Automata-Theoretic Approach to Planning for Temporally Extended Goals

Giuseppe De Giacomo[1] and Moshe Y. Vardi[2]

[1] Dip. Informatica e Sistemistica, Univ. Roma "La Sapienza",
Via Salaria 113, I-00198 Roma, Italy
degiacomo@dis.uniroma1.it

[2] Department of Computer Science, Rice University,
P.O. Box 1892, Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu

Abstract. We study an automata-theoretic approach to planning for temporally extended goals. Specifically, we devise techniques based on nonemptiness of Büchi automata on infinite words, to synthesize sequential and conditional plans in a generalized setting in which we have that: goals are general temporal properties of desired execution; dynamic systems are represented by finite transition systems; incomplete information on the initial situation is allowed; and states are only partially observable. We prove that the techniques proposed are optimal wrt the worst case complexity of the problem. Thanks to the scalability of the nonemptiness algorithms, the techniques presented here promise to be applicable to fairly large systems, notwithstanding the intrinsic complexity of the problem.

1 Introduction

Artificial Intelligence has always been interested in the analysis and synthesis of dynamic systems behavior. In particular, the research area of reasoning about actions has been concerned with representing and reasoning on such systems in order to *analyze* interesting properties of their behavior; the area of planning has instead been concerned with the *synthesis* of devices (*plans*) in order to control the system behavior so as to achieve desired conditions (*goals*). The two areas have developed their research in quite different directions. In reasoning about actions a lot of work has been done in finding ways to represent and reason on dynamic systems and dynamic properties of increasing generality [39, 42, 41]. In the area of planning the focus has been in achieving effectiveness of the planning process (and lately notable results have been obtained [29, 5]), while being contented with limited capabilities both in modeling dynamic system and in the kind of goals considered.

In this paper, we study synthesis of *sequential* and *conditional* plans[1] in a setting which is close to that considered in the area of reasoning about actions.

[1] We call *sequential plans* plans that are sequences of primitive actions, and *conditional plans* plans that include choice points to be resolved by ascertaining given conditions at runtime.

In particular following, in spirit, Reiter's formalization of dynamic systems in the Situation Calculus [39], we assume (i) *incomplete information* on the initial situation (several initial states are compatible with the information available on the initial situation); (ii) *deterministic actions*, that is, performing an action in a state brings about a univocally determined next state; (iii) actions with *conditional effects*, that is, the effects of an action depend on the state in which the action is performed. Also, we allow for *partially observable states*. In other words, the agent can observe only part of the state and hence its choices on the action to perform next may depend only on that part. The kind of goals we consider are *temporally extended goals*, i.e., goals that specify acceptable *sequences of states* as in [2, 27]. This kind of goal subsumes the usual goals expressing *reachability* of desired conditions, as well as generalized goals as *don't-disturb* and *restore* requirements [47]. More generally, complex temporal properties typically used in the specification of processes can be expressed [16, 31, 46]. Observe that as we deal with goals expressing general temporal properties, even sequential plans may in fact involve loops, since goals of this form may require *infinite executions*. Consider, for example, a plan to satisfy the requirement that whenever certain triggering conditions are met within a finite (but undetermined) number of steps, a specified state of affairs must be brought about in which the triggering conditions are met again. The major compromise we accept, in order to get effective planning techniques in the outlined setting, is to restrict our attention to systems that have a *finite number of states*. Although this is a radical simplification wrt [39], it is a compromise that is widely accepted in planning.

Formally, we model dynamic systems as finite *transition systems*. A transition system can be thought as a graph, where *nodes* represent states and are labeled by the part of the state that is observable, while *edges* represent state transitions and are labeled by actions that cause the state transitions. Several representation formalisms in AI are based on transition systems. For example, both STRIPS-like formalisms [18, 3] and Action Languages [22] are formalisms to compactly represent transition systems (of different generality). Furthermore, formalizations based on logics such as the Situation Calculus [34, 39], or Dynamic Logics [40, 13], are also tightly related to transition systems.

As goal specification we adopt automata on infinite words: the desirable traces correspond to the language accepted by the automaton. This way of specifying goals is very close to adopting linear time temporal logic (LTL) [16] as goal specification language, since every LTL-based specification can be translated into an automaton-based one [46].

In such a framework we establish techniques and characterize the worst-case computational complexity of synthesizing sequential and conditional plans by adopting an approach based on the theory of automata on infinite objects (infinite words in our case), an approach that is widely used in verification of hardware and control-intensive (as opposed to data-intensive) software [45, 44].

