, and cook dinner before its battery dies, but dinner is most critical, followed by dishes).

In this paper we explore finite LTL synthesis with environment assumptions and quality guarantees. In doing so, we uncover important observations regarding the form and nature of \( LTL_\Sigma \) synthesis specifications, how resulting strategies are computed, and the nature of the guarantees we can provide regarding the resulting strategies. In Section 3 we examine the problem of \( LTL_\Sigma \) synthesis with environment assumptions, introducing the notion of constrained \( LTL_\Sigma \) synthesis in Section 4. In Section 5, we propose algorithms for constrained \( LTL_\Sigma \) synthesis, including a reduction to Deterministic Büchi Automata games for the fragment of environment constraints that are conjunctions of safe and co-safe LTL formulae. In Section 6, we examine the problem of augmenting constrained \( LTL_\Sigma \) synthesis with quality mea-
2 Preliminaries

We recall the syntax and semantics of linear temporal logic for both infinite and finite traces, as well as the basics of finite state automata and the link between LTL and automata.

2.1 Linear Temporal Logic (LTL)

Given a set \( \mathcal{P} \) of propositional variables, LTL formulae are defined as follows:

\[
\varphi ::= T \mid \bot \mid p \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid \phi \mid \varphi_1 \varphi_2 \mid \varphi_1 \forall \varphi_2 \mid \varphi_1 \exists \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \U \varphi_2
\]

where \( p \in \mathcal{P} \). Here, \( \land, \lor, \forall, \exists, \rightarrow \) are the usual Boolean connectives, and \( \U \) (until) and \( \R \) (release) are temporal operators. The formula \( \Box \varphi \) states that \( \varphi \) must hold in the next timepoint, \( \varphi_1 \U \varphi_2 \) stipulates that \( \varphi_1 \) must hold until \( \varphi_2 \) becomes true, and \( \varphi_1 \R \varphi_2 \) expresses that \( \varphi_2 \) remains true until and including the point in which \( \varphi_1 \) is made true (or forever if \( \varphi_2 \) never becomes true). For concision, we do not include logical implication (\( \rightarrow \)), eventually (\( \varphi \), ‘sometime in the future’) and always (\( \Box \), ‘at every point in the future’) in the core syntax, but instead view them as abbreviations:

\[
\sigma \rightarrow \beta := \neg \sigma \land \beta, \quad \varphi \Box := \top \varphi \quad \text{and} \quad \varphi \R := \bot \varphi.
\]

LTL formulae are traditionally interpreted over infinite traces \( \pi \), i.e., infinite words over the alphabet \( 2^\mathcal{P} \). Intuitively, an infinite trace \( \pi \) describes an infinite sequence of (time)steps, with the \( i \)-th symbol in \( \pi \), written \( \pi(i) \), specifying the propositional symbols that hold at step \( i \). We use \( \pi \subseteq \pi' \) to indicate that \( \pi \) is a prefix of \( \pi' \). We define what it means for an infinite trace \( \pi \) to satisfy an LTL formula \( \varphi \) at step \( i \), denoted \( \pi \models \varphi \), as follows:

- \( \pi \models T \) if and only if \( \pi \models p \) for each \( p \in \mathcal{P} \);
- \( \pi \models \neg \varphi \) if \( \pi \not\models \varphi \);
- \( \pi \models \varphi_1 \land \varphi_2 \) if \( \pi \models \varphi_1 \) and \( \pi \models \varphi_2 \);
- \( \pi \models \neg \varphi_1 \lor \varphi_2 \) if \( \pi \models \varphi_2 \) or \( \pi \not\models \varphi_1 \);
- \( \pi \models \Box \varphi \) if \( \pi \models \varphi \) for all \( i \); and
- \( \pi \models \varphi_1 \varphi_2 \) if there exists \( j \geq i \) such that \( \pi \models \varphi_2 \) and (for each \( i \leq k < j \), \( \pi \models \varphi_1 \))

A formula \( \varphi \) is satisfied in \( \pi \), written \( \pi \models \varphi \), if \( \pi \models \varphi \). Two formulas \( \varphi \) and \( \psi \) are equivalent if \( \varphi \models \psi \) and \( \psi \models \varphi \) for all traces \( \pi \). Observe that, in addition to the usual Boolean equivalences, we have: \( \varphi_1 \varphi_2 \equiv \neg (\neg \varphi_1 \neg \varphi_2) \) and \( \neg \varphi \equiv \Box \neg \varphi \).

We consider two well-known syntactic fragments of LTL. The safe fragment is defined as follows (Sistla 1994):

\[
\varphi ::= T \mid \bot \mid \varphi_1 \land \varphi_2 \mid \neg \varphi_1 \lor \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid \Box \varphi \mid \varphi_1 \R \varphi_2
\]

The complementary co-safe fragment is similarly defined, using \( \U \) in place of \( \R \). It is known that if \( \varphi \) is a safe formula and \( \pi \not\models \varphi \), then there is a finite bad prefix \( \pi_b \subseteq \pi \) such that \( \pi' \not\models \varphi \) for every infinite trace \( \pi' \) with \( \pi_b \subseteq \pi' \). Similarly, if \( \varphi \) is a co-safe formula and \( \pi \models \varphi \), then there exists a finite good prefix \( \pi_g \subseteq \pi \) such that \( \pi' \models \varphi \) for every infinite trace \( \pi' \) with \( \pi_g \subseteq \pi' \). This means that violation of safe formulae and satisfaction of co-safe formulae can be shown by exhibiting a suitable finite prefix (Kupferman and Vardi 2001).

In this paper, our main focus will be on a more recently studied finite version of LTL, denoted LTLf (De Giacomo and Vardi 2013), in which formulae are interpreted over finite traces (finite words over \( 2^\mathcal{P} \)). We will reuse the notation \( \pi(i) \) (\( i \)-th symbol) and introduce the notation \( |\pi| \) for the length of \( \pi \). LTLf has precisely the same syntax as LTL and the same semantics for the propositional constructs, but it differs in its interpretation of the temporal operators:

- \( \pi \models \Box \psi \) if \( |\pi| > i \) and \( \pi \models \psi \);
- \( \pi \models \neg \varphi_1 \U \varphi_2 \) if there exists \( i \leq j \leq |\pi| \) such that \( \pi \models \varphi_2 \), and \( \pi \models \varphi_1 \).
- \( \pi \models \neg \varphi_1 \R \varphi_2 \) if for all \( i \leq j \leq |\pi| \) either \( \pi \models \varphi_2 \) or there exists \( i \leq k < j \) such that \( \pi \models \varphi_1 \).

We introduce the weak next operator (\( \varphi \)) as an abbreviation: \( \varphi \Box := \Box \varphi \land \neg \top \). Thus, \( \varphi \Box \) holds if \( \varphi \) holds in the next step or we have reached the end of the trace. Over finite traces, \( \neg \varphi \equiv \neg \varphi \), but we do have \( \neg \varphi \equiv \neg \varphi \).

As before, we say that \( \varphi \) is satisfied in \( \pi \), written \( \pi \models \varphi \), if \( \pi \models \varphi \). Note that we can unambiguously use the same notation for LTL and LTLf so long as we specify whether the considered trace is finite or infinite.

2.2 Finite State Automata

We recall that a non-deterministic finite-state automaton (NFA) is a tuple \( A = (\Sigma, Q, \delta, \delta_0, F) \), where \( \Sigma \) is a finite alphabet of input symbols, \( Q \) is a finite set of states, \( \delta_0 \in Q \) is a set of initial states, \( F \subseteq Q \) is a set of accepting states, and \( \delta : \Sigma \times Q \rightarrow 2^Q \) is the transition function. NFAs are evaluated on finite words, i.e., elements of \( \Sigma^* \). A run of \( A \) on a word \( w = \theta_1 \ldots \theta_n \) is a sequence \( q_0 \ldots q_n \) of states, such that \( q_0 = \delta_0 \), and \( q_i = \delta(q_{i-1}, \theta_i) \) for all \( 0 \leq i \leq n \). A run \( q_0 \ldots q_n \) accepts if \( q_n \in F \), and \( A \) accepts \( w \) if some run of \( A \) on \( w \) is accepting. The language of an automaton \( A \), denoted \( L(A) \), is the set of words accepted by \( A \).

Deterministic finite-state automata (DFAs) are NFAs in which \( |\delta(q, \theta)| \leq 1 \) for all \( (q, \theta) \in \Sigma \times Q \). When \( A \) is a DFA, we define \( q \Rightarrow Q \times \Sigma \rightarrow Q \) and \( q = \delta(q, \theta) \) in place of \( q' = \delta(q, \theta) \) when \( q_0 = q \), and we will simply write \( q_0 \) (without the set notation). For every NFA \( A \), there exists a DFA that accepts the same language as \( A \) and whose size is at most single exponential in the size of \( A \). The powerset construction is a well-known technique to determinize NFAs (Rabin and Scott 1959).

Non-deterministic Büchi automata (NBA) are defined like NFAs but evaluated on infinite words, that is, elements of \( \Sigma^\omega \). A run of \( A \) on an infinite word \( w = \theta_1 \theta_2 \ldots \) is a sequence \( \rho = q_0 q_1 q_2 \ldots \) of states, such that \( q_0 = \delta_0 \), and \( q_{i+1} = \delta(q_i, \theta_{i+1}) \) for every \( i \geq 0 \). A run \( \rho \) is accepting if \( \text{irf}(\rho) \cap F \neq \emptyset \), where \( \text{irf}(\rho) \) is the set of states that appear infinitely often in \( \rho \). We say that an NBA \( A \) accepts \( w \) if some run of \( A \) on \( w \) is accepting. Analogous definitions apply to deterministic Büchi automata (DBAs).
We will also consider deterministic finite-state transducers (also called Mealy machines, later abbreviated to ‘transducers’), given by tuples \( T = \langle \Sigma, \Omega, Q, \delta, \omega, q_0 \rangle \), where \( \Sigma \) and \( \Omega \) are respectively the input and output alphabets, \( Q \) is the set of states, \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function, \( \omega : Q \times \Sigma \rightarrow \Omega \) is the output function, and \( q_0 \) the initial state. The run of \( T \) on \( \sigma = \omega(q_0, \theta_1) \ldots \omega(q_i, \theta_{i+1}) \) for every \( i \geq 0 \), and the output sequence of \( T \) on \( \sigma = \omega(q_0, \theta_1) \ldots \omega(q_i, \theta_{i+1}) \) for every \( i \geq 0 \).

Given an LTL formula \( \varphi \), one can construct an NFA that accepts precisely those finite traces \( \pi \) with \( \pi \models \varphi \) (e.g. (De Giacomo and Vardi 2015)). For every safe formula \( \varphi_i \) (resp. co-safe formula \( \varphi_c \)), one can construct an NFA that accepts all bad prefixes of \( \varphi_i \) (resp. good prefixes of \( \varphi_c \)) (Kupferman and Vardi 2001). In these constructions, the NFAs are worst case single exponential in the size of the formula. By determining these NFAs, we can obtain DFAs of double-exponential size that recognize the same languages.

3 LTL and LTLf Synthesis

To set the stage for our work, we recall the definition of LTL synthesis in the infinite and finite trace settings and the relationship between planning and synthesis.

