Synthesizing Agent Protocols From LTL Specifications Against Multiple Partially-Observable Environments

Giuseppe De Giacomo and **Paolo Felli**
Sapienza Università di Roma, Italy
{degiacomo,felli}@dis.uniroma1.it

Alessio Lomuscio
Imperial College London, UK
a.lomuscio@imperial.ac.uk

Abstract

We consider the problem of synthesizing an agent protocol satisfying LTL specifications for multiple, partially-observable environments. We present a sound and complete procedure for solving the synthesis problem in this setting and show it is computationally optimal from a theoretical complexity standpoint. While this produces perfect-recall, hence unbounded, strategies we show how to transform these into agent protocols with bounded number of states.

Introduction

A key component of any intelligent agent is its ability of reasoning about the actions it performs in order to achieve a certain goal. If we consider a single-agent interacting with an environment, a natural question to ask is the extent to which an agent can derive a plan to achieve a given goal. Under the assumption of full observability of the environment, the methodology of LTL synthesis enables the automatic generation, e.g., through a model checker, of a set of rules for the agent to achieve a goal expressed as an LTL specification. This is a well-known decidable setting but one that is 2EXPTIME-complete (Pnueli and Rosner 1989; Kupferman and Vardi 2000) due to the required determinisation of non-deterministic Büchi automata. Solutions that are not complete but computationally attractive have been put forward (Harding, Ryan, and Schobbens 2005). Alternative approaches focus on subsets of LTL, e.g., GR(1) formulas as in (Piterman, Pnueli, and Sa'ar 2006).

Work in AI on planning has of course also addressed this issue albeit from a rather different perspective. The emphasis here is most often on sound but incomplete heuristics that are capable of generating effective plans on average. Differently from main-stream synthesis approaches, a well-explored assumption here is that of partial information, i.e., the setting where the environment is not fully observable by the agent. Progress here has been facilitated by the relatively recent advances in the efficiency of computation of Bayesian expected utilities and Markov decision processes. While these approaches are attractive, differently from work in LTL-synthesis, they essentially focus on goal reachability (Bonet and Geffner 2009).

An automata-based approach to planning for full-fledged LTL goals covering partial information was put forward in (De Giacomo and Vardi 1999) where an approach based on non-emptiness of Büchi-automata on infinite words was presented. An assumption made in (De Giacomo and Vardi 1999) is that the agent is interacting with a single deterministic environment which is only partially observable. It is however natural to relax this assumption, and study the case where an agent has the capability of interacting with multiple, partially-observable, environments sharing a common interface. A first contribution in this direction was made in (Hu and De Giacomo 2011) which introduced generalized planning under multiple environments for reachability goals and showed that its complexity is EXPSPACE-complete. The main technical results of this paper is to give a sound and complete procedure for solving the generalized planning problem for LTL goals, within the same complexity bound.

A further departure from (Hu and De Giacomo 2011) is that we here ground the work on interpreted systems (Fagin et al. 1995; Parikh and Ramanujam 1985), a popular agent-based semantics. This enables us to anchor the framework to the notion of *agent protocol*, i.e., a function returning the set of possible actions in a given local state, thereby permitting implementations on top of existing model checkers (Gammie and van der Meyden 2004; Lomuscio, Qu, and Raimondi 2009). As as already remarked in the literature (van der Meyden 1996), interpreted systems are particularly amenable to incorporating observations and uncertainty in the environment. The model we pursue here differentiates between visibility of the environment states and observations made by the agent (given what is visible). Similar models have been used in the past, e.g, the VSK model discussed in (Wooldridge and Lomuscio 2001). Differently from the VSK model, here we are not concerned in epistemic specifications nor do we wish to reason at specification level about what is visible, or observable. The primary aim, instead, is to give sound and complete procedures for solving the generalized planning problem for LTL goals.

With a formal notion of agent, and agent protocol at hand, it becomes natural to distinguish several ways to address generalized planning for LTL goals, namely:

- *State-based solutions*, where the agent has the ability of choosing the "right" action toward the achievement of the LTL goal, among those that its protocol allows, exploiting the current observation, but avoiding memorizing previous observations. So the resulting plan is essentially a *state-based strategy*, which can be encoded in the agent protocol itself. While this solution is particularly simple, is also often too restrictive.