The rest of the paper is structured as follows. We first introduce transition systems and Büchi automata on infinite words. Then, we study sequential planning when the initial state is unique (we have complete information on the initial situation) and states are fully observable. In Section 4 and Section 5, we study sequential planning and conditional planning when the initial state is not unique and states are only partially observable. Then, we briefly discuss algorithmic techniques, and related works. Finally, we draw some conclusions.

2 Preliminaries

2.1 Transition Systems

A (finite) *transition system* $\mathcal{T}$ is defined as $\mathcal{T} = (W, w_0, Act, R, Obs, \pi)$ where:

- W is the finite set of possible states.
- $W_0 \subseteq W$ is the finite set of possible initial states
- Act is the set of possible actions.
- $R : W \times Act \rightarrow W$ is the transition function (actions are deterministic), i.e., a function that given a state and an action return the next state.[2]
- Obs is the finite set of possible observations, which model the observable part of states.
- $\pi : W \rightarrow Obs$ is the *observability* function, which returns the current observation, i.e., the observable part of the current state.

An *execution* on the transition system is an infinitive sequence of states $w_0, w_1, w_2, \ldots$ s.t. $w_0 \in W_0$ and $w_{i+1} = R(w_i, a)$ for some $a \in Act$. A *trace* is what we can observe of an execution. For example, $\pi(w_0), \pi(w_1), \pi(w_2), \ldots$ is the trace corresponding to the execution $w_0, w_1, w_2, \ldots$. The *observable behavior* of the dynamic system is the set of all possible traces of the transition system.[3]

2.2 Automata on infinite words

Given a finite nonempty alphabet Σ, an *infinite word* is an element of Σ^ω, i.e., an infinite sequence $a_0, a_1, a_n, \ldots$ of symbols from Σ.

A *Büchi automaton* is a tuple $\mathcal{A} = (\Sigma, S, S_0, \rho, F)$ where:

- Σ is the alphabet of the automaton.
- S is the finite set of possible states.
- $S_0 \subseteq S$ is the set of possible initial states.
- $\rho : S \times \Sigma \rightarrow 2^S$ is the transition function of the automaton (the automaton need not to be deterministic).
- $F \subseteq S$ is the set of accepting states.

[2] For simplicity and wlog, we assume that R is a total function. We can model the case in which a transition does not exists by making R return a special dummy state.

[3] This way of describing the observable behavior of a dynamic system corresponds to the so called *linear time view of dynamic system*, and is to be contrasted to the so called *branching time view* –see [16] for a discussion.

The *input words* of $\mathcal{A}$ are infinite words $a_0, a_1, a_2, \ldots \in \Sigma^\omega$. A *run* of $\mathcal{A}$ on a infinite word $a_0, a_1, a_2, \ldots$ is an infinite sequences of states $s_0, s_1, s_2, \ldots \in S^\omega$ s.t. $s_0 \in \mathcal{S}_0$ and $s_{i+1} \in \rho(s_i, a_i)$. A run r is *accepting* iff $lim(r) \cap F \neq \emptyset$, where $lim(r) = \{s \mid s \text{ occurs in } r \text{ infinitely often}\}$. In other word a run is accepting if it gets into F infinitely many times, which in turn means, being F finite, that there is at least one state $s_f \in F$ that is visited infinitely often. The *language* accepted by $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of words for which there is an accepting run.

The *nonemptiness* problem for an automaton is to decide given an automaton $\mathcal{A}$ whether $L(A) \neq \emptyset$, i.e., if the automaton accepts at least one word.

Proposition 1. [46] *The nonemptiness problem for Büchi automata is NLOGSPACE-complete.*

Algorithms for nonemptiness are based on *fair reachability* on graphs. The idea behind the algorithms is best explained by the following three line Prolog implementation:

```
nonempty :- ini(X),cn(X,Y),acc(Y),cn(Y,Y).
cn(X,Y) :- rho(X,A,Y).
cn(X,Y) :- rho(X,A,Z),cn(Z,Y).
```

where `ini` denotes the elements in S_0, `acc` denotes the elements in F, `rho` denotes the relation corresponding to the transition function, and `cn` denotes that two states are connected by a `rho`-chain (`cn` is the transitive closure of `rho`).[4]

In other words an automaton is nonempty if starting from some initial state we can reach an accepting state from where there is a cycle back to itself.

A *nondeterministic algorithm for nonemptiness* can then work as follows: it nondeterministically chooses an initial state x and an accepting state y and then checks that x is connected to y and y is connected to itself. To run the algorithm we only need to store the state y, as well as the current and next states, plus a constant number of control bits. To encode states as bit vectors we need only $O(\log(|S|))$ bits. This gives the NLOGSPACE bound.

