Synthesis for LTL and LDL on Finite Traces

Giuseppe De Giacomo
Sapienza Università di Roma
Roma, Italy
degiacomo@dis.uniroma1.it

Moshe Y. Vardi
Rice University,
Houston, TX, USA
vardi@cs.rice.edu

Abstract
In this paper, we study synthesis from logical specifications over finite traces expressed in LTL and its extension LDL. Specifically, in this form of synthesis, propositions are partitioned in controllable and uncontrollable ones, and the synthesis task consists of setting the controllable propositions over time so that, in spite of how the value of the uncontrollable ones changes, the specification is fulfilled. Conditional planning in presence of declarative and procedural trajectory constraints is a special case of this form of synthesis. We characterize the problem computationally as 2EXPTIME-complete and present a sound and complete synthesis technique based on DFA (reachability) games.

1 Introduction
LTL, Linear Temporal Logic on finite traces, has been extensively used in AI and CS, see [De Giacomo and Vardi, 2013] for a survey. For example, it is at the base of trajectory constraints in PDDL 3.0 [Bacchus and Kabanza, 2000; Gerevini et al., 2009], the de-facto standard formalism for representing planning problems. Also LTL is used as a declarative formalism to specify (terminating) services and processes by the business process modeling community [Pesic and van der Aalst, 2006; Montali et al., 2010; Sun et al., 2012; De Giacomo et al., 2014].

In this paper we study synthesis from specifications expressed in LTL and its extension LDL, Linear Dynamic Logic on finite traces, that allows for regular expressions in the temporal operators [De Giacomo and Vardi, 2013]. We consider propositions partitioned into two sets: the first under the control of our agent and the second not (e.g., variables in the second set are controlled by the environment). The specification consists of an LTL/LDL formula (or finite set of formulas), which expresses how both kinds of propositions should evolve over time. The problem of interest is checking whether there are strategies to set the controllable propositions over time so that, in spite of the values assumed by the uncontrollable ones, the specification is fulfilled. If such strategies exist, it is of interest to compute one.

Notice that conditional planning with full observability is indeed a special case of such a problem. Consider a non-deterministic domain with a given initial state and reachability goal. It is easy to encode actions as propositions and then express the nondeterministic domain as a finite set of simple LTL formulas, the initial state as a propositional formula, and the reachability goal φ as the LTL formula 2φ. Here, the controllable propositions are the actions, while the uncontrollable ones are all the others, which we may call fluents. Then, planning amounts to solving a special case of the LTL synthesis problem. Interestingly, if we add temporal constraints (also specifiable in LTL) to be satisfied while reaching the goal [Gerevini et al., 2009], we still get a special case of the LTL synthesis problem, as we do if we consider temporally extended goals [Bacchus and Kabanza, 1996], instead of reachability goals (as long as we require eventual termination). If we consider procedural execution constraints as in [Fritz and McIlraith, 2007; Baier et al., 2008], which are typically expressible as regular expressions that must be fulfilled by traces, we can resort to LDL/LTL (LTL is not expressive enough in this case) and planning becomes a special form of the LDL synthesis problem studied here.

Technically, our synthesis problem relates to the classical Church realizability problem [Church, 1963; Vardi, 1996] and it has been thoroughly investigated in the infinite setting, starting from [Pnueli and Rosner, 1989]. Unfortunately, while theoretically well investigated, algorithmically it still appears to be prohibitive in the infinite setting, not so much due to the high complexity of the problem, which is 2EXPTIME-complete, but for the difficulties of finding good algorithms for automata determinization, a crucial step in the solution, see, e.g., [Fogarty et al., 2013].

In the finite setting, the difficulties of determinization disappear and hence a theoretical solution to this problem, as the one provided here, actually promises to be appealing for effective practical implementation.

Specifically, we present a general sound and complete solution technique for the synthesis based on DFA (reachability) games. These are games played over a deterministic automaton whose alphabet is formed by possible propositional interpretations. These games can be solved in linear time in the number of states of the automaton. We show how to transform an LTL or LDL specification first into an NFA (on possible propositional interpretations), and then, through determinization, into a DFA on which to play the DFA game.
The presented technique is double exponential in general but optimal, since it can be shown that the problem itself is 2EXPTIME-complete. Notice that in some cases, e.g., in the conditional planning setting, the determination step is not required and hence our technique becomes single exponential, which is the complexity of conditional planning with full observability [Rintanen, 2004].

2 LTLf and LDLf

In this paper, we adopt standard LTL and its variant LDL interpreted on finite traces. For lack of space, as our main reference we use [De Giacomo and Vardi, 2013], which, apart from introducing LDL on finite traces, surveys the main known results and techniques in the finite traces setting.