3.1 LTL Synthesis

An LTL specification is a tuple \( \langle X, Y, \varphi \rangle \) where \( \varphi \) is an LTL formula over uncontrollable variables \( X \) and controllable variables \( Y \). A strategy is a function \( \sigma : (2^X)^{\ast} \rightarrow 2^Y \). The infinite trace induced by \( \sigma \) is denoted \( \text{traces}(\sigma) = \{ \pi \mid \pi \in (2^X)^{\ast} \} \) and \( \pi \models \varphi \).

The set of all infinite traces induced by \( \sigma \) is denoted \( \text{traces}(\sigma) = \{ \pi \mid \pi \in (2^X)^{\ast} \} \). The realizability problem \( \langle X, Y, \varphi \rangle \) consists in determining whether there exists a winning strategy, i.e., a strategy \( \sigma \) such that \( \pi \models \varphi \) for every \( \pi \in \text{traces}(\sigma) \). The synthesis problem is to compute such a winning strategy when one exists.

LTL synthesis can be viewed as a 2-player game between the environment \( X \) and the agent \( Y \). In each turn, the environment makes a move by selecting \( X \subseteq X \), and the agent replies by selecting \( Y \subseteq Y \). The aim is to find a strategy \( \sigma \) for the agent that guarantees the resulting trace satisfies \( \varphi \).

3.2 Finite LTL Synthesis

We now recall LTL realizability and synthesis, where LTL specification is interpreted on finite traces. An LTL specification is a tuple \( \langle X, Y, \varphi \rangle \) where \( \varphi \) is an LTL formula over uncontrollable variables \( X \) and controllable variables \( Y \). A strategy is a function \( \sigma : (2^X)^{\ast} \rightarrow (2^Y)^{\ast} \) such that for each finite sequence \( X = \{X_1, \ldots, X_{n-1}\} \) of subsets of \( X \), there is exactly one integer \( n_{X,1} \geq 1 \) with \( \text{end} \in \sigma(X_1 \cdots X_{n_{X,1}}) \). The induced infinite trace \( \pi_i[\sigma, X] \) is defined as before, and the finite trace induced by \( X \) and \( \sigma \) is

\[ \pi_i[\sigma, X] = (X_1 \cup \sigma(X_1)) \cdots (X_{n_{X,1}} \cup \sigma(X_1 \cdots X_{n_{X,1}})) \]

but with \( \text{end} \) removed from \( \sigma(X_1 \cdots X_{n_{X,1}}) \). The set of all finite traces induced by \( \sigma \) is denoted \( \text{finite traces}(\sigma) = \{ \pi_i[\sigma, X] \mid X \in (2^X)^{\ast} \} \). A finite trace \( \pi \) is compatible with \( \sigma \) if \( \pi \in \pi' \) for some \( \pi' \in \text{finite traces}(\sigma) \), with \( \text{finite traces}(\sigma) \) (‘\( \pi' \) for ‘partial’) the set of all such traces. We call \( \sigma \) a winning strategy for an LTL specification \( \langle X, Y, \varphi \rangle \) if \( \pi \models \varphi \) for every \( \pi \in \text{finite traces}(\sigma) \). The realizability and synthesis problems for LTLf are then defined in the same way as for LTL.

Comparison with prior formulations

Prior work on LTLd synthesis defined strategies as functions \( \sigma : (2^X)^{\ast} \rightarrow 2^Y \) that do not explicitly indicate the end of the trace (De Giacomo and Vardi 2015; Zhu et al. 2017; Camacho et al. 2018a). In these works, a strategy \( \sigma \) is winning iff for each \( \pi \in \text{traces}(\sigma) \) there exists some finite prefix \( \pi' \subseteq \pi \) such that \( \pi' \models \varphi \). Note that in general, multiple prefixes \( \pi' \) that satisfy \( \varphi \) may exist. We believe that it is cleaner mathematically to be precise about which trace is produced, and it will substantially simplify our technical developments. The two definitions give rise to the same notion of realizability, and existing results and algorithms for LTLf synthesis transfer to our slightly different setting.

3.3 Planning as LTLf Synthesis

It has been observed that different forms of automated planning can be recast as LTLf synthesis (see e.g. (De Giacomo and Vardi 2015; D’Ippolito, Rodríguez, and Sardinha 2018; Camacho et al. 2018b)). We recall that planning problems are specified in terms of a set of fluents (i.e., atomic facts whose value may change over time), a set of actions which can change the state of the world, an action theory whose axioms give the preconditions and effects of the actions (i.e., which fluents must hold for an action to be executable, and how do the fluents change as a result of performing an action), a description of the initial state, and a goal. In classical planning, actions are deterministic (i.e. there is a unique state resulting from performing an action in a given state), and the aim is to produce a sequence of actions leading from the initial state to a goal state. In fully observable non-deterministic (FOND) planning, actions have non-deterministic effects, meaning that there may be multiple possible states that result from performing a given action in a given state (with the effect axioms determining which states are possible results). Strong solutions are policies (i.e., functions that map states into actions) that guarantee eventual achievement of the goal.

We briefly describe how FOND planning can be reduced to LTLf synthesis,\(^\dagger\) as the reduction crucially relies on the use of environment assumptions. We will use the set \( \mathcal{F} \) of fluents as the uncontrollable variables, and the set of actions \( \mathcal{A} \) for the controllable variables. The high-level structure of the LTLf specification formula is: \( \Phi = (\Psi_{\text{sat}} \land \Psi_{\text{eff}}) \rightarrow (\Psi_{\text{pre}} \land \Psi_{\text{goal}}) \). Intuitively, \( \Phi \) states that under the assumption that the environment sets the fluents in accordance with the initial state and effect axioms (captured by \( \Psi_{\text{sat}} \) and \( \Psi_{\text{eff}} \)), the agent can choose a single action per turn (\( \Psi_{\text{one}} \)) in such a way that the preconditions are obeyed (\( \Psi_{\text{pre}} \)) and the goal is achieved (\( \Psi_{\text{goal}} \)). We set \( \Psi_{\text{goal}} = 0(\varphi \land \neg \mathcal{T}) \).

\(^\dagger\)Our high-level presentation combines elements of the reductions in (De Giacomo and Vardi 2015; Camacho et al. 2018a). Its purpose is to illustrate the general form and components of an LTLf encoding of planning (not to provide the most efficient encoding).
we further assume that the cat eventually leaves each of the rooms ($\Psi_{\text{leaves}} = \Diamond \neg \text{catIn}(BR) \land \Diamond \neg \text{catIn}(LR)$), there is an obvious solution: vacuum a cat-free room, and then simply wait until the other room is cat-free and then vacuum it. However, rather unexpectedly, adding $\Psi_{\text{leaves}}$ to $\Psi_{\text{env}}$ makes the specification $\Psi_{\text{env}} \rightarrow \Psi_{\text{robot}}$ realizable in a trivial and unintended way: by ending execution in the first move, $\neg \Psi_{\text{env}}$ trivially holds in the resulting length-1 trace $\pi$. Indeed, there are three possibilities: (i) $\pi \models \text{catIn}(BR)$ (so $\pi \models \neg \Diamond \neg \text{catIn}(BR)$), (ii) $\pi \models \Diamond \neg \text{catIn}(LR)$ (so $\pi \models \Diamond \neg \text{catIn}(LR) \land \neg \Diamond \neg \text{catIn}(BR)$), or (iii) $\pi \models \Diamond \neg \Diamond \neg \text{catIn}(LR) \land \Diamond \neg \Diamond \neg \text{catIn}(BR)$). Clearly, this length-1 strategy is not the strategy that we wanted to synthesize. In Section 4, we propose a new framework for handling environment assumptions which avoids the generation of such trivial strategies and makes it possible to find the desired strategies.

4 Constrained LTL Synthesis

To the aim of properly handling environment assumptions, we introduce a generalization of LTL synthesis, in which the assumptions are separated from the rest of the specification formula and given a different interpretation. Essentially, the idea is that the environment is allowed to satisfy the assumption over the whole infinite trace, rather than on the finite prefix chosen by the agent. This can be accomplished using LTL semantics for the environment assumption, but keeping LTL semantics for the formula describing the objective.

Formally, a constrained LTL specification is a tuple $(X, Y, \alpha, \varphi)$, where $X$ and $Y$ are the uncontrollable and controllable variables, $\alpha$ is an LTL formula over $X \cup Y$, and $\varphi$ is an LTL formula over $X \cup Y$. Here $\varphi$ describes the desired agent behavior when the environment behaves so as to satisfy $\alpha$. We will henceforth call $\varphi$ the objective, and will refer to $\alpha$ as the (environment) assumption or constraint (as it acts to constrain the allowed environment behaviors).

A strategy for $(X, Y, \alpha, \varphi)$ is a function $\sigma : (2^X)^\omega \rightarrow (2^Y)^{1+\omega}$ such that for each finite sequence $S = \langle X_i \rangle_{i \leq \omega} \in (2^X)^\omega$ of subsets of $X$, there is at most one integer $n_S \geq 1$ with $\sigma(S) = \sigma(X_1 \cdots X_{n_S})$. If none exists, we write $n_S = \infty$. To account for traces that do not contain $\sigma$, we redefine traces of $\sigma$ as follows: $\sigma(S) = \sigma(X_1 \cdots X_{n_S})$. A strategy $\sigma$ is an $\alpha$-strategy if for every $X \in (2^X)\omega$, either $n_{\sigma(X)} < \infty$ or $\pi(S) \models \alpha$, i.e. $\sigma$ terminates on every trace that satisfies $\alpha$. A winning strategy (w.r.t. $(X, Y, \alpha, \varphi)$) is an $\alpha$-strategy such that $\pi \models \alpha$ for every $\pi \in \text{traces}(\sigma)$. In other words, winning strategies are those that guarantee the satisfaction of the objective $\varphi$ under the assumption that the environment behaves in a way that constraint $\alpha$ is satisfied. The realizability and synthesis problems for constrained LTL specifications are defined as before, using this notion of winning strategy.

Because the constraints are interpreted using infinite LTL semantics, we are now able to correctly handle liveliness constraints ($\Diamond \varphi$) and fairness constraints as studied in LTL synthesis ($\Diamond \Diamond \varphi$ and FOND planning ($\Diamond \Diamond \varphi_1 \rightarrow \Diamond \Diamond \varphi_2$) (D’Ippolito, Rodríguez, and Sardina 2018).

Example 1. Returning to our earlier example, consider the constrained synthesis problem with assumption $\Psi_{\text{env}}$ (which
includes ϕ(leaf)) and objective Ψalt_robbed. The obvious strategy (vacuum dirty rooms as soon as they are cat-free) gives rise to a winning strategy, in which we output end and if we manage to clean both rooms, and otherwise, produce an infinite trace without end in which Ψalt_robbed is not true. Trivial strategies that terminate immediately will not be winning strategies, as there will be infinite traces that satisfy the constraint but where the length-1 finite trace falsifies the objective.

We remark that if we are not careful about how we write the constraint α, we may unintentionally allow the agent to block the environment from fulfilling α.

Remark 1. Suppose that instead of using Equation 1 to encode the effects of actions, we employ the simpler □((κ ∧ y) → 0/ϕ). While intuitive, this alternative formulation does not properly encode FOND planning, as the specification may be realized in an unintended way: by performing multiple actions with conflicting effects, or a single action whose precondition is not satisfied, the agent can force the environment to satisfy a contradictory set of formulae β in the next state, causing the assumption to be violated.