A *linear time deterministic algorithm for nonemptiness* is the following (i) decompose the transition graph of the automaton into *maximally strongly connected components (mscc)* (linear cost [11]); (ii) verify that one of the mscc's intersects with F (linear cost).

Büchi automata are widely used in verification to specify properties of dynamic systems [31, 46]. Given transition system representing a dynamic system,

[4] Observe the strong similarity with the following naive algorithm to check plan existence in more traditional approaches (which is in fact *reachability* on graphs):

```
planexis :- ini(X),cn(X,Y),goal(Y).
cn(X,Y) :- result(X,A,Y).
cn(X,Y) :- result(X,A,Z),cn(Z,Y).
```

where `ini` denotes the initial states (typically one), `goal` the states where the goal is satisfied, `result` corresponds to the result function that return a state resulting from executing an action (operator) in the current state, `cn` its transitive closure.

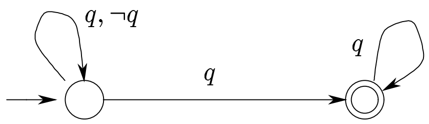

Fig. 1. Automaton for "eventually always q"

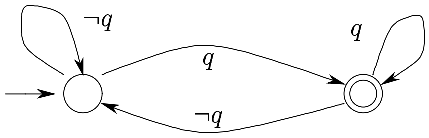

Fig. 2. Automaton for "always eventually q"

the words accepted by the automaton can be put in correspondence with traces of the transitions system that have specified properties. Two examples of such specifications are shown in Figure 1 and Figure 2. The automaton in Figure 1 accepts traces where from a certain point on the property q will hold forever. The automaton in Figure 2, instead, accepts traces where at every point of the trace it is guaranteed that sooner or later a certain property q will hold. More generally any property expressible in propositional linear time temporal logic (LTL) can be expressed as a Büchi automaton, but not vice-versa.[5]

In the following, we will also make use of generalized Büchi automata. A *generalized Büchi automaton* $\mathcal{A}_g = (\Sigma, S, S_0, \rho, \{F_0, \ldots, F_{k-1}\})$ is a variant of Büchi automata which has k sets of accepting states $F_0, \ldots, F_{k-1}$ instead of one, and whose acceptance condition for a run r is $lim(r) \cap F_i \neq \emptyset$ for $i = 0, \ldots, k-1$. Given a generalized Büchi automaton $\mathcal{A}_g = (\Sigma, S, S_0, \rho, \{F_0, \ldots, F_{k-1}\})$, it can be transformed into an equivalent[6] Büchi automaton $\mathcal{A}_b = (\Sigma, S', S'_0, \rho', F')$ where:

- $S' = S \times \{0, \ldots, k-1\}$
- $S'_0 = S_0 \times \{0\}$
- $\rho'((s,i),a) = \rho(s,a) \times \{i\}$ if $s \not\in F_i$, and $\rho'((s,i),a) = \rho(s,a) \times \{(i+1 \, mod \, k)\}$ if $s \in F_i$
- $F' = F_0 \times \{0\}$

3 Planning with complete information

We start our investigation by considering a simplified case. We assume, that we have *complete information* on initial situation and that we have *full observability* on the state. The only kind of plans of interest in this case are sequential ones (sequences of actions), since a conditional plan exists iff a sequential plan does.

[5] There are standard techniques to transform LTL formulas into Büchi automata. The size of the resulting automaton is worst-case exponential wrt the formula [46].

[6] In the sense that $L(\mathcal{A}_g) = L(A_b)$.

We model the dynamic system of interest as a transition system $\mathcal{T} = (W, W_0, Act, R, Obs, \pi)$ where:

- $W_0 \subseteq W$ is a singleton set containing the initial state (which is unique since we are assuming complete information on the initial situation).
- $Obs = W$, and $\pi : W \to Obs$ is simply the identity function (since we are assuming full observability).

Let $\mathcal{A}$ be a Büchi automaton specifying the behavior of the desired executions of the system. Formally, $\mathcal{A} = (Obs, S, S_0, \rho, F)$ where:

- Obs plays the role of the alphabet of the automaton.
- S is the finite set of possible states of the automaton.
- $S_0 \subseteq S$ is the set of possible initial states.
- $\rho : S \times Obs \to 2^S$ is the transition function of the automaton (the automaton need not to be deterministic).
- $F \subseteq S$ is the set of accepting states.

