Finite LTL Synthesis is EXPTIME-complete

Jorge A. Baier
Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile

Alberto Camacho
University of Toronto, Toronto, Ontario

Christian Muise
MIT CSAIL, Massachusetts, USA

Sheila A. McIlraith
University of Toronto, Toronto, Ontario

Abstract

LTL synthesis – the construction of a function to satisfy a logical specification formulated in Linear Temporal Logic – is a 2EXPTIME-complete problem with relevant applications in controller synthesis and a myriad of artificial intelligence applications. In this research note we consider De Giacomo and Vardi’s variant of the synthesis problem for LTL formulas interpreted over finite rather than infinite traces. Rather surprisingly, given the existing claims on complexity, we establish that LTL synthesis is EXPTIME-complete for the finite interpretation, and not 2EXPTIME-complete as previously reported. Our result coincides nicely with the planning perspective where non-deterministic planning with full observability is EXPTIME-complete and partial observability increases the complexity to 2EXPTIME-complete; a recent related result for LTL synthesis shows that in the finite case with partial observability, the problem is 2EXPTIME-complete.

1. Introduction

Church’s synthesis problem, a classical problem in computer science, calls for the automatic construction of a procedure which, given a logical specification \( \varphi(I,O) \) between input strings \( I \) and output strings \( O \), determines whether there exists an operator \( F \) that implements the specification such that \( \varphi(I,F(I)) \) holds for all inputs \( I \). It was first posed

Email addresses: jabaier@ing.puc.cl (Jorge A. Baier), acamacho@cs.toronto.edu (Alberto Camacho), cjmuise@mit.edu (Christian Muise), sheila@cs.toronto.edu (Sheila A. McIlraith)

Under review for publication
by Church in 1957 in the context of synthesizing digital circuits from a logical specification [1, 2] and is considered one of the most challenging problems in reactive systems [3]. Büchi and Landweber first solved the problem in 1969 [4]. Soon after, Rabin proposed a solution exploiting automata on infinite trees [5]. In the years that have followed, two common approaches to solving the problem have emerged: reducing the problem to the emptiness problem of tree automata, and characterizing the problem in terms of a two-player game.

In 1989, Pnueli and Rosner examined the problem of reactive synthesis using Linear Temporal Logic (LTL) [6] as the specification language, viewing the problem as a two-player game, and showing that this problem was 2EXPTIME-complete [7]. Pnueli and Rosner's approach required two translations, first from the LTL formula, \( \varphi \), to a Büchi automaton, \( B_\varphi \), and then determinization of \( B_\varphi \) to a deterministic Rabin automaton. This double translation gives the procedure its worst case double exponential complexity in the size of \( \varphi \). The subsequent game over the Rabin automata can be solved in \( n^k \), where \( n \) is the number of states of the automaton and \( k \) is the number of accepting pairs.

Over the years, this discouraging result has been mitigated by the identification of several restricted classes of LTL for which the complexity of the synthesis problem need not be so high. For example, in 1998 Asarin, Maler, Pnueli and Sifakis presented efficient polynomial (\( N^2 \)) solutions to games, and therefore synthesis problems, where the acceptance condition was restricted to an LTL formula of the form \( \square p, \lozenge q, \square \lozenge p \) or \( \diamond \lozenge q \). In 2004, Alur and La Torre also identified what they argued to be a compelling fragment of LTL that restricts to Boolean combinations of either \( (\lozenge, \land) \) or \( (\lozenge, \land, \lor) \), proving that the synthesis problem was PSPACE-complete or EXPSPACE, respectively, utilizing a parameterization in terms of the so-called longest distance (length of the longest simple path in the Büchi automaton) [8]. Further, in 2006, Piterman, Pnueli, and Sa’ar examined the synthesis of reactive designs when the LTL specification was restricted to the class of so-called Generalized Reactivity(1) (GR1) formulae, presenting an \( N^3 \)-time algorithm which checks whether the formula is realizable, and in the case where it is, constructs an automaton representing one of the possible implementing circuits [3].

In this paper, we identify another important and rich class of LTL formulae for which the synthesis problem does not require a double exponential solution. In particular, we show that when the specification, \( \varphi \), is restricted to formulae expressed in LTL interpreted over finite traces, so-called finite LTL or LTL\(_f\), the synthesis problem is EXPSPACE-complete. The notion of interpreting LTL over finite traces dates back at least 20 years to its use expressing domain control knowledge or temporally extended goals and preferences in artificial intelligence (AI) automated plan generation tasks. Various techniques for generating plans with finite LTL goals have been developed for actions with deterministic effects (e.g., [9, 10, 11, 12, 13, 14]) and more recently for Fully Observable Non-Deterministic Planning (FOND) problems, where actions have non-deterministic effects (e.g., [15]). Finite LTL generally exploits a weak-next operator (\( \blacklozenge \)) and often includes a special modality to designate the final state of the finite trace, distinguishing its syntax and expressivity from LTL interpreted over infinite traces (e.g., [16, 17, 18]). Finite LTL supports the specification
of an important classes of planning and synthesis problems. The majority of automated planning applications require solutions that are finite, such as a plan for a fleet of trucks to deliver a set of packages. Similarly, the class of synthesis problems for which solutions are finite is large, including a myriad of tasks from component robot skills or terminating software activities such as web services and the processing of business transactions (e.g., \cite{19,20}). Indeed, LTL$_f$ is used extensively in business process specifications \cite{21}.

In 2015, De Giacomo and Vardi investigated synthesis from LTL specifications interpreted over finite traces, characterizing the problem computationally as 2EXPTIME-complete and presenting a sound and complete synthesis technique based on Deterministic Finite Automata (reachability) games. This paper shows that synthesis from LTL specifications interpreted over finite traces is EXPTIME-complete. Similar in spirit to the work of Asarin et al., Alur and La Torre, and Piterman et al., this result is important not only for clarifying the complexity of the finite LTL synthesis task, but also because it identifies another rich fragment of LTL for which synthesis avoids the discouraging double exponential originally described by Pnueli and Rosner.

The paper proceeds as follows. First, we provide necessary background on alternating Turing machines and finite LTL, LTL$_f$. Next, we briefly describe a technique by the Torres and Baier’s for translating LTL$_f$ into alternating automata that has been used in support of automated plan generation with LTL$_f$ temporally extended goals \cite{22}. In the next section we introduce the problem of LTL$_f$ synthesis, and present the central contribution of this paper: a proof that LTL$_f$ synthesis is EXPTIME-complete. The proof is based on two lemmas. In Lemma 1 we show that LTL$_f$ synthesis is in EXPTIME by means of an algorithm, inspired in Torres and Baier’s alternating automata compilation, that shows that LTL$_f$ synthesis can be solved using alternating Turing machines in polynomial space. In Lemma 2 we show that LTL$_f$ synthesis is EXPTIME-hard. The technique we use transforms an alternating Turing machine into LTL$_f$ synthesis, and gains some inspiration from methods used by Rintanen to prove the complexity of non-deterministic planning with full and partial observability \cite{23}.

