Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

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_f and its extension LDL_f. 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_f, *Linear Temporal Logic on finite traces*, has been extensively used in AI an 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_f 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_f and its extension LDL_f, *Linear Dynamic Logic over 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_f/LDL_f 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 nondeterministic 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_f formulas, the initial state as a propositional formula, and the reachability goal ϕ as the LTL_f formula $\Diamond\phi$. 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_f synthesis problem. Interestingly, if we add temporal constraints (also specifiable in LTL_f) to be satisfied while reaching the goal [Gerevini *et al.*, 2009], we still get a special case of the LTL_f 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 constrains 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_f (LTL_f is not expressive enough in this case) and planning becomes a special form of the LDL_f 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_f or LDL_f 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 determinization step is not required and hence our technique becomes single exponential, which is the complexity of conditional planning with full observability [Rintanen, 2004].

2 LTL$_f$ and LDL$_f$

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.

Liner Temporal Logic on finite traces (LTL$_f$). LTL on finite traces, or LTL$_f$, has essentially the same syntax as LTL on infinite traces [Pnueli, 1977], namely, given a set of $\mathcal{P}$ of propositional symbols, LTL$_f$ formulas are obtained as follows:

$$\begin{array}{rcl}\varphi & ::= & \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \circ\varphi \mid \bullet\varphi \mid \\ & & \Diamond\varphi \mid \Box\varphi \mid \varphi_1 \mathcal{U} \varphi_2\end{array}$$

where ϕ is a propositional formula over $\mathcal{P}$, $\circ$ is the *next* operator, $\bullet$ is *weak next*, i.e., $\bullet\varphi$ is an abbreviation for $\neg\circ\neg\varphi$[1], $\Diamond$ is *eventually*, $\Box$ is *always*, and $\mathcal{U}$ is *until*. We make use of the standard boolean abbreviations, including $true$ and $false$. The semantics of LTL$_f$ is given in terms of *finite traces*, i.e., finite words π, denoting a finite nonempty sequence of consecutive steps, over the alphabet of $2^{\mathcal{P}}$, consisting of possible interpretations of the propositional symbols in $\mathcal{P}$. 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 π, we inductively define when an LTL$_f$ formula φ *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 $\pi(i) \models \phi$ (ϕ propositional);
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \varphi_1 \vee \varphi_2$ iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$;
- $\pi, i \models \circ\varphi$ 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 $\pi, j \models \varphi$;
- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ 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 $\varphi_1 \vee \varphi_2 \equiv \neg\varphi_1 \wedge \neg\varphi_2$; furthermore we have that $\bullet\varphi \equiv \neg\circ\neg\varphi$, $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$, and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. It is known that LTL$_f$ 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].

[1]In LTL$_f$, unlike LTL on infinite traces, $\neg\circ\neg\varphi \not\equiv \circ\varphi$.

Linear Dynamic Logic on finite traces (LDL$_f$). *Linear Dynamic Logic on finite traces*, or LDL$_f$ is as natural as LTL$_f$, but with the full expressive power of Monadic Second Order logic over finite traces [De Giacomo and Vardi, 2013]. LDL$_f$ is obtained by merging LTL$_f$ 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 interpreted over infinite traces, introduced in [Vardi, 2011].

Formally, LDL$_f$ formulas are built as follows:

$$\begin{array}{rcl}\varphi & ::= & \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \langle\rho\rangle\varphi \mid [\rho]\varphi \\ \rho & ::= & \phi \mid \varphi? \mid \rho_1 + \rho_2 \mid \rho_1; \rho_2 \mid \rho^*\end{array}$$

where ϕ is a propositional formula over $\mathcal{P}$; ρ denotes path expressions, which are regular expressions over propositional formulas ϕ with the addition of the test construct $\varphi?$ typical of PDL; and φ stands for LDL$_f$ formulas built by applying boolean connectives and the modal connectives $\langle\rho\rangle\varphi$ and $[\rho]\varphi$. In fact $[\rho]\varphi \equiv \neg\langle\rho\rangle\neg\varphi$.