A *plan* p for $\mathcal{T}$ is an infinite sequence of actions $a_0, a_1, a_2, \ldots \in Act^\omega$. The *execution* of p (starting from the initial state w_0) is the infinite sequence of states $w_0, w_1, w_2, \ldots \in W^\omega$ s.t. $w_0 \in W_0$ and $w_{i+1} = R(w_i, a_i)$. The *trace*, $tr(p, w_0)$, of p (starting from the initial state w_0) is the infinite sequence $\pi(w_0), \pi(w_1), \pi(w_2), \ldots$. A plan p *realizes* a specification $\mathcal{A}$ iff $tr(p, w_0) \in L(\mathcal{A})$.

How can we synthesize such a plan? We check for nonemptiness the following Büchi automaton $\mathcal{A}_\mathcal{T} = (Act, S_\mathcal{T}, S_{\mathcal{T}0}, \rho_\mathcal{T}, F_\mathcal{T})$ where:

- Act is the alphabet of the automaton
- $S_\mathcal{T} = S \times W$
- $S_{\mathcal{T}0} = S_0 \times \{w_0\}$
- $(s_j, w_j) \in \rho_\mathcal{T}((s_i, w_i), a)$ iff $w_j = R(w_i, a)$ and $s_j \in \rho(s_i, \pi(w_i))$
- $F_\mathcal{T} = F \times W$

For $\mathcal{A}_\mathcal{T}$ we get the following result:

Theorem 1. *A plan p for $\mathcal{T}$ realizing the specification $\mathcal{A}$ exists iff $L(\mathcal{A}_\mathcal{T}) \neq \emptyset$.*

Notably the nonemptiness algorithm can be easily modified to return a plan if a plan exists. The plan returned always consists of two parts: a sequence arriving to a certain state, and a second sequence that forms a cycle back into that state. Thus, such plans have finite representations.

As an immediate consequence of the construction we get:

Theorem 2. *Planning in the setting above is decidable in NLOGSPACE.*

Proof. The automaton $\mathcal{A}_\mathcal{T}$ can be built on the fly, thus for checking nonemptiness using a nondeterministic algorithm we only need $O(\log(|W|) + \log(|S|))$ bits.

Observe that if we adopt a compact (i.e., logarithmic) representation of the transition system, for example by using propositions to denote states and computing the transitions directly on such propositions,[7],then planning in the above setting becomes PSPACE. This is the complexity of planning in STRIPS [8], which can be seen as a special case of the setting considered here – reachability of a desired state of affairs is the only kind of goal considered in STRIPS; moreover, only certain transition systems are (compactly) representable.

Moreover considering that STRIPS is PSPACE-hard [8], we can conclude:

Theorem 3. *Planning in the setting above is NLOGSPACE-complete (PSPACE-complete wrt a compact representation of $\mathcal{T}$).*

4 Sequential planning with incomplete information

Next we consider the more general case. We assume to have only partial information on the initial situation, and we assume that only part of the state is observable. In this section we consider generating sequential plans, in the next section we turn to conditional plans.

We model the dynamic system of interest as a general transition system $\mathcal{T} = (W, W_0, R, Act, Obs, \pi)$ defined as in Section 2.1. Such a transition system has several initial states $W_0 = \{w_{00}, \ldots, w_{0k-1}\}$, for $k > 1$, reflecting our uncertainty about the initial situation.

As in the previous section we specify the behavior of the desired executions of the system by a Büchi automaton $\mathcal{A}$.

A *plan* p for $\mathcal{T}$ is an infinite sequence of actions $a_0, a_1, a_2, \ldots \in Act^\omega$. The *execution* of p starting from w_{oh} is the infinite sequence of states $w_{0h}, w_{1h}, w_{2h}, \ldots \in W^\omega$ s.t. $w_{0h} \in W_0$ and $w_{i+1h} = R(w_{ih}, a_i)$. The *trace*, $tr(p, w_{0h})$, of the plan p in $\mathcal{T}$ is the infinite sequence $\pi(w_{0h}), \pi(w_{1h}), \pi(w_{2h}), \ldots$. A plan p *realizes* a specification $\mathcal{A}$ iff $tr(p, w_{0h}) \in L(\mathcal{A})$ for $h = 0, \ldots k-1$.

How can we synthesize such a plan? Again we check for nonemptiness a Büchi automaton. This time, however, the construction is slightly more involved.