LTLf on finite traces (LTLf). LTL on finite traces, or LTLf, has essentially the same syntax as LTL on infinite traces [Pnueli, 1977], namely, given a set of $P$ of propositional symbols, LTLf formulas are obtained as follows:

$$\varphi ::= \phi \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid O \varphi \mid \bullet \varphi \mid \Diamond \varphi \mid \varphi U \varphi_2$$

where $\phi$ is a propositional formula over $P$, $O$ is the next operator, $\bullet$ is weak next, i.e., $\bullet \phi$ is an abbreviation for $\neg O \neg \phi$\footnote{In LTLf, unlike LTL on infinite traces, $\neg O \neg \phi \not\equiv O \phi$.}, $\Diamond \phi$ is eventually, $\square$ is always, and $U$ is until. We make use of the standard boolean abbreviations, including true and false.

The semantics of LTLf is given in terms of finite traces, i.e., finite words $\pi$, denoting a finite nonempty sequence of consecutive steps, over the alphabet of $2^P$, consisting of possible interpretations of the propositional symbols in $P$. We denote by $length(\pi)$ the length of the trace, and by $\pi(i)$, with $1 \leq i \leq length(\pi)$, the propositional interpretation at the $i$-th position in the trace. Given a finite trace $\pi$, we inductively define when an LTLf formula $\varphi$ is true at a step $i$ with $1 \leq i \leq length(\pi)$, written $\pi, i \models \varphi$, as follows:

- $\pi, i \models \phi$ iff $\pi(i) \models \phi$ (formula propositional);
- $\pi, i \models \neg \varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \land \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \varphi_1 \lor \varphi_2$ iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$;
- $\pi, i \models O \varphi$ iff $i < length(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \bullet \varphi$ iff $i < length(\pi)$ implies $\pi, i+1 \models \varphi$;
- $\pi, i \models \Diamond \varphi$ iff for some $j$ s.t. $i \leq j \leq length(\pi)$, we have $\pi, j \models \varphi$;
- $\pi, i \models \Box \varphi$ iff for all $j$ s.t. $i \leq j \leq length(\pi)$, we have $\pi, j \models \varphi$;
- $\pi, i \models \varphi_1 U \varphi_2$ iff for some $j$ s.t. $i \leq j \leq length(\pi)$, we have $\pi, j \models \varphi_2$, and for all $k$ s.t. $i \leq k < j$, we have $\pi, k \models \varphi_1$.

Notice that we have the usual boolean equivalences such as $\varphi_1 \lor \varphi_2 \equiv \neg \varphi_1 \land \neg \varphi_2$; furthermore we have that $\bullet \varphi \equiv \neg O \neg \varphi$, $\Diamond \varphi \equiv true U \varphi$, and $\Box \varphi \equiv \neg \Diamond \neg \varphi$. It is known that LTLf is as expressive as First Order Logic over finite traces and star-free regular expressions, so strictly less expressive than regular expressions, which in turn are as expressive as Monadic Second Order logic over finite traces. On the other hand, regular expressions are a too low level formalism for expressing temporal specifications, since, for example, they miss a direct construct for negation and for conjunction, see, e.g., [De Giacomo and Vardi, 2013].

Linear Dynamic Logic on finite traces (LDLf). Linear Dynamic Logic on finite traces, or LTLf, is as natural as LTLf, but with the full expressive power of Monadic Second Order logic over finite traces [De Giacomo and Vardi, 2013]. LTLf is obtained by merging LTLf with regular expressions through the syntax of the well-known logic of programs PDL, Propositional Dynamic Logic [Fischer and Ladner, 1979; Harel et al., 2000], but adopting a semantics based on finite traces. This logic is an adaptation of LDL on infinite traces, introduced in [Vardi, 2011].

Formally, LTLf formulas are built as follows:

$$\varphi ::= \phi \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid (\rho)\varphi \mid [\rho]\varphi$$

where $\phi$ is a propositional formula over $P$, $\rho$ denotes path expressions, which are regular expressions over propositional formulas $\phi$ with the addition of the test construct $\rho$ typical of PDL; and $\varphi$ stands for LTLf formulas built by applying boolean connectives and the modal connectives $(\rho)\varphi$ and $[\rho]\varphi$. In fact $(\rho)\varphi \equiv \neg (\rho) \neg \varphi$.

Intuitively, $(\rho)\varphi$ states that, from the current step in the trace, there exists an execution satisfying the regular expression $\rho$ such that its last step satisfies $\varphi$, while $[\rho]\varphi$ states that, from the current step, all executions satisfying the regular expression $\rho$ are such that their last step satisfies $\varphi$. Tests are used to insert into the execution path checks for satisfaction of additional LTLf formulas.

As for LTLf, the semantics of LTLf is given in terms of finite traces, i.e., finite words, denoting a finite, nonempty sequence of consecutive steps $\pi$ over the alphabet of $2^P$. As above, we use the notation $length(\pi)$ and $\pi(i)$, and in addition we denote by $\pi(i, j)$ the segment of the trace $\pi$ starting at the $i$-th step end ending at the $j$-th step. If $j > length(\pi)$, we get the segment from $i$-th step to the end.

