Synthesizing Good-Enough Strategies for LTL$_f$ Specifications

Yong Li[1] , **Andrea Turrini**[1,2] , **Moshe Y. Vardi**[3] and **Lijun Zhang**[1,2]

[1]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
[2]Institute of Intelligent Software, Guangzhou
[3]Department of Computer Science, Rice University
{liyong, turrini,zhanglj}@ios.ac.cn, vardi@cs.rice.edu

Abstract

We consider the problem of synthesizing good-enough (GE)-strategies for Linear Temporal Logic (LTL) over *finite traces* or LTL$_f$ for short. The problem of synthesizing GE-strategies for an LTL formula φ over *infinite traces* reduces to the problem of synthesizing winning strategies for the formula $(\exists\mathcal{O}\varphi) \implies \varphi$, where $\mathcal{O}$ is the set of propositions controlled by the system. We first prove that this reduction does *not* work for LTL$_f$ formulas. Then we show how to synthesize GE-strategies for LTL$_f$ formulas via the Good-Enough (GE)-synthesis of LTL formulas. Unfortunately, this requires to construct deterministic parity automata on *infinite words*, which is computationally expensive. We then show how to synthesize GE-strategies for LTL$_f$ formulas by a reduction to solving games played on deterministic Büchi automata, based on an easier construction of deterministic automata on *finite words*. We show empirically that our specialized synthesis algorithm for GE-strategies outperforms the algorithms going through GE-synthesis of LTL formulas by orders of magnitude.

1 Introduction

Reactive synthesis is the automated construction of a reactive system from a given specification φ [Church, 1957], typically written as a Linear Temporal Logic (LTL) formula over a set of input signals $\mathcal{I}$ and a set of outputs $\mathcal{O}$ [Pnueli, 1977]. Reactive synthesis can be used to solve a number of different problems in AI, in particular planning. For instance, several variants of conditional planning problems with fully observability can be reduced to LTL synthesis problems, see, e.g., [Aminof *et al.*, 2019; Camacho *et al.*, 2019]. The task of synthesis is to construct a system M such that for each infinite input sequence $\alpha \in (2^{\mathcal{I}})^\omega$, M is able to output, in a step-by-step fashion, an output sequence $M(\alpha)$ such that the combined input and output sequence satisfies φ, i.e., $\alpha \otimes M(\alpha) \models \varphi$ [Pnueli and Rosner, 1989]. The extracted system M is a *winning strategy*. The synthesis problem of winning strategies for an LTL formula φ has been proved to be 2EXPTIME-complete and it is usually reduced to solving a game between the environment controlling the input signals and the system controlling the output, on a deterministic Parity automaton (DPA) representing φ [Pnueli and Rosner, 1989].

The requirement for constructing a system producing a satisfying computation of φ for each input sequence α is, however, sometimes too strong and unrealistic: there may be input sequences making φ unsatisfiable, whatever the outputs are. Therefore, researchers have proposed [Almagor and Kupferman, 2020; Damm and Finkbeiner, 2011] to relax this requirement and focused on synthesizing a system that makes best effort to respond to the input signals from the environment, which is termed as the *good-enough* (GE) synthesis in [Almagor and Kupferman, 2020]. More precisely, for GE-synthesis, the synthesized system M is required to output a sequence $M(\alpha)$ such that $\alpha \otimes M(\alpha) \models \varphi$ whenever possible. That is, if there does not exist an output sequence β such that $\alpha \otimes \beta \models \varphi$, M can just respond with an arbitrary output sequence. We call such a system a *good-enough (GE)-strategy*. The synthesis problem of GE-strategies for LTL formulas has been reduced to the synthesis of winning strategies for the formula $(\exists\mathcal{O}\varphi) \implies \varphi$ [Almagor and Kupferman, 2020; Damm and Finkbeiner, 2011], where the assumption $\exists\mathcal{O}\varphi$ restricts us to the input sequences that can be combined with an output sequence such that the resultant sequences satisfy φ. More precisely, given a formula φ with free input/environment variables $\mathcal{I}$ and free output/system variables $\mathcal{O}$, $\exists\mathcal{O}\varphi$ is the formula where only input variables in φ are left free, while output variables are existentially quantified; $\exists\mathcal{O}\varphi$ stands for $\exists o_1 \cdots \exists o_k \varphi$, where $\mathcal{O} = \{o_1, \ldots, o_k\}$ and $k \geq 1$, and it has the same meaning as in standard first order logic. We refer to [Sistla *et al.*, 1987] for more details of LTL with $\exists$ (existential) and $\forall$ (universal) quantifiers.

In recent years, the version of LTL over finite traces [Baier and McIlraith, 2006; De Giacomo and Vardi, 2013], or LTL$_f$ for short, emerged as another popular logic for specifications, because many settings, such as planning [Baier and McIlraith, 2006], assume that an execution stops after the specification has been achieved; we refer to [De Giacomo and Vardi, 2013] for more applications of LTL$_f$ in AI. Later, synthesizing winning strategies for an LTL$_f$ formula φ was proved to be 2EXPTIME-complete and can be reduced to solving a game played on a deterministic finite automaton (DFA) accepting φ [De Giacomo and Vardi, 2015]. LTL$_f$ synthesis has since then found applications in specifying task plans in robotics [He *et al.*, 2017; Lahijanian *et al.*, 2015], business

processes [Pesic *et al.*, 2010], and more.

This work looks into the synthesis problem of GE-strategies for LTL_f formulas. Our contributions are threefold and summarized as follows. First, we show that the reduction idea in the LTL setting to the synthesis of winning strategies for the formula $(\exists \mathcal{O} \varphi) \implies \varphi$ does not work for LTL_f formulas. Second, we show that synthesizing GE-strategies for LTL_f formulas can be reduced to the synthesis for LTL formulas. In practice, however, solving LTL_f synthesis via a translation to LTL cannot compete with native approaches specialized for LTL_f formulas, as shown in [Zhu *et al.*, 2017; Wells *et al.*, 2020]. This is because in practice, DFAs are expressive enough to accept the language of an LTL_f formula and specialized algorithms for constructing automata on finite words are easier and more efficient than those for building automata on infinite words for an LTL formula [Zhu *et al.*, 2017; Wells *et al.*, 2020]. Moreover, LTL synthesis requires solving parity (DPA) games, while LTL_f synthesis only depends on solving DFA games; it is known that solving a DFA game is much easier than solving a DPA game between the environment and the system, since DFA games are solvable in polynomial time [Mazala, 2002], while whether DPA game is doable in polynomial time is still an open problem [Calude *et al.*, 2017]. Therefore, it is of importance to obtain a specialized algorithm for the synthesis of GE-strategies for LTL_f formulas to achieve better scalability. Third, and our main contribution, we propose to synthesize GE-strategies for LTL_f formulas by a reduction to solving a game played on deterministic Büchi automata, based on construction of DFAs. We also prove that the problem of synthesizing GE-strategies for LTL_f formulas is 2EXPTIME-complete. Moreover, we conduct a comprehensive empirical evaluation on benchmarks from synthesis competitions and literature. We show that our specific synthesis algorithm for GE-strategies outperforms the algorithm going through GE-synthesis of LTL formulas by orders of magnitude, regarding the runtime and the number of solved cases.