2. Preliminaries

In this section we review alternating Turing machines and LTL$_f$. We also overview a technique for translating LTL$_f$ into alternating automata. These concepts are necessary for the proofs that follow. The reader familiar with some or all of these topics can safely skip the corresponding subsections.

2.1. Alternating Turing Machines

Alternating Turing Machines (ATMs) where introduced by Chandra et al. (1981). Throughout the paper, we use a variant of the original definition, and follow the notation that has been used by Rintanen (2004).
Definition 1 (Alternating Turing Machine) An Alternating Turing Machine (ATM) is a tuple $\langle \Sigma, Q, q_0, g \rangle$ where:

- $\Sigma$ is a finite alphabet
- $Q$ is a finite set of internal states
- $\delta : \Sigma \cup \{ |, \square \} \times Q \to 2^{\Sigma \cup \{ | \} \times \{ L, N, R \}}$ is a transition function, where $|$ and $\square$ are, respectively, the end-of-tape and blank symbols. Additionally, we require:
  - $s' = |$ and $m = R$ for all $\langle s', q', m \rangle \in \delta(|, q)$
  - $s' \in \Sigma$ for all $\langle s', q', m \rangle \in \delta(s, q)$ where $s \in \Sigma$
- $q_0 \in Q$ is the initial internal state
- $g : Q \to \{ \text{accept}, \text{reject}, \exists, \forall \}$ is a labeling function. States $q \in Q$ with $g(q) = \forall$ are called universal, whereas states with $g(q) = \exists$ are called existential.

A configuration of an ATM $A$ is defined by the content of its tape cells, its internal state, and the position of the R/W head over the tape cells (see Definition 2). In the initial configuration, the R/W head is positioned over the second tape cell. The first tape cell has an end-of-tape symbol, and the following cells have, in order, the value of the elements in the input string $\sigma$. The remaining cells of the tape are assumed to be blank ($\square$).

For a given configuration $C$ of an ATM $A$, each element $\langle s', q', m \rangle$ in $\delta(q, s)$ identifies a successor configuration. Intuitively, the content of the tape cell under the R/W head is updated to $s'$. The internal state of $A$ switches to $q'$, and $m$ describes the movement of the R/W head – which moves either one cell right, left, or stays in the same position (see Definition 3).

Definition 2 (Configuration of an ATM) A configuration of an ATM $A = \langle \Sigma, Q, q_0, g \rangle$ is a tuple $\langle q, \sigma_l, \sigma_r \rangle$ where $q \in Q$ is the current state of $A$, $\sigma_l$ is a word containing the tape cells to the left head, including the symbol under the head, $\sigma_r$ is a word describing the contents of the tape to the right of the head. Elements in $\sigma_l$ and $\sigma_r$ have values in $\Sigma \cup \{ |, \square \}$. For an input string $\sigma = s_0, s_1, \ldots, s_n$, the initial configuration of $A$ is $\langle q_0, |s_0, s_1 \ldots s_n \rangle$.

Definition 3 (Successor Configurations of an ATM) For a configuration $C$ of an ATM $A = \langle \Sigma, Q, q_0, g \rangle$, the set of successor configurations is the smallest set that has the following elements:

- $\langle q', \sigma, s' \sigma' \rangle$ if $\langle s', q', L \rangle \in \delta(s, q)$ and $C = \langle q, \sigma s, \sigma' \rangle$
- $\langle q', \sigma s', \sigma' \rangle$ if $\langle s', q', N \rangle \in \delta(s, q)$ and $C = \langle q, \sigma s, \sigma' \rangle$
- $\langle q', \sigma s', \sigma' \rangle$ if $\langle s', q', R \rangle \in \delta(s, q)$ and $C = \langle q, \sigma s, t \sigma' \rangle$
• \( \langle q', \sigma s, \sigma' \rangle \) if \( \delta(s, q) \cap \sigma = \{ (s', q, R) \} \), \( C = \langle q, \sigma, \sigma' \rangle \), and \( |\sigma'| = 0 \)

A configuration \( \langle q, \sigma l, \sigma r \rangle \) of n ATM \( A = \langle \Sigma, Q, q_0 \rangle \) is final if \( g(q) \in \{ \text{accept}, \text{reject} \} \). The acceptance of a string \( \sigma \) by \( A \) is defined inductively as follows. A final configuration is 0-accepting if \( g(q) = \text{accept} \). A non-final configuration is \( n \)-accepting, for \( n > 0 \), if one of these conditions hold: (1) \( g(q) = \forall \), at least one successor configuration is \( (n-1) \)-accepting, and all other successor configurations are \( m \)-accepting for some \( m < n \); or (2) \( g(q) = \exists \), at least one successor configuration is \( (n-1) \)-accepting, and no other successor configurations are \( m \)-accepting for any \( m < n-1 \). A configuration is accepting if it is \( n \)-accepting for some \( n \geq 0 \). In particular, we say that \( A \) accepts string \( \sigma \) if the initial configuration is accepting.

Definition 4 (Computation Subtrees) The computation subtree of an ATM \( A = \langle \Sigma, Q, q_0 \rangle \) is the set \( T \) of accepting configurations defined recursively as follows:

• \( \langle q_0, \sigma \rangle \in T \), i.e., \( T \) contains the initial configuration of the ATM

• If \( \langle q, \sigma' \rangle \in T \) and \( g(q) = \forall \), then all successor configurations of \( \langle q, \sigma' \rangle \) are in \( T \)

• If \( \langle q, \sigma' \rangle \in T \), \( q \) is existential, and \( \langle q, \sigma' \rangle \) is \( n \)-accepting, then there exists at least one successor configuration of \( \langle q, \sigma' \rangle \) that is in \( T \) and is \( m \)-accepting for some \( m < n \).

A Deterministic Turing Machine (DTM) is an ATM without universal states, and such that \( |\delta(q, s)| = 1 \) for all \( q \in Q \) and \( s \in \Sigma \cup \{ \} \). EXPTIME is the class of decision problems that can be solved by DTMs that use a number of tape cells bounded by an exponential on the input length \( n \). APSPACE is the class of decision problems that can be solved by ATMs that use a number of tape cells bounded by a polynomial on the input length \( n \). It is well-known that \( \text{EXPTIME} = \text{APSPACE} \) [24].

2.2. Finite LTL

Finite LTL, or \( \text{LTL}_f \), is a variant of LTL interpreted over finite traces. The syntax of finite LTL differs only slightly from standard LTL, save the addition of a “weak next” operator \( (\bullet) \). The definition follows.