The semantics of LTLf is defined by simultaneous induction on formulas and path expressions as follows: given a finite trace $\pi$, an LTLf formula $\varphi$ is true at a step $i$, with $1 \leq i \leq length(\pi)$, in symbols $\pi, i \models \varphi$, if:

- $\pi, i \models \phi$ iff $\pi(i) \models \phi$ (formula propositional);
- $\pi, i \models \neg \varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \land \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \varphi_1 \lor \varphi_2$ iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$;
- $\pi, i \models (\rho)\varphi$ iff there exists $j \geq i$ such that $\pi(i, j) \in L(\rho)$.

where the relation $\pi(i, j) \in L(\rho)$ is as follows:
\[\pi(i, j) \in \mathcal{L}(\phi) \text{ if } j = i + 1, j \leq \text{length}(\pi), \text{ and } \pi, i \models \phi \text{ (\phi propositional);}\]
\[\pi(i, j) \in \mathcal{L}(\phi) \text{ if } j = i \text{ and } \pi, i \models \neg \phi;\]
\[\pi(i, j) \in \mathcal{L}(\rho_1 + \rho_2) \text{ if } \pi(i, j) \in \mathcal{L}(\rho_1) \text{ or } \pi(i, j) \in \mathcal{L}(\rho_2);\]
\[\pi(i, j) \in \mathcal{L}(\rho_1; \rho_2) \text{ if there exists } k, \text{ with } i \leq k \leq j, \text{ such that } \pi(i, k) \in \mathcal{L}(\rho_1) \text{ and } \pi(k, j) \in \mathcal{L}(\rho_2);\]
\[\pi(i, j) \in \mathcal{L}(\rho^*) \text{ if } j = i \text{ or there exists } k, \text{ with } i \leq k \leq j, \text{ such that } \pi(i, k) \in \mathcal{L}(\rho) \text{ and } \pi(k, j) \in \mathcal{L}(\rho^*).\]

Notice that we have the usual boolean equivalences, and in addition \([\rho] \neg \phi \equiv \neg(\rho) \neg \phi\). We use the abbreviation last, standing for [true]false, to denote the last step of the trace.

It is easy to encode LTLf into LDLf: it suffices to observe that we can express the various LTLf operators by recursively applying the following translations:

- \(\diamond \phi\) translates to \((\text{true}) \diamond \phi;\)
- \(\lnot \phi\) translates to \(\neg(\text{true}) \lnot \phi = [\text{true}] \lnot \phi;\)
- \(\Box \phi\) translates to \((\text{true}) \Box \phi;\)
- \(\lnot(\phi_1 \lor \phi_2)\) translates to \(\langle (\phi_1 \lor \phi_2)^* \rangle \phi_2.\)

It is also easy to encode into LDLf regular expressions, used as a specification formalism for traces. Assuming all traces end with a special making symbol end (i.e., the last element of the trace is not part of regular expressions), a regular expression specification \(\rho\) translates to \(\langle \rho \rangle (\text{last} \land \text{end})\).

We say that a trace satisfies an LTLf/LDLf formula \(\phi\), written \(\pi \models \phi\), if \(\pi, 1 \models \phi\). Also sometimes we denote by \(\mathcal{L}(\phi)\) the set of traces that satisfy \(\phi\): \(\mathcal{L}(\phi) = \{\pi \mid \pi \models \phi\}\).

### 3 LDLf Automata

We can associate with each LDLf formula \(\phi\) an (exponential) NFA \(A_\phi\) that accepts exactly the traces that make \(\phi\) true. Here, we provide a simple algorithm for computing the NFA corresponding to an LDLf formula. The correctness of the algorithm is based on the fact that \(i\) can associate each LDLf formula \(\phi\) with a polynomial alternating automaton on words (AFW) \(A_\phi\) that accepts exactly the traces that make \(\phi\) true, and \(i\) every AFW can be transformed into an NFA, see, e.g., [De Giacomo and Vardi, 2013]. However, to formulate the algorithm, we do not need these notions, but we can work directly on the LDLf formula. In order to proceed, we put LDLf formulas \(\phi\) in negation normal form \(\text{nnf}(\phi)\) by exploiting equivalences and pushing negation inside as much as possible, until negation signs are eliminated except in front of propositional formulas. Note that computing \(\text{nnf}(\phi)\) can be done in linear time. Then, we define an auxiliary function \(\delta\) (which corresponds directly to the transition function of the AFW \(A_\phi\)) that takes an LDLf formula \(\psi\) in negation normal form and a propositional interpretation \(P\) for \(\mathcal{P}\), including explicitly a special proposition last to denote the last element of the trace, and returns a positive boolean formula whose atoms are subformulas of \(\psi\):

\[\delta(\psi, \Pi) = \begin{cases} \text{true} & \text{if } \Pi \models \psi \text{ (\psi propositional)}; \\ \text{false} & \text{if } \Pi \not\models \psi \end{cases}\]