We first build the generalized Büchi automaton $\mathcal{A}_\mathcal{T} = (Act, S_T, S_{T0}, \rho_\mathcal{T}, F_T)$ where:

- $S_T = S^k \times W^k$
- $S_{T0} = S_0^k \times \{(w_{00}, \ldots, w_{0k-1})\}$
- $(\boldsymbol{s_j}, \boldsymbol{w_j}) \in \rho_\mathcal{T}((\boldsymbol{s_i}, \boldsymbol{w_i}), a)$ iff $w_{jh} = R(w_{ih}, a)$ and $s_{jh} \in \rho(s_{ih}, \pi(w_{ih}))$ for $h = 0, \ldots, k-1$.
- $F_T = \{F \times S^{k-1} \times W^k, \ldots, S^{k-1} \times F \times W^k\}$

[7] We want to stress that assuming that there are formalisms able to represent *every transition system* compactly is not realistic. Indeed, the number of possible transition functions is $|W|^{|W|}$, while the number of transition functions distinguishable with $O(\log(|W|))$ bits is $2^{O(\log(|W|))} = |W|^{O(1)}$. In many cases, however, compact representations of transitions systems do exist, e.g., digital circuits are often described compactly by means of hardware description languages.

From such an automaton we get an equivalent Büchi automaton $\mathcal{A}^b_\mathcal{T} = (Act, S^b_T, S^b_{T0}, \rho^b_\mathcal{T}, F^b_T)$ where:

- $S^b_T = S^k \times W^k \times \{0,\ldots,k-1\}$
- $S^b_{T0} = S^k_0 \times \{(w_{00},\ldots,w_{0k-1})\} \times \{0\}$
- $(\boldsymbol{s_j}, \boldsymbol{w_j}, \ell_j) \in \rho^b_\mathcal{T}((\boldsymbol{s_i}, \boldsymbol{w_i}, \ell_i), a)$ iff $w_{jh} = R(w_{ih}, a)$ and $s_{jh} \in \rho(s_{ih}, \pi(w_{ih}))$ for $h = 0,\ldots,k-1$, and $\ell_j = (\ell_i+1)\, mod\, k$ if $s_{ip_i} \in F$ and $\ell_j = \ell_i$ otherwise.
- $F^b_\mathcal{T} = F \times S^{k-1} \times W^k \times \{0\}$

Theorem 4. *A plan p for $\mathcal{T}$ realizing the specification $\mathcal{A}$ exists iff $L(\mathcal{A}_\mathcal{T}) = L(\mathcal{A}^b_\mathcal{T}) \neq \emptyset$.*

Again the nonemptiness algorithm can be easily modified to return a plan if a plan exists. The plan again consist of two parts: a sequence arriving to a certain state, and a second sequence that forms a cycle back into that state. Note that the possibility of expressing the plan as a finite sequence and a cycle is guaranteed in spite of the uncertainty about the initial state.

Building the automaton $\mathcal{A}_\mathcal{T}$ on the fly, we can check nonemptiness with a nondeterministic algorithm needing $O(k \cdot \log(|W|) + \log(|S|))$ bits, where k is bounded by the size of $|W|$. Considering that NPSPACE=PSPACE, we get:

Theorem 5. *Planning in the setting above is decidable in PSPACE.*

If we adopt a compact representation of the transition system, then planning in the above setting becomes EXPSPACE. What about lower bounds? The following theorem says that our upper bounds are tight.

Theorem 6. *Planning in the setting above is PSPACE-complete (EXPSPACE-complete wrt a compact representation of $\mathcal{T}$).*

Proof. We only need to prove the hardness. Consider that the problem of finding a string that is accepted by the intersection of k deterministic finite state automata over the same alphabet is PSPACE-complete [19]. It is easy to reduce such a problem to planning in the above setting. In particular, the reduction works even with the following two restrictions: (i) $Obs = \mathcal{S}$ and π is the identity function; (ii) the specification automaton denotes an achievement goal. When the transition system is represented compactly, techniques from [25] can be used to lift the PSPACE lower bound to EXPSPACE lower bound.

Note that plan existence in STRIPS with incomplete information on the initial situation is PSPACE-complete [3] – polynomial reduction to the case where the initial situation is completely known. This means that we do pay a price this time in generalizing the setting wrt more traditional approaches. Observe that the reduction used in the proof of Theorem 6 tells us that the increase in the complexity is essentially due to coping with the general form of transition systems once we allow for several possible initial states, and not to the partial observability of states or the more general form of goals considered here.

5 Conditional planning with incomplete information

Now we turn to synthesis of conditional plans in the general setting introduced in the previous section.

Let $\mathcal{T}$ be the transition system and $\mathcal{A}$ be the specification automata both defined as in the previous section.