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

As for LTL$_f$, the semantics of LDL$_f$ is given in terms of *finite traces*, i.e., finite words, denoting a finite, nonempty, sequence of consecutive steps π over the alphabet of $2^{\mathcal{P}}$. 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 π 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 LDL$_f$ is defined by simultaneous induction on formulas and path expressions as follows: given a finite trace π, an LDL$_f$ formula φ *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 $\pi(i) \models \phi$ (ϕ propositional);
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \varphi_1 \vee \varphi_2$ iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$;
- $\pi, i \models \langle\rho\rangle\varphi$ iff there exists $j \geq i$ such that $\pi(i,j) \in \mathcal{L}(\rho)$ and $\pi, j \models \varphi$;
- $\pi, i \models [\rho]\varphi$ iff for all $j \geq i$ such that $\pi(i,j) \in \mathcal{L}(\rho)$, we have $\pi, j \models \varphi$;

where the relation $\pi(i,j) \in \mathcal{L}(\rho)$ is as follows:

- $\pi(i,j) \in \mathcal{L}(\phi)$ if $j = i+1$, $j \leq \mathit{length}(\pi)$, and $\pi, i \models \phi$ (ϕ propositional);
- $\pi(i,j) \in \mathcal{L}(\varphi?)$ if $j = i$ and $\pi, i \models \varphi$;
- $\pi(i,j) \in \mathcal{L}(\rho_1 + \rho_2)$ if $\pi(i,j) \in \mathcal{L}(\rho_1)$ or $\pi(i,j) \in \mathcal{L}(\rho_2)$;
- $\pi(i,j) \in \mathcal{L}(\rho_1; \rho_2)$ if there exists k, with $i \leq k \leq j$, such that $\pi(i,k) \in \mathcal{L}(\rho_1)$ and $\pi(k,j) \in \mathcal{L}(\rho_2)$;
- $\pi(i,j) \in \mathcal{L}(\rho^*)$ if $j = i$ or there exists k, with $i \leq k \leq j$, such that $\pi(i,k) \in \mathcal{L}(\rho)$ and $\pi(k,j) \in \mathcal{L}(\rho^*)$.

Notice that we have the usual boolean equivalences, and in addition $[\rho]\varphi \equiv \neg\langle\rho\rangle\neg\varphi$. We use the abbreviation *last*, standing for $[\mathit{true}]\mathit{false}$, to denote the last step of the trace.

It is easy to encode LTL$_f$ into LDL$_f$: it suffices to observe that we can express the various LTL$_f$ operators by recursively applying the following translations:

- $\circ\varphi$ translates to $\langle \mathit{true}\rangle\varphi$;
- $\bullet\varphi$ translates to $\neg\langle \mathit{true}\rangle\neg\varphi = [\mathit{true}]\varphi$;
- $\Diamond\varphi$ translates to $\langle \mathit{true}^*\rangle\varphi$;
- $\Box\varphi$ translates to $[\mathit{true}^*]\varphi$;
- $\varphi_1 \,\mathcal{U}\, \varphi_2$ translates to $\langle(\varphi_1?; \mathit{true})^*\rangle\varphi_2$.

It is also easy to encode into LDL$_f$ 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 ρ translates to $\langle\rho\rangle(\mathit{last} \wedge \mathit{end})$.

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

3 LDL$_f$ Automata

We can associate with each LDL$_f$ formula φ an (exponential) NFA A_φ that accepts exactly the traces that make φ true. Here, we provide a simple direct algorithm for computing the NFA corresponding to an LDL$_f$ formula. The correctness of the algorithm is based on the fact that (*i*) we can associate each LDL$_f$ formula φ with a polynomial *alternating automaton on words* (AFW) $\mathcal{A}_\varphi$ that accepts exactly the traces that make φ true, and (*ii*) 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 LDL$_f$ formula. In order to proceed, we put LDL$_f$ formulas φ in negation normal form $\mathit{nnf}(\varphi)$ 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 $\mathit{nnf}(\varphi)$ can be done in linear time. Then, we define an auxiliary function δ (which corresponds directly to the transition function of the AFW $\mathcal{A}_\varphi$) that takes an LDL$_f$ formula ψ in negation normal form and a propositional interpretation Π 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 ψ:

$$
\begin{aligned}
\mathit{delta}(\texttt{"}\phi\texttt{"}, \Pi) &= \begin{cases} \mathit{true} & \text{if } \Pi \models \phi \\ \mathit{false} & \text{if } \Pi \not\models \phi \end{cases} \quad (\phi \text{ prop.})\\
\delta(\texttt{"}\varphi_1 \wedge \varphi_2\texttt{"}, \Pi) &= \delta(\texttt{"}\varphi_1\texttt{"}, \Pi) \wedge \delta(\texttt{"}\varphi_2\texttt{"}, \Pi)\\
\delta(\texttt{"}\varphi_1 \vee \varphi_2\texttt{"}, \Pi) &= \delta(\texttt{"}\varphi_1\texttt{"}, \Pi) \vee \delta(\texttt{"}\varphi_2\texttt{"}, \Pi)\\
\delta(\texttt{"}\langle\phi\rangle\varphi\texttt{"}, \Pi) &= \begin{cases} \texttt{"}\varphi\texttt{"} & \text{if } \mathit{last} \notin \Pi \text{ and } \Pi \models \phi \\ \mathit{false} & \text{otherwise} \quad (\phi \text{ prop.}) \end{cases}\\
\delta(\texttt{"}\langle\psi?\rangle\varphi\texttt{"}, \Pi) &= \delta(\texttt{"}\psi\texttt{"}, \Pi) \wedge \delta(\texttt{"}\varphi\texttt{"}, \Pi)\\
\delta(\texttt{"}\langle\rho_1 + \rho_2\rangle\varphi\texttt{"}, \Pi) &= \delta(\texttt{"}\langle\rho_1\rangle\varphi\texttt{"}, \Pi) \vee \delta(\texttt{"}\langle\rho_2\rangle\varphi\texttt{"}, \Pi)\\
\delta(\texttt{"}\langle\rho_1; \rho_2\rangle\varphi\texttt{"}, \Pi) &= \delta(\texttt{"}\langle\rho_1\rangle\langle\rho_2\rangle\varphi\texttt{"}, \Pi)\\
\delta(\texttt{"}\langle\rho^*\rangle\varphi\texttt{"}, \Pi) &= \begin{cases} \delta(\texttt{"}\varphi\texttt{"}, \Pi) & \text{if } \rho \text{ is test-only} \\ \delta(\texttt{"}\varphi\texttt{"}, \Pi) \vee \delta(\texttt{"}\langle\rho\rangle\langle\rho^*\rangle\varphi\texttt{"}, \Pi) & \text{o/w} \end{cases}\\
\delta(\texttt{"}[\phi]\varphi\texttt{"}, \Pi) &= \begin{cases} \texttt{"}\varphi\texttt{"} & \text{if } \mathit{last} \notin \Pi \text{ and } \Pi \models \phi \\ \mathit{true} & \text{otherwise} \quad (\phi \text{ prop.}) \end{cases}\\
\delta(\texttt{"}[\psi?]\varphi\texttt{"}, \Pi) &= \delta(\texttt{"}\mathit{nnf}(\neg\psi)\texttt{"}, \Pi) \vee \delta(\texttt{"}\varphi\texttt{"}, \Pi)\\
\delta(\texttt{"}[\rho_1 + \rho_2]\varphi\texttt{"}, \Pi) &= \delta(\texttt{"}[\rho_1]\varphi\texttt{"}, \Pi) \wedge \delta(\texttt{"}[\rho_2]\varphi\texttt{"}, \Pi)\\
\delta(\texttt{"}[\rho_1; \rho_2]\varphi\texttt{"}, \Pi) &= \delta(\texttt{"}[\rho_1][\rho_2]\varphi\texttt{"}, \Pi)\\
\delta(\texttt{"}[\rho^*]\varphi\texttt{"}, \Pi) &= \begin{cases} \delta(\texttt{"}\varphi\texttt{"}, \Pi) & \text{if } \rho \text{ is test-only} \\ \delta(\texttt{"}\varphi\texttt{"}, \Pi) \wedge \delta(\texttt{"}[\rho][\rho^*]\varphi\texttt{"}, \Pi) & \text{o/w} \end{cases}
\end{aligned}
$$