Chatterjee, Henzinger, and Jobstmann (2008) discuss this phenomenon in the context of LTL synthesis, and suggest that a reasonable environment constraint is one which is realizability for the environment. We note that the constraints we considered in Section 3 all satisfy this property.

Correspondence with Finite LTL Synthesis We begin by observing that (plain) LTL synthesis is a special case of constrained LTL synthesis in which one uses the trivial constraint ⊤ for the environment assumption:

Theorem 1. Every winning strategy α for the LTL specification (X, Y, α, ϕ) is a winning strategy for the constrained LTL specification (X, Y, α, ϕ), and vice-versa. In particular, (X, Y, α, ϕ) is realizable iff (X, Y, α, ϕ) is realizable.

A natural question is whether a reduction in the other direction exists. Indeed, it is well-known that in the infinite setting, assume-guarantee LTL synthesis\footnote{We refer to assume-guarantee synthesis as considered in (Chatterjee, Henzinger, and Jobstmann 2008; Almagor et al. 2017), where given a pair (α, ϕ), the aim is to construct a strategy such that every induced finite trace either violates α or satisfies ϕ. This is different from the assume-guarantee synthesis of (Chatterjee and Henzinger 2007), in which N agents each have their own goals, and the objective is for each agent to satisfy its own goals.} with an assumption α and objective ϕ corresponds to classical LTL synthesis w.r.t. α → ϕ (that is, the two synthesis problems have precisely the same winning strategies). The following negative result shows that a simple reduction via implication does not work in the finite trace setting:

Theorem 2. There exists an unrealizable constrained LTL specification S = ⟨X, Y, α → ϕ⟩ such that the LTL specifications S_{\alpha} = ⟨X, Y, α → ϕ⟩ is realizable.

Proof. Consider the constrained LTL specification S = ⟨X, Y, α, ϕ⟩ with α = ¬x ∧ δx and ϕ = Ø(x ∧ y). We claim that S is unrealizable. Indeed, take any X = X_1X_2 ... such that x \not\in X_1, x \in X_2, and x’ \not\in X_i for all i ≥ 1. Then no matter which strategy ϕ is used, the infinite trace π[ϕ, X] will satisfy α, and the induced finite trace π[f|σ, X], if it exists, will falsify ϕ (as x’ never holds).

Next consider S_α = ⟨X, Y, α → ϕ⟩, and observe that α → ϕ ≡ x ∨ (¬¬x ∨ 0(x ∧ y)). A simple winning strategy exists: output end in the first step time. Indeed, every induced trace has length 1 and hence trivially satisfies x ∨ □¬x.

With the next theorem, we observe a more fundamental difficulty in reducing constrained LTL synthesis problems to standard LTL synthesis: winning strategies for constrained problems may need an unbounded number of time steps to realize the specification, a phenomenon that does not occur in standard LTL synthesis.

Theorem 3. An LTL specification is realizable iff it admits a winning strategy, i.e. a strategy for which there exists B > 0 such that every induced finite trace has length at most B. There exist realizable constrained LTL specifications that do not possess any bounded winning strategy.

Proof. A straightforward examination of the LTL synthesis algorithm\footnote{The algorithm can be easily modified to output end once ϕ has been satisfied to match our definition of strategy.} in (De Giacomo and Vardi 2015) shows that when (X, Y, α, ϕ) is realizable, the produced strategy guarantees achievement of ϕ in a number of time steps bounded by the number of states in a DFA for ϕ.

For the second point, consider the constrained LTL specification S = ⟨X, Y, α → ϕ⟩. Assume for a contradiction that there is a winning strategy σ for S and constant B > 0 such that n_{σ,X} ≤ B for every X ∈ X^ω. Define X^B = X_1X_2 ... as follows: X^i = {x} if i = B + 1 and X^i = ∅ otherwise. The induced trace π = π[σ, X^B] has length at most B and hence does not contain x. It follows that π ∉ ϕ, contradicting our assumption that σ is a winning strategy.

While the implication-based approach does not work in general, we show that it can be made to work for environment assumptions that belong to the safe fragment:

Theorem 4. When α is a safe formula, the constrained LTL specification S = ⟨X, Y, α → ϕ⟩ is realizable iff the LTL specification S = ⟨X, Y, α’ → ϕ⟩ is realizable, where α’ is obtained from α by replacing every occurrence of ϕ by δϕ.

Proof sketch. Let α’ be a winning strategy for (X, Y, α’ → ϕ), with α a safe formula. To define a winning strategy σ for (X, Y, α, ϕ), we set σ(X_1 · · · X_n) equal to

σ(X_1 · · · X_n) \{ end \}, when end \in σ(X_1 · · · X_n) and

• σ(X_1 ∪ σ(X_1)) ... (X_n ∪ σ(X_1 · · · X_n)) \neq α’;

• σ(X_1 · · · X_n), otherwise.

For the other direction, given a winning strategy σ for (X, Y, α, ϕ), we can define a winning strategy σ’ for (X, Y, α’ → ϕ) by setting σ’(X_1 · · · X_n) equal to

σ(X_1 · · · X_n) \{ end \}, if end \in σ’(X_1 · · · X_n) and

• σ(X_1 · · · X_n) ∪ \{ end \}, if end \in σ’(X_1 · · · X_n) for k < n;

• σ’(X_1 · · · X_n) ∪ \{ end \}, if end \in σ’(X_1 · · · X_n) for some k < n;

• σ’(X_1 · · · X_n) = σ(X_1 · · · X_n), otherwise.

□
The following example shows that it is essential in the preceding theorem to use \( \alpha' \rightarrow \varphi \) rather than \( \alpha \rightarrow \varphi \):

**Example 2.** If we let \( \alpha = \Box(\neg x \vee \Box y) \) and \( \varphi = \neg x \wedge y \), then the constrained specification \((X, Y, \alpha, \varphi)\) is not realizable (as the environment can output \( x \) in the first step), but the LTL specification \((X, Y, \alpha \rightarrow \varphi)\) is realizable with a strategy that outputs \( y \) and \( x \) in the induced length-1 trace; if we have \( \neg x \) instead, then \( \varphi \) holds.

Note however that the negative result in the general case (Theorem 2) continues to hold if \( \alpha' \rightarrow \varphi \) is used instead of \( \alpha \rightarrow \varphi \), since the formulas in that proof do not involve \( \Box \).

Another interesting observation is the environment assumptions \( \Psi_{\text{out}} \) and \( \Psi_{\text{eff}} \) used to encode the initial state and the action effects in planning are safe formulas. This explains why these constraints can be properly encoded in LTL\(_f\) using implication and \( \bullet \) rather than \( \Box \). We note that if we encode planning using constrained LTL\(_f\) synthesis, then we can use \( \Box \) in the effect axioms, which is arguably more natural.

**Reduction to LTL Synthesis** Every LTL\(_f\) formula \( \varphi \) over \( P \) can be polynomially transformed into an LTL formula \( \varphi_{\text{end}} \) over \( P \cup \{\text{alive}\} \) such that for any prefix \( \pi' \models \varphi \) for some finite prefix \( \pi' \models \varphi \), \( (\text{De Giacomo and Vardi 2013}) \). Intuitively, \( \text{alive} \) holds for the duration of the (simulated) finite trace. Formally, \( \varphi_{\text{end}} : = (\varphi \wedge \text{alive}) \cup (\Box \varphi \wedge \neg \text{alive}) \), where:

\[
\tau(p) = \begin{cases} 
\tau(\neg \varphi) \quad &\text{if } \neg \varphi \quad \text{for some finite prefix } \pi' \models \varphi, \text{ or } \pi \models \varphi \text{ and } \neg \varphi \quad \text{finitely many times} \\
\tau(\varphi) \quad &\text{for infinitely many times}
\end{cases}
\]

\[
\tau(\Box \varphi) = (\Box \varphi \wedge \text{alive} \cup \Box \neg \varphi) \wedge \neg \text{alive} 
\]

We extend this transformation as follows:

\[
\psi_{\text{end}} : = (\psi(\text{end} \leftrightarrow \text{alive} \wedge \Box \neg \text{alive})) \wedge (\Box (\text{end} \rightarrow \Box \neg \psi_{\text{end}}))
\]

Here \( \psi_{\text{end}} \) forces the agent to trigger variable \( \text{end} \) when the end of the trace is simulated and also ensures that \( \text{end} \) occurs at most once. Formula \( \psi_{\text{end}} \) ensures that \( \varphi_{\text{end}} \) is satisfied – i.e., a finite trace that satisfies \( \varphi \) and \( \text{end} \) is simulated – when either the environment assumption \( \alpha \) holds or \( \text{end} \) occurs.

**Theorem 6.** The constrained LTL\(_f\) specification \( S = (X, Y, \alpha, \varphi) \) is realizable if \( \text{LTL specification } S' = (X, Y, \{\text{alive}\}, \psi_{\text{end}}) \psi_{\text{end}} \) is realizable. Moreover, for every winning strategy \( \sigma \) for \( S' \), the strategy \( \sigma' \) defined by \( \sigma'(X_1 \cdots X_n) : = \sigma(X_1 \cdots X_n) \cup \{\text{alive}\} \) is a winning strategy for \( S \).

**5 Algorithms for Constrained LTL\(_f\) Synthesis**

LTL\(_f\) realizability is both 2EXP-complete (Pnueli and Rosner 1989; De Giacomo and Vardi 2015), and we can show the same holds for constrained LTL\(_f\) problems. The upper bound exploits the reduction to LTL (Theorem 5), and the lower bound is inherited from (plain) LTL\(_f\) synthesis, which is a special case of constrained LTL\(_f\) synthesis (Theorem 1).

**Theorem 6.** Constrained LTL\(_f\) realizability (resp. synthesis) is 2EXP-complete (resp. in 2EXP).

It follows from Theorem 6 that the reduction to infinite LTL realizability and synthesis yields worst-case optimal algorithms. However, we argue that the reduction to LTL does not provide a practical approach. Indeed, while LTL and LTL\(_f\) synthesis share the same worst-case complexity, recent experiments have shown that LTL\(_f\) is much easier to handle in practice (Zhu et al. 2017). Indeed, state-of-the-art approaches to LTL synthesis rely on first translating the LTL formula into a suitable infinite-word automata, then solving a two-player game on the resulting automaton. The computational bottleneck is the complex transformations of infinite-word automata, for which no efficient implementations exist. Recent approaches to LTL\(_f\) synthesis also adopt an automata-game approach, but LTL\(_f\) formulae require only finite-word automata (NFAs and DFAs), which can be manipulated more efficiently.

The preceding considerations motivate us to explore an alternative approach to constrained LTL\(_f\) synthesis, which involves a reduction to DBA games. Importantly, the DBA can be straightforwardly constructed from DFAs for the constraint and objective formulae, allowing us to sidestep the difficulties of manipulating infinite-word automata.

**5.1 DBA for Constrained Specifications**

For the rest of this section, we assume \( \alpha = \alpha_s \wedge \alpha_c \), where \( \alpha_s \) is a safe formula and \( \alpha_c \) is a co-safe formula, both defined over \( X \cup Y \). Safe and co-safe formulae are well-known LTL fragments (Kupferman and Vardi 2001) of proven utility. Safe formulae are prevalent in LTL specifications and a key part of the encoding of planning as LTL\(_f\) synthesis (see Section 3.3); the usefulness of co-safe formulae can be seen from our example (Section 3.4) and their adoption in work on robot planning (see e.g. (Lahijanian et al. 2015)).