A *vector plan* $\boldsymbol{p}$ is an infinite sequence of vectors of actions $\boldsymbol{a_0}, \boldsymbol{a_1}, \boldsymbol{a_2}, \ldots \in (Act^k)^\omega$. The *execution*, $ex_h(\boldsymbol{p}, w_{0h})$ of its h-component (starting from the initial state w_{0h}) is the infinite sequence of states $w_{0h}, w_{1h}, w_{2h}, \ldots \in W^\omega$ s.t. $w_{0h} \in W_0$ and $w_{i+1h} = R(w_{ih}, a_{ih})$. The *trace*, $tr_h(\boldsymbol{p}, w_{0h})$, of its h-component is the infinite sequence $\pi(w_{0h}), \pi(w_{1h}), \pi(w_{2h}), \ldots$. A vector plan $\boldsymbol{p}$ *realizes* a specification $\mathcal{A}$ iff $tr_h(\boldsymbol{p}, w_{0h}) \in L(\mathcal{A})$ for $h = 0, \ldots k{-}1$.

A vector plan is not a conditional plan yet, it is simply the parallel compositions of k sequential plans, one for each initial state. *Conditional plans* are vector plans whose actions agree on executions with the same observations.

To formally define conditional plans, we introduce the following notion of equivalence on finite traces. Let $w_{0l}, \ldots, w_{n1}$ and $w_{0m}, \ldots, w_{nm}$ be two finite traces, then

$$\begin{aligned}&\langle w_{0l}, \ldots, w_{n1}\rangle \sim \langle w_{0m}, \ldots, w_{nm}\rangle \text{ iff}\\ &\qquad \langle \pi(w_{0l}), \ldots, \pi(w_{n1})\rangle = \langle \pi(w_{0m}), \ldots, \pi(w_{nm})\rangle.\end{aligned}$$

A *conditional plan* $\boldsymbol{p}$ is a vector plan such that given the executions $w_{0l}, w_{1l}, w_{2l}, \ldots$ and $w_{0m}, w_{1m}, w_{2m}, \ldots$ of a pair of components l and m, we have that $a_{nl} = a_{nm}$ whenever $\langle w_{0l}, \ldots, w_{n1}\rangle \sim \langle w_{0m}, \ldots, w_{nm}\rangle$.

Intuitively a conditional plan can be though of as composed by an (infinite) sequence of *case* instructions that at each step on the base of the observations select how to proceed.

How can we synthesize a conditional plan? We follow the line of the construction in the previous section, checking nonemptiness of a Büchi automaton which this time has Act^k as alphabet. Specifically, we build the generalized Büchi automaton $\mathcal{A}_\mathcal{T} = (Act^k, S_T, S_{T0}, \rho_\mathcal{T}, F_T)$ where:

- $S_T = S^k \times W^k \times \mathcal{E}_k$, where $\mathcal{E}_k$ is the set of equivalence relations on the set $\{0, \ldots, k-1\}$,
- $S_{T0} = S_0^k \times \{(w_{00}, \ldots, w_{0k-1})\} \times \equiv_0$, where $i \equiv_0 j$ iff $w_{0i} = w_{0j}$,
- $(\boldsymbol{s_j}, \boldsymbol{w_j}, \equiv') \in \rho_\mathcal{T}((\boldsymbol{s_i}, \boldsymbol{w_i}), \boldsymbol{a}, \equiv)$ iff
  - $w_{jh} = R(w_{ih}, a_h)$ and $s_{jh} \in \rho(s_{ih}, \pi(w_{ih}))$
  - if $l \equiv m$ then $a_l = a_m$
  - $l \equiv' m$ iff $l \equiv m$ and $\pi(w_{jl}) = \pi(w_{jm})$
- $F_\mathcal{T} = \{F \times S^{k-1} \times W^k \times \mathcal{E}_k, \ldots, S^{k-1} \times F \times W^k \times \mathcal{E}_k\}$

Such automaton can be transformed into a Büchi automaton $\mathcal{A}^b_\mathcal{T}$ as before.

Theorem 7. *A conditional plan* $\boldsymbol{p}$ *for* $\mathcal{T}$ *realizing the specification* $\mathcal{A}$ *exists iff* $L(\mathcal{A}_\mathcal{T}) = L(\mathcal{A}^b_\mathcal{T}) \neq \emptyset$.

The nonemptiness algorithm can again be immediately modified to return a plan if a plan exists. The plan returned again consists of two parts: a sequence arriving to a certain state and a second sequence that forms loop over that state (however, this time the element of the sequences are k-tuples of actions). Observe that even if formally we still deal with vectors of sequential plans, the conditional plan returned can be put in a more convenient form using *case* instructions and loops.