2 Preliminaries

2.1 Linear Temporal Logic over Finite Traces

We focus here on Linear Temporal Logic over finite traces (LTL_f) [Baier and McIlraith, 2006; De Giacomo and Vardi, 2013], which is a variant of LTL [Pnueli, 1977] with the same syntax but it is interpreted over finite instead of infinite traces. The syntax of an LTL_f formula over a finite set of propositions $\mathcal{P}$ is defined as $\varphi ::= a \in \mathcal{P} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \text{X}\varphi \mid \varphi\text{U}\varphi \mid \text{F}\varphi \mid \text{G}\varphi$. Here X (strong Next), U (Until), F (Finally/Eventually), and G (Globally/Always) are temporal operators interpreted over finite traces. Note that X is a strong next operator such that $\text{X}\varphi$ requires the tail of the finite trace to satisfy φ, while we use N to denote the weak next operator such that $\text{N}\varphi$ demands that if the tail of the finite trace is not empty, then it satisfies φ. Consequently, $\text{N}\varphi := \neg\text{X}\neg\varphi$. The negation of an LTL_f formula φ, i.e., $\neg\varphi$ is also an LTL_f formula. As usual, true and false represent a tautology and a falsum, respectively. We denote by $|\varphi|$ the length of φ, i.e., the number of temporal operators and connectives in φ. We refer interested readers to [Pnueli, 1977] and [De Giacomo and Vardi, 2013] for the semantics of LTL and LTL_f, respectively. The language of an LTL/LTL_f formula φ, denoted as $\mathcal{L}(\varphi)$, is the set of infinite/finite words over $2^{\mathcal{P}}$ that satisfy φ.

2.2 NFA, DFA, DBA, and DPA

A *nondeterministic finite automaton* (NFA) is a tuple $A = (\Sigma, S, \iota, \Delta, F)$, where S is a finite set of states, $\iota \in S$ is the initial state, $\Delta\colon S \times \Sigma \to 2^S$ is the nondeterministic transition function, and $F \subseteq S$ is the set of accepting states. A run of A on a word $u = u_0 u_1 \cdots u_n \in \Sigma^*$ is a sequence of states $\rho = s_0 \cdots s_{n+1} \in S^+$ such that $s_0 = \iota$ and $s_{i+1} \in \Delta(s_i, u_i)$ holds for all $i \in \{0, \ldots, n\}$. The run ρ is *accepting* if $s_{n+1} \in F$. A accepts a word u if there is an accepting run of A on u. The language $\mathcal{L}(A)$ of A is the set of all accepted words. An NFA D is said to be a *deterministic* finite automaton (DFA) if for each $s \in S$ and $a \in \Sigma$, $|\Delta(s, a)| \leq 1$, i.e., we have $\Delta\colon S \times \Sigma \to S$.

For an LTL_f formula φ one can construct, with a single-exponential blow-up, an NFA A over the alphabet $\Sigma = 2^{\mathcal{P}}$ from φ such that $\mathcal{L}(A) = \mathcal{L}(\varphi)$ [De Giacomo and Vardi, 2013]. An NFA A can be converted into an equivalent DFA D with an exponential blowup by subset construction (see, e.g., [Hopcroft *et al.*, 2007]). Consequently, an LTL_f formula φ can be converted to a DFA D whose number of states is at most doubly exponential in the size of φ.

A *deterministic Büchi automaton* (DBA) is a tuple $B = (\Sigma, S, \iota, \Delta, F)$ as in the definition of DFAs; the difference lies in the fact that a DBA only accepts infinite words. The run of B on $u = u_0 u_1 \cdots u_n \cdots \in \Sigma^\omega$ is the infinite sequence $\rho = s_0 \cdots s_{n+1} \cdots \in S^\omega$ of states such that $s_0 = \iota$ and $s_{i+1} = \Delta(s_i, u_i)$ holds for all $i \geq 0$. ρ is *accepting* if ρ visits some state in F *infinitely many* times. B accepts an infinite word u if the run of B on u is accepting. The language $\mathcal{L}(B)$ of B is the set of all accepted infinite words.

A *deterministic Parity automaton* (DPA) is a tuple $P = (\Sigma, S, \iota, \Delta, p)$, where S, ι and Δ are as in DBAs while $p\colon S \to \mathbb{N}$ is a function defining the acceptance condition. The run ρ of P over u is *accepting* if the minimal color induced by p occurring infinitely often in ρ is *even*. Note that one can construct a DPA P for every LTL formula φ such that $\mathcal{L}(P) = \mathcal{L}(\varphi)$, while there exists an LTL formula ψ such that no DBA accepts $\mathcal{L}(\psi)$ [Baier and Katoen, 2008].

2.3 Winning and Good-Enough Strategies

Let A and B be two finite sets and $\alpha = a_0, a_1, \cdots$ and $\beta = b_0, b_1, \cdots$ be two finite/infinite words over 2^A and 2^B, respectively. We denote by $\alpha \otimes \beta$ the combined word $(a_0 \cup b_0), (a_1 \cup b_1) \cdots$ over $2^{A \cup B}$ and by $\alpha \ominus \beta$ the reduced word $(a_0 \setminus b_0), (a_1 \setminus b_1) \cdots$ over $2^{A \setminus B}$, provided that α, β are of the same length. We denote by $\alpha[i]$ the element a_i of α and we use $\alpha[i..k]$ to denote the subword of α between a_i and a_k, included, when $i \leq k$ and the empty word ε when $i > k$; lastly, we denote by $\alpha[i..]$ the suffix of α starting from a_i.

In this work we consider LTL/LTL_f formulas over $\mathcal{P} = \mathcal{I} \cup \mathcal{O}$, where $\mathcal{I}$ and $\mathcal{O}$ are two disjoint sets of propositions/variables. The set of input variables $\mathcal{I}$ is controlled by the environment while the set of output variables $\mathcal{O}$ is controlled by the system. A *strategy* γ is a total function

$\gamma\colon (2^{\mathcal{I}})^+ \to 2^{\mathcal{O}}$. For every word $\alpha = I_0, I_1, \cdots \in (2^{\mathcal{I}})^\omega$ of interpretations over $\mathcal{I}$, the strategy γ induces the word $(I_0 \cup \gamma(I_0)), (I_1 \cup \gamma(I_0, I_1)), \ldots, (I_m \cup \gamma(I_0, \ldots, I_m)) \cdots \in (2^{\mathcal{P}})^\omega$ of interpretations over $\mathcal{P}$, which we call the *computation* induced by γ on α, denoted as $\alpha \otimes \gamma(\alpha)$.