Our aim is to construct a DBA that accepts infinite traces over \( 2^{X \cup Y \cup \{\text{end}\}} \) such that either (i) \( \pi \) contains a single occurrence of \( \text{end} \) which induces a finite prefix \( \pi' \models \varphi \), or (ii) \( \pi \) does not contain \( \text{end} \) and \( \pi \not\models \alpha_s \wedge \alpha_c \). Such a DBA \( \mathcal{A}_{\alpha_s,\alpha_c} \) can be defined by combining three DFAs:

\[
\mathcal{A}_s = (2^P, Q_s, \delta_s, (q_0), F_s)
\]

accepts the bad prefixes of \( \alpha_s \); \( \mathcal{A}_c = (2^P, Q_c, \delta_c, (q_0), F_c) \) accepts the good prefixes of \( \alpha_c \); and \( \mathcal{A}_\tau = (2^P, Q_\tau, \delta_\tau, (q_0), F_\tau) \) accepts models of \( \varphi \).

Recall that these DFAs can be built in double-exponential time.

Formally, we let \( \mathcal{A}_{\alpha_s,\alpha_c} = (2^P \cup \{\text{end}\}, Q, \delta, q_0, F) \), where \( Q, q_0, F \) and \( \delta \) are defined as follows:

- \( Q = (Q_s \cup \{q_{\text{bad}}\} \times Q_c \cup \{q_{\text{good}}\} \cup \{q_{\text{bad}}\}) \cup \{q_T, q_\perp\} \)
- \( q_0 = ((q_0), (q_0), (q_0)) \)
- \( F = \{q_{\text{bad}}, q_{\text{good}}, q_T\} \cup \{q_{\text{good}}\} \cup \{q_T\} \)

For ‘regular’ symbols \( \theta \) in \( 2^P \) (i.e., end \( \notin \theta \) ), we set \( \delta((q_s, q_c, q_\tau), \theta) = (\delta_s(q_s, \theta), \delta_c(q_c, \theta), \delta_\tau(q_\tau, \theta)) \) where:

\[
\delta_s(q_s, \theta) = \begin{cases} 
q_{\text{bad}} \quad &\text{if } q_s = q_{\text{bad}} \text{ or } \delta_s(q_s, \theta) \in F_s \\
q_s \quad &\text{otherwise}
\end{cases}
\]

\[
\delta_c(q_c, \theta) = \begin{cases} 
q_{\text{good}} \quad &\text{if } q_c = q_{\text{good}} \text{ or } \delta_c(q_c, \theta) \in F_c \\
q_c \quad &\text{otherwise}
\end{cases}
\]

For \( \theta \) with end \( \in \theta \) , we set \( \delta((q_s, q_c, q_\tau), \theta) = q_T \) if \( \delta_c(q_c, \theta) \in F_c \) and \( \delta((q_s, q_c, q_\tau), \theta) = q_\perp \) in all other cases. Accepting state \( q_T \) is quasi-absorbing: \( \delta((q_s, \theta) \in F_c \) when end \( \notin \theta \) , and \( \delta(q_T, \theta) = q_\perp \) otherwise. This forces winning
strategies to output variable end at most once. Finally, q<sub>1</sub> is an absorbing state: δ(q<sub>1</sub>, δ) = q<sub>1</sub> for every δ ∈ 2<supΧ×Θ</sup>.

**Theorem 7.** The DBA δ<sub>α</sub>-DBA accepts infinite traces π such that either: (i) π(1) · · · π(n) = ϕ and end occurs only in π(n), or (ii) π ⌊ α₁ ∧ α₂, and end does not occur in π. δ<sub>α</sub>-DBA can be constructed in double-exponential time in |α₁| + |α₂| + |ϕ|.

### 5.2 DBA Games

Once a specification has been converted into a DBA, realizability and synthesis can be reduced to DBA games. We briefly recall next the definition of such games and how winning strategies can be computed.

A **DBA (or Büchi) game** (see e.g. (Chatterjee, Henzinger, and Piterman 2006)) is a two-player game given by a tuple ⟨X, Φ, A⟩, where X and Φ are disjoint finite sets of variables and A is a DBA with alphabet 2<supΧ×Θ</sup>. A play is an infinite sequence of rounds, where in each round, Player I selects X<sub>i</sub> ⊆ X, then Player II selects Y<sub>i</sub> ⊆ Φ. A play is winning if it yields a word (X<sub>1</sub> ∪ Y<sub>1</sub>) X<sub>2</sub> ∪ Y<sub>2</sub> ... that belongs to L(A). A game is **winning** if there exists a strategy σ : 2<supΧ</sup> → 2<supΦ</sup> such that for every infinite sequence X<sub>i</sub> ∈ X<supΧ</sup>, the word (X<sub>1</sub>∪ Y<sub>1</sub>)(X<sub>2</sub>∪ Y<sub>2</sub>...(X<sub>i</sub>∪ Y<sub>i</sub>)) obtained by following σ belongs to L(A). In this case, we call σ a winning strategy.

Existence of a winning strategy for a DBA game \( G = \langle X, \Phi, A \rangle \) based upon \( A = \langle 2^{X \times \Theta}, \delta, \delta_0, F \rangle \) can be determined by computing the winning region of G. This is done in two steps. First, we compute the set RA(G) of recurring accepting states, i.e. those q ∈ F such that Player II has a strategy from state q to revisit F infinitely often. Next, we define the **winning region** Win(G) of A as those states in q ∈ Q for which Player II has a strategy for reaching a state in RA(G). The sets RA(G) and Win(G) can be computed in polynomial time by utilizing the controllable predecessor operator: CPre(S) = {q ∈ Q | ∀ X ⊆ X ∀ Y ⊆ Φ : δ(q, X ∪ Y) ∈ S}. We set Reach<sup>0</sup>(S) = S and Reach<sup>n+1</sup>(S) = Reach<sup>n</sup>(S) ∪ CPre(Reach<sup>n</sup>(S)). Intuitively, Reach<sup>n</sup>(S) contains those states from which Player II has a strategy for reaching (or returning to) S in at most n rounds. The limit lim<sub>n→∞</sub> Reach<sup>n</sup>(S) exists because Reach<sup>n</sup>(S) ⊆ Reach<sup>n+1</sup>(S), and convergence is achieved in a finite number of iterations bounded by |Q|. To compute RA(G), we set S<sub>1</sub> = F and let S<sub>n+1</sub> = S<sub>n</sub> ∩ lim<sub>n→∞</sub> Reach<sup>n</sup>(S<sub>n</sub>). The set S<sub>n</sub> contains those accepting states from which Player II has a strategy for visiting S<sub>n</sub> less than k times. The limit lim<sub>n→∞</sub> S<sub>n</sub> exists because S<sub>n</sub> ⊆ S<sub>n+1</sub>, and convergence is achieved in a finite number of iterations bounded by |F|. RA(G) is the finite limit of S<sub>n</sub>, and the set Win(G) is then the finite limit of Reach<sup>n</sup>(RA(G)). It is easy to see that Win(G) can be computed in polynomial time w.r.t. the size of the DBA A. The following well-known result shows how we can use Win(G) to decide if G is winning.

**Theorem 8.** G is winning iff q<sub>0</sub> ∈ Win(G).

We sketch the proof of the right-to-left implication here, since it will be needed for later results. We suppose that q<sub>0</sub> ∈ Win(G) and show how to construct a transducer T<sub>G</sub> that implements a winning strategy. Intuitively, the transducer’s output function ensures that the transducer stays within Win(G), always reducing the ‘distance’ to RA(G).

More precisely, we can define T<sub>G</sub> as (2<supΧ</sup>, 2<supΦ</sup>, Q, δ', ω, q<sub>0</sub>), where: the set of states Q and initial state q<sub>0</sub> are the same as for the DBA A, and the transition function δ' mirrors the transition function δ of A. δ'(q, X) = δ(q, X ∪ ω(q, X)). We define the output function ω as follows:

- Case 1: there exists Y such that δ(q, X ∪ Y) ∈ Win(G).
  In this case, we let ω(q, X) be any Y ∈ 2<supΦ</sup> such that (δ(q, X ∪ Y) ∈ Reach<sup>n</sup>(RA(G)), and (b) there is no Y with δ(q, X ∪ Y) ∈ Reach<sup>n</sup>(RA(G)).
- Case 2: no such Y exists. We let ω(q, X) be any Y ∈ 2<supΦ</sup>.

According to this definition, after reading X, the transducer T<sub>G</sub> chooses an output symbol Y that allows the underlying automaton A to transition from the current state via X ∪ Y to a state in Win(G) (if some symbol exists). Moreover, among the immediately reachable winning states, preference is given to those that are closest to RA(G), i.e. those belonging to Reach<sup>n</sup>(RA(G)) for the minimal value of n.

### 5.3 Constrained LTL₁ Synthesis via DBA Games

Given a constrained LTL₁ synthesis specification ⟨X, Φ, α, ϕ⟩ with α = α₁ ∧ α₂, we proceed as follows:

1. Construct the DBA game G<sub>α</sub>-DBA = (X, Φ ∪ {end}, δ<sub>α</sub>-DBA).
2. Determine whether G<sub>α</sub>-DBA is winning: build Win(G<sub>α</sub>-DBA) and check whether (q<sub>0</sub>, q<sub>0</sub>) ∈ Win(G<sub>α</sub>-DBA).
3. If G<sub>α</sub>-DBA is not winning, return ‘unrealizable’.
4. Otherwise, compute a winning strategy for G<sub>α</sub>-DBA using the transducer from Section 5.2.

Using Theorems 7 and 8, we can show that this method is correct and yields optimal complexity:

**Theorem 9.** Consider a constrained LTL₁ specification S = (X, Φ, α, ϕ) where α₁ (resp. α₂) is a safe (resp. co-safe) formula. Then:

- S is realizable if the DBA game G<sub>α</sub>-DBA is winning;
- Every winning strategy for G<sub>α</sub>-DBA is a winning strategy for S, and vice-versa;
- Deciding whether G<sub>α</sub>-DBA is winning, and constructing a winning strategy when one exists, can be done in 2EXP.

### 6 Synthesis of High-Quality Strategies

This section explores the use of a quantitative specification language to compare strategies based upon how well they satisfy the specification. We adopt the LTL₁[ϕ] language from (Almagor, Boker, and Kupferman 2016) and propose a new more refined way of defining optimal strategies.

#### 6.1 The Temporal Logic LTL₁[ϕ]

We recall here the language LTL₁[ϕ] proposed by Almagor, Boker, and Kupferman (2016). The basic idea is that instead of a formula being either totally satisfied or totally violated by a trace, a value between 0 and 1 will indicate its degree of satisfaction. In order to allow for different ways of aggregating formulae, the basic LTL syntax is augmented with a set F ⊆ {f : [0, 1] → [0, 1]} | k ∈ N| of functions, with the

\[^{\text{Several } Y \text{ may satisfy the conditions, and choosing any such } Y \text{ yields a suitable transducer. Alternatively, one can use non-deterministic transducers (called strategy generators in (De Giacomo and Vardi 2015)) to encode a family of deterministic transducers.}}\]
choice of which functions to include in $F$ being determined by the application at hand.