Using the auxiliary function \(\delta\), we can build the NFA \(A_\phi\) of an LDLf formula \(\phi\) in a forward fashion according to the following algorithm:

```
algorithm LDLf2NFA
input LTLf formula \(\phi\)
output NFA \(A_\phi = (2^P \times S, \{s_0\}, \{s_f\}, \delta_{\phi})\)
\[s_0 \leftarrow \{\text{false}\}\] single initial state
\[s_f \leftarrow \emptyset\] single final state
while \((S \text{ or } \delta \text{ change})\) do
  if \((q \in S\) and \(q' \models \phi\) then
    \[S \leftarrow S \cup \{q'\}\] update set of states
    \[q \leftarrow q \cup \{q, (q, q')\}\] update transition relation
```

In the algorithm LDLf2NFA, states \(S\) of \(A_\phi\) are sets of atoms, where each atom consists of quoted subformulas of \(\phi\). Such sets of atoms are interpreted as conjunctions and in particular \(\emptyset\), i.e., the empty conjunction, stands for true. Formula \(\bigwedge_{\psi' \in S} \delta(\psi', \Pi)\) is a positive boolean formula whose atoms are again quoted subformulas of \(\phi\). Finally, \(q'\) is a set of atoms, consisting again of quoted subformulas of \(\phi\); such that, when seen as a propositional interpretation, makes \(\bigwedge_{\psi' \in S} \delta(\psi', \Pi)\) true. (In fact, we do not need to get all \(q'\) such that \(q' \models \phi\); but only the minimal ones.) Notice that trivially we have \(\emptyset, \emptyset, \emptyset\) for every \(a \in \Sigma\). The algorithm LDLf2NFA terminates in at most an exponential number of steps, and generates a set of states \(S\) whose size is at most exponential in the size of \(\phi\).

**Theorem 1.** Let \(\phi\) be an LDLf formula and \(A_\phi\) the NFA constructed by algorithm LDLf2NFA. Then \(\pi \models \phi\) if and only if \(\pi \in \mathcal{L}(A_\phi)\) for every finite trace \(\pi\).

To see why the above theorem holds consider that given a LDLf formula \(\phi\), \(\delta\) grounded on the subformulas of \(\phi\) becomes the transition function of the AFW, with initial state.
and no final states, corresponding to \( \varphi \). LDL\(_2\)2NFA essentially amounts to the procedure that transforms an AFW into an NFA, cf. [De Giacomo and Vardi, 2013]. Notice that we never need to construct the AFW using the above algorithm. We directly build the NFA using the rules \( \delta \). Finally, if we want to remove the special proposition \( \text{last} \) from \( \mathcal{P} \), we can easily transform the obtained automaton \( A_\pi = (2^P, S, \{\text{"false"}, \emptyset, \{\emptyset\}) \) into the new automaton \( A'_\pi \) by adding a special state \( \text{end} \):

\[
A'_\pi = (2^P - \{\text{last}\}, S \cup \{\text{end}\}, \{\text{"false"}, \emptyset, \{\emptyset, \text{end}\})
\]

where \((q, \Pi', q') \in g'\) if and only if \((q, \Pi', q') \in g\) or \((q, \Pi' \cup \{\text{last}\}, \text{true}) \in g\) and \(q' = \text{end}\).

It is easy to see that the NFA obtained can be built on-the-fly while checking for nonemptiness, hence we have:

**Theorem 2.** Satisfiability of an LDL\(_2\) formula can be checked in PSPACE by nonemptiness of \( A_\pi \) (or \( A'_\pi \)).

Considering that it is known that satisfiability in LDL\(_2\) is a PSPACE-complete problem, we can conclude that the proposed construction is optimal wrt computational complexity for satisfiability, as well as for validity and logical implication, which are linearly reducible to satisfiability in LDL\(_2\), cf. [De Giacomo and Vardi, 2013].

### 4 LTL\(_f\) and LDL\(_f\) Synthesis

In this section, we study the general form of synthesis for LTL\(_f\) and LDL\(_f\), sometimes called realizability, or Church problem, or simply reactive synthesis [Vardi, 1996; Pnueli and Rosner, 1989]. We partition the set \( \mathcal{P} \) of propositions into two disjoint sets \( X \) and \( Y \). We assume to have no control on the truth value of the propositions in \( X \), while we can control those in \( Y \).