Using the auxiliary function δ, we can build the NFA A_φ of an LDL$_f$ formula φ in a forward fashion according to the following algorithm:

algorithm LDL$_f$2NFA
input LTL$_f$ formula φ
output NFA $A_\varphi = (2^{\mathcal{P}}, \mathcal{S}, \{s_0\}, \varrho, \{s_f\})$
$s_0 \leftarrow \{\texttt{"}\varphi\texttt{"}\}$ ▷ single initial state
$s_f \leftarrow \emptyset$ ▷ single final state
$\mathcal{S} \leftarrow \{s_0, s_f\}$, $\varrho \leftarrow \emptyset$
while ($\mathcal{S}$ or ϱ change) **do**
  if ($q \in \mathcal{S}$ and $q' \models \bigwedge_{(\texttt{"}\psi\texttt{"} \in q)} \delta(\texttt{"}\psi\texttt{"}, \Pi)$) **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{q'\}$ ▷ update set of states
    $\varrho \leftarrow \varrho \cup \{(q, \Pi, q')\}$ ▷ update transition relation

In the algorithm LDL$_f$2NFA, states $\mathcal{S}$ of A_φ are sets of atoms, where each atom consists of quoted subformulas of φ. Such sets of atoms are interpreted as conjunctions and in particular $\emptyset$, i.e., the empty conjunction, stands for *true*. Formula $\bigwedge_{(\texttt{"}\psi\texttt{"} \in q)} \delta(\texttt{"}\psi\texttt{"}, \Pi)$ is a positive boolean formula whose atoms are again quoted subformulas of φ. Finally, q' is a set of atoms, consisting again of quoted subformulas of φ, such that, when seen as a propositional interpretation, makes $\bigwedge_{(\texttt{"}\psi\texttt{"} \in q)} \delta(\texttt{"}\psi\texttt{"}, \Pi)$ true. (In fact, we do not need to get all q' such that $q' \models \bigwedge_{(\texttt{"}\psi\texttt{"} \in q)} \delta(\texttt{"}\psi\texttt{"}, \Pi)$, but only the minimal ones.) Notice that trivially we have $(\emptyset, a, \emptyset) \in \varrho$ for every $a \in \Sigma$. The algorithm LDL$_f$2NFA terminates in at most an exponential number of steps, and generates a set of states $\mathcal{S}$ whose size is at most exponential in the size of φ.

Theorem 1. *Let φ be an* LDL$_f$ *formula and A_φ the* NFA *constructed by algorithm* LDL$_f$2NFA*. Then $\pi \models \varphi$ iff $\pi \in \mathcal{L}(A_\varphi)$ for every finite trace π.*

To see why the above theorem holds consider that given a LDL$_f$ formula φ, δ grounded on the subformulas of φ becomes the transition function of the AFW, with initial state

"φ" and no final states, corresponding to φ. LDL$_f$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 δ. Finally, If we want to remove the special proposition *last* from $\mathcal{P}$, we can easily transform the obtained automaton $A_\varphi = (2^{\mathcal{P}}, \mathcal{S}, \{"\varphi"\}, \varrho, \{\emptyset\})$ into the new automaton A'_φ by adding a special state *ended*:

$$A'_\varphi = (2^{\mathcal{P}-\{last\}}, \mathcal{S} \cup \{ended\}, \{"\varphi"\}, \varrho', \{\emptyset, ended\})$$

where $(q, \Pi', q') \in \varrho'$ if and only if $(q, \Pi', q') \in \varrho$ or $(q, \Pi' \cup \{last\}, true) \in \varrho$ and $q' = ended$.