Formally, the set of $\text{LTL}_i[F]$ formulae is obtained by adding $f(\varphi_1, \ldots, \varphi_k)$ to the grammar for $\varphi$, for every $f \in F$. We assign a satisfaction value to every $\text{LTL}_i[F]$ formula, finite trace $\pi$, and time step $1 \leq i \leq |\pi|$, as follows$^6$:

$$
\begin{align*}
\llbracket \pi, \top \rrbracket & = 1 \\
\llbracket \pi, \bot \rrbracket & = 0 \\
\llbracket \pi, \rho \rrbracket & = \begin{cases} 
1 & \text{if } p \in \pi(i) \\
0 & \text{otherwise}
\end{cases}
\end{align*}
$$

$\llbracket \pi, \neg \varphi \rrbracket = 1 - \llbracket \pi, \varphi \rrbracket$

$\llbracket \pi, \varphi_1 \land \varphi_2 \rrbracket = \min \{\llbracket \pi, \varphi_1 \rrbracket, \llbracket \pi, \varphi_2 \rrbracket\}$

$\llbracket \pi, f(\varphi_1, \ldots, \varphi_k) \rrbracket = f(\llbracket \pi, \varphi_1 \rrbracket, \ldots, \llbracket \pi, \varphi_k \rrbracket)$

$\llbracket \pi, \varphi_1 \cup \varphi_2 \rrbracket = \max \{\min \{\llbracket \pi, \varphi_1 \rrbracket, \min \{\llbracket \pi, \varphi_1 \rrbracket \}\}\}$

The (satisfaction) value of $\varphi$ on $\pi$, written $\llbracket \pi, \varphi \rrbracket$, is $\llbracket \pi, \varphi \rrbracket$.

We define $V(\varphi) \subseteq [0, 1]$ as the set of values $\llbracket \pi, \varphi \rrbracket$, ranging over all traces $\pi$ and steps $1 \leq i \leq |\pi|$. The following proposition, proven by (Almagor, Boker, and Kupferman 2016), shows that an $\text{LTL}_i[F]$ formula can take on only exponentially many different values.

**Proposition 1.** For every $\text{LTL}_i[F]$ formula $\varphi$, $|V(\varphi)| \leq 2^{2^{|F|}}$.

The functions $f$ allow us to capture a diversity of methods for combining a set of potentially competing objectives (including classical preference aggregation methods like weighted sums and lexicographic ordering).

**Example 3.** For illustration purposes, consider two variants of our robot vacuum example, with specifications $\varphi_1$ and $\varphi_2$, in which the goal $\phi(\text{clean}(LR) \land \text{clean}(BR))$ is replaced by $\phi(\text{clean}(LR))$ and $\phi(\text{clean}(BR))$, respectively. We can include in $F$ a binary weighted sum operator $\text{sum}_{0,0,0.7}$, where the satisfaction value of $\text{sum}_{0,0,0.7}(\varphi_1, \varphi_2)$ on trace $\pi$ is 0.3 if $\pi \models \varphi_1 \land \neg \varphi_2$, 0.7 if $\pi \models \neg \varphi_1 \land \varphi_2$, 1 if $\pi \models \varphi_1 \land \varphi_2$, and zero otherwise. We can thus express that we’d like to clean both rooms, but give priority to the bedroom.

### 6.2 Defining Optimal Strategies

Henceforth, we consider a constrained synthesis $\text{LTL}_i[F]$ problem $(X, Y, \alpha, \varphi)$, defined as before except that now $\varphi$ is an $\text{LTL}_i[F]$ formula. Such formulae assign satisfaction values to traces, allowing us to rank traces according to the extent to which they satisfy the expressed preferences. It remains to lift this preference order to strategies.

Perhaps the most obvious way to rank strategies is to consider the minimum value of any trace induced by the strategy, preferring strategies that can guarantee the highest worst-case value. This is the approach adopted by (Almagor, Boker, and Kupferman 2016) for $\text{LTL}_i[F]$ synthesis. We formalize it for constrained $\text{LTL}_i[F]$ synthesis as follows:

**Definition 1.** The best guaranteed value of strategy $\sigma$, denoted $\text{bgv}(\sigma)$, is the minimum of $\llbracket \pi, \varphi \rrbracket$ over all $\pi \in \text{traces}(\sigma)$ (or 0 if $\text{traces}(\sigma) = \emptyset$). A strategy $\sigma$ is $\text{bgv}$-optimal w.r.t. $(\alpha, \varphi)$ if it is an $\alpha$-strategy and no $\alpha$-strategy $\sigma'$ exists with $\text{bgv}(\sigma') > \text{bgv}(\sigma)$.

Optimizing for the best guaranteed value seems natural, but can be insufficiently discriminative. Consider a simple scenario with $X = \{x\}$ and $Y = \{y\}$. If the environment plays $x$, then we get value 0 no matter what, and if $\neg x$ is played, a value of 1 is achieved by playing $y$, and 0 if $\neg y$ is played. Clearly, we should prefer to play $y$ after $\neg x$, yet the strategy that plays $\neg y$ following $\neg x$ is bgv-optimal, since like every strategy, its bgv is 0. This motivates us to introduce a stronger, context-aware, notion of optimality:

**Definition 2.** Given a strategy $\sigma$, trace $\pi \in \text{traces}(\sigma)$ that does not contain end, and $X \in 2^X$, the best guaranteed value of $\sigma$ starting from $\pi, X$, written $\text{bgv}_{\text{End}}(\sigma)$, is the minimum of $\llbracket \pi, \varphi \rrbracket$ over all traces $\pi' \in \text{traces}(\sigma)$ such that $\pi \cdot (X \cup Y) = \pi'$ for some $Y \in Y$ (or 0 if no such trace exists). A strategy $\sigma$ is strongly $\text{bgv}$-optimal w.r.t. $(\alpha, \varphi)$ if it is an $\alpha$-strategy, and there is no $\alpha$-strategy $\sigma'$, trace $\pi \in \text{traces}(\sigma') \cap \text{traces}(\sigma')$ without end, and $X \in 2^X$ such that $\text{bgv}_{\text{End}}(\sigma') > \text{bgv}_{\text{End}}(\sigma)$.

Strongly bgv-optimal strategies take advantage of any favorability during execution to improve the best worst-case value. In the preceding example, they allow us to say that the first strategy is better than the second.

### 7 Algorithms: High-Quality $\text{LTL}_i[F]$ Synthesis

In this section, we present novel techniques to compute bgv-optimal and strongly bgv-optimal strategies for a constrained $\text{LTL}_i[F]$ synthesis problem $(X, Y, \alpha, \varphi)$. As in Section 5.1, we focus on the case where $\sigma$ is a conjunction $\alpha_1 \land \alpha_2$ of safe and co-safe formulae.

#### 7.1 Automaton for $\text{LTL}_i[F]$

It has been shown in (Almagor, Boker, and Kupferman 2016) how to construct, for a given $\text{LTL}_i[F]$ formula $\varphi$ and set of values $V \subseteq [0, 1]$, an NFA $\mathcal{A}_{\varphi, V} = (2^F, \delta, \delta_0, V)$ that accepts finite traces $\pi$ with $\llbracket \pi, \varphi \rrbracket \in V$. We briefly recall the construction here. We denote by $\text{sub}(\varphi)$ the set of subformulas of $\varphi$, and let $\mathcal{C}_\varphi$ be the set of all functions $g : \text{sub}(\varphi) \to [0, 1]$ such that $g(\psi) \in V(\psi)$ for all $\psi \in \text{sub}(\varphi)$. $Q$ contains all consistent functions in $\mathcal{C}_\varphi$, where a function $g$ is consistent if, for every $\psi \in \text{sub}(\varphi)$, the following hold:

- if $\psi = \top$, then $g(\psi) = 1$, and if $\psi = \bot$, then $g(\psi) = 0$
- if $\psi \in P$ then $g(\psi) \in [0, 1]$
- if $\psi = f(\psi_1, \ldots, \psi_k)$, then $g(\psi) = f(g(\psi_1), \ldots, g(\psi_k))$

The transition function $\delta$ is such that $g' \in \delta(g, \sigma)$ whenever:

- $\sigma = \{p \in P \mid g(p) = 1\}$
- $g(\psi_1) \in \text{sub}(\varphi)$
- $g(\psi_1, \psi_2) \in \text{sub}(\varphi)$

Finally, the set of initial states is $Q_0 = \{g \in Q \mid g(\psi) \in V, \forall \psi \in \text{sub}(\varphi)\}$, and $F = \{g \mid g(\psi_1, \psi_2) \in \text{sub}(\varphi) \land \forall \psi, \psi_1, \psi_2 \in \text{sub}(\varphi) \land \forall \psi \in \text{sub}(\varphi)\}$.

The NFA $\mathcal{A}_{\varphi, V}$ can be constructed in single-exponential time, and $L(\mathcal{A}_{\varphi, V}) = \{\llbracket \pi, \varphi \rrbracket \in V\}$ (Almagor, Boker, and Kupferman 2016). By determining $\mathcal{A}_{\varphi, V}$, we obtain a DFA $\mathcal{A}_{\varphi, V}$ that accepts the same language and can be constructed in double-exponential time. In what follows, $V$ will always take the form $[b, 1]$, so we’ll use $\mathcal{A}_{\varphi, b}$ in place of $\mathcal{A}_{\varphi, [b, 1]}$. 

---

$^6$We omit $\top$ and $\bot$, as they can be defined using $\neg$, $\land$, and $\lor$. 

---

---
7.2 Synthesis of bgv-optimal strategies

We describe how to construct a bgv-optimal strategy. First note that given $b \in [0,1]$, we can construct a DBA $\mathcal{A}_\rho$ that recognizes traces such that either (i) $\alpha = \alpha_f \land \alpha_v$ is violated and $\text{end}$ does not occur, or (ii) $\text{end}$ occurs exactly once and the induced finite trace $\pi$ is such that $[\pi, \rho] \ni b$. Indeed, we simply reuse the construction from Section 5.1, replacing the DFA $\mathcal{A}_b$ with the DFA $\mathcal{A}_\rho$. We next observe that an $\alpha$-strategy $\sigma$ with $\text{bgv}(\sigma) \geq b$ exists if the DBA game $\langle X, Y, \emptyset \cup \{\text{end}\}, \mathcal{A}_\rho \rangle$ is winning. Thus, by iterating over the values in $V(\rho)$ in descending order, we can determine the maximal $b^*$ for which an $\alpha$-strategy $\sigma$ with $\text{bgv}(\sigma) \geq b^*$ exists. A bgv-optimal strategy can be computed by constructing a winning strategy for the DBA game $\langle X, Y, \emptyset \cup \{\text{end}\}, \mathcal{A}_\rho \rangle$, using the approach in Section 5.2. As there are only exponentially many values in $V(\rho)$ (Prop. 1), the overall construction takes double-exponential time.