Definition 1 (Winning strategy)**.** *Let γ be a strategy. (1) γ is a* winning strategy *for an LTL formula φ if for every infinite word $\alpha = I_0, I_1, \cdots \in (2^{\mathcal{I}})^\omega$ of interpretations over $\mathcal{I}$, the computation $\alpha \otimes \gamma(\alpha)$ satisfies φ. (2) γ is a* winning strategy *for an LTL$_f$ formula φ if for every infinite word $\alpha = I_0, I_1, \cdots \in (2^{\mathcal{I}})^\omega$ of interpretations over $\mathcal{I}$, there exists an integer $m \geq 0$ such that $\alpha[0..m] \otimes \gamma(\alpha)[0..m]$ satisfies φ.*

Intuitively, for an LTL$_f$ formula φ, the environment, controlling the input variables, provides an unbounded sequence of inputs α; a strategy γ, in order to be winning for φ, must generate an appropriate sequence of output variables $\gamma(\alpha)$ such that, once combined with α as $\alpha \otimes \gamma(\alpha)$, there is a finite prefix of $\alpha \otimes \gamma(\alpha)$ of length $m+1$ that satisfies φ.

We say that φ is *realizable* if there is a winning strategy for φ. The problem of LTL synthesis [Pnueli and Rosner, 1989] is to decide whether φ is realizable and *construct* a winning strategy if so. Intuitively, LTL synthesis can be regarded as a DPA game corresponding to φ [Pnueli and Rosner, 1989] played between an external environment and the desired system that take turns to assign values to input and output propositions, respectively. The game is won by the system if we can construct a winning strategy guaranteeing that the resultant infinite input-output sequence satisfies φ. Similarly, the problem of LTL$_f$ synthesis is to decide whether the LTL$_f$ formula φ is *realizable* and to construct a winning strategy if so. Different from that of LTL formulas, LTL$_f$ synthesis for φ can be reduced to solving a game between the system and the environment played on a DFA accepting φ [De Giacomo and Vardi, 2015], rather than on a DPA.

In practice, however, it is not always possible to construct a winning strategy for the system. For instance, let $\varphi = \mathrm{F}i \wedge \mathrm{F}o$ where i and o are input and output variables, respectively. φ requires the computation to make both i and o eventually hold, in no predefined order. There is no winning strategy since the environment can generate $\neg i$ all the time. As proposed in [Damm and Finkbeiner, 2011; Almagor and Kupferman, 2020], one can relax the goal for winning strategies and aim to obtain the *best* strategy we can get among all alternative strategies. We call such strategy a *good-enough (GE)-strategy* and extend this notion to LTL$_f$.

Let $\alpha \in (2^{\mathcal{I}})^\omega$: for an LTL formula φ, α is said to be *φ-hopeful* if there exists an infinite word $\beta \in (2^{\mathcal{O}})^\omega$ such that $\alpha \otimes \beta \models \varphi$; for an LTL$_f$ formula φ, we say that α is *φ-f-hopeful* if there exists an integer $m \geq 0$ and a finite word $\beta = O_0 \cdots O_m \in (2^{\mathcal{O}})^+$ such that $\alpha[0..m] \otimes \beta \models \varphi$.

Definition 2 (Good-enough (GE)-strategy)**.** *Let γ be a strategy. (1) γ is a* GE-strategy *for an LTL formula φ if for every φ-hopeful infinite input word $\alpha \in (2^{\mathcal{I}})^\omega$, we have that $\alpha \otimes \gamma(\alpha) \models \varphi$. (2) γ is a* GE-strategy *for an LTL$_f$ formula φ if for every φ-f-hopeful input word $\alpha \in (2^{\mathcal{I}})^\omega$, there exists an integer $m \geq 0$ such that $\alpha[0..m] \otimes \gamma(\alpha)[0..m] \models \varphi$.*

The problem of *good-enough* (GE) LTL synthesis [Almagor and Kupferman, 2020] is to decide whether there exists a GE-strategy γ for φ and construct γ if so. We say a strategy γ GE-realizes φ if γ is a GE-strategy for φ; we also say that φ is *GE-realizable* if there exists a GE-strategy for φ. Analogous notions extend to LTL$_f$ formulas as below.

Definition 3 (GE-synthesis for LTL$_f$ formulas)**.** *The* good-enough *(GE)-synthesis problem for LTL$_f$ formulas is to decide whether there exists a GE-strategy γ for a given LTL$_f$ formula φ and construct γ if so.*

Clearly, a winning strategy is a GE-strategy; a GE-strategy, however, is not necessarily a winning strategy since, e.g., $\varphi = \mathrm{F}i \wedge \mathrm{F}o$ is not realizable but has a GE-strategy that outputs o at least once. We have the following result about the synthesis of GE-strategies for LTL.

Theorem 1 ([Almagor and Kupferman, 2020])**.** *The problem of the GE-synthesis for an LTL formula φ can be reduced to finding a winning strategy γ for the formula $(\exists \mathcal{O}\varphi) \implies \varphi$. Moreover, we can construct a DPA accepting the language of the formula $(\exists \mathcal{O}\varphi) \implies \varphi$ with a doubly exponential blow-up with respect to $|\varphi|$.*

Thus we can reduce the GE-synthesis problem for the LTL formula φ to the game played on a DPA for $(\exists \mathcal{O}\varphi) \implies \varphi$. Intuitively, for each infinite word $\alpha \in (2^{\mathcal{I}})^\omega$, the computation $\alpha \otimes \gamma(\alpha)$ induced by the winning strategy γ for $(\exists \mathcal{O}\varphi) \implies \varphi$ satisfies either $\forall \mathcal{O} \neg \varphi$ or φ, i.e., either α is not φ-hopeful or $\alpha \otimes \gamma(\alpha) \models \varphi$ when α is φ-hopeful, respectively. Consequently, a winning strategy γ for the formula $(\exists \mathcal{O}\varphi) \implies \varphi$ is a GE-strategy for φ.

3 Good-Enough Synthesis for LTL$_f$ Formulas

In this work, we study the problem of good-enough synthesis for LTL$_f$ formulas, i.e., synthesizing a GE-strategy for a given LTL$_f$ formula φ over $\mathcal{P} = \mathcal{I} \cup \mathcal{O}$, whenever possible.