- *History-based solutions*, where the agent has the ability of remembering all observations made in the past, and use them to decide the "right" action toward the achivement of the LTL goal, again among those allowed by its protocol. In this case we get a *perfect-recall strategy*. These are the most general solutions, though in principle such solutions could be infinite and hence unfeasible. One of the key result of this paper however is that if a perfect-recall strategy exists, then there exist one which can be represented with a finite number of states.
- *Bounded solutions*, where the agent decides the "right" course of actions, by taking into account only a fragment of the observation history. In this case the resulting strategies can be encoded as a new agent protocol, still compatible with the original one, though allowing the internal state space of the original agent to grow so as to incorporate the fragment of history to memorize. Since we show that if a perfect-recall strategy exists, there exist one that is finite state (and hence makes use only of a fragment of the history), we essentially show that wlog we can restrict our attention to bounded solution (for a suitable bound) and incorporate such solutions in the agent protocol.

The rest of the paper is organized as follows. First we give the formal framework for our investigations, and we look at state-based solutions. Then, we move to history based solutions and prove the key technical results of our paper. We give a sound, complete, and computationally optimal (wrt worst case complexity) procedure for synthesizing perfect-recall strategies from LTL specification. Moreover, we observe that the resulting strategy can be represented with finite states. Then, we show how to encode such strategies (which are in fact bounded) into agent protocol. Finally, we briefly look at an interesting variant of the presented setting where the same results hold.

Framework

Much of the literature on knowledge representation for multi-agent systems employs modal temporal-epistemic languages defined on semantic structures called *interpreted systems* (Fagin et al. 1995; Parikh and Ramanujam 1985). These are refined versions of Kripke semantics in which the notions of computational states and actions are given prominence. Specifically, all the information an agent has at its disposal (variables, facts of a knowledge base, observations from the environment, etc.) is captured in the *local state* relating to the agent in question. *Global states* are tuples of local states, each representing an agent in the system as well as the environment. The environment is used to represent information such as messages in transit and other components of the system that are not amenable to an intention-based representation.

An interesting case is that of a single agent interacting with an environment. This is not only interesting in single-agent systems, or whenever we wish to study the single-agent interaction, but it is also a useful abstraction in loosely-coupled systems where all of the agents' interactions take place with the environment. Also note that the modular reasoning approach in verification focuses on the single agent case in which the environment encodes the rest of the system and its potential influence to the component under analysis. In this and other cases it is useful to model the system as being composed by a single agent only, but interacting with multiple environments, each possibly modelled by a different finite-state machine.

With this background we here pursue a design paradigm enabling the refinement of a generic agent program following a planning synthesis procedure given through an LTL specification, such that the overall design and encoding is not affected nor invalidated. We follow the interpreted system model and formalise an agent as being composed of *(i)* a set of local states, *(ii)* a set of actions, *(iii)* a *protocol* function specifying which actions are available in each local state, *(iv)* an observation function encoding what perceptions are being made by the agent, and *(v)* a local evolution function. An environment is modelled similarly to an agent; the global transition function is defined on the local transition functions of its components, i.e., the agent's and the environment's.

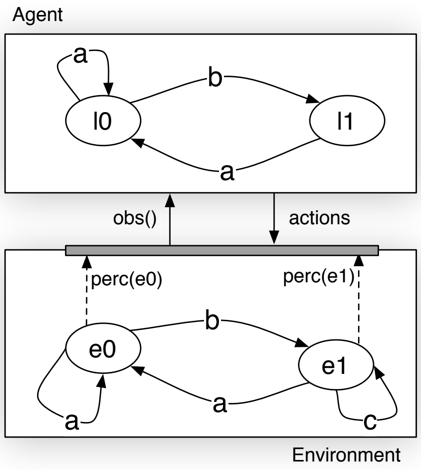


Figure 1: Interaction between Ag and Env

We refer to Figure 1 for a pictorial description of our setting in which the agent is executing actions on the environment, which, in turn, respond to these actions by changing its state. The observations of the agent depend on what perceived of the states of the environment. Notice that environments are only partially observable through what perceivable of their states. So two different environments may give rise to the same observations in the agent.