The problem then becomes: can we control the values of \( Y \) in such a way that for all possible values of \( X \) a certain LTL\(_f\)/LDL\(_f\) formula remains true? More precisely, traces now assume the form \( \pi = (X_0, Y_0)(X_1, Y_1)(X_2, Y_2) \cdots (X_n, Y_n) \), where \((X_i, Y_i)\) is the propositional interpretation at the \( i \)-th position in \( \pi \), now partitioned in the propositional interpretation \( X_i \) for \( X \) and \( Y_i \) for \( Y \). Let us denote by \( \pi|_i \) the interpretation \( \pi \) projected only on \( X \) and truncated at the \( i \)-th position (included), i.e., \( \pi|_i = (X_0, X_1, \ldots, X_i) \). The realizability problem checks the existence of a function \( f : (2^X)^* \rightarrow 2^Y \) such that for all \( \pi \) with \( Y_i = f(\pi|_i) \), we have that \( \pi \) satisfies the formula \( \phi \). The synthesis problem consists of actually computing such a function. Observe that in realizability/synthesis we have no way of constraining the value assumed by the propositions in \( X \); the function we are looking for only acts on propositions in \( Y \). Realizability and synthesis for LTL on infinite traces has been introduced in [Pnueli and Rosner, 1989] and shown to be 2EXPTIME-complete for arbitrary LTL formulas.

We are going to devise a technique in this section to do the same kind of synthesis in the case of finite traces for LTL\(_f\) and LDL\(_f\). To do so, we will rely on DFA games, introduced below (see [Mazala, 2002] for a survey on game based approaches in the infinite trace setting).

**DFA Games.** DFA games are games between two players, here called respectively the environment and the controller, that are specified by a DFA. We have a set of \( X \) of uncontrollable propositions, which are under the control of the environment, and a set \( Y \) of controllable propositions, which are under the control of the controller. A round of the game consists of both the controller and the environment setting the values of the propositions they control. A (complete) play is a word in \((2^X \times 2^Y)^*\) describing how the controller and environment set their propositions at each round until the game stops.

The specification of the game is given by a DFA \( \mathcal{G} \) of the form \( \mathcal{G} = (2^X \times 2^Y, S, s_0, \delta, F) \), where:

- \( 2^X \times 2^Y \) is the alphabet of the game;
- \( S \) are the states of the game;
- \( s_0 \) is the initial state of the game;
- \( \delta : S \times 2^X \times 2^Y \rightarrow S \) is the transition function of the game; given the current state \( s \) and a choice of propositions \( X \) and \( Y \), respectively for the environment and the controller, \( \delta(s, (X, Y)) = s' \) is the resulting state of the game;
- \( F \) are the final states of the game, where the game can be considered terminated.

A play is winning for the controller if such a play leads from the initial to a final state. A strategy for the controller is a function \( f : (2^X)^* \rightarrow 2^Y \) that, given a history of choices from the environment, decides which propositions \( Y \) to set to true/false next. A winning strategy is a strategy \( f : (2^X)^* \rightarrow 2^Y \) such that for all \( \pi \) with \( Y_i = f(\pi|_i) \) we have that \( \pi \) leads to a final state of \( \mathcal{G} \). The realizability problem consists of checking whether there exists a winning strategy. The synthesis problem amounts to actually computing such a strategy.

We now give a sound and complete technique to solve realizability for DFA games. We start by defining the controllable preimage \( \text{PreC}(\mathcal{E}) \) of a set \( \mathcal{E} \) of states \( \mathcal{E} \) of \( \mathcal{G} \) as the set of states \( s \) such that there exists a choice of values for propositions \( Y \) such that for all choices of values for propositions \( X \), game \( \mathcal{G} \) progresses to states in \( \mathcal{E} \). Formally:

\[
\text{PreC}(\mathcal{E}) = \{s \in S \mid \exists Y \in 2^Y. \forall X \in 2^X. \delta(s, (X, Y)) \in \mathcal{E}\}
\]

Using such a notion, we define the set \( \text{Win}(\mathcal{G}) \) of winning states of a DFA game \( \mathcal{G} \), i.e., the set formed by the states from which the controller can win the DFA game \( \mathcal{G} \). Specifically, we define \( \text{Win}(\mathcal{G}) \) as a least-fixpoint, making use of approximates \( \text{Win}_i(\mathcal{G}) \) denoting all states where the controller wins in at most \( i \) steps:

- \( \text{Win}_0(\mathcal{G}) = F \) (the final states of \( \mathcal{G} \));
- \( \text{Win}_{i+1}(\mathcal{G}) = \text{Win}_i(\mathcal{G}) \cup \text{PreC}(\text{Win}_i(\mathcal{G})) \).

Then, \( \text{Win}(\mathcal{G}) = \bigcup_i \text{Win}_i(\mathcal{G}) \). Notice that computing \( \text{Win}(\mathcal{G}) \) requires linear time in the number of states in \( \mathcal{G} \). Indeed, after at most a linear number of steps \( \text{Win}_{i+1}(\mathcal{G}) = \text{Win}_i(\mathcal{G}) \cup \text{Win}_{i+1}(\mathcal{G}) \).