3.1 Reduction to $(\exists \mathcal{O}\varphi) \implies \varphi$ Does Not Work for LTL$_f$ Formulas

As aforementioned, the GE-synthesis problem for an LTL formula φ can be reduced to a game played on a DPA for the formula $(\exists \mathcal{O}\varphi) \implies \varphi$. One may wonder whether the GE-synthesis problem for an LTL$_f$ formula φ can analogously be reduced to a game played on a DFA for the formula $(\exists \mathcal{O}\varphi) \implies \varphi$. Unfortunately, this is not the case, since the finite trace semantics of LTL$_f$ can be exploited to synthesize a strategy γ for $(\exists \mathcal{O}\varphi) \implies \varphi$ that does not work for φ, as formalized by Theorem 2 below.

Theorem 2. *There is an LTL$_f$ formula φ such that a winning strategy for the formula $(\exists \mathcal{O}\varphi) \implies \varphi$ is not necessarily a GE-strategy for φ.*

For LTL formulas, the reduction to $(\exists \mathcal{O}\varphi) \implies \varphi$ works because LTL is interpreted over infinite words. Thus, the quantified LTL formula $\exists \mathcal{O}\varphi$ [Sistla *et al.*, 1987], as an assumption, naturally allows us to focus only on the φ-hopeful infinite input words, while for LTL$_f$ formulas, the quantified LTL$_f$ formula $\exists \mathcal{O}\varphi$ is interpreted over finite words over $2^{\mathcal{I}}$. In particular, the assumption $\exists \mathcal{O}\varphi$ restricts us to finite input words over $2^{\mathcal{I}}$ rather than the φ-f-hopeful words. The definition of GE-synthesis for LTL$_f$ formulas, however, is defined

with respect to infinite sequences over $2^{\mathcal{I}}$. Consequently, we cannot reduce the synthesis of a GE-strategy for the LTL$_f$ formula φ to the synthesis of the winning strategy for the quantified LTL$_f$ formula $(\exists \mathcal{O}\varphi) \implies \varphi$.

Example 1. *An LTL$_f$ formula justifying Theorem 2 is $\varphi = Fi \wedge Fo$, where i and o are the input and output variables, respectively. There are GE-strategies for φ, such as the one outputting o on seeing the input i or the one continuously outputting o. There are, however, also winning strategies for $\exists\mathcal{O}\varphi \implies \varphi$ that are not GE-strategies for φ, such as the one that outputs o/$\neg o$ if the first input is i/$\neg i$, respectively, and then only outputs $\neg o$.*

3.2 Good-Enough Synthesis as Reduction to LTL

As shown by [De Giacomo and Vardi, 2013], given an LTL$_f$ formula φ over $\mathcal{P}$, one can construct an LTL formula $t(\varphi)$ over propositions $\mathcal{P} \cup \{\mathsf{tail}\}$ in linear time, where $\mathsf{tail} \notin \mathcal{P}$ is a fresh variable, such that φ is satisfiable if and only if $t(\varphi)$ is satisfiable. That is, one can reduce the LTL$_f$ satisfiability of φ to the LTL satisfiability of $t(\varphi)$. The variable tail is used to indicate when a finite word in $\mathcal{L}(\varphi)$ terminates. The resulting LTL formula $t(\varphi)$ has the following language.

Lemma 1 ([De Giacomo and Vardi, 2013])**.** $\mathcal{L}(t(\varphi)) = \{\, \lambda \otimes \{\mathsf{tail}\}^* \mid \lambda \in \mathcal{L}(\varphi) \,\} \cdot (2^{\mathcal{P}})^{\omega}$.

Intuitively, given a finite word $u \in \mathcal{L}(\varphi)$, tail holds for the first $|u|$ positions and $\neg\mathsf{tail}$ holds forever afterwards.

For synthesis, the fresh proposition $\mathsf{tail} \notin \mathcal{P}$ is under the control of the system. It is further shown in [Zhu *et al.*, 2017] that this reduction also works for LTL$_f$ realizability and synthesis, i.e., φ is realizable if and only if $t(\varphi)$ is realizable; also, a winning strategy for $t(\varphi)$ yields a winning strategy for the LTL$_f$ formula φ. We now show that this reduction works also for the GE-synthesis for LTL$_f$ formulas.

Lemma 2. *Let φ be an LTL$_f$ formula and $\alpha \in (2^{\mathcal{I}})^{\omega}$. Then α is φ-f-hopeful if and only if α is $t(\varphi)$-hopeful.*

Lemma 3. *Let φ be an LTL$_f$ formula. Then (1) φ is GE-realizable iff $t(\varphi)$ is GE-realizable; and (2) a GE-strategy γ for $t(\varphi)$ is also a GE-strategy for φ.*

We present now the main result of this subsection, which is a direct consequence of Lemma 3.

Theorem 3. *The GE-synthesis problem for an LTL$_f$ formula φ with respect to $\langle \mathcal{I}, \mathcal{O} \rangle$ can be reduced to the GE-synthesis problem for the LTL formula $t(\varphi)$ with respect to $\langle \mathcal{I}, \mathcal{O} \cup \{\mathsf{tail}\} \rangle$.*

Thanks to Theorem 3, we can reduce the GE-synthesis problem for LTL$_f$ formulas to the GE-synthesis problem for LTL formulas. This reduction, however, is not computationally competitive when compared with our specialized synthesis algorithm for LTL$_f$ formulas, given below in Section 3.3, as experiments show (cf. Section 4).

3.3 Synthesizing GE-Strategies for LTL$_f$ Formulas via DBA Games

By Theorem 2, for LTL$_f$ formulas, the problem of GE-synthesis for φ cannot be reduced to the classical synthesis problem for $(\exists\mathcal{O}\varphi) \implies \varphi$. Nonetheless, due to Lemma 3, we can just synthesize a winning strategy γ for $\exists(\mathcal{O} \cup \{\mathsf{tail}\})t(\varphi) \implies t(\varphi)$, since γ is also a GE-strategy for the LTL$_f$ formula φ. The key issue with this approach is that one has to, directly or indirectly, construct a DPA for the formula $\exists(\mathcal{O} \cup \{\mathsf{tail}\})t(\varphi) \implies t(\varphi)$, which is generally believed to be the main bottleneck of the synthesis procedure [Kupferman, 2012].

Our key observation for the synthesis of a GE-strategy for an LTL$_f$ formula is that we can construct a DBA accepting the language of $\exists(\mathcal{O} \cup \{\mathsf{tail}\})t(\varphi) \implies t(\varphi)$ based on the construction of NFAs and DFAs from φ. This allows us to exploit algorithms, specialized for LTL$_f$, for constructing automata on finite words, which are easier and more efficient in practice than those building automata on infinite words [Zhu *et al.*, 2017; Wells *et al.*, 2020]. In the remainder of the section we show how this is achieved; we start by formalizing the language of $\exists(\mathcal{O} \cup \{\mathsf{tail}\})t(\varphi) \implies t(\varphi)$, following directly from Lemma 2 since $t(\varphi)$ is an LTL formula and $\neg(\exists(\mathcal{O} \cup \{\mathsf{tail}\})t(\varphi)) \equiv \forall(\mathcal{O} \cup \{\mathsf{tail}\})\neg t(\varphi)$ recognizes the *infinite* words over $\mathcal{I}$ that are not $t(\varphi)$-hopeful.