Environment. An *environment* is a tuple $Env = \langle E, Act_e, Perc, perc, \delta, e_0 \rangle$ such that:

- $E = \{e_0, \ldots\}$ is a finite set of local states of the environment;

- Act_e is the alphabet of the environment's actions;
- $Perc$ is the alphabet of perceptions;
- $perc : E \to Perc$ is the perceptions (output) function;
- $\delta : E \times Act_e \to E$ is a (partial) transition function;
- $e_o \in E$ is the initial state.

Notice that, as in classical planning, such an environment is deterministic.

A *trace* is a sequence $\tau = e_0\alpha_1 e_1 \alpha_2 \dots \alpha_n e_n$ of environment's states such that $e_{i+1} = \delta(e_i, \alpha_{i+1})$ for each $0 \leq i < n$. The corresponding *perceivable* trace is the trace obtained by applying the perception function: $perc(\tau) = perc(e_0), \dots, perc(e_n)$.

Similarly, an agent is represented as a finite machine, whose state space is obtained by considering the agent's internal states, called *configurations*, together with all additional information the agent can access, i.e., the observations it makes. We take the resulting state space as the agent's *local states*.

Agent. An *agent* is a tuple $Ag = \langle Conf, Act_a, Obs, obs, L, poss, \delta_a, c_0 \rangle$ where:

- $Conf = \{c_0, \dots\}$ is a finite set of the agent's configurations (internal states);
- Act_a is the finite set of agent's actions;
- Obs is the alphabet of observations the agent can make;
- $obs : Perc \to Obs$ is the observation function;
- $L = Conf \times Obs$ is the set of agent's local states;
- $poss : L \to \wp(Act_a)$ is a protocol function;
- $\delta_a : L \times Act_a \to Conf$ is the (partial) transition function;
- $c_0 \in Conf$ is the initial configuration. A local state $l = \langle c, o \rangle$ is called *initial* iff $c = c_0$.

Agents defined as above are deterministic: given a local state l and an action α, there exists a unique next configuration $c' = \delta_a(l, \alpha)$. Nonetheless, observations introduce nondeterminism when computing the new local state resulting from action execution: executing action α in a given local state $l = \langle c, o \rangle$ results into a new local state $l' = \langle c', o' \rangle$ such that $c' = \delta_a(l, \alpha)$ and o' is the new observation, which can not be foreseen in advance.

Consider the agent and the environment depicted in Figure 1. Assume that the agent is in its initial configuration c_0 and that the current environment's state is e_0. Assume also that $obs(perc(e_0)) = o$, i.e., the agent receives observation o. Hence, the current local state is $l_0 = \langle c_0, o \rangle$. If the agent performs action b (with $b \in poss(l_0)$), the agent moves from configuration c_0 to configuration $c_1 = \delta_a(\langle c_0, o \rangle, b)$. At the same time, the environment changed its state from e_0 to e_1, so that the new local state is $l_1 = \langle c_1, o' \rangle$, where $o' = obs(perc(e_1))$.

Notice that the protocol function is not defined with respect to the transition function, i.e., according to transitions available to the agent. In fact, we can imagine an agent having its own behaviours, in terms of transitions defined over configurations, that can be constrained according to some protocol, which can in principle be modified or substituted. Hence, we say that a protocol is *compatible* with an agent iff it is compatible with its transition function, i.e., $\alpha \in poss(\langle c, o \rangle) \to \exists c' \in Conf \mid \delta_a(\langle c, o \rangle, \alpha) = c'$. Moreover, we say that a protocol $poss$ is an *action-deterministic protocol* iff it always returns a singleton set, i.e., it allows only a single action to be executed for a given local state. Finally, an agent is *nonblocking* iff it is equipped with a compatible protocol function $poss$ and for each sequence of local states $l_0\alpha_1 l_1 \alpha_2 \dots \alpha_n l_n$ such that $l_i = \langle c_i, o_i \rangle$ and $\alpha_{i+1} \in poss(\langle c_i, o_i \rangle)$ for each $0 \leq i < n$, we have $poss(l_n) \neq \emptyset$. So, an agent is nonblocking iff it has a compatible protocol function which always provides a nonempty set of choices for each local state that is reachable according to the transition function and the protocol itself.