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$_f$ *formula can be checked in PSPACE by nonemptiness of* A_φ *(or* A'_φ*).*

Considering that it is known that satisfiability in LDL$_f$ 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$_f$, 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 $\mathcal{X}$ and $\mathcal{Y}$. We assume to have *no control* on the truth value of the propositions in $\mathcal{X}$, while we can control those in $\mathcal{Y}$. The problem then becomes: *can we control the values of $\mathcal{Y}$ in such a way that for all possible values of $\mathcal{X}$ 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 π, now partitioned in the propositional interpretation X_i for $\mathcal{X}$ and Y_i for $\mathcal{Y}$. Let us denote by $\pi_{\mathcal{X}}|_i$ the interpretation π projected only on $\mathcal{X}$ and truncated at the i-th element (included), i.e., $\pi_{\mathcal{X}}|_i = X_0 X_1 \cdots X_i$. The *realizability problem* checks the existence of a function $f : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ such that for all π with $Y_i = f(\pi_{\mathcal{X}}|_i)$, we have that π satisfies the formula ϕ. 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 $\mathcal{X}$: the function we are looking for only acts on propositions in $\mathcal{Y}$. 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 $\mathcal{X}$ of *uncontrollable propositions*, which are under the control of the environment, and a set $\mathcal{Y}$ 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 $(2^{\mathcal{X}\times\mathcal{Y}})^*$ describing how the controller and environment set their propositions at each round till the game stops. The *specification* of the game is given by a DFA $\mathcal{G}$ of the form $\mathcal{G} = (2^{\mathcal{X}\times\mathcal{Y}}, S, s_0, \delta, F)$, where:

- $2^{\mathcal{X}\times\mathcal{Y}}$ is the alphabet of the game;
- S are the states of the game;
- s_0 is the initial state of the game;
- $\delta : S \times 2^{\mathcal{X}\times\mathcal{Y}} \to S$ is the transition function of the game: given the current state s and a choice of propositions X and Y, respectively for the enviroment 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 $f : (2^{\mathcal{X}})^* \to 2^{Y}$ that, given a history of choices from the environment, decides which propositions $\mathcal{Y}$ to set to true/false next. A *winning strategy* is a strategy $f : (2^{\mathcal{X}})^* \to 2^{Y}$ such that for all π with $Y_i = f(\pi_{\mathcal{X}}|_i)$ we have that π 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* $PreC(\mathcal{E})$ 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 $\mathcal{Y}$ such that for all choices of values for propositions $\mathcal{X}$, game $\mathcal{G}$ progresses to states in $\mathcal{E}$. Formally:

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

Using such a notion, we define the set $Win(\mathcal{G})$ 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 $Win(\mathcal{G})$ as a least-fixpoint, making use of approximates $Win_i(\mathcal{G})$ denoting all states where the controller wins in at most i steps:

- $Win_0(\mathcal{G}) = F$ (the final states of $\mathcal{G}$);
- $Win_{i+1}(\mathcal{G}) = Win_i(\mathcal{G}) \cup PreC(Win_i(\mathcal{G}))$.

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

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

Next, we turn to actually computing the strategy. To do so, we define a *strategy generator* based on the winning sets $Win_i(\mathcal{G})$. This is a nondeterministic transducer, where nondeterminism is of the kind "don't-care": all nondeterministic choices are equally good. The strategy generator $\mathcal{T}_\mathcal{G} = (2^{\mathcal{X}\times\mathcal{Y}}, S, s_0, \varrho, \omega)$ is as follows:

- $2^{\mathcal{X}\times\mathcal{Y}}$ is the alphabet of the trasducer;
- S are the states of the trasducer;
- s_0 is the initial state;
- $\varrho : S \times 2^{\mathcal{X}} \to 2^S$ is the transition function such that

$$\varrho(s, X) = \{s' \mid s' = \delta(s, (X, Y)) \text{ and } Y \in \omega(s)\};$$