Definition 5 (LTLf formulas) The set of LTLf formulas over a set of propositions \( P \), \( \text{LTL}_f(P) \), is inductively defined as follows:

• \( T \), \( \bot \), and \( p \) are in \( \text{LTL}_f(P) \), for every \( p \in P \).

• If \( \varphi \) and \( \psi \) are in \( \text{LTL}_f(P) \) then so are \( \neg \varphi \), \( (\varphi \land \psi) \), \( (\varphi \lor \psi) \), \( (\varphi \circ \psi) \), \( (\varphi U \psi) \), and \( (\varphi R \psi) \).

The standard LTL operators eventually \( (\Diamond) \) and always \( (\square) \) are defined as macros \( \Diamond \varphi \equiv (T U \varphi) \) and \( \square \varphi \equiv (\bot R \varphi) \). As usual, \( T \) evaluates to true and \( \bot \) evaluates to false.
Definition 6 (Subformulas of an LTL\textsubscript{f} Formula)  Given a formula \( \varphi \) over a set of propositions \( \mathcal{P} \), we define the set of subformulae of \( \varphi \), denoted \( \text{sub}(\varphi) \), inductively as follows.

1. If \( \varphi \) is a proposition in \( \mathcal{P} \), then \( \text{sub}(\varphi) = \{ \varphi \} \).
2. If \( \varphi = \psi \ast \chi \), where \( \ast \) is a unary connective in \( \{ \neg, \circ, 
\ast \} \), then \( \text{sub}(\varphi) = \{ \varphi \} \cup \text{sub}(\psi) \cup \text{sub}(\chi) \).
3. If \( \varphi = (\psi \ast \chi) \) where \( \ast \) is any binary connective in \( \{ \land, \lor, U, R \} \) then \( \text{sub}(\varphi) = \{ \varphi \} \cup \text{sub}(\psi) \cup \text{sub}(\chi) \).

Finally, the size of a formula \( \varphi \), denoted by \(|\varphi|\), is defined as the number of connectives in \( \varphi \) plus the number of atomic formulas in \( \varphi \); thus the size of \( (p \lor \neg p) \) is 4.

Proposition 1 The cardinality of \( \text{sub}(\varphi) \) is at most \(|\varphi|\).

Proof: By induction on the construction of \( \varphi \). The base case, that is, \( \varphi \) is a propositional variable, is straightforward. For the induction, if \( \varphi = \psi \ast \chi \), for a unary connective \( \ast \), we observe that \(|\text{sub}(\varphi)| \leq 1 + |\text{sub}(\psi)| \), from where the results follows immediately by using the inductive hypothesis. The proof for binary connectives is analogous. If \( \varphi = (\psi \ast \chi) \), then \(|\text{sub}(\varphi)| \leq 1 + |\text{sub}(\psi)| + |\text{sub}(\chi)| \). In both cases, \(|\text{sub}(\varphi)| \leq |\varphi| \).

The truth of an LTL\textsubscript{f} formula over \( \mathcal{P} \) is evaluated over a finite word of states, where each state is an element over alphabet \( 2^{\mathcal{P}} \).

Definition 7 Given a finite word of states \( \sigma = s_0 \ldots s_n \), and a formula \( \varphi \in \text{LTL}\textsubscript{f}(\mathcal{P}) \), we say that \( \sigma \) satisfies \( \varphi \), denoted as \( \sigma \models \varphi \), iff it holds that \( \sigma,0 \models \varphi \), where, for every \( i \in \{0, \ldots, n\} \):

1. \( \sigma, i \models \top \).
2. \( \sigma, i \models p \) iff \( p \in s_i \), when \( p \in \mathcal{P} \).
3. \( \sigma, i \models \neg \varphi \) iff \( \sigma, i \not\models \varphi \)
4. \( \sigma, i \models (\psi \land \chi) \) iff \( \sigma, i \models \psi \) and \( \sigma, i \models \chi \)
5. \( \sigma, i \models (\psi \lor \chi) \) iff \( \sigma, i \models \psi \) or \( \sigma, i \models \chi \)
6. \( \sigma, i \models \circ \psi \) iff \( i < n \) and \( \sigma, (i+1) \models \psi \)
7. \( \sigma, i \models \bullet \psi \) iff \( i = n \) or \( \sigma, (i+1) \models \psi \)
8. \( \sigma, i \models (\psi U \chi) \) iff there exists \( k \in \{i, \ldots, n\} \) such that \( \sigma, k \models \chi \) and for each \( j \in \{i, \ldots, k-1\} \), it holds that \( \sigma, j \models \psi \)
9. \( \sigma, i \models (\psi R \chi) \) iff for each \( k \in \{i, \ldots, n\} \) it holds that \( \sigma, k \models \chi \) or there exists a \( j \in \{i, \ldots, k-1\} \) such that \( \sigma, j \models \psi \)

2.3. An Alternating Automaton for LTL\textsubscript{f}

We present Torres and Baier’s construction of alternating automata from LTL\textsubscript{f} formulae [22]. Given an LTL\textsubscript{f} formula \( \varphi \), this construction defines an alternating automaton \( A_\varphi \) that accepts precisely the models of \( \varphi \). We commence with some definitions and then present the construction.
2.3.1. Alternating Automata

Definition 8 (Positive Boolean Formula) The set of positive formulae over a set of propositions $P$—denoted by $B^+(P)$—is the set of all Boolean formulae over $P$ and constants $\bot$ and $\top$ that do not use the connective “$\neg$”.

Definition 9 (Alternating Automata) An alternating automata (AA) over words is a tuple $A = (Q, \Sigma, \delta, I, F)$, where $Q$ is a finite set of states, the alphabet $\Sigma$ is a finite set of symbols, $\delta : Q \times \Sigma \to B^+(Q)$ is the transition function, $I \subseteq Q$ are the initial states, and $F \subseteq Q$ is a set of final states.

For a set of states $Q' \subseteq Q$, we define $\delta(Q', s) \overset{\text{def}}{=} \bigwedge_{q \in Q'} \delta(q, s)$. In what follows, a word is an ordered finite sequence $x_1, x_2, \ldots, x_n$ of symbols in $\Sigma$.

Definition 10 (Run of an AA over a Finite String) A run of an AA $A = (Q, \Sigma, \delta, I, F)$ over word $x_1x_2\ldots x_n$ is a sequence $Q_0Q_1\ldots Q_n$ of subsets of $Q$, where $Q_0 = I$, and $Q_i \models \delta(Q_{i−1}, x_i)$, for every $i \in \{1, \ldots, n\}$.

Definition 11 A word $w$ is accepted by an AA, $A$, iff there is a run $Q_0\ldots Q_n$ of $A$ over $w$ such that $Q_n \subseteq F$.