Lemma 4. *Let φ be an LTL$_f$ formula; then it holds that $\mathcal{L}(\exists(\mathcal{O} \cup \{\mathsf{tail}\})t(\varphi) \implies t(\varphi)) = \mathcal{L}(t(\varphi)) \cup \{\, \alpha \in (2^{\mathcal{I}})^{\omega} \mid \alpha$ is not φ-f-hopeful $\,\}$.*

In the following, we show that instead of constructing a DPA for $\mathcal{L}(t(\varphi)) \cup \{\, \alpha \in (2^{\mathcal{I}})^{\omega} \mid \alpha$ is not φ-f-hopeful $\,\}$, we can construct a DBA accepting it. Consequently, the problem of synthesizing a GE-strategy for φ is reduced to solving a game played on a DBA B rather than on a DPA, which is much easier to solve [Mazala, 2002].

In order to construct the desired DBA B, we first build a DBA B_1 for $\mathcal{L}(t(\varphi))$, which accounts for the situation where a GE-strategy must generate a computation satisfying φ on a given φ-f-hopeful input sequence. Then we build a DBA B_2 for $\{\, \alpha \in (2^{\mathcal{I}})^{\omega} \mid \alpha$ is not φ-f-hopeful $\,\}$. Finally, we take their union product B accepting exactly the computations for synthesizing GE-strategies.

Theorem 4. *Let γ be a winning strategy for the system that produces computations in $\mathcal{L}(B) = \mathcal{L}(B_1) \cup \mathcal{L}(B_2)$. Then γ is a GE-strategy for φ.*

3.4 DBA Construction

We show how to construct the DBAs B_1, B_2, and B in detail.

Construction of B_1

In order to construct the DBA B_1 accepting all computations generated by a GE-strategy on a given φ-f-hopeful infinite input sequence, which clearly satisfy φ, we first construct a DFA D accepting $\mathcal{L}(\varphi)$, with a doubly exponential blow-up [De Giacomo and Vardi, 2015]. Then we get the DBA B_1 from D so that it accepts the language $\{\, \lambda \otimes \{\mathsf{tail}\}^* \mid \lambda \in \mathcal{L}(\varphi) = \mathcal{L}(D) \,\} \cdot (2^{\mathcal{P}})^{\omega}$, where tail is a fresh variable not present in $\mathcal{P}$. Note that in the DBA game, the fresh variable tail is controlled by the system as an output variable.

Definition 4. *Let $D = (2^{\mathcal{P}}, S, \iota, \Delta, F)$ be the DFA for φ. Then the DBA B_1 is the tuple $(2^{\mathcal{P} \cup \{\mathsf{tail}\}}, S_1, \iota_1, \Delta_1, F_1)$, where $S_1 = S \cup \{\top\}$ with $\top \notin S$ being a fresh state, $\iota_1 = \iota$ is the initial state, $F_1 = \{\top\}$ is the set of accepting states, and Δ_1 is defined as follows: (1) if $\Delta(q, a) = q'$, then*

$\Delta_1(q, a \cup \{\mathsf{tail}\}) = q'$; *(2) if* $q \in F$, *then* $\Delta_1(q, a) = \top$ *for each* $a \in 2^{\mathcal{P}}$; *and (3)* $\Delta_1(\top, a) = \top$ *for each* $a \in 2^{\mathcal{P}}$.

Note that the transition function Δ_1 may be undefined for some state $q \in S_1$ and assignment $a \in (2^{\mathcal{P} \cup \{\mathsf{tail}\}})$. Intuitively, in Definition 4, we just use "tail" to mark when a finite word in $\mathcal{L}(\varphi)$ terminates. Since the number of states in D is at most doubly exponential in the length of φ [De Giacomo and Vardi, 2015], we directly obtain the following result.

Lemma 5. *Let* B_1 *be the DBA constructed in Definition 4. Then* $\mathcal{L}(B_1) = \{\, \lambda \otimes \{\mathsf{tail}\}^* \mid \lambda \in \mathcal{L}(\varphi) \,\} \cdot (2^{\mathcal{P}})^\omega$ *and* B_1 *has* $2^{2^{\mathcal{O}(|\varphi|)}}$ *states.*

Construction of B_2

We now construct the DBA B_2 that accepts all infinite sequences $\lambda \in (2^{\mathcal{I}})^\omega$ for which no strategy can generate a computation satisfying φ. That is, $\mathcal{L}(B_2) = \{\, \alpha \in (2^{\mathcal{I}})^\omega \mid \alpha$ is not φ-f-hopeful $\}$. To construct the DBA B_2, we take the following three steps.

Step 1. Construct an NFA N accepting $\mathcal{L}(\varphi)$ with a single exponential blow-up [De Giacomo and Vardi, 2015] and then make all accepting states of N sink accepting states with a self-loop transition labeled with $\mathsf{true} = 2^{\mathcal{P} \cup \{\mathsf{tail}\}}$. Thus we have that $\mathcal{L}(N) = \mathcal{L}(\varphi) \cdot \mathsf{true}^*$ and N has $2^{\mathcal{O}(|\varphi|)}$ states.

Step 2. Existentially quantify out all $\mathcal{O}$-variables on the transitions of N and let N' be the resultant NFA. Let Δ and Δ' be the transition functions of N and N', respectively. Then if $q' \in \Delta(q, a)$ for $a \in 2^{\mathcal{P}}$, then we have that $q' \in \Delta'(q, a \setminus \mathcal{O})$. We then treat the modified NFA N' as an NBA A. Intuitively, the NBA A accepts all input sequences $\alpha = I_0, I_1, \cdots \in (2^{\mathcal{I}})^\omega$ that are φ-f-hopeful. That is, we have that $\mathcal{L}(A) = \{\, \alpha \in (2^{\mathcal{I}})^\omega \mid \alpha$ is φ-f-hopeful $\}$. Then we just need to complement the NBA A at the next step since the desired DBA B_2 accepts the complementary language of A. Note that all accepting states of A are sink states with a self-loop on true.