Finally, given a perceivable trace of an environment Env, the *observation history* of an agent Ag is defined as the sequence obtained by applying the observation function: $obs(perc(\tau)) = obs(perc(e_0)), \dots, obs(perc(e_n))$. Given one such history $h \in Obs^*$, we denote with $last(h)$ its latest observation: $last(h) = obs(perc(e_n))$.

LTL Specifications against Multiple Environments

In this paper, we consider an *agent Ag* and a *finite set $\mathcal{E}$ of environments*, all sharing common actions and the same perception alphabet. Such environments are indistinguishable by the agent, in the sense that the agent is not able to identify which environment it is actually interacting with, unless through observations. The problem we address is thus to synthesize a (or customize the) agent protocol so as to fulfill a given LTL specification (or goal) in all environments. A planning problem for an LTL goal is a triple $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G} \rangle$, where Ag is an agent, $\mathcal{E}$ an environment set, and $\mathcal{G}$ an LTL goal specification. This setting is that of *generalized planning* (Hu and De Giacomo 2011), extended to deal with *long-running goals*, expressed as arbitrary LTL formulae. This is also related to *planning for* LTL *goals* under partial observability (De Giacomo and Vardi 1999).

Formally, we call *environment set* a finite set of environments $\mathcal{E} = \{Env_1, \dots, Env_k\}$, with $Env_i = \langle E_i, Act_{ei}, Perc, perc_i, \delta_i, e_{0i} \rangle$. Environments share the same alphabet $Perc$ of the agent Ag. Moreover the Ag's actions must be executable in the various environments: $Act_a \subseteq \bigcap_{i=1,\dots,k} Act_{ei}$. This is because we are considering an agent acting in environments with similar "interface".

As customary in verification and synthesis (Baier and Katoen 2008), we represent an LTL goal with a Büchi automaton[1] $\mathcal{G} = \langle Perc, G, g_0, \gamma, G^{acc} \rangle$, describing the desired behaviour of the system in terms of perceivable traces, where:

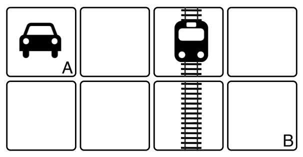
- $Perc$ is the finite alphabet of perceptions, taken as input;
- G is a finite set of states;
- g_0 is the initial state;

[1]In fact, while any LTL formula can be translated into a Büchi automaton, the converse is not true. Our results hold for any goal specified as a Büchi automaton, though for ease of exposition we give them as LTL.

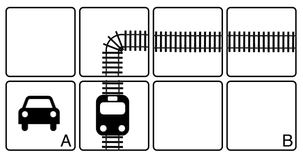
- $\gamma : G \times Perc \to 2^G$ is the transition function;
- $G^{acc} \subseteq G$ is the set of accepting states.

A run of $\mathcal{G}$ on an input word $w = p_0, p_1, \ldots \in Perc^\omega$ is an infinite sequence of states $\rho = g_0, g_1, \ldots$ such that $g_i \in \gamma(g_{i-1}, p_i)$, for $i > 0$. Given a run ρ, let $inf(\rho) \subseteq G$ be the set of states occurring infinitely often, i.e., $inf(\rho) = \{g \in G \mid \forall i\ \exists j > i \text{ s.t. } g_j = g\}$. An infinite word $w \in Perc^\omega$ is accepted by $\mathcal{G}$ iff there exists a run ρ on w such that $inf(\rho) \cap G^{acc} \neq \emptyset$, i.e., at least one accepting state is visited infinitely often during the run. Notice that, since the alphabet $Perc$ is shared among environments, the same goal applies to all of them. As we will see later, we can characterize a variant of this problem by considering the environment's internal states as $\mathcal{G}$'s alphabet.