For example, if the definition of an AA, $A$, is such that $\delta(q, b) = (s \land t) \lor r$, and $I = \{q\}$, then both $\{q\}\{s, t\}$ and $\{q\}\{r\}$ are runs of $A$ over word $b$.

2.3.2. Torres and Baier’s Automaton

Given a formula $\psi$ in $\mathsf{LTL}_f(P)$, the first step is to convert it into an equivalent formula, $\varphi$, in negation normal form (NNF). NNF is a form in which negations, if any, are applied only over propositional variables. This can be done in linear time by applying successively the substitutions given by $\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta)$, $\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta)$, $\neg\Box \alpha \equiv \Diamond \neg\alpha$, $\neg(\alpha \mathbf{U} \beta) \equiv (\neg\alpha \mathbf{R} \neg\beta)$, $\neg\Diamond \alpha \equiv \Box \neg\alpha$. Henceforth we assume that all $\mathsf{LTL}$ formulae are in NNF.

For a formula $\varphi$ Torres and Baier’s automaton is a tuple $A_\varphi = (Q, 2^P, \delta, q_0, \{q_F\})$, where $Q = \{\alpha \mid \alpha \in \text{sub}(\varphi)\} \cup \{q_F\}$ and:
This AA is based on Muller et al.’s AA for infinite [25]. Unlike theirs, this AA has a state $q_F$, which, besides from being a final state, has the intuitive meaning of “forcing the input to finish now”. Indeed, when the automaton is in $q_F$, processing any input symbol gets the automaton into a rejection configuration.

**Theorem 1 (Correctness of $A_{\varphi}$ [22])** $\sigma \models \varphi$ iff $A_{\varphi}$ accepts $\sigma$.

3. **$\text{LTL}_f$ Synthesis**

$LTL$ synthesis is a 2EXPTIME-complete problem. In this section, we introduce the problem of $LTL_f$ synthesis. The community has assumed that $LTL_f$ synthesis was a 2EXPTIME-complete problem [21]. We prove that, contrary to the common thought, $LTL_f$ synthesis is an EXPTIME-hard problem.

**Definition 12 ($LTL_f$ Realizability)** Given two disjoint sets of variables $X$ and $Y$, and a formula $\varphi$ over $LTL_f(X \cup Y)$, we say that $\varphi$ is realizable iff there exists a function $f : (2^X)^* \to 2^Y$ such that for every infinite word $X_1X_2\ldots$ over subsets of $X$, there exists a natural $n$ such that when $\pi = (X_1 \cup f(X_1))(X_2 \cup f(X_2X_1))\ldots(X_n \cup f(X_nX_{n-1}\ldots X_1))$, it holds that $\pi \models \varphi$.

**Lemma 1** $LTL_f$ realizability is in EXPTIME.

**Proof:** Algorithm 1 defines an alternating, non-deterministic procedure where $\text{REALIZABLE}(\{\varphi\})$ accepts iff $\varphi$ is realizable. Moreover, as we show below, it requires memory that is linear in $|\varphi|$. This proves that the problem is in APSPACE, the class of problems that can be decided with an ATM that requires polynomial space on its input. In addition, because $\text{APSPACE}=\text{EXPTIME}$ [24], we conclude that $LTL_f$ realizability is in EXPTIME.
Algorithm 1: An Algorithm for Deciding Realizability

1 \textbf{procedure} Realizable(\(Q\))
2 \hspace{1em} if \(Q \subseteq \{q_F\}\) then \hspace{1em} accept
3 \hspace{2em} if \(q_F \in Q\) then \hspace{1em} reject
4 \hspace{1em} forall subsets \(X\) of \(X\) do
5 \hspace{2em} guess a subset \(Y\) of \(Y\) do
6 \hspace{3em} s \leftarrow X \cup Y
7 \hspace{3em} Progress(Q, \emptyset, s)

10 \textbf{procedure} Progress(Q_{old}, Q_{new}, s)
11 \hspace{1em} if \(Q_{old} = \emptyset\) then \hspace{1em} Realizable(Q_{new})
12 \hspace{2em} Remove a formula \(\varphi\) from \(Q_{old}\) (different from \(q_F\))
13 \hspace{1em} switch \(\varphi\) do
14 \hspace{2em} case \(\varphi\) is a literal \hspace{1em} if \(s \nmid \varphi\) then \hspace{1em} reject
15 \hspace{2em} case \(\varphi\) is of the form \(\alpha \lor \beta\) \hspace{1em} choose
16 \hspace{3em} either Progress(Q_{old} \cup \{\alpha\}, Q_{new}, s)
17 \hspace{3em} or Progress(Q_{old} \cup \{\beta\}, Q_{new}, s)
18 \hspace{2em} case \(\varphi\) is of the form \(\alpha \land \beta\) \hspace{1em} Progress(Q_{old} \cup \{\alpha, \beta\}, Q_{new}, s)
19 \hspace{2em} case \(\varphi\) is of the form \(\alpha \lor \beta\) \hspace{1em} Progress(Q_{old}, Q_{new} \cup \{\alpha\}, s)
20 \hspace{2em} case \(\varphi\) is of the form \(\alpha \land \beta\) \hspace{1em} progress(Q_{old}, Q_{new} \cup \{\alpha\}, s)
21 \hspace{2em} case \(\varphi\) is of the form \(\alpha \lor \beta\) \hspace{1em} choose
22 \hspace{3em} either Progress(Q_{old}, Q_{new} \cup \{q_F\}, s)
23 \hspace{3em} or Progress(Q_{old} \cup \{\alpha\}, Q_{new}, s)
24 \hspace{2em} case \(\varphi\) is of the form \(\alpha \land \beta\) \hspace{1em} Progress(Q_{old} \cup \{\alpha\}, Q_{new} \cup \{\alpha \land \beta\}, s)
25 \hspace{2em} case \(\varphi\) is of the form \(\alpha \lor \beta\) \hspace{1em} Progress(Q_{old}, Q_{new} \cup \{\alpha \lor \beta\}, s)
26 \hspace{2em} case \(\varphi\) is of the form \(\alpha \land \beta\) \hspace{1em} progress(Q_{old}, Q_{new} \cup \{\beta\}, s)
27 \hspace{2em} case \(\varphi\) is of the form \(\alpha \lor \beta\) \hspace{1em} progress(Q_{old}, Q_{new} \cup \{q_F\}, s)
28 \hspace{2em} case \(\varphi\) is of the form \(\alpha \land \beta\) \hspace{1em} Progress(Q_{old} \cup \{\alpha\}, Q_{new}, s)
29 \hspace{2em} case \(\varphi\) is of the form \(\alpha \lor \beta\) \hspace{1em} Progress(Q_{old}, Q_{new} \cup \{\alpha \lor \beta\}, s)
Before continuing with the proof, we provide some intuition for both procedures. Let us first denote the set of infinite words over alphabet $2^X$ by $W_\infty$. The objective of the algorithm is to prove that for every infinite word $w = X_1X_2 \ldots \in W_\infty$, there exists a finite word $Y_1Y_2 \ldots Y_n$ over alphabet $2^Y$ such that when $\pi = (X_1 \cup Y_1)(X_2 \cup Y_2) \ldots (X_N(w) \cup Y_N(w))$, where $N$ is a function from $W_\infty$ to the natural numbers, it holds that $\pi \models \varphi$. To prove this, via successive calls, the recursive algorithm $\text{REALIZABLE}$ attempts to compute an accepting run $Q_0Q_1 \ldots Q_{N_\infty(w)}$ of the automaton $A_\varphi$ for each possible $w \in W_\infty$, by guessing sets $Y_0Y_1 \ldots Y_{N_\infty(w)}$. In the $k$-th recursive call, as we prove below, the set of states $Q$ used as the argument for the procedure corresponds to $Q_k$.