**Theorem 3.** A DFA game \( \mathcal{G} \) admits a winning strategy iff \( s_0 \in \text{Win}(\mathcal{G}) \).

Next, we turn to actually computing the strategy. To do so, we define a strategy generator based on the winning
sets $Win_i(G)$. This is a non-deterministic transducer, where non-determinism is of the kind “don’t-care”: all non-deterministic choices are equally good. The strategy generator $T_G = (2^{X \times Y}, S, s_0, \rho, \omega)$ is as follows:

- $2^{X \times Y}$ is the alphabet of the transducer;
- $S$ are the states of the transducer;
- $s_0$ is the initial state;
- $\rho: S \times 2^X \rightarrow 2^Y$ is the transition function such that $\rho(s, X) = \{s' | s' = \delta(s, (X, Y)) \text{ and } Y \in \omega(s)\};$
- $\omega: S \rightarrow 2^X$ is the output function such that $\omega(s) = \{Y | \text{ if } s \in Win_{i+1}(G) - Win_i(G) \text{ then } \forall X. \delta(s, (X, Y)) \in Win_i(G)\}.$

The transducer $T_G$ generates strategies in the following sense: for every way of further restricting $\omega(s)$ to return only one of its values (chosen arbitrarily), we get a strategy.

**Synthesis in LTL$_f$ and LDL$_f$.** To do synthesis in LTL$_f$ or LDL$_f$, we translate an LDL$_f$/LTL$_f$ specification $\varphi$ into an NFA $A_{\varphi}$, e.g., using the algorithm LDL$_f$/NFA presented above. This is an exponential step. Then, we transform the resulting NFA into a DFA $A_{\varphi}^d$, e.g., using the standard determinization algorithm based on the subset construction [Rabin and Scott, 1959]. This will cost us another exponential. At this point we view the resulting DFA $A_{\varphi}^d$ as a DFA game, considering exactly the separation between controllable and uncontrollable propositions in the original LDL$_f$/LTL$_f$ specification, and we solve it by computing $Win(A_{\varphi}^d)$ and the corresponding strategy generator $T_{A_{\varphi}^d}$. This is a linear step. The following theorem assesses the correctness of this technique:

**Theorem 4.** Realizability of an LDL$_f$/LTL$_f$ specification $\varphi$ can be solved by checking whether $s_0 \in Win(A_{\varphi}^d)$. Synthesis can be solved by returning any strategy generated by $T_{A_{\varphi}^d}$.

Moreover, considering the cost of each of the steps above, we get the following worst-case computational complexity upper bound.

**Theorem 5.** Realizability in LDL$_f$/LTL$_f$ can be solved in $2EXPTIME$, and synthesis (i.e., returning a winning strategy) can be done in time double exponential.

A matching lower-bound can be shown by reduction from EXPSPACE alternating Turing machines.

**Theorem 6.** Realizability (and synthesis) in LDL$_f$/LTL$_f$ is $2EXPTIME$-hard.

## 5 Relationship with planning

In this section, we discuss the relationship between LTL$_f$/LTL$_f$ synthesis and conditional planning with full observability. We can characterize an action domain by the set of allowed evolutions, each represented as a sequence of states of affair of the domain, which with a little abuse of terminology, we may call situations [Reiter, 2001]. To do so, we typically introduce a set of atomic facts, called fluents, whose truth value changes as the system evolves from one situation to the next one because of actions. Since LTL$_f$/LTL$_f$ do not provide a direct notion of action, we use propositions to denote them, as in [Calvanese et al., 2002]. Hence, we partition $P$ into fluents $F$ and actions $A$, adding structural constraints such as $\Box(\bigvee_{a \in A} a) \land \Box(\bigwedge_{a \in A} (a \rightarrow \bigwedge_{b \in A, b \neq a} \neg b))$, to specify that one action must be performed to get to a new situation, and that a single action at a time can be performed. Then, the initial situation is described by a propositional formula $\varphi_{\text{init}}$ involving only fluents, while effects can be modelled as:

$$\Box(\varphi \rightarrow \Box(a \rightarrow \psi))$$

where $a \in A$, while $\varphi$ and $\psi$ are arbitrary propositional formulas involving only fluents. Such a formula states that performing action $a$ under the conditions denoted by $\varphi$ brings about the conditions denoted by $\psi$. A formula like $\Box(\varphi \rightarrow \Box(a \rightarrow \varphi))$ corresponds to a frame axiom expressing that $\varphi$ does not change when performing $a$. Alternatively, we can formalize effects through Reiter’s successor state axioms [Reiter, 2001] (which implicitly provide a solution to the frame problem), as in [Calvanese et al., 2002; De Giacomo and Vardi, 2009], by translating the (instantiated) successor state axiom $F(do(a, s)) \equiv \varphi^+(s) \lor (F(s) \land \neg \varphi^-(s))$, where $\varphi^+(s)$ (resp. $\varphi^-(s)$) is the condition that makes the fluent $F$ true (resp. false) after action $a$, into the LTL$_f$ formula:

$$\Box(\Box(a \rightarrow F \equiv \varphi^+ \lor (F \land \neg \varphi^-))).$$

In general, to specify effects we need LTL$_f$ formulas that talk only about the current and the next state, to capture how the domain transitions from the current to the next state, as above.