Finally, It is easy to verify that the same complexity bounds of the previous case still hold.[8]

Theorem 8. *Finding a conditional plan in the setting above is PSPACE-complete (EXPSPACE-complete wrt a compact representation of $\mathcal{T}$).*

6 Practical algorithms

The results above show that planning can be reduced to nonemptiness of Büchi automata. Algorithms for checking nonemptiness of Büchi automata have proved to be well suited for scaling up to very large systems [7]. A breakthrough technology has been the use of *symbolic methods* [35], based on the idea being of encoding states as bit vectors, representing sets of states and transitions symbolically as Boolean functions on the encoding, and using ordered binary decision diagrams (OBDDs) to efficiently manipulate Boolean functions [6]. Industrial strength system used in hardware and protocol verification have been developed and used commercially with success [24, 26].

This indicates that notwithstanding the worst-case complexity, it should be actually possible to implement planners even for the most general setting considered here. The experimental results in [9, 10] on adopting symbolic techniques for planning are quite promising (see also [12]). The focus there is on attaining propositional goals when actions can be nondeterministic, but the symbolic techniques can be adapted to our framework.

7 Related work

The need of dealing with incomplete information has often put forward in the area of planning, e.g., [33, 36, 17, 23, 32], as has the need of going beyond goals that specify the reachability of desired state of affair, e.g., [15, 47, 27, 2]. In particular, in [27] a planning setting close to the one considered here is studied, where: dynamic systems are represented by transition systems with a single initial state, nondeterministic actions (which allow for modeling incomplete information), and fully observable states; goals are temporally extended goals, expressed in a variant of LTL that includes a metric over time; plans generated are reactive (conditional) plans. In [2, 4] an analogous planning setting is studied, under the additional assumption of deterministic actions: [2] focuses on generating finite sequential plans only, while [4] considers plans consisting of possibly

[8] Observe that we only need k bits to represent an equivalence relation on $\{0, \ldots, k-1\}$.

infinite sequences of actions. The approach adopted in [27, 2, 4] for obtaining planning algorithms is somewhat ad-hoc (it is based on formula decomposition). The approach proposed here is based on the fundamental relationship between LTL and Büchi automata. The automata-theoretic approach separates the logical and the algorithmic aspects of the planning problem, resulting in clean and optimal algorithms. As we demonstrated, our approach is quite flexible and can be easily adapted to various planning scenarios. Also, neither of [27, 2, 4] studies the intrinsic complexity of the specific planning problem they tackle. In particular, no complexity lower bounds are established.

It is also worth mentioning that there are some similarities between the automata theoretic approach adopted here and approaches to planning based on techniques from operations research, such as MDPs and POMDPs, which are considered quite promising in dealing with incomplete information and generalized goals in stochastic domains [28, 1, 21, 20]. Precise relationships, however, are yet to be established. In particular, to the best of our knowledge, encoding general temporally extended goals, as those expressible with Büchi automata or LTL, as a MDPs/POMDPs rewarding function still remains an open problem.

Finally, automata on infinite objects have already been studied for synthesis of hardware and control-software [37, 43, 30]. Also, automata theoretic techniques have been used in synthesizing discrete controllers [38, 14]. The setting studied here (incomplete information on the initial situation plus deterministic actions), however, which naturally arise in planning and reasoning about actions, is simpler than the general synthesis framework, enabling us to obtain algorithms that are both simper and of lower computational complexity.

8 Conclusions

In this paper we have studied planning for temporally extended goals when incomplete information on the initial situation is available, states are only partially observable, and the number of possible states is finite. We have devised techniques based on nonemptiness of Büchi automata on infinite words, to synthesize sequential and conditional plans, and have characterized the worst case computational complexity. The techniques introduced here in an abstract framework can be easily specialized to a wide range of formalisms for reasoning about actions that are based on transition systems. Moreover, in spite of the high worst-case complexity, the scalability of the practical algorithms involved promises to make the automata-theoretic approach to planning actually feasible even in the most general setting considered here.

Acknowledgments We would like to thank Yves Lesperance, who participated to the discussion that set the stage for this work at ESSLLI'98, and Amedeo Cesta, who gave us precious comments on how to relate our material to traditional planning. This work was partially supported by the NSF grants CCR-9628400 and CCR-9700061, by ASI, by MURST, and by ESPRIT LTR Project No. 22469 DWQ.

References

[1] F. Bacchus, C. Boutilier, and A. Grove. Structured solution methods for non-markovian decision processes. In *Proc. of AAAI'97*, 112–117, 1997.

[2] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. of Math. and AI*, 22:5–27, 1998.

[3] C. Backstrom. Equivalence and tractability results for SAS+ planning. In *Proc. of KR'92*, 1992.

[4] M. Barbeau, F. Kabanza, and R. St-Denis. Synthesizing plant controllers using real-time goals. In *Proc. of IJCAI'95*, 791–798, 1995.