In each call to $\text{REALIZABLE}$, both variables for $X$ and $Y$ are chosen non-deterministically (Lines 6 and 7). Observe, however, that variables from $X$ are chosen using a universal transition. (In fact, Line 9 is the only universal transition in the algorithm.) Once state $s$ is built, the algorithm invokes procedure $\text{PROGRESS}$.

Procedure $\text{PROGRESS}$ is also a recursive procedure whose objective is to build the next set of states that conforms to a run (that is, it computes $Q_{k+1}$ given $Q_k$ and the $(k+1)$-th symbol of the input string). To do this it uses two arguments: $Q_{\text{old}}$ and $Q_{\text{new}}$. The former set represents the set of states $A_\varphi$ is in before processing symbol $s$. The latter is the new set of states. Through multiple iterations, $\text{PROGRESS}$ builds $Q_{\text{new}}$ and yields the control back to $\text{REALIZABLE}$ when $Q_{\text{old}}$ is empty (Line 12).

Now we argue that the algorithm we propose is correct, that is, it accepts iff $\varphi$ is realizable. We start by arguing that $\text{PROGRESS}$ is correct. Specifically we prove that if $\text{PROGRESS}$ is called in Line 9 with $Q$ as the first argument and $s$ as the third argument, and, after a number of self-recursive calls of $\text{PROGRESS}$, the next call to $\text{REALIZABLE}$ in Line 12 is made with argument $Q'$, then $Q' \models \delta(Q, s)$. This follows from the fact that $\text{PROGRESS}$ adds to $Q_{\text{new}}$ precisely the formulae that are added in a branch of a Tableaux proof tree for the formula $\delta(Q, s)$. In the successive calls for $\text{PROGRESS}$ when $Q_{\text{old}}$ is empty, $Q_{\text{new}}$ thus contains a model of the formula $\delta(Q, s)$. Observe also that a $\text{PROGRESS}$ call never reaches Line 12 and rejects: if the Tableaux proof tree branch would have added $\bot$ to the Tableaux set, it would mean that there is no model for the formula.

Now we prove that $\text{REALIZABLE}$ is correct too, that is, it accepts iff $\varphi$ is realizable. For the $\Rightarrow$ direction, assume $\text{REALIZABLE}$ accepts. Then the accepting run of $\text{REALIZABLE}$ can be conceptualized as a finite tree, as shown in Figure 1, which every internal node has one children for each element of $2^X$. Each of these sets is accompanied by an element of $2^Y$. Thus each branch of the tree of length $n$ defines words $X_1X_2 \ldots X_n$ and $Y_1Y_2 \ldots Y_n$, respectively in $(2^X)^*$ and $(2^Y)^*$, together with a sequence $Q_0Q_1 \ldots Q_n$ of sets of states of $A_\varphi$, such that $Q_0 = \{ \varphi \}$ and $Q_{k+1} \models \delta(Q_k, s)$, for every $k \in \{0, \ldots, n-1\}$, and such that $Q_n \subseteq \{ q_f \}$. From the correctness of the automaton, this means that $(X_1 \cup Y_1)(X_2 \cup Y_2) \ldots (X_n \cup Y_n) \models \varphi$. Finally, it only remains to define function $f$: for every path of the tree labeled with $(X_1, Y_1)(X_2, Y_2) \ldots (X_m, Y_m)$, we define $f(X_1 \ldots X_1) = Y_m$. This proves that $\varphi$ is realizable since for every infinite word $w_\text{X}$ in $2^X$ there is a branch in the tree that defines a (finite) prefix of $w_\text{X}$. 10
Figure 1: A finite tree that represents a (finite) accepting run of procedure \texttt{Realizable}. Each node represents a call to \texttt{Realizable}, and thus is associated to a set of states of A. \( N \) is the cardinality of \( 2^X \).

Each \( X^\sigma \) set is in \( 2^X \) and all “\( X \)-sets” that share the same parent are different; i.e., \( X^{\sigma,i} = X^{\sigma,j} \Rightarrow i = j \).

Analogously, each \( Y^\sigma \) set is in \( 2^Y \) however two “\( Y \)-sets” sharing the same parent need not be different.

For the \( \Leftarrow \) direction, recall we denote the set of infinite words over alphabet \( 2^X \) as \( W_\infty \). Because \( \varphi \) is realizable, for each \( w = X_1 X_2 \ldots \) in \( W_\infty \) there exists a finite word \( Y_1 \ldots Y_{N(w)} \) in \( (2^Y)^* \) such that \( (X_1 \cup Y_1) \ldots (X_{N(w)} \cup Y_{N(w)}) \models \varphi \), where \( N \) is a function from words in \( W_\infty \) to the natural numbers. This means that to prove realizability, we only need to check a (finite) number of finite words rather than over all words in \( W_\infty \); in fact, we need to check at most all words in \( (2^X)^* \) whose size is bounded by \( \max_{w \in W_\infty} N(w) \). In other words, this means that for each infinite word \( w \) in \( W_\infty \), there is a prefix of \( w \) that is sufficient to characterize all words in \( W_\infty \) that share the same prefix. Then it is clear that if one can provide a finite proof for realizability, it is possible to represent this proof as a finite tree, analogous to the one shown Figure 1. This means that \texttt{Realizable}({\{\varphi\}}) will accept.

We have proven that \texttt{Realizable}({\{\varphi\}}) accepts iff \( \varphi \) is realizable. Now it remains to prove that a call to \texttt{Realizable}({\{\varphi\}}) requires only polynomial memory on \( |\varphi| \). Clearly, all operations in the \texttt{Progress} procedure require polynomial space. To see this, observe that the cardinality of both \( Q_{\text{old}} \) and \( Q_{\text{new}} \) is at most \( |\text{sub}(\varphi)|+1 \). This implies that \texttt{Realizable} requires memory linear in \( \varphi \) to store its \( Q \) argument. Moreover, the other operations in \texttt{Realizable}, namely the \texttt{forall} and \texttt{guess} can be implemented in polynomial space by choosing, in sequence, whether or not each variable is included in the set \( X \).