**Theorem 10.** A bgv-optimal strategy can be constructed in double-exponential time.

7.3 Synthesis of strongly bgv-optimal strategies

To compute a strongly bgv-optimal strategy, we build a transducer that runs in parallel DBAs $\mathcal{A}_\rho$ for different values $b$, and selects outputs symbols so as to advance within the 'best' applicable winning region. This idea can be formalized as follows. As in Section 7.2, we first determine the maximal $b^* \in V(\rho)$ for which an $\alpha$-strategy $\sigma$ with $\text{bgv}(\sigma) \geq b^*$ exists, and set $B = V(\rho) \cap [b^*, 1]$. In the process, we will compute, for each $b \in B$, the sets $\text{Win}(\mathcal{G}_b)$ and $\text{RA}(\mathcal{G}_b)$ for the DBA game $\mathcal{G}_b = \langle X, Y, \emptyset \cup \{\text{end}\}, \mathcal{A}_\rho \rangle$ based on the DBA $\mathcal{A}_\rho = (2^{X \cup \{\text{end}\}}, Q_0, \delta_\text{str}, \omega_\text{str}, \emptyset)$. In the sequel, we will assume that the elements of $B$ are ordered as follows: $b_1 < b_2 \ldots < b_m$, with $b^* = b_1$ and $b_m = 1$.

We now proceed to the definition of the desired transducer $T_{\text{str}} = (2^X \cdot 2^Y \cup \{\text{end}\}, Q_{\text{str}}, \delta_{\text{str}}, \omega_{\text{str}}, q_0^{\text{str}})$, obtained by taking the cross product of the set of DBAs $\mathcal{A}_\rho$ with $b \in B$, in order to keep track of the current states in these automata:

- $q_0^{\text{str}} = (q_0^b, q_0^b, \ldots, q_0^b)$ and $Q_{\text{str}} = \{ q_0^b \times \ldots \times q_0^b \}$
- $\delta_{\text{str}}((q_1, \ldots, q_m), X) = (\delta_b(q_1, X \cup Y), \ldots, \delta_b(q_m, X \cup Y))$, where $Y = \omega_{\text{str}}((q_1, q_2, \ldots, q_m), X)$

After reading $X$, the output function identifies the maximal value $b \in B$ such that current state $q_b$ of $\mathcal{A}_\rho$ can transition, via some symbol $X \cup Y$, into a state in $\text{Win}(\mathcal{G}_b)$, and it returns the same output as the transducer $T_{\text{str}}$ in state $q_b$:

- $\omega_{\text{str}}((q_1, \ldots, q_m), X) = \omega_b(q_b, X)$, where $b = \max_b \{ b \in B \mid \exists Y \in \text{Win}(\mathcal{G}_b) \}$

We note that the transducer $T_{\text{str}}$ can be defined as in Section 5.2 even when $q_0 \not\in \text{Win}(\mathcal{G}_b)$, but it only returns 'sensible' outputs when it transitions to $\text{Win}(\mathcal{G}_b)$.

**Theorem 11.** $T_{\text{str}}$ implements a strongly bgv-optimal strategy and can be constructed in double-exponential time.

8 Discussion and Concluding Remarks

It has been widely remarked in the (infinite) LTL synthesis literature that environment assumptions are ubiquitous: the existence of winning strategies is almost always predicated on some kind of environment assumption. This was observed in the work of (Chatterjee, Henzinger, and Jobstmann 2008), motivating the introduction of the influential assume-guarantee synthesis model, and in work on rational synthesis (Fisman, Kupferman, and Lustig 2010), where the environment is assumed to act as a rational agent, and synthesis necessitates finding a Nash equilibrium. Interesting reflections on the role of assumptions in LTL synthesis, together with a survey of the relevant literature, can be found in (Bloem et al. 2014).

In this paper, we explored the issue of handling environment assumptions in LTL synthesis (De Giacomo and Vardi 2013), the counterpart of LTL synthesis for programs that terminate. Our starting point was the observation that the standard approach to handling assumptions in LTL synthesis (via logical implication) fails in the finite-trace setting. This led us to propose an extension of LTL synthesis that explicitly accounts for environment assumptions. The key insight underlining the new model of constrained LTL synthesis is that while the synthesized program must realize the objective in a finite number of steps, the environment continues to exist after the program terminates, so environment assumptions should be interpreted under infinite LTL semantics.

We studied the relationships holding between constrained LTL synthesis and (standard) LTL synthesis. In particular, we identified a fundamental difficulty in reducing constrained LTL synthesis to LTL synthesis — the former problem can require unbounded strategies, while bounded strategies suffice for the latter. Nevertheless, when the constraints were restricted to the safe LTL fragment, a reduction from constrained LTL synthesis to LTL synthesis is possible. Interestingly, this explains why planning — more naturally conceived as a constrained LTL synthesis problem — can also be encoded as LTL synthesis. The connection between synthesis and planning has been remarked in several works (see e.g., (De Giacomo and Vardi 2015; D’Ippolito, Rodríguez, and Sardiña 2018; Camacho et al. 2018a; 2018b; 2018c)). We also showed how to reduce constrained LTL synthesis to (infinite) LTL synthesis, which provides a worst-case optimal means of solving constrained LTL synthesis problems, in the general case, using (infinite) LTL synthesis tools. In the case where our constraint is comprised of a conjunction of safe and co-safe formulae, we showed that the constrained LTL synthesis problem can be reduced to DBA games and the winning strategy determined from the winning region. What makes our approach interesting is that the DBA is constructed via manipulation of DFAs, much easier to handle in practice than infinite-word automata.

We next turned our attention to the problem of augmenting constrained LTL synthesis with quality measures. We were motivated by practical concerns surrounding the ability to differentiate and synthesize high-quality strategies in settings where we may have a collection of mutually unrealizable objective formulae and alternative strategies of differing quality. Our work builds on results for the infinite case, e.g., (Almagor, Boker, and Kupferman 2016; Almagor et al. 2017; Kupferman 2016) with and without environment assumptions. We adopted LTL synthesis as our language for specifying quality measures. While the syntax of LTL synthesis is utilitarian, many more compelling preference
languages are reducible to this core language. We defined two different notions of optimal strategies – bgv-optimal and strongly bgv-optimal. The former adapts a similar definition in (Almagor, Boker, and Kupferman 2016) and the latter originates with us. We focused again on assumptions that can be expressed as conjunctions of safe and co-safe formulae and provided algorithms to compute bgv- and strongly bgv-optimal strategies with optimal (2EXP) complexity.

Proper handling of environment assumptions and quality measures, together with the design of efficient algorithms for such richer specifications, is essential to putting LTL synthesis into practice. The present paper makes several important advances in this direction and also suggests a number of interesting topics for future work including: the study of other types of assumptions in the finite-trace setting (e.g. rational synthesis), the exploitation of more compelling KR languages for specifying preferences, and the exploration of further ways of comparing and ranking strategies (perhaps incorporating notions of cost or trace length).

**Relation to Conference Version**

This paper appears in the Proceedings of the 16th International Conference on Knowledge Representation and Reasoning (KR 2018) without the appendix proofs. The body of this paper is the same as the KR 2018 paper, except that a minor typographic error in the translation of LTL into LTL₁ (the definition of τ(ϕ₁ U ϕ₂) on page 6), which originally appeared in (De Giacomo and Vardi 2013) and was repeated in the KR 2018 paper, has been corrected.

**Acknowledgements**

This work was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the ANR project GoAsQ (ANR-15-CE23-0022).

**References**


Theorem 4. When $\alpha$ is a safe formula, the constrained LTL$_{\omega}$ specification $S = \langle X, Y, \alpha, \varphi \rangle$ is realizable iff the LTL$_{\omega}$ specification $S' = \langle X, Y, \alpha' \rightarrow \varphi \rangle$ is realizable, where $\alpha'$ is obtained from $\alpha$ by replacing every occurrence of $\Box \varphi$ by $\Diamond \varphi$.

Proof. Let $\sigma'$ be a winning strategy for $S' = \langle X, Y, \alpha' \rightarrow \varphi \rangle$, with $\sigma$ a safe formula. We define a strategy $\sigma$ for $S = \langle X, Y, \alpha, \varphi \rangle$ by setting $\sigma(X_1 \cdots X_n) = \sigma'$.

- $\sigma'(X_1 \cdots X_n) \setminus \{\text{end}\}$, when end $\in \sigma'(X_1 \cdots X_n)$ and $(X_1 \cup \sigma(X_1)) \cdots (X_n \cup \sigma(X_1 \cdots X_n)) \not\equiv \alpha'$;
- $\sigma'(X_1 \cdots X_n)$, otherwise.

To show that $\sigma$ is a winning strategy, take some $X \in (2^X)^\omega$. Then $n_{\sigma,X} < \infty$, and the finite trace $\pi' = \pi'[\sigma',X]$ is such that (i) $\pi' \models \neg \alpha'$, or (ii) $\pi' \models \varphi$. If (i) holds, then $n_{\sigma,X} = \infty$ (since we will remove end), and $\pi = \pi'[\sigma,X]$ contains $\alpha'$ as a prefix. It can then use the LTL$_{\omega}$ equivalence $\neg \alpha' \equiv \Box \neg \varphi$ and the fact that $\sigma$ is safe to derive $\pi \models \neg \alpha$. If (ii) holds (and it does not), then it follows from the definition of $\sigma$ that $n_{\sigma,X} = n_{\sigma,X}$ and $\pi = \pi'[\sigma,X] = \pi'[\sigma',X]$, so $\sigma \models \varphi$.

For the other direction, let $\sigma$ be a winning strategy for $S$. Define a strategy $\sigma'$ for $S'$ by setting $\sigma'(X_1 \cdots X_n)$ equal to

- $\sigma(X_1 \cdots X_n) \cup \{\text{end}\}$, if $(X_1 \cup \sigma(X_1)) \cdots (X_n \cup \sigma(X_1 \cdots X_n))$ is a bad prefix for $\sigma$, and end $\not\in \sigma'(X_1 \cdots X_n)$ for $k < n$;
- $\sigma(X_1 \cdots X_n)$, if end $\in \sigma'(X_1 \cdots X_n)$ for some $k < n$;
- $\sigma(X_1 \cdots X_n) = \sigma(X_1 \cdots X_n)$, otherwise.

Basically, $\sigma'$ mimics $\sigma$ and outputs end as soon as a bad prefix is reached or $\sigma$ terminates the execution due to satisfaction of $\varphi$, whichever situation occurs first. The construction of $\sigma'$ is such that executions always terminate – which is not necessarily true for $\sigma$ – and induce finite traces that satisfy the LTL$_{\omega}$ formula $\alpha' \rightarrow \varphi$.

To show that $\sigma'$ is a winning strategy for $S'$, take some $X \in (2^X)^\omega$. Consider the infinite induced trace $\pi = \pi'[\sigma,X]$, and let $k_X$ be the minimum $k$ such that $\pi(1) \cdots \pi(k)$ is a bad prefix for $\alpha$ ($k_X = \infty$ if $\pi$ has no bad prefix). We examine these cases separately:

(i) $n_{\sigma,X} < \infty$ and $k_X \leq n_{\sigma,X}$, (ii) $n_{\sigma,X} < \infty$ and $k_X > n_{\sigma,X}$, and (iii) $n_{\sigma,X} = \infty$.

If (i) holds, then $n_{\sigma,X} = k_X$, and $\pi' = \pi'[\sigma',X] = \pi(1) \cdots \pi(k_X)$. From the fact that $\pi(1) \cdots \pi(k_X)$ is a bad prefix for the safe formula $\alpha$ and the LTL$_{\omega}$ equivalence $\neg \alpha' \equiv \Box \neg \varphi$, we obtain $\pi \models \neg \alpha'$.