[5] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2):281–300, 1997.

[6] R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[7] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[8] T. Bylander. Tractability and artificial intelligence. *J. of Experimental and Theoretical Computer Science*, 3:171–178, 1991.

[9] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking. In *Proc. of the ECP'97*, 1997.

[10] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. of AAAI'98*, 875–881, 1998.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[12] M. Daniele and P. T. M. Y. Vardi. Strong cyclic planning revisited. submitted, 1999.

[13] G. De Giacomo, L. Iocchi, D. Nardi, and R. Rosati. Moving a robot: the KR&R approach at work. In *Proc. of KR'96*, 1996.

[14] A. Deshpande and P. Varaiya. Sementic tableau for control of PLTL formulae. In *Proc. of the 35th Conf. on Decision and Control*, 2243–2248. IEEE, 1996.

[15] M. Drummond. Situated control rules. In *Proc. of KR'89*, 103–113, 1989.

[16] E. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, 997–1072, 1990.

[17] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. *Proc. of KR'92*, 1992.

[18] R. Fikes and N. J. Nilsson. A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4), 1971.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

[20] H. Geffner. Classical, probabilistic and contingent planning: Three models, one algorithm. In *Proc. AIPS'98 Work. on Planning as Combinatorial Search*, 1998.

[21] H. Geffner and B. Bonet. High-level planning and control with incomplete information using POMDPs. In *Proc. AIPS'98 Work. on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, 1998.

[22] M. Gelfond and V. Lifschitz. Action languages. *Linköping Electronic Articles in Computer and Information Science*, 3(16), 1998.

[23] K. Golden and D. S. Weld. Representing sensing actions: The middle ground revisited. *Proc. of KR'96*, 174–185, 1996.

[24] R. Hardin, Z. Har'el, and R. Kurshan. COSPAN. In *Computer Aided Verification, Proc. 8th Int'l Conf*, LNCS 1102, 423–427. Springer-Verlag, 1996.

[25] D. Harel, O. Kupferman, and M. Y. Vardi. On the complexity of verifying concurrent transition systems. In *Proc. 8th Int'l Conf. on Concurrency Theory*, LNCS 1243, 258–272, Warsaw, July 1997. Springer-Verlag.

[26] G. Holtzmann. Tutorial: proving correctness of concurrent systems with spin. In *Proc. 6th Int'l Conf. on Concurrency Theory*, 453–455. Springer-Verlag, 1995.

[27] F. Kabanza, M. Barbeau, , and R. St-Denis. Planning control rules for reactive agents. *Artif. Intell.*, 95(1):67–113, 1997.

[28] L. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101:99–134, 1998.

[29] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proc. of KR'96*, 1996.

[30] O. Kupferman and M. Y. Vardi. Synthesis with incomplete informatio. In *2nd Int'l Conf. on Temporal Logic*, 91–106, Manchester, July 1997. Kluwer Academic Publishers.

[31] R. Kurshan. *Computer Aided Verification of Coordinating Processes.* Princeton Univ. Press, 1994.

[32] H. J. Levesque. What is planning in presence of sensing? In *Proc. of AAAI'96*, 1139–1149. AAAI Press/The MIT Press, 1996.

[33] Z. Manna and R. J. Waldinger. How to clear a block: A theory of plans. *J. of Automated Reasoning*, 3(4), 1987.

[34] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[35] K. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[36] E. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proc. of KR'89*, 324–332, 1989.

[37] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, Austin, January 1989.

[38] P. Ramadge and W. Wonham. The control of discrete event systems. *Proc. of IEEE*, 77(1):81–98, Jan. 1989.

[39] R. Reiter. *Knowledge in Action: Logical Foundation for Describing and Implementing Dynamical Systems.* 1998. In preparation.

[40] S. J. Rosenschein. Plan synthesis: A logical perspective. In *Proc. of IJCAI'81*, 331–337, 1981.

[41] E. Sandewall. *Features and Fluents. The Representation of Knowledge about Dynamical Systems. Volume I.* Oxford University Press, 1994.

[42] M. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia.* The MIT Press, 1997.

[43] M. Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In P. Wolper, editor, *Computer Aided Verification, Proc. 7th Int'l Conf.*, LNCS 939, 267–292. Springer-Verlag, Berlin, 1995.

[44] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, LNCS 1043, 238–266. Springer-Verlag, Berlin, 1996.

[45] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, 322–331, Cambridge, June 1986.

[46] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[47] D. S. Weld and O. Etzioni. The first law of robotics (a call to arms). In *Proc. of AAAI'94*, 1042–1047, 1994.