We conclude that \texttt{LTL}_f realizability can be decided in \texttt{APSAFE} and thus that it is in \texttt{EXPTIME}.

\begin{lemma}
\texttt{LTL}_f realizability is \texttt{EXPTIME}-hard.
\end{lemma}
Proof:

We show that LTL\(_f\) synthesis is at least as hard as the problem of deciding plan existence in ATMs with a polynomial space bound \(p(x)\) on the size of the input string. We do so by introducing a polynomial time transformation, similar to the one used by Rintanen to prove that non-deterministic planning with partial observability is 2EXPTIME-hard \([23]\).

First, we describe the transformation from a given ATM \(A = \langle \Sigma, Q, \delta, q_0, g \rangle\) with input string \(\sigma = s_1, \ldots, s_n\) – therefore, space bound \(p(n)\) – to an LTL\(_f\) synthesis specification. Finally, we show that accepting ATM computation trees correspond one-to-one with solutions to the LTL\(_f\) synthesis problem.

Controllable Variables

The set of controllable variables in the LTL\(_f\) synthesis problem consists of the following sets:

- \(\{v_q \mid q \in Q\}\), denoting the internal state of the ATM
- \(\{v_{s,i} \mid s \in \Sigma \cup \{|, \boxcheck\} \text{ and } i \in \{0, \ldots, p(n)\}\}\), denoting the state of each tape cell
- \(\{v_i \mid i \in \{0, \ldots, p(n) + 1\}\}\), denoting the position of the R/W head
- \(Z = \{z_1, z_2, \ldots, z_\Delta\}\), where \(\Delta = \max\{|\delta(s, q)| \mid s \in \Sigma \cup \{|, \boxcheck\}, q \in Q\}\)

The value of these variables simulate the configuration of the ATM. Intuitively, \(v_q\) is true when the internal state of \(A\) is \(q\), \(v_{s,i}\) is true when the value in the \(i\)-th tape cell is \(s\), and \(v_i\) is true when the R/W head is over the \(i\)-th tape cell. The values of variables in \(Z\) are used to uniquely identify one of the successor configurations of the ATM when the internal state is existential.

Uncontrollable Variables

The set of uncontrollable variables is \(X = \{x_1, x_2, \ldots, x_\Delta\}\).

For convenience, we define the formulae \(\text{eval}_X(i, k)\) as follows:

\[
\text{eval}_X(i, k) = \begin{cases} 
  x_i \land \bigwedge_{j=1..i-1} \neg x_j & \text{if } i < k \\
  \bigwedge_{j=1..i-1} \neg x_j & \text{if } i = k \\
  \text{undefined} & \text{if } i > k 
\end{cases}
\]

Intuitively, \(\text{eval}_X(i, k)\) evaluates true when \(i\) is the lowest index of the first \(k\) variables in \(X\) that evaluates true. \(\text{eval}_X(k, k)\) also evaluates true when none of the first \(k\) variables in \(X\) evaluates true. We will use the truth of these formulae to distinguish one among \(k\) possible transitions. For the set of controllable variables \(Z\), \(\text{eval}_Z(i, k)\) is defined analogously.

Initial Configuration

Let \(C = (q', \sigma_l, \sigma_r)\) be a configuration of the ATM \(A\), where the internal state is \(q'\), the values in the tape cells are \(s_0, \ldots, s_{p(n)}\), and the R/W head is over \(s_j\). We simulate configuration \(C\) with the formula:

\[
\varphi_C = \bigwedge V_Q \land \bigwedge V_S \land \bigwedge V_H
\]

The sets of variables that constitute each of the subformulas in \(\varphi_C\) are defined as follows:
\( V_Q = \{ v_q' \} \cup \{ \neg v_q \mid q \in Q \setminus \{ q' \} \} \)

\( V_S = \{ v_{s,i} \mid s \in \Sigma \cup \{ |, \square \}, s_i \in \sigma', s_i = s \} \cup \{ \neg v_{s,i} \mid s \in \Sigma \cup \{ |, \square \}, s_i \in \sigma', s_i \neq s \} \)

\( V_H = \{ v_j \} \cup \{ \neg v_i \mid i \in \{ 0, 1, \ldots, p(n) + 1 \}, i \neq j \} \)

In order to simulate the initial configuration of the ATM with input string \( \sigma = s_1, \ldots, s_n \), we consider the formula \( \varphi_C \) with \( C = \langle q_0, \sigma_l, \sigma_r \rangle \) as follows. The R/W head is placed over \( s_1 \), so \( \sigma_l = | s_1 \). The values of the remaining tape cells are \( \sigma_r = s_2 \ldots s_{p(n)} \).

Here, we set \( s_i = \square \) for \( n < i < p(n) + 1 \), as the value of the tape cells in positions greater than \( n \) are initially blank cells. Note that the value of controllable variables \( Z \) is not set.

**Goal Configuration**

The goal configuration of the ATM is captured in Formula (2), which simulates achievement of an accepting state of the ATM.

\[ \diamond \bigvee_{q \in Q, \rho(q) = \text{accept}} v_q \tag{2} \]

**Transitions**

For all \( s \in \Sigma \cup \{ |, \square \} \) and \( q \in Q \), the transition function \( \delta \) maps the pair \( \langle s, q \rangle \) into a set of tuples \( \delta(s, q) = \{ (s'_1, q'_1, m_1), \ldots, (s'_k, q'_k, m_k) \} \), where \( s'_i \in \Sigma \cup \{ | \} \), \( q'_i \in Q \), and \( m \in \{ L, R, N \} \). Let \( \alpha, \kappa, \) and \( \theta \) be functions such that, for each \( \langle s'_\rho, q'_\rho, m_\rho \rangle \in \delta(s, q) \), we have \( s'_\rho = \alpha(s, q, \rho) \), \( q'_\rho = \kappa(s, q, \rho) \), and \( m_\rho = \theta(s, q, \rho) \) for \( \rho = 1..|\delta(s, q)| \), and undefined for \( \rho > |\delta(s, q)| \).

The successor configurations of the ATM are captured with the families of LTLf formulae described below. Each family of formulae can be viewed as successor state axioms [26], that describe the updates in the value of each variable after each time step.