Step 3. Determinize the NBA A with a standard subset construction [Hopcroft *et al.*, 2007] and set all states that do not contain a sink accepting state of A as accepting states, which yields the DBA B_2, which we assume without loss of generality to be complete. Since all accepting states of A are sink states, all states reachable from a nonaccepting state of B_2 will also be nonaccepting, which makes it possible to complement the NBA A with a simple subset construction and then to reverse the set of accepting states. Note that complementing general NBAs is much more complicated and expensive [Vardi, 2007; Yan, 2008] than a subset construction. Intuitively, B_2 accepts all infinite sequences $\alpha \in (2^{\mathcal{I}})^\omega$ that are not φ-f-hopeful, i.e., no strategy at all can generate a computation for α that satisfies φ.

Lemma 6. *Let* B_2 *be the DBA constructed above for* φ. *Then* $\mathcal{L}(B_2) = \{\, \alpha \in (2^{\mathcal{I}})^\omega \mid \alpha$ *is not* φ*-f-hopeful* $\}$ *and* B_2 *has* $2^{2^{\mathcal{O}(|\varphi|)}}$ *states.*

Construction of B

The DBA B is then the union product of B_1 and B_2 such that $\mathcal{L}(B) = \mathcal{L}(B_1) \cup \mathcal{L}(B_2)$. Since the accepting states in B_1 and B_2 are sink states, we can just compute the union product as it is done for DFAs [Hopcroft *et al.*, 2007]. Let $B_1 = (2^{\mathcal{P}} \cup \{\mathsf{tail}\}, S_1, \iota_1, \Delta_1, F_1)$ and $B_2 = (2^{\mathcal{I}}, S_2, \iota_2, \Delta_2, F_2)$. Then we have $B = (2^{\mathcal{P}} \cup \{\mathsf{tail}\}, S_1 \times S_2, (\iota_1, \iota_2), \Delta_1 \times \Delta_2, (S_1 \times F_2) \cup (F_1 \times S_2))$. We remark that this union product can be efficiently performed symbolically.

Strategy Extraction

To extract a winning strategy for the system on the DBA B, we use a DBA game solving algorithm, working in polynomial time with respect to the number of states of B [Mazala, 2002]. The algorithm determines whether from the initial state the system is sure to win the game. In this case, a winning strategy is given as a Mealy machine producing the appropriate system's output for the current state and input. See [Mazala, 2002] for more details about solving DBA games and extracting winning strategies.

3.5 Correctness and Complexity

Theorem 5. *Let* φ *be an LTL*$_f$ *formula and* B *the DBA constructed in Section 3.4. (1) There exists a winning strategy for the DBA game played on* B *if and only if* φ *is GE-realizable. (2) A winning strategy* γ *for the DBA game played on* B *is also a GE-strategy for* φ.

It is clear that the LTL$_f$ GE-realizability/GE-synthesis problem is in 2EXPTIME. Similarly to [Rosner, 1991, Section 3.2] in the LTL setting, one can also construct from a given 2EXPTIME Turing machine M an LTL$_f$ formula φ_M that is realizable if and only if M accepts the empty tape, where all input sequences are φ_M-f-hopeful. Hence checking GE-realizability of φ_M is equivalent to checking its realizability. Thus the 2EXPTIME-hardness result follows.

Theorem 6. *The synthesis problem of GE-strategies for LTL*$_f$ *specifications is 2EXPTIME-complete.*

4 Experimental Evaluation

4.1 Implementation

We have extended the LISA tool described in [Bansal *et al.*, 2020] with an implementation of the synthesis algorithm described in Section 3; we used MONA [Henriksen *et al.*, 1995] to construct DFAs and NFAs from an LTL$_f$ formula, as described in [Zhu *et al.*, 2019]. The DFAs and NFAs are first constructed explicitly, then encoded symbolically by means of well-known standard techniques (see, e.g., [Ferrara *et al.*, 2005; Bansal *et al.*, 2020]); this allows us to build their union product efficiently.

We implemented two versions of the construction for the automaton B_2 in LISA: one using NFAs as in Section 3 and one using only DFAs (without affecting the correctness of our algorithm). While this adds in theory an exponential blowup, in practice working with DFAs often yields better performance than with NFAs [Tabajara and Vardi, 2020]. The comparison between the two approaches given below in Section 4.3 confirms this, so we use DFAs by default.

4.2 Evaluation Setting

To evaluate the effectiveness of the strategy synthesis algorithm proposed in Section 3, we compared LISA with a modified version of BOSY [Faymonville *et al.*, 2017] and of LTL-

Outcome	LISA		BOSY		LTLSYNT	
	strong	weak	strong	weak	strong	weak
realizable	1840	1838	832	831	1123	1105
unrealizable	49	50	0	0	18	16
timeout	4	6	598	566	339	366
out of memory	9	8	70	66	4	3
other failures	9	9	411	448	427	421

Table 1: Summary of the outcomes of the experiments

SYNT, part of SPOT [Duret-Lutz *et al.*, 2016], so to allow them to support GE-synthesis.

Since BOSY and LTLSYNT do not support LTL_f natively, we used SPOT to convert each LTL_f formula φ as $t(\varphi)$. The modified BOSY first constructs a universal co-Büchi automaton accepting $(\exists(\mathcal{O} \cup \{\mathsf{tail}\})t(\varphi)) \implies t(\varphi)$ and then uses the built-in bounded synthesis techniques [Faymonville *et al.*, 2017] on this automaton to synthesize the GE-strategy. The modified LTLSYNT first constructs a DPA that accepts $(\exists(\mathcal{O} \cup \{\mathsf{tail}\})t(\varphi)) \implies t(\varphi)$ and then solves the DPA game with built-in solvers [Duret-Lutz *et al.*, 2016].

LTLSYNT got the second place in SYNTCOMP 2020[1]; we did not compare with the winner Strix [Meyer *et al.*, 2018] since its architecture does not allow us to add support for GE-synthesis with reasonable effort, as we did for the other tools.

As benchmarks, we collected all benchmarks available on the past SYNTCOMP competitions website[2], resulting in 294 benchmark files. We then used SYFCO[3] to translate them to the JSON format accepted by each tool. We considered also the 1617 benchmarks used in [Bansal *et al.*, 2020], already in the appropriate JSON format, for a total of 1911 executions. We considered all formulas in the benchmarks as LTL_f formulas and then generated two versions of the files according to the strong/weak semantics of the X next operator (cf. Section 2), which are encoded in the benchmark files by the operators `X[!]` and `X`, respectively.

We performed our experiments on a desktop PC equipped with a 3.4 GHz Intel i7-6700 processor and 8 GB of RAM, of which 5 GB were assigned to the experiments; the operating system was Ubuntu 18.04. We used BENCHEXEC[4] to trace the execution of the tools; we imposed a timeout of 10 minutes for each benchmark. We use the following conventions in the scatter plots presented in the remainder of this section: a mark placed on the horizontal/vertical dotted line at time 600 means that the corresponding tool went timeout; a mark on the dashed line, instead, stands for the tool went out of memory during the execution; a mark on the top/right border of the plot indicates that the tool failed for other motivations.