- $\omega : S \to 2^{\mathcal{Y}}$ is the output function such that

$$\omega(s) = \{Y \mid \text{ if } s \in Win_{i+1}(\mathcal{G}) - Win_i(\mathcal{G}) \text{ then } \forall X.\delta(s, (X, Y)) \in Win_i(\mathcal{G})\}.$$

The transducer $\mathcal{T}_\mathcal{G}$ 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 φ into an NFA A_φ, e.g., using the algorithm LDL$_f$2NFA presented above. This is an exponential step. Then, we transform the resulting NFA into a DFA A^d_φ, e.g., using the standard determinization algorithm based on the subset construction [Rabin and Scott, 1959]. This will cost us another exponential. At this point we view the resulting DFA A^d_φ as a DFA game, considering exactly the separation between controllable and uncontrollable propositions in the original LDL$_f$/LTL$_f$ specification, and we solve it by computing $Win(A^d_\varphi)$ and the corresponding strategy generator $\mathcal{T}_{A^d_\varphi}$. This is a linear step. The following theorem assesses the correctness of this technique:

Theorem 4. *Realizability of an* LDL$_f$/LTL$_f$ *specification* φ *can be solved by checking whether* $s_0 \in Win(A^d_\varphi)$*. Synthesis can be solved by returning any strategy generated by* $\mathcal{T}_{A^d_\varphi}$*.*

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$_f$/LDL$_f$ 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/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 $\mathcal{P}$ into fluents $\mathcal{F}$ and actions $\mathcal{A}$, adding structural constraints such as $\Box(\bigvee_{a\in\mathcal{A}} a) \wedge \Box(\bigwedge_{a\in\mathcal{A}}(a \to \bigwedge_{b\in\mathcal{A},b\neq a} \neg b))$, 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 φ_{init} involving only fluents, while effects can be modelled as:

$$\Box(\varphi \to \bullet(a \to \psi))$$

where $a \in \mathcal{A}$, while φ and ψ are arbitrary propositional formulas involving only fluents. Such a formula states that performing action a under the conditions denoted by φ brings about the conditions denoted by ψ. A formula like $\Box(\varphi \to \bullet(a \to \varphi))$ corresponds to a frame axiom expressing that φ 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, 2013], by translating the (instantiated) successor state axiom $F(do(a, s)) \equiv \varphi^+(s) \vee (F(s) \wedge \neg\varphi^-(s))$, 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:

$$\Box(\bullet a \to (\bullet F \equiv \varphi^+ \vee (F \wedge \neg\varphi^-))).$$

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 $\varphi_{goal} = \Diamond\phi$. where ϕ is a propositional formula on fluents.

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

$$\begin{aligned}
(\texttt{at end}\ \phi) &\doteq \Diamond(\mathit{last} \wedge \phi)\\
(\texttt{always}\ \phi) &\doteq \Box\phi\\
(\texttt{sometime}\ \phi) &\doteq \Diamond\phi\\
(\texttt{within}\ n\ \phi) &\doteq \textstyle\bigvee_{0\leq i\leq n} \underbrace{\bigcirc\cdots\bigcirc}_{i}\phi\\
(\texttt{hold-after}\ n\ \phi) &\doteq \underbrace{\bigcirc\cdots\bigcirc}_{n+1}\Diamond\phi\\
(\texttt{hold-during}\ n_1\ n_2\ \phi) &\doteq \underbrace{\bigcirc\cdots\bigcirc}_{n_1}(\textstyle\bigwedge_{0\leq i\leq n_2}\underbrace{\bigcirc\cdots\bigcirc}_{i}\phi)\\
(\texttt{at-most-once}\ \phi) &\doteq \Box(\phi \to \phi\,\mathcal{W}\,\neg\phi)\\
(\texttt{sometime-after}\ \phi_1\ \phi_2) &\doteq \Box(\phi_1 \to \Diamond\phi_2)\\
(\texttt{sometime-before}\ \phi_1\ \phi_2) &\doteq (\neg\phi_1 \wedge \neg\phi_2)\,\mathcal{W}(\neg\phi_1 \wedge \phi_2)\\
(\texttt{always-within}\ n\ \phi_1\ \phi_2) &\doteq \Box(\phi_1 \to \textstyle\bigvee_{0\leq i\leq n}\underbrace{\bigcirc\cdots\bigcirc}_{i}\phi_2)
\end{aligned}$$