For each \( q \in Q \) such that \( g(q) = \exists \), and each \( s \in \Sigma \cup \{ |, \square \} \):

\[ \bigwedge_{j=1..p(n)+1} \square(\bigcirc v_{s,j} \leftrightarrow \bigvee_{i=1..p(n)} \neg v_i \land v_j) \land \bigvee_{\rho=1..|\delta(s, q)|} (v_{s,j} \land v_j \land v_q \land \text{eval}_Z(\rho, |\delta(s, q)|)) \tag{3} \]

\[ \square(\bigcirc v_{q'} \leftrightarrow \bigvee_{\rho=1..|\delta(s, q)|} (v_{s,i} \land v_i \land v_q \land \text{eval}_Z(\rho, |\delta(s, q)|))) \tag{4} \]
For each $q \in Q$ such that $g(q) = \forall$, and each $s \in \Sigma \cup \{|, \square\}$:

\[
\bigwedge_{j=1..p(n)+1} \Box (\bigcirc v_j \leftrightarrow \bigvee_{i=1..p(n), i \neq j} (v_{s,i} \wedge v_i \wedge \neg v_j) \vee (v_{s,j} \wedge v_j \wedge v_{q} \wedge \text{eval}_Z(\rho, |\delta(s, q)|)))
\]

(5)

Intuitively, the family of formulae in (3) simulate the update of the value in the tape cell under the R/W head according to one of the successor configurations of the ATM. Similarly, the family of formulae in (4) simulate the update of the internal state of the ATM. Finally, the family of formulae in (5) simulate the update of the position of the R/W head over the cell tapes of the ATM. In all cases, the selection of a particular successor configuration is simulated by $\text{eval}_Z(s, q)$, which depends on the value of the variables in $Z$. The families of Formulae (6), (7), and (8) are analogous. This time, $q$ is an universal state and all successor
configurations of the ATM are plausible and need to be considered. This is simulated by means of $\text{eval}_X(s, q)$, whose truth value depends on the set of uncontrollable variables $X$.

**Memory bounds** The formulae above simulate the transitions of the ATM, but do not guarantee the memory bounds are preserved. The memory bound is maintained if the R/W head never reaches the position $p(n) + 1$. We simulate this requirement with the following formula:

$$\square \neg v_{p(n)+1}$$

For a configuration $C$ of ATM $\mathcal{A}$, we consider the synthesis problem $\text{synth}(\mathcal{A}, C)$ whose specification is given by the conjunction of all formulae defined above. In particular, Formula (1) is such that simulates configuration $C$. Such transformation can be computed in polynomial time. We claim that if the ATM $\mathcal{A}$ with input string $\sigma$ accepts without violating the space bound, then the associated LTL synthesis problem also has a solution, and vice-versa.

The first implication we prove is, that from a computation tree of a $t$-accepting configuration $C$ of an ATM $\mathcal{A} = (\Sigma, Q, \delta, q_0, g)$ we can extract a solution for the LTL synthesis problem $\text{synth}(\mathcal{A}, C)$. We prove it via induction on $t$. If $t = 0$, then $g(q'_0) = \text{accept}$ and, by construction, the initial state of $\text{synth}(\mathcal{A}, C)$ is accepting. For the general case, we distinguish two cases:

- If $g(q'_0) = \exists$, then let $(s'_\rho, q'_\rho, m_\rho) \in \delta(s_j, q'_0)$ be a transition that leads to a successor configuration in the ATM computation tree that is $(t - 1)$-accepting. Let $Z_0$ be the assignment of variables in $Z$ that satisfy $\text{eval}_Z(\rho, |\delta(s_j, q'_0)|)$. The first assignment to controllable variables in $\text{synth}(\mathcal{A}, C)$ assigns $Z_0$ to variables in $Z$. The values of the remaining controllable variables simulate the configuration $C$ of the ATM.

Let $C'$ be the successor configuration of $C$ that is associated with transition $(s'_\rho, q'_\rho, m_\rho) \in \delta(s_j, q'_0)$. Such configuration is $(t - 1)$-accepting and, by inductive hypothesis, we can construct a solution to $\text{synth}(\mathcal{A}, C')$. It is easy to see that the assignment to controllable variables defined above, followed by the the solution strategy to $\text{synth}(\mathcal{A}, C')$ is a solution to $\text{synth}(\mathcal{A}, C)$.

- If $g(q'_0) = \forall$, then the first assignment to controllable variables in $\text{synth}(\mathcal{A}, C)$ assigns any value to variables in $Z$. As in the previous case, the value of the remaining controllable variables simulates the configuration $C$ of the ATM.

Let $C'$ be the successor configuration of $C$ that is associated with transition $(s'_\rho, q'_\rho, m_\rho) \in \delta(s_j, q'_0)$. Such configuration is $t'$-accepting, for certain $t' < t$ and, by inductive hypothesis, we can construct a solution to $\text{synth}(\mathcal{A}, C')$. As in the previous case, the assignment to controllable variables defined above, followed by the the solution strategy to $\text{synth}(\mathcal{A}, C')$ is a solution to $\text{synth}(\mathcal{A}, C)$.
For the other direction, we first prove one of the contrapositives. If the ATM violates the space bounds $p(n)$, then at some point the variable $v_{p(n)+1}$ has to become true, representing that the R/W head moves to a tape cell position that violates the space bounds. At this point, the Formula (9) is not satisfied.

It remains to prove that from a solution to the \( \text{LTL}_f \) synthesis problem \( \text{synth}(\mathcal{A}, C_0) \), where \( C_0 \) is the initial configuration of the ATM \( \mathcal{A} \), we can construct a computation tree. This can be done by constructing the tree directly by unfolding a simulation of the \( \text{LTL}_f \) synthesis problem. By construction, assignments to variables in the \( \text{LTL}_f \) problem describe configurations of the ATM. Satisfaction of Formula (2) guarantees that the ATM is accepting.

We have shown that deciding whether or not an ATM with a polynomial space bound accepts can be converted to an \( \text{LTL}_f \) synthesis problem in polynomial time. Therefore, existence of solutions to \( \text{LTL}_f \) synthesis is \( \text{APSPACE} \)-hard or, equivalently, \( \text{EXPTIME} \)-hard.

\textbf{Theorem 2} \( \text{LTL}_f \) realizability is \textit{EXPTIME-complete}.

\textbf{Proof:} Follows directly from Lemmas 1 and 2.

4. Conclusion

In this research note we have presented a proof that the synthesis problem with logical specification represented in \( \text{LTL}_f \), \( \text{LTL} \) interpreted over finite traces, is \textit{EXPTIME}-complete. This result improves upon the \textit{2EXPTIME}-complete result reported in [21]. Our complexity result completes the landscape of complexity results that connect \( \text{LTL}_f \) synthesis and non-deterministic planning. With full observability, \( \text{LTL}_f \) synthesis and non-deterministic planning are \textit{EXPTIME}-complete. With partial observability, \( \text{LTL}_f \) synthesis and non-deterministic planning are \textit{2EXPTIME}-complete. This latter result was recently reported in [27].