4.3 Experimental Results

Overview of the results. Table 1 summarizes the outcomes of the different tools on the two versions of the benchmarks. As we can see, LISA has been able to solve considerably more experiments than both BOSY and LTLSYNT; out of 1911 benchmarks, LISA failed only on less than 25; BOSY and LTLSYNT, instead, failed on more than 1000 and 750 experiments, respectively, of which roughly half caused by timeout. It is interesting to observe in Table 1 that BOSY has never returned unrealizable. This happens also for cases on which both LISA and LTLSYNT returned unrealizable; so it seems BOSY returns some wrong answer. Given these differences in the outcomes, we exclude BOSY from the remainder of our experimental evaluation.

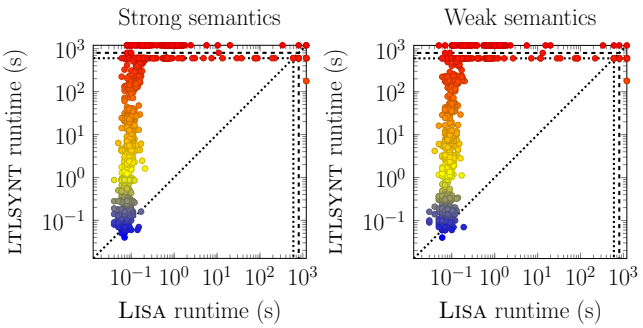

Figure 1: Comparison of LISA and LTLSYNT on the benchmarks

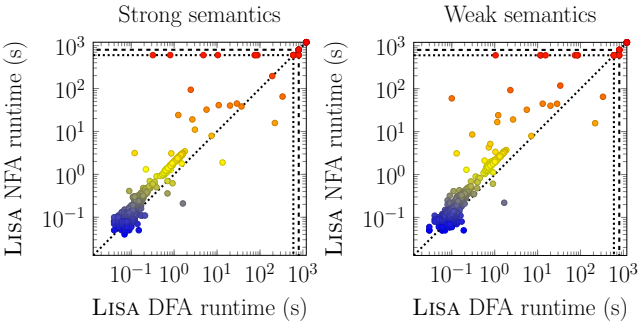

Figure 2: Comparison of LISA using DFA and NFA construction

Comparison between LISA and LTLSYNT. In Figure 1 we compare the execution running time of LISA and LTLSYNT; the left plot is relative to the strong X semantics while the right plot to the weak X semantics. As we can see, in both semantics LISA is almost always considerably much faster than LTLSYNT in either synthesizing a GE-strategy or finding that there is no GE-strategy, as indicated by the large majority of the points above the diagonal dotted line. There are few cases on which LISA crashed, corresponding to the points on the plot right border, as already mentioned in the summary of the experiments above.

Comparison of LISA using DFA and NFA construction. We also compared the running time of LISA when using a DFA instead of an NFA for the construction of the DBA B_2 (cf. Sections 3.4 and 4.2); it is shown in Figure 2. As we can see from the plots, even though constructing the NFA is usually faster than generating the DFA, the latter gives a better overall performance, as indicated by the majority of the points above the diagonal, in particular for executions requiring more than 0.25 seconds. This can be motivated by the blowup caused

[1] http://www.syntcomp.org/syntcomp-2020-results/
[2] https://syntcomp.react.uni-saarland.de/
[3] https://github.com/reactive-systems/syfco/
[4] https://github.com/sosy-lab/benchexec/

by the determinization step in the construction of the DBA B_2 (cf. Section 3.4), which can be mitigated when working with DFAs from the beginning.

5 Related Work

The GE-synthesis problem for LTL_f formulas can be seen as a synthesis problem for an LTL_f formula φ under the quantified LTL assumption $\exists\mathcal{O}t(\varphi)$, though without explicitly giving the assumption. There are works [Camacho *et al.*, 2018; Zhu *et al.*, 2020; Giacomo *et al.*, 2020] about the synthesis problem where assumptions are explicitly given as an LTL formula ψ; ψ is also used to restrict the synthesis problem to certain environment behaviours we care about: in [Zhu *et al.*, 2020], the assumption is restricted to simple fairness and stability, which is then reduced to solving a so-called fair and stable DFA game, respectively. The assumption is further extended in [Giacomo *et al.*, 2020] to the whole set of LTL formulas, which however relies on solving DPA games. In [Camacho *et al.*, 2018], the assumption is specified as a conjunction of a safe LTL formula ψ_s and a co-safe LTL formula ψ_c, i.e., the LTL formula $\psi_s \wedge \psi_c$.

Our work and the works in [Camacho *et al.*, 2018; Zhu *et al.*, 2020; Giacomo *et al.*, 2020] consider a similar problem (i.e., the synthesis of a strategy so to satisfy a given LTL_f formula given that the environment behaves nicely) in two different scenarios; the main difference is that the environment assumptions in their problems are given explicitly as LTL formulas, while our problem does not need to take an environment assumption as input. Even if we see the φ-f-hopeful sequences as an assumption, there is no trivial way to model them as an LTL formula from the given LTL_f formula φ, so to formalize the concept of φ-f-hopeful sequence.

Below we focus on a detailed comparison with [Camacho *et al.*, 2018], where the LTL_f synthesis problem for φ under the safe and co-safe assumption $\psi_s \wedge \psi_c$ has also been reduced to solving a DBA game. Thus, they can use one DFA for ψ_s, one DFA for ψ_c and one DFA for φ, and then compose them via automata product operation to obtain the DBA game. In contrast, our constructed NFA and DFA are both obtained from the input LTL_f formula φ, our only input. A synthesis problem for a kind of LTL_f formulas with quality measures, denoted by $\mathbf{LTL}_f[\mathcal{F}]$, has been considered in [Camacho *et al.*, 2018]: instead of a formula being either totally satisfied or totally violated by a computation, a value between 0 and 1 by the weight functions $f \in \mathcal{F}$ indicates its degree of satisfaction. Different strategies can then be compared according to the so called *best guaranteed value* (bgv), the minimum value among the values of the resultant computations induced by a strategy. A strongly bgv-optimal strategies for $\mathbf{LTL}_f[\mathcal{F}]$ is a strategy that desires to obtain the best worst-case value. We conjecture that strongly bgv-optimal strategies degenerate to our GE-strategies when the quality is set to one and the LTL assumption $\psi_s \wedge \psi_c$ is discarded; we leave as future work the confirmation of this conjecture. Note that the GE-synthesis problem we consider here, though possibly identified as a subclass of synthesis of strongly bgv-optimal strategies now, is still novel just like the interesting subclass called GR(1) [Bloem *et al.*, 2012] of full LTL synthesis, which allows for specialized and practical algorithms and draws a lot of researchers to work on it.