where ϕ is a propositional formula on fluents, and $\varphi_1\,\mathcal{W}\,\varphi_2 \doteq (\varphi_1\,\mathcal{U}\,\varphi_2 \vee \Box\varphi_1)$ is the so-called *weak until*, which states that φ_1 holds until φ_2 or forever.

Finally, we turn to *procedural constraints*. Typically, these cannot be expressed in LTL$_f$ but they can be expressed in LDL$_f$. 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]:

a	atomic action
$\phi?$	test for a condition
$\delta_1; \delta_2$	sequence
if ϕ **then** δ_1 **else** δ_2	conditional
while ϕ **do** δ	while loop
$\delta_1 \mid \delta_2$	nondeterministic branch
$\pi x.\delta$	nondeterministic choice of argument[2]
δ^*	nondeterministic iteration

One can translate such programs into regular expressions:

$$\begin{array}{rcl} tr(a) & = & true; a? \\ tr(\phi?) & = & \phi? \qquad (\phi \text{ propositional}) \\ tr(\delta_1; \delta_2) & = & tr(\delta_1); tr(\delta_2) \\ tr(\textbf{if}\, \phi\, \textbf{then}\, \delta_1\, \textbf{else}\, \delta_2) & = & \phi?; tr(\delta_1) + \neg\phi?; tr(\delta_2) \\ tr(\textbf{while}\, \phi\, \textbf{do}\, \delta) & = & (\phi?; tr(\delta_1))^*; \neg\phi? \\ tr(\delta_1 \mid \delta_2) & = & tr(\delta_1) + tr(\delta_2) \\ tr(\pi x.\delta) & = & \sum_x tr(\delta(x)) \\ tr(\delta^*) & = & tr(\delta)^* \end{array}$$

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 GOLOG procedural constraints such as: δ must be executed at least once during the plan, as $\langle tr(\delta)\rangle true$; plans for goal ϕ must be an executions of δ, as $\langle tr(\delta)\rangle\phi$; along the plan every time δ is executed ψ must hold, as $[tr(\delta)]\psi$.

Given a $\text{LTL}_f/\text{LDL}_f$ formulas Ψ_{domain} describing the action domain (a transition formula), ϕ_{init} describing the initial state (a propositional formula), $\Phi_{constraints}$ describing trace constraints, and φ_{goal} describing the goal, conditional planning amonts to solving the synthesis problem for the following specification:

$$\phi_{init} \wedge \Psi_{domain} \wedge \Phi_{constraints} \wedge \varphi_{goal}$$

where action propositions $\mathcal{A}$ are controllable and fluent propositions $\mathcal{F}$ are uncontrollable. The same holds if we consider arbitrary temporally extended goals expressed in $\text{LTL}_f/\text{LDL}_f$. In other words, by using the synthesis technique presented here we can generate plans in a very general setting: nondeterministic domains, arbitrary $\text{LTL}_f/\text{LDL}_f$ goals, arbitrary $\text{LTL}_f/\text{LDL}_f$ 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 $\text{LTL}_f/\text{LDL}_f$, 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 $\text{LTL}_f/\text{LDL}_f$ formula corresponding to the domain, initial state, and reachability goal (corresponding to the conditional planning problem) is indeed *deterministic*. So determinization, 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, determinization may be required for more complex $\text{LTL}_f/\text{LDL}_f$ goals or trajectory constraints, since the goals or constrains themselves may give rise to NFAs over traces of propositional interpretations.