If (ii) holds, then it follows from the definition of $\sigma'$ that $n_{\sigma',X} = n_{\sigma,X} + \pi'[\sigma',X] = \pi(1) \cdots \pi(n_{\sigma,X})$. Since $\sigma'$ is a winning strategy for $S$ with $n_{\sigma,X} < \infty$, we must have $\pi(1) \cdots \pi(n_{\sigma,X}) \models \varphi$, hence $\pi'[\sigma',X] \models \varphi$.

If (iii) holds, then it must be the case that $\pi \not\models \alpha$ (because $\varphi$ is a winning strategy). As $\sigma$ is a safe formula, $\pi$ must contain a bad prefix for $\alpha$, so $k_X < \infty$. Using a similar argument as in case (i), we can show that $\pi \models \neg \alpha'$.

Theorem 5. The constrained LTL$_{\omega}$ specification $S = \langle X, Y, \alpha, \varphi \rangle$ is realizable iff LTL$_{\omega}$ specification $S'' = \langle X, Y \cup \{\text{alive}\} \cup \{\text{end}\}, \varphi, \sigma \rangle$ is realizable. Moreover, for every winning strategy $\sigma$ for $S''$, the strategy $\sigma'$ defined by $\sigma'(X_1 \cdots X_n) := \sigma(X_1 \cdots X_n) \setminus \{\text{alive}\}$ is a winning strategy for $S$.

Proof. For the first direction, suppose that $S = \langle X, Y, \alpha, \varphi \rangle$ is realizable, and let $\sigma$ be a winning strategy for $S$. By definition, for every $X \in (2^X)^\omega$, the trace $\pi[\sigma,X]$ satisfies one of the following: (i) it has a finite prefix of length $n_{\sigma,X} < \infty$ that satisfies $\varphi$, or (ii) $n_{\sigma,X} = \infty$ and $\pi[\sigma,X] \not\models \alpha$. We define a strategy $\sigma''$ for $S'' = \langle X, Y \cup \{\text{alive}\}, \varphi, \sigma \rangle$ as follows:

- $\sigma''(X_1 \cdots X_n) = \sigma(X_1 \cdots X_n) \cup \{\text{alive}\}$, if there is no $k < n$ such that end $\in \sigma(X_1 \cdots X_n)$;
- $\sigma''(X_1 \cdots X_n) = \sigma(X_1 \cdots X_n)$, otherwise.

We claim that $\sigma''$ is a winning strategy for $S''$. Take some $X = X_1X_2 \cdots$, let $\sigma = \sigma[\sigma,X]$ and $\pi'' = \pi[\sigma'',X]$. We first show that $\pi'' \models \varphi_{end}$. First note that if $\sigma$ does not $\models \varphi_{end}$, then end $\in \pi''(i)$, which means end $\not\models \varphi(i + 1)$. We have thus shown that $\pi'' \models \varphi_{end}$. Next suppose that alive $\not\models \varphi(i + 1)$. This can only occur if end $\in \pi''(i)$, which implies that end $\not\models \varphi(i)$. We have thus shown that $\pi'' \models \varphi_{end}$.

For the other direction, suppose $S''$ is realizable, and let $\sigma''$ be a winning strategy for $S$. We define $\sigma$ as follows: $\sigma(X_1 \cdots X_n) = \sigma''(X_1 \cdots X_n) \setminus \{\text{alive}\}$. Our aim is to show that $\sigma$ is a winning strategy for $S$. Consider some $X = X_1X_2 \cdots$, and let $\pi'' = \pi[\sigma'',X]$ and $\pi = \pi[\sigma,X]$. Then $\pi'' = \pi[\sigma'',X]$ and $\pi = \pi[\sigma,X]$ contains at most one occurrence for end, and the same holds for $\sigma$. It follows that $\sigma$ is a valid strategy. Next suppose that $\sigma$ is a strategy. Consider the case where end does not occur in $\pi''$. We know that $\pi'' \not\models \varphi_{end}$. Note that we cannot have $\pi'' \not\models \varphi_{end}$ since it implies, when combined with $\pi'' \models \varphi_{end}$, that alive $\not\models \varphi(i + 1)$, which follows that $\pi'' \not\models \varphi_{end}$ or equivalently, $\pi'' \not\models \varphi_{end}$ does not contain end. From this, we can derive that $\pi \not\models \alpha$ and $\sigma$ does not contain $\text{end}$. We have thus established that $\sigma$ is a winning strategy.

Theorem 7. The DBA $A_{\omega}^{\varphi,\alpha}$ accepts infinite traces $\pi$ such that either: (i) $\pi(1) \cdots \pi(n) \models \varphi$ and end occurs only in $\pi(n)$,
that the finite prefix
prefixes for
any state (enters a state (the set of traces that satisfy one of the conditions (i) and (ii)). The construction of $A^\omega_{\text{free}}$ is polynomial in the size of $A$, $A_\ell$, and $A_\ell$, which can be constructed in double-exponential time in $|\alpha_1| + |\alpha_2| + |\varphi|$. 

Theorem 9 Consider a constrained LTL specification $S = (X, Y, \alpha_1 \land \alpha_2, \varphi)$ where $\alpha_1$ (resp. $\alpha_2$) is a safe (resp. co-safe) formula. Then:

- $S$ is realizable iff the DBA $G^\omega_{\varphi,\alpha_1}$ is winning;
- Every winning strategy for $G^\omega_{\varphi,\alpha_1}$ is a winning strategy for $S$, and vice-versa;
- Deciding whether $G^\omega_{\varphi,\alpha_1}$ is winning, and constructing a winning strategy when one exists, can be done in 2EXP.

Theorem 10 A bgv-optimal strategy can be constructed in double-exponential time.

Proof. We first argue that the described construction yields an bgv-optimal strategy. We know from (Almagor, Boker, and Kupferman 2016) that $A_{\text{free}}$ accepts finite traces $\pi$ such that $[\pi, \varphi] \geq b$. By reusing the arguments from the proof of Theorem 7, we can show that $A_{\text{free}}$ recognizes infinite traces such that either (i) $\alpha = \alpha_1 \land \alpha_2$ and $\ell$ does not occur, or (ii) $\ell$ occurs exactly once and the induced finite trace $\pi$ is such that $[\pi, \varphi] \geq b$. It follows that $\alpha$-strategy $\sigma$ with $\text{bgv}(\sigma) \geq b$ exists iff the DBA game $(X, Y \cup \{\ell\}, A_{\text{free}})$ is winning. We can use the techniques described in Section 5.2 to decide whether such a DBA is winning. By considering the values in $V(\varphi)$ in descending order, we can determine the maximal $b^*$ for which an $\alpha$-strategy $\sigma$ with $\text{bgv}(\sigma) \geq b^*$ exists. We can then compute such a strategy, for the identified best value $b^*$, by constructing a winning strategy for the DBA game $(X, Y \cup \{\ell\}, A_{\text{free}})$, again using the techniques from Section 5.2.

For each value $b$, we can construct the DFA $A_{b'}$ in double-exponential time, and the DFAs for safe and co-safe constraints can also be constructed in double-exponential
time. It follows that $\mathcal{A}_{y'}$, can also be constructed in double-

exponential time. Determining whether the DBA game $⟨X, Y \cup \{\text{end}\}, \mathcal{A}_{y_0}⟩$ is winning, and constructing a win-

ning strategy for the game, is also in double-exponential time. Finally, we note that all of the preceding double-

exponential time operations are performed at most once per value $b \in V(φ)$, so the overall procedure runs in double-

exponential time.

The next two lemmas will be used to prove Theorem 11. In what follows, it will be convenient to slightly abuse no-

tation and use $d(q, π)$, with $π$ a finite trace, to indicate the automata state resulting from reading $π$ starting from state $q$ (and similarly for the output function $ω$ of transducers on a finite string $X_1 \ldots X_n$).

Lemma 1. For every $π = (X_1 \cup Y_1) \ldots (X_n \cup Y_n) \in 2^{X \cup Y}$, $X_{n+1} \in 2^X$, and $b \in [0, 1]$, the following are equivalent:

1. there exists an $α$-strategy $σ$ such that $π \in \text{ptrac}(σ)$ and $bvg_{π}X_{n+1}(σ, b) ≥ b$
2. $δ_b(q_{n+1}^b, π(X_n \cup Y_n)) \in \text{Win}(G_b)$ for some $Y_{n+1} \in 2^{Y \cup \{\text{end}\}}$.

Proof. ($⇒$) Suppose that $σ$ is an $α$-strategy such that $π \in \text{traces}(σ)$ and $bvg_{π}X_{n+1}(σ, b) ≥ b$. Set $Y_{n+1} = σ(X_1 \ldots X_nX_{n+1})$, and let $q^* = δ_b(q_{n+1}^b, π(X_n \cup Y_n))$. Consider the DBA $\mathcal{A}_{y_0} ≡ (2^{X \cup Y \cup \{\text{end}\}}, Q_0, δ_0, ω, F_b)$, which is the same as $\mathcal{A}_{y_0}$ but with $q^*$ for the initial state. Define a strategy $σ'$ for the DBA game $G_b = (X, Y \cup \{\text{end}\}, \mathcal{A}_{y_0})$ as follows:

$σ'(X_1 \ldots X_nX_{n+1}) = σ(X_1 \ldots X_nX_{n+1}X_{n+1})$

We claim that $σ'$ is a winning strategy for $G_b$. To see why, take any $X \in (2^Y)^*$, and let $π_σ = σ(\langle 1, X \rangle)$. We need to show that $π_σ$ is accepted by $\mathcal{A}_{y_0}$. Let us consider $π_σ = π(σ(X_1 \ldots X_nX_{n+1}))$, and let $δ_0S_1S_2 \ldots$ be the infinite run of $\mathcal{A}_{y_0}$ on $π_σ$. Note that since $π ∈ \text{traces}(σ)$, we know that $π ∈ π_σ$, hence either (i) $π_σ$ does not contain end and $π ≠ α$, or (ii) $π_σ$ contains exactly one occurrence of end (at some position $k \leq n + 1$, since $π$ does not contain end) and the induced finite trace $π_σ^k = π(σ(X_1 \ldots X_nX_{n+1}))$ is such that $[π_N, φ] ≥ b$. It follows that $π_σ$ is accepted by $\mathcal{A}_{y_0}$, and thus the infinite run $δ_0S_1S_2 \ldots$ of $\mathcal{A}_{y_0}$ on $π_σ$ contains infinitely many $s_j \in F_b$. We then observe that $s^*_j = q^*$. Since $\mathcal{A}_{y_0}$ has the same transitions as $\mathcal{A}_{y_0}$, it follows that $s_{n+1} = q^*$. As $\mathcal{A}_{y_0}$ has the same transitions as $\mathcal{A}_{y_0}$, it must also contain infinitely many $s_j \in F_b$, so it is an accepting run. We have thus shown that $σ'$ is a winning strategy for $G_b$.

As the game $G_b$ is winning, we must have $q^* \in \text{Win}(G_b)$. We then remark that since $G_b$ and $G_b$ only differ in their initial states, the two games have precisely the same win-

ning regions. We thus obtain $q^* = δ_b(q_{n+1}^b, π(X_n \cup Y_n)) \in \text{Win}(G_b)$.