We remark that the 2EXPTIME-completeness established in [Camacho *et al.*, 2018] applies only to LTL_f synthesis with safe and co-safe LTL assumptions and no such result is claimed for synthesizing strongly bgv-optimal strategies from LTL_f formulas with explicit assumptions and quality measures. Our 2EXPTIME-completeness is for synthesizing GE-strategies only for LTL_f formulas. We note, however, that, if the conjecture above is correct, then our hardness result extends to the synthesis of strongly bgv-optimal strategies.

6 Conclusion and Future Work

To the best of our knowledge, this is the first time the problem of GE-synthesis for LTL_f formulas has been considered. The main contribution of this work is that we proposed to solve the GE-synthesis problem for LTL_f formulas by reducing it to a DBA game. Our specific GE-synthesis algorithm takes advantage of the construction of NFAs and DFAs, which is much easier than the direct or indirect construction of DPAs by going through LTL synthesis. The empirical evaluation on benchmarks from SYNTCOMP and literature confirmed the merits of our specific approach. We observe that currently the bottleneck of our approach is the size of the DBAs, which invites for further improvement as future work.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable suggestions and comments about this paper. Work supported in part by the Guangdong Science and Technology Department (Grant No. 2018B010107004), NSF grants IIS-1527668, CCF-1704883, IIS-1830549, an award from the Maryland Procurement Office and the National Natural Science Foundation of China (Grants Nos. 61532019, 61761136011, 61572024).

References

[Almagor and Kupferman, 2020] Shaull Almagor and Orna Kupferman. Good-enough synthesis. In *CAV*, pages 541–563, 2020.

[Aminof *et al.*, 2019] Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Planning under LTL environment specifications. In *ICAPS*, pages 31–39, 2019.

[Baier and Katoen, 2008] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[Baier and McIlraith, 2006] Jorge A. Baier and Sheila McIlraith. Planning with temporally extended goals using heuristic search. In *ICAPS*, pages 342–345, 2006.

[Bansal *et al.*, 2020] Suguman Bansal, Yong Li, Lucas M. Tabajara, and Moshe Y. Vardi. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *AAAI*, pages 9766–9774, 2020.

[Bloem *et al.*, 2012] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of

reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[Calude *et al.*, 2017] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *STOC*, pages 252–263, 2017.

[Camacho *et al.*, 2018] Alberto Camacho, Meghyn Bienvenu, and Sheila A. McIlraith. Finite LTL synthesis with environment assumptions and quality measures. In *KR*, pages 454–463. AAAI Press, 2018.

[Camacho *et al.*, 2019] Alberto Camacho, Meghyn Bienvenu, and Sheila A. McIlraith. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*, pages 58–67, 2019.

[Church, 1957] Alonzo Church. Applications of recursive arithmetic to the problem of circuit synthesis. *J. Symbol. Logic*, 28(4):289–290, 1957.

[Damm and Finkbeiner, 2011] Werner Damm and Bernd Finkbeiner. Does it pay to extend the perimeter of a world model? In *FM*, pages 12–26, 2011.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860, 2013.

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, pages 1558–1564, 2015.

[Duret-Lutz *et al.*, 2016] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 – a framework for LTL and ω-automata manipulation. In *ATVA*, pages 122–129, 2016.

[Faymonville *et al.*, 2017] Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. BoSy: An experimentation framework for bounded synthesis. In *CAV*, pages 325–332, 2017.

[Ferrara *et al.*, 2005] Andrea Ferrara, Guoqiang Pan, and Moshe Y. Vardi. Treewidth in verification: Local vs. global. In *LPAR*, pages 489–503, 2005.

[Giacomo *et al.*, 2020] Giuseppe De Giacomo, Antonio Di Stasio, Moshe Y. Vardi, and Shufang Zhu. Two-stage technique for LTLf synthesis under LTL assumptions. In *KR*, pages 304–314, 2020.

[He *et al.*, 2017] Keliang He, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. Reactive synthesis for finite tasks under resource constraints. In *IROS*, pages 5326–5332, 2017.

[Henriksen *et al.*, 1995] Jesper G. Henriksen, Jakob Jensen, Michael Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. Mona: Monadic second-order logic in practice. In *TACAS*, pages 89–110, 1995.

[Hopcroft *et al.*, 2007] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Addison-Wesley, 2007.

[Kupferman, 2012] Orna Kupferman. Recent challenges and ideas in temporal synthesis. In *SOFSEM*, pages 88–98, 2012.

[Lahijanian *et al.*, 2015] Morteza Lahijanian, Shaull Almagor, Dror Fried, Lydia E. Kavraki, and Moshe Y. Vardi. This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In *AAAI*, pages 3664–3671, 2015.

[Mazala, 2002] René Mazala. Infinite games. In *Automata, Logics, and Infinite Games*, pages 23–38. Springer, 2002.

[Meyer *et al.*, 2018] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *CAV*, pages 578–586, 2018.

[Pesic *et al.*, 2010] Maja Pesic, Dragan Bosnacki, and Wil M. P. van der Aalst. Enacting declarative languages using LTL: Avoiding errors and improving performance. In *SPIN*, pages 146–161, 2010.

[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.

[Rosner, 1991] Roni Rosner. *Modular synthesis of reactive systems*. The Weizmann Institute of Science (Israel), 1991. PhD thesis.

[Sistla *et al.*, 1987] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for büchi automata with appplications to temporal logic. *Theor. Comput. Sci.*, 49:217–237, 1987.

[Tabajara and Vardi, 2020] Lucas M. Tabajara and Moshe Y. Vardi. LTLf synthesis under partial observability: From theory to practice. In *GANDALF*, pages 1–17, 2020.

[Vardi, 2007] M. Y. Vardi. The Büchi complementation saga. In *STACS*, pages 12–22, 2007.

[Wells *et al.*, 2020] Andrew M. Wells, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. LTLf synthesis on probabilistic systems. In *GANDALF*, pages 166–181, 2020.

[Yan, 2008] Qiqi Yan. Lower bounds for complementation of ω-automata via the full automata technique. *Log. Methods Comput. Sci.*, 4(1:5), 2008.

[Zhu *et al.*, 2017] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTLf synthesis. In *IJCAI*, pages 1362–1369, 2017.

[Zhu *et al.*, 2019] Shufang Zhu, Geguang Pu, and Moshe Y. Vardi. First-order vs. second-order encodings for LTLf-to-automata translation. In *TAMC*, pages 684–705, 2019.

[Zhu *et al.*, 2020] Shufang Zhu, Giuseppe De Giacomo, Geguang Pu, and Moshe Y. Vardi. LTLf synthesis with fairness and stability assumptions. In *AAAI*, pages 3088–3095. AAAI Press, 2020.