[2]Notice that the GOLOG 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.

6 Conclusion

We have studied $\text{LTL}_f/\text{LDL}_f$ 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 $\text{LTL}_f/\text{LDL}_f$ on finite traces is the same as the one for infinite trace, namely, 2EXPTIME-complete. Also the procedure for computing it is analogus [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 espressive 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 $\text{LTL}_f/\text{LDL}_f$ 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 syntesis 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 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

[Alur *et al.*, 2002] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5), 2002.

[Bacchus and Kabanza, 1996] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. In *AAAI*, 1996.

[Bacchus and Kabanza, 2000] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2), 2000.

[Baier and McIlraith, 2006] J Baier and S McIlraith. Planning with First-Order Temporally Extended Goals using Heuristic Search. In *AAAI*, 2006.

[Baier *et al.*, 2008] Jorge A. Baier, Christian Fritz, Meghyn Bienvenu, and Sheila A. McIlraith. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *AAAI*, 2008.

[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), 2012.

[Calvanese *et al.*, 2002] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. Reasoning about actions and planning in LTL action theories. In *KR*, 2002.

[Church, 1963] A. Church. Logic, arithmetics, and automata. In *Proc. International Congress of Mathematicians, 1962*. institut Mittag-Leffler, 1963.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. of IJCAI*, 2013.

[De Giacomo *et al.*, 2010] Giuseppe De Giacomo, Paolo Felli, Fabio Patrizi, and Sebastian Sardiña. Two-player game structures for generalized planning and agent composition. In *Proc. AAAI*, 2010.

[De Giacomo *et al.*, 2014] Giuseppe De Giacomo, Riccardo De Masellis, Marco Grasso, Fabrizio Maria Maggi, and Marco Montali. Monitoring business metaconstraints based on LTL and LDL for finite traces. In *Proc. of BPM*, 2014.

[Fischer and Ladner, 1979] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18, 1979.

[Fogarty *et al.*, 2013] Seth Fogarty, Orna Kupferman, Moshe Y. Vardi, and Thomas Wilke. Profile trees for Büchi word automata, with application to determinization. In *GandALF*, 2013.

[Fritz and McIlraith, 2007] Christian Fritz and Sheila A. McIlraith. Monitoring plan optimality during execution. In *ICAPS*, 2007.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[Gerevini *et al.*, 2009] Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artif. Intell.*, 173(5-6), 2009.

[Harel *et al.*, 2000] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[Jobstmann and Bloem, 2006] Barbara Jobstmann and Roderick Bloem. Optimizations for LTL synthesis. In *FMCAD*, 2006.

[Levesque *et al.*, 1997] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming*, 31, 1997.

[Mazala, 2002] René Mazala. Infinite games. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

[McDermott and others, 1998] Drew McDermott et al. PDDL – the planning domain definition language – version 1.2. Technical report, TR-98-003, Yale Center for Computational Vision and Control, 1998.

[Montali *et al.*, 2010] Marco Montali, Maja Pesic, Wil M. P. van der Aalst, Federico Chesani, Paola Mello, and Sergio Storari. Declarative specification and verification of service choreographies. *ACM Trans. on the Web*, 2010.

[Pesic and van der Aalst, 2006] Maja Pesic and Wil M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proc. of the BPM 2006 Workshops*, volume 4103 of *LNCS*. Springer, 2006.

[Pnueli and Rosner, 1989] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, 1989.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *FOCS*, 1977.

[Rabin and Scott, 1959] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2), April 1959.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[Rintanen, 2004] Jussi Rintanen. Complexity of planning with partial observability. In *ICAPS*, 2004.

[Sun *et al.*, 2012] Yutian Sun, Wei Xu, and Jianwen Su. Declarative choreographies for artifacts. In *Proc. of ICSOC'12*, LNCS. Springer, 2012.

[Vardi, 1996] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*. Springer, 1996.

[Vardi, 2011] M.Y. Vardi. The rise and fall of linear time logic. In *GandALF*, 2011.