Our proofs leverage recent advances from the automated planning community. In particular, the algorithm used to prove \textit{EXPTIME} membership in Lemma 1 is inspired by recent compilations based on alternating automata [22, 15]. Lemma 2 proves that \( \text{LTL}_f \) synthesis is \textit{EXPTIME}-hard, with a polynomial transformation of ATMs into \( \text{LTL}_f \) specifications.

AI automated planning with temporally extended goals specified in \( \text{LTL}_f \) is a practically motivated task of broad interest. The synthesis problem is an extension of plan synthesis where aspects of state are uncontrollable by the agent’s actions, and the synthesized procedure must fulfill the specification regardless of how the uncontrollable aspects of state change. Like it’s automated planning counterpart, \( \text{LTL}_f \) synthesis is well-motivated by myriad finite controller applications. Indeed, conditional planning with an \( \text{LTL}_f \) goal
is a special case of the synthesis problem that can be realized by LTL\(f\) FOND planning algorithms such as those proposed by the authors in related work [15]. The techniques developed in that work provide the computational core for the development of efficient algorithms for the realization of LTL\(f\) synthesis – a topic of current exploration. There is renewed interest to link planning and LTL synthesis, as demonstrated by [Sardina and D’Ippolito] where non-deterministic planning is modeled as a synthesis problem [28]. Our approach is complementary, as we leverage planning techniques to establish complexity results in synthesis.

Acknowledgements: The authors gratefully acknowledge funding from Fondecyt (grant number 1150328), and from the Natural Sciences and Engineering Research Council of Canada (NSERC) under the Discovery Grant and Postdoctoral Fellowship programs.

References

[1] A. Church, Applications of recursive arithmetic to the problem of circuit synthesis, Summaries of the Summer Institute of Symbolic Logic, Cornell University 1957 1 (1957) 3–50.

[2] A. Church, Logic, arithmetic and automata, in: Proceedings of the international congress of mathematicians, 23–35, 1962.

[3] N. Piterman, A. Pnueli, Y. Sa’ar, Synthesis of Reactive(1) Designs, in: Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings, 364–380, 2006.

[4] J. R. Büchi, L. H. Landweber, Solving Sequential Conditions by Finite-State Strategies, Transactions of the American Mathematical Society 138 (1969) 295–311.

[5] M. O. Rabin, Automata on Infinite Objects and Church’s Problem, American Mathematical Society, Boston, MA, USA, 1972.

[6] A. Pnueli, The temporal logic of programs, in: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 46–57, 1977.

[7] A. Pnueli, R. Rosner, On the Synthesis of a Reactive Module, in: Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989, 179–190, 1989.

[8] R. Alur, S. La Torre, Deterministic Generators and Games for LTL Fragments, ACM Trans. Comput. Logic 5 (1) (2004) 1–25.

[9] F. Bacchus, F. Kabanza, Using temporal logics to express search control knowledge for planning, Artificial Intelligence 116 (1-2) (2000) 123–191.
[10] P. Doherty, J. Kvarnström, TALplanner: A Temporal Logic-Based Planner, AI Magazine 22 (3) (2001) 95–102.

[11] S. Cresswell, A. M. Coddington, Compilation of LTL Goal Formulas into PDDL, in: R. L. de Mántaras, L. Saitta (Eds.), Proceedings of the 16th European Conference on Artificial Intelligence, IOS Press, Valencia, Spain, 985–986, 2004.

[12] J. A. Baier, S. A. McIlraith, Planning with First-Order Temporally Extended Goals using Heuristic Search, in: Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA, 788–795, 2006.

[13] M. Bienvenu, C. Fritz, S. A. McIlraith, Planning with Qualitative Temporal Preferences, in: Proceedings of the 10th International Conference on Knowledge Representation and Reasoning, 134–144, 2006.

[14] S. Edelkamp, On the Compilation of Plan Constraints and Preferences, in: Proceedings of the 16th International Conference on Automated Planning and Scheduling, 374–377, 2006.

[15] A. Camacho, E. Triantafillou, C. J. Muise, J. A. Baier, S. A. McIlraith, Non-Deterministic Planning with Temporally Extended Goals: Completing the Story for Finite and Infinite LTL (Amended Version), in: Proceedings of the Workshop on Knowledge-based Techniques for Problem Solving and Reasoning co-located with 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York City, USA, July 10, 2016., 2016.

[16] A. Gerevini, P. Haslum, D. Long, A. Saetti, Y. Dimopoulos, Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners, Artificial Intelligence 173 (5-6) (2009) 619–668.

[17] G. De Giacomo, M. Y. Vardi, Linear Temporal Logic and Linear Dynamic Logic on Finite Traces, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, 854–860, 2013.

[18] G. De Giacomo, R. D. Masellis, M. Montali, Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness, in: Proceedings of the 28th AAAI Conference on Artificial Intelligence, 1027–1033, 2014.

[19] S. A. McIlraith, T. C. Son, H. Zeng, Semantic Web Services, IEEE Intelligent Systems 16 (2) (2001) 46–53.

[20] G. De Giacomo, R. De Masellis, M. Grasso, F. M. Maggi, M. Montali, Monitoring Business Metaconstraints Based on LTL and LDL for Finite Traces, in: Business Pro-
[21] G. De Giacomo, M. Y. Vardi, Synthesis for LTL and LDL on Finite Traces, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, 1558–1564, 2015.

[22] J. Torres, J. A. Baier, Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals, in: Q. Yang, M. Wooldridge (Eds.), International Joint Conference on Artificial Intelligence, AAAI Press, Buenos Aires, Argentina, 1696–1703, 2015.

[23] J. Rintanen, Complexity of Planning with Partial Observability, in: Proceedings of the 14th International Conference on Automated Planning and Scheduling, 345–354, 2004.

[24] A. K. Chandra, D. Kozen, L. J. Stockmeyer, Alternation, Journal of the ACM 28 (1) (1981) 114–133.

[25] D. E. Muller, A. Saoudi, P. E. Schupp, Weak Alternating Automata Give a Simple Explanation of Why Most Temporal and Dynamic Logics are Decidable in Exponential Time, in: Proceedings of the 3rd Annual Symposium on Logic in Computer Science, 422–427, 1988.

[26] R. Reiter, Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems, MIT Press, Cambridge, MA, 2001.

[27] G. De Giacomo, M. Y. Vardi, LTLf and LDLf Synthesis under Partial Observability, in: Proceedings of the 25th International Joint Conference on Artificial Intelligence, 1044–1050, 2016.

[28] S. Sardina, N. D’Ippolito, Towards Fully Observable Non-deterministic Planning as Assumption-based Reactive Synthesis, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 3200–3206, 2015.