Turning to goals, usual reachability goals can be expressed as $\varphi_{\text{goal}} = \Diamond \varphi$, where $\varphi$ is a propositional formula on fluents.

**PDDL trajectory constraints** [McDermott and others, 1998; Gerevini et al., 2009] are also representable in LTL$_f$/LTL$_f$. As an example:

$$\begin{align*}
(at\;end\;\varphi) & \equiv \Diamond(last\;\land\;\varphi) \\
(always\;\varphi) & \equiv \Box \varphi \\
(some\;time\;\varphi) & \equiv \Diamond \varphi \\
(within\;n\;\varphi) & \equiv \bigvee_{0 \leq i \leq n} \Diamond \varphi \\
\hspace{1cm} (hold\;after\;n\;\varphi) & \equiv \bigvee_{0 \leq i \leq n} \Diamond \varphi \\
\hspace{1cm} (hold\;during\;n_1\;n_2\;\varphi) & \equiv \bigvee_{0 \leq i \leq n_1} \Diamond \bigvee_{0 \leq i \leq n_2} \Diamond \varphi \\
\hspace{1cm} (at\;most\;once\;\varphi) & \equiv \Box(a \rightarrow \Diamond \neg \varphi) \\
\hspace{1cm} (some\;time\;before\;\varphi_1\;\varphi_2) & \equiv \Box(\varphi_1 \rightarrow \Diamond \varphi_2) \\
\hspace{1cm} (any\;time\;in\;n_1\;n_2) & \equiv \bigvee_{0 \leq i \leq n_1} \Diamond (\Box (0 \leq i \leq n_2) \Diamond \varphi_2) \\
\hspace{1cm} (always\;within\;n\;\varphi_1\;\varphi_2) & \equiv \Box(a \rightarrow \Diamond \neg \varphi_2)
\end{align*}$$

where $\varphi$ is a propositional formula on fluents, and $\varphi_1 W \varphi_2 = (\varphi_1 \land \varphi_2) \lor (\varphi_1 \land \varphi_2)$ is the so-called weak until, which states that $\varphi_1$ holds until $\varphi_2$ or forever.

Finally, we turn to procedural constraints. Typically, these cannot be expressed in LTL$_f$ but they can be expressed in LTL$_f$. For example, let us consider them expressed in a propositional variant of GOLOG [Levesque et al., 1997; Fritz and McIlraith, 2007; Baier et al., 2008]:

$$\begin{align*}
\begin{align*}
\end{align*}
\end{align*}$$
\( a \) atomic action
\( \phi \) test for a condition
\( \delta_1; \delta_2 \) sequence
if \( \phi \) then \( \delta_1 \) else \( \delta_2 \) conditional
while \( \phi \) do \( \delta \) while loop
\( \delta_1|\delta_2 \) nondeterministic branch
\( \pi.x.\delta \) nondeterministic choice of argument
\( \delta^* \) nondeterministic iteration

One can translate such programs into regular expressions:

\[
\begin{align*}
tr(a) &= true; a \\
tr(\phi?) &= \phi? \quad (\phi \text{ propositional}) \\
tr(\delta_1; \delta_2) &= tr(\delta_1); tr(\delta_2) \\
tr(\begin{cases} \phi \text{ then } \delta_1 \text{ else } \delta_2 \end{cases}) &= \phi?; tr(\delta_1) + \neg\phi?; tr(\delta_2) \\
tr(\begin{cases} \text{while } \phi \text{ do } \delta \end{cases}) &= (\phi?; tr(\delta_1))^*; \neg\phi? \\
tr(\delta_1|\delta_2) &= tr(\delta_1) + tr(\delta_2) \\
tr(\pi.x.\delta) &= \sum_{\pi} tr(\delta(x)) \\
tr(\delta^*) &= tr(\delta)^* 
\end{align*}
\]

Note that the atomic action \( a \) is translated into a test of an action proposition \( a \) (meaning action \( a \) has just been performed) in the next state. Then one can express GLOGO procedural constraints such as: \( \delta \) must be executed at least once during the plan, as \( (tr(\delta)); true; \) plans for goal \( \phi \) must be an executions of \( \delta \), as \( tr(\delta(x)); \phi \) along the plan every time \( \delta \) is executed \( \psi \) must hold, as \( tr(\delta)); \psi \).