($⇐$) Suppose that $q^* = δ_b(q_{n+1}^b, π(X_n \cup Y_n)) \in \text{Win}(G_b)$. By following the strategy of the transducer $T_{G_b}$ from this point on, we are guaranteed to produce a trace that is ac-

cepted by the automaton $\mathcal{A}_{y_0}$. More precisely, consider the strategy $σ$ defined as follows:

- $σ(X'_1 \ldots X'_k) = Y_k$, if $X'_1 \ldots X'_k = X_1 \ldots X_k (1 ≤ k ≤ n + 1)$
- $σ(X'_1 \ldots X'_nX_{n+1} = ω_b(δ_0(q^*, X'_1 \ldots X'_nX_{n+1})), X'_{n+1})$, if $X'_1 \ldots X'_nX_{n+1} = X_1 \ldots X_{n+1}$
- $σ(X'_1) = \text{end}$, if $X'_1 ≠ X_1$
- $σ(X'_1X'_2 \ldots X'_k) = \text{end}$, if $X'_1 = X_1$ for $1 ≤ k < n + 1$
- $σ(X'_1X'_2 \ldots X'_k) = 0$, in all other cases

where $δ^*_b$ and $ω_b$ are respectively the transition and output functions of the transducer $T_{G_b}$. The first bullet concerns prefixes of $X_1 \ldots X_{n+1}$ and ensures that $(X_1 \cup Y_1) \ldots (X_n \cup Y_n) ∈ \text{traces}(σ)$). The second bullet states that once $X_1 \ldots X_{n+1}$ has been read, we start following the transducer $T_{G_b}$. The remaining points ensure that all infinite traces $π \in \text{traces}(σ)$ such that $π(X_n \cup Y_n) \notin π'$ contain a single occurrence of end.

We claim that $σ$ is an $α$-strategy such that $(X_1 \cup Y_1) \ldots (X_n \cup Y_n) ∈ \text{traces}(σ)$ and $bvg_{π}X_{n+1}(σ, b) ≥ b$. As noted above, the first bullet of the definition ensures that $(X_1 \cup Y_1) \ldots (X_n \cup Y_n) ∈ \text{traces}(σ)$. The last three bullets make sure that every $π' ∈ \text{traces}(σ)$ with $(X_1 \cup Y_1) \ldots (X_n \cup Y_n) \notin π'$ contains exactly one occurrence of end. It remains to consider the infinite traces that begin with $(X_1 \cup Y_1) \ldots (X_n \cup Y_n)$. Consider some such trace $π' = (X_1 \cup Y_1)(X_2 \cup Y_2) \ldots$ ∈ traces(σ), and let $s_{01}s_2 \ldots$ be the infinite run of $\mathcal{A}_{y_0}$ on $π'$. Because $(X_1 \cup Y_1) \ldots (X_n \cup Y_n) \notin π'$, we know that $s_{n+1} = q^*$. The latter can be combined with the second bullet to show that for every $i ≥ n + 1$, $Y_{n+1} = \omega_b(s_i, X_{n+1})$ and $s_{n+1} = δ_0(s_i, X_{n+1})$. As $s_{n+1} = q^* \in \text{Win}(G_b)$, and $ω_b$ is defined so as to always remain within $\text{Win}(G_b)$, we have $s_i \in \text{Win}(G_b)$ for every $i ≥ n + 1$. Furthermore, since $ω_b$ always reduces the distance to $R(\mathcal{A})$, there are infinitely many $i ≥ n + 1$ such that $s_i ∈ F_b$. This establishes that $s_0s_1s_2 \ldots$ is an accepting run, and thus $π'$ must either be such that $π' \notin α$, or it contains a single occurrence of end such that the finite induced trace $π''$ is such that $[π'', φ] ≥ b$. We have thus shown that every infinite trace produced by $σ$ either violates $α$ or contains end, i.e. $σ$ is an $α$-strategy. Moreover, for every $π'' ∈ \text{traces}(σ)$ with $(X_1 \cup Y_1) \ldots (X_n \cup Y_n) \notin π''$, we have $[π'', φ] ≥ b$, as desired.

In the following lemma, we use $σ_π$ to denote the strategy implemented by the transducer $T_{G_b}$. To simplify the formul-

ation, we extend the notation $bvg_{π}(σ)$ to all finite traces that can be produced by strategy $σ$ (recall that Definition 2 only defines this notation for compatible traces that do not contain end). Formally, given a trace $π = (X_1 \cup Y_1) \ldots (X_n \cup Y_n)$ that is a prefix of some infinite trace in $π[σ, X]$ and con-

tains end at position $k ≤ n$, we set $bvg_{π}(σ) = [π', φ]$, where $π' = (X_1 \cup Y_1) \ldots (X_k \cup Y_k \cup \text{end})$.

Lemma 2. For every $X_1 \ldots X_n \in (2^Y)^*$, $X_{n+1} \subseteq X$, and $v \in [0, 1]$, if $π = (X_1 \cup \omega_b(q_0^b, \text{str}) \ldots X_n) \cup \omega_b(q_0^b, X_1 \ldots X_n)$, $Q_1, Q_2, \ldots, Q_m = δ_0b(\text{str}^b, X_1 \ldots X_n)$, and $u$ is such that
\( \omega_{\#}(q_1, q_2, \ldots, q_m, X_{n+1}) \) was set equal to \( \omega_0(q_n, X_{n+1}) \), then \( \text{bgv}_{\varepsilon, X_{n+1}}(\sigma_{\text{str}}) \geq u \).

**Proof.** Fix \( X_1, \ldots, X_n \in (2^X)^s \) and \( X_{n+1} \subseteq X \). Consider some \( X \in (2^X)^s \) such that \( X = X_1 \ldots X_n X_{n+1} X_{n+2} \ldots \).

Let \( \pi' = \pi[\varepsilon_{\#}, X] \) = \((X_1 \cup Y_1)(X_2 \cup Y_2) \ldots \) and for all \( i \geq 0 \) let \((q_1', \ldots, q^i_{n+1}) = \delta_{\varepsilon_{\#}}(q_0^i, X_1 \ldots X_{n+1}) \).

Define a function \( \zeta : \mathbb{N} \to [b_1, \ldots, b_m] \) by letting \( \zeta(i) \) be the unique value \( b \) in \([b_1, \ldots, b_m]\) such that \( \omega_{\#}(q_1', \ldots, q^i_{n+1}), X_1 \ldots X_n, X_{n+1} \) was set equal to \( \omega_0(q^i_{n+1}, X_{n+1}) \).

Basically, \( \zeta(i) = b \) means that we used transducer \( T_{\varepsilon_{\#}} \) to generate \( Y_{n+1} \) after reading \( X_{n+1} \) from state \((q_1', \ldots, q^i_{n+1}) \).

Note that the value \( u \) from the lemma statement is equal to \( \zeta(n) \).

**Claim:** For all \( i \geq 0 \), \( \zeta(i+1) \geq \zeta(i) \).

**Proof of claim:** Suppose that \( \zeta(i) = b \), which means that \( b = \max\{v \in B \mid \exists Y \delta_i(q^i_{n+1}, X_{n+1}) \in \text{Win}(G_i)\} \).

Then we have \( Y_{n+1} = \omega_0(q^i_{n+1}, X_{n+1}) \).

From the definition of \( \omega_0 \) and the fact that there exists some \( Y \) with \( \delta_i(q^i_{n+1}, X_{n+1}) \in \text{Win}(G_i) \), we know that \( \delta_i(q^i_{n+1}, X_{n+1}) \in \text{Win}(G_i) \).

It follows that there must exist \( Y' \) such that \( \delta_i(q^i_{n+1}, X_{n+1} \cup Y') \in \text{Win}(G_i) \), and hence \( \max\{v \in B \mid \exists Y' \delta_i(q^i_{n+1}, X_{n+1} \cup Y') \in \text{Win}(G_i)\} \geq b \).

Thus, \( \zeta(i+1) \geq b = \zeta(i) \). (**end proof of claim**)

Due to the preceding claim, and the finiteness of \( B \), there exists \( b \geq 0 \) such that \( \zeta(i) = \zeta(b) \) for all \( i \geq b \), and \( \zeta(i) < \zeta(b) \) for \( i < b \). Let \( b_X = \zeta(b) \). Observe that \( q_0^b, q_1^b, q_2^b \ldots \) is the run of \( \mathcal{A}_{b_X} \) on \( \pi' \) and that \( q_0^b \in \text{Win}(G_{b_X}) \).

From the definition of \( \omega_{b_X} \), we can infer that \( q_1^b \in \text{Win}(G_{b_X}) \) for all \( i > b \), and further that there are infinitely many \( q_1^b \in F_{b_X} \). This means that \( q_0^b, q_1^b, q_2^b \ldots \) is an accepting run of \( \mathcal{A}_{b_X} \) on \( \pi' \).

Thus, either \( \pi' \) does not contain \( \text{end} \) and violates \( \alpha \), or \( \pi' \) contains \( \text{end} \) and the induced finite trace \( \pi'[\sigma_{\text{str}}, X] \) has value at least \( b_X = \zeta(b) \geq \zeta(n) \).

We have shown that every trace \( \pi'' \in \text{traces}(\sigma_{\text{str}}) \) is such that \( \|\pi'', \varphi\| \geq \zeta(n) = u \), and hence that \( \text{bgv}_{\varepsilon, X_{n+1}}(\sigma_{\text{str}}) \geq u \).

We now proceed to the proof of Theorem 11.

**Theorem 11** \( \mathcal{T}^{\text{str}} \) implements a strongly bgv-optimal strategy and can be constructed in double-exponential time.

**Proof.** We first show that \( \mathcal{T}^{\text{str}} \) implements a strongly bgv-optimal strategy. Suppose for a contradiction that this is not the case. Then there exists a finite trace \( \pi = (X_1 \cup Y_1) \ldots (X_n \cup Y_n) \) in \( \text{traces}(\sigma) \) that does not contain \( \text{end} \), \( X \in X \), and an \( \alpha \)-strategy such that \( \pi \in \text{traces}(\sigma') \) and \( \text{bgv}_{\varepsilon, \lambda}(\sigma') > \text{bgv}_{\varepsilon, \lambda}(\sigma) \).

Suppose that \( \text{bgv}_{\varepsilon, \lambda}(\sigma) = b \) and \( \text{bgv}_{\varepsilon, \lambda}(\sigma') = b' \).

From Lemma 1, we know that \( \delta_{\varepsilon}(q_0^i, \pi(X_n \cup Y_{n+1})) \in \text{Win}(G_{b_X}) \) for some \( Y_{n+1} \in \mathcal{V}(\text{end}) \).

It follows that \( \omega_{\#}(\delta_{\varepsilon}(q_0^i, X_1 \ldots X_n), X_{n+1}) = \omega_0(q^i_{n+1}, X_{n+1}) \) for some \( v \geq b' \).

From Lemma 2, \( \text{bgv}_{\varepsilon, X_{n+1}}(\sigma_{\text{str}}) \geq v \geq b' \), a contradiction.

We know from Section 7.2 and Theorem 10 that for each \( v \in V(\varphi) \), the DBA \( \mathcal{A}^{\text{str}}_{v} \) is winning region \( \text{Win}(G_{v}) \), and transducer \( T_{\varepsilon} \) can be constructed in double-exponential time.

As there are only single exponentially many values in \( V \) (hence \( B \)), the transducer \( \mathcal{T}^{\text{str}} \) can also be constructed double-exponential time. \( \square \)