Given a LTL/\( f \)/LDL\(_ f \) formulas \( \Psi_{domain} \) describing the action domain (a transition formula), \( \phi_{init} \) describing the initial state (a propositional formula), \( \Phi_{\text{constraints}} \) describing trace constraints, and \( \varphi_{goal} \) describing the goal, conditional planning amounts to solving the synthesis problem for the following specification:

\[
\phi_{init} \land \Psi_{domain} \land \Phi_{\text{constraints}} \land \varphi_{goal}
\]

where action propositions \( A \) are controllable and fluent propositions \( F \) are uncontrollable. The same holds if we consider arbitrary temporally extended goals expressed in LTL/\( f \)/LDL\(_ f \). In other words, by using the synthesis technique presented here we can generate plans in a very general setting: nondeterministic domains, arbitrary LTL/\( f \)/LDL\(_ f \) goals, arbitrary LTL/\( f \)/LDL\(_ f \) trajectory constraints.

Notice that in case we drop (declarative and procedural) trajectory constraints and we concentrate our attention to reachability goals of the form \( \Diamond \phi \), we get the classical conditional planning setting, which is known to be EXPTIME-complete [Rintanen, 2004]. Interestingly, our technique in this case matches the EXPTIME complexity, and hence is an optimal technique (wrt worst case computational complexity) for solving conditional planning problems. To see this, consider that the goal \( \Diamond \phi \) gives rise to a linear size DFA over the alphabet of propositional interpretations. The domain itself, once represented in LTL/\( f \)/LDL\(_ f \), also gives rise to a deterministic automaton over the alphabet of propositional interpretations: given the current state, which is a propositional

interpretation, and the next propositional interpretation, the domain can only accept it if it conforms to its transition rules or discard it if it does not. Indeed, the “nondeterminism” of the planning domain corresponds to the uncontrollability of the fluents and not to the nondeterminism of the automata on traces of propositional interpretations. As a result, the automaton that we get by translating the LTL/\( f \)/LDL\(_ f \) formula corresponding to the domain, initial state, and reachability goal (corresponding to the conditional planning problem) is indeed deterministic. So determination, which in general costs an exponential, is skipped, and we can directly solve the DFA game, getting overall a single exponential procedure as in [Rintanen, 2004].

However, determination may be required for more complex LTL/\( f \)/LDL\(_ f \) goals or trajectory constraints, since the goals or constrains themselves may give rise to NFAs over traces of propositional interpretations.

### 6 Conclusion

We have studied LTL/\( f \)/LDL\(_ f \) synthesis, assessing its complexity and devising an effective procedure to check the existence of a winning strategy and to actually compute it if it exists. The complexity characterization of the problem in the case of LTL/\( f \)/LDL\(_ f \) on finite traces is the same as the one for infinite trace, namely, 2EXPTIME-complete. Also the procedure for computing it is analogous [Mazala, 2002]. Starting from the logical specification, we get a nondeterministic automaton (NFA in the finite case, Büchi in the infinite case), we determinize it (in the infinite case, we change the automaton, e.g., to a parity one, since Büchi automata are not closed under determinization, indeed deterministic Büchi automata are strictly less expressive that nondeterministic ones), and then we solve the corresponding game (DFA games for finite traces, parity games for the infinite ones) considering which propositions are controllable and which are not. However, in the case of infinite traces, the determinization remains a very difficult step, and there are no good algorithms yet to perform it [Fogarty et al., 2013]. As a consequence, virtually no tools are available (an exception is Lily [Jobstmann and Bloem, 2006]). In the finite case, determinization is much easier: it requires the usual subset constructions and good algorithms are available. So effective tools can indeed be developed.

With respect to the latter, it would be particularly interesting to study how techniques developed for planning could be used to actually translate an LTL/\( f \)/LDL\(_ f \) specification and determinize the automaton on-the-fly while playing the DFA game [Baier and McIlraith, 2006; Geffner and Bonet, 2013].

Our work is also related to other forms of synthesis that are well established in AI, such as ATL model checking [Alur et al., 2002] or 2-player game structure model checking, see e.g., [De Giacomo et al., 2010]. The latter was used to solve our kind of synthesis for GR(1) LTL formulas in the infinite trace setting [Bloem et al., 2012]. The key difference between that kind of synthesis and the general one considered here is that there the game arena is given while the goal to play for is determined by the ATL or mu-calculus formula. In our case, instead, the game arena, i.e., the DFA game, is computed from the formula, while the goal to play for is fixed to

\footnote{Notice that the GLOGO nondeterministic pick construct \( \pi.x.\delta(x) \) makes sense only if the propositions in the domain are generated by finitely instantiating parametrized propositions as, for instance, in PDDL.}
be reachability of the final state of the automaton.

Acknowledgements. The authors would like to thank anonymous reviewers for their comments and Diego Calvanese for insightful discussions on the topic of the paper.

References