Example 1. Consider a simple environment Env_1 constituted by a grid of 2x4 cells, each denoted by e_{ij}, a train, and a car. A railroad crosses the grid, passing on cells e_{13}, e_{23}. Initially, the car is in e_{11} and the train in e_{13}. The car is controlled by the agent, whereas the train is a moving obstacle moving from e_{13} to e_{23} to e_{13} again and so on. The set of actions is $Act_{e1} = \{goN, goS, goE, goW, wait\}$. The train and the car cannot leave the grid, so actions are allowed only when feasible. The state space is then $E_1 = \{e_{11}, \ldots, e_{24}\} \times \{e_{13}, e_{23}\}$, representing the positions of the car and the train. We consider a set of perceptions $Perc = \{\texttt{posA}, \texttt{posB}, \texttt{danger}, \texttt{dead}, \texttt{nil}\}$, and a function $perc_1$ defined as follows: $perc_1(\langle e_{11}, e_t\rangle) = \texttt{posA}$, $perc_1(\langle e_{24}, e_t\rangle) = \texttt{posB}$, $perc_1(\langle e_{13}, e_{23}\rangle) = perc_1(\langle e_{23}, e_{13}\rangle) = \texttt{danger}$ and $perc_1(\langle e_{13}, e_{13}\rangle) = perc_1(\langle e_{23}, e_{23}\rangle) = \texttt{dead}$. $perc_1(\langle e_c, e_t\rangle) = \texttt{nil}$ for any other state.



(a) Env_1



(b) Env_2

Figure 2: Environments Env_1 and Env_2

We consider a second environment Env_2 similar to Env_1 as depicted in Figure 2b. We skip details about its encoding, as it is analogous to Env_1.

Then, we consider a third environment Env_3 which is a variant of the Wumpus world (Russell and Norvig 2010), though sharing the interface (in particular they all share perceptions and actions) with the other two. Following the same convention used before, the hero is initially in cell e_{11}, the Wumpus in e_{31}, gold is in e_{34} and the pit in e_{23}. The set of actions is $Act_{e3} = \{goN, goS, goE, goW, wait\}$. Recall that the set of perceptions $Perc$ is instead shared with Env_1 and Env_2. The state space is $E_3 = \{e_{11}, \ldots, e_{34}\}$, and the function $perc_3$ is defined as follows: $perc_3(e_{11}) = \texttt{posA}$, $perc_3(e_{34}) = \texttt{posB}$, $perc_3(e_{23}) = perc_3(e_{31}) = \texttt{dead}$, $perc_3(e) = \texttt{danger}$ for $e \in \{e_{13}, e_{21}, e_{22}, e_{24}, e_{32}, e_{33}\}$, whereas $perc_3(e) = \texttt{nil}$ for any other state. This example allows us to make some observation about our framework. Consider first the perceptions $Perc$. They are intended to represent signals coming from the environment, which is modeled as a "black box". If we could distinguish between perceptions (instead of having just a $\texttt{danger}$ perception), we would be able to identify the current environment as Env_3, and solve such a problem separately. Instead, in our setting the perceptions are not informative enough to discriminate environments (or the agent is not able to observe them); so all environments need to be considered together. Indeed, Env_3 is similar to Env_1 and Env_2 at the interface level, and it is attractive to try to synthesize a single strategy to solve them all. In some sense, crashing in Env_1 or Env_2 corresponds to falling into the pit or being eaten by the Wumpus; the same holds for danger states with the difference that perceiving $\texttt{danger}$ in Env_1 or Env_2 can not be used to prevent an accident.

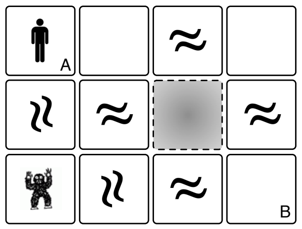


Figure 3: Environment Env_3

Notice that in our framework we design the environments without taking into account the agent Ag that will interact with them. Likewise, the same holds when designing Ag. Indeed, agent Ag is encoded as an automaton with a single configuration c_0, and all actions being loops. In particular, let $Act_a = Act_{ei}$, $i = 1, 2, 3$. Notice also that, by suitably defining the observation function obs, we can model the agent's sensing capabilities, i.e., its ability to observe perceptions coming from the environment. Suppose that Ag can sense perceptions $\texttt{posA}, \texttt{danger}, \texttt{dead}$, but it is unable to sense $\texttt{posB}$, i.e., its sensors can not detect such signal. To account for this, it is enough to consider the observation function obs as a "filter" (i.e. $Obs \subseteq Perc$), such that $Obs = \{\texttt{posA}, \texttt{danger}, \texttt{dead}, \texttt{nil}\}$ and $obs(\texttt{posA}) = \texttt{posA}$, $obs(\texttt{danger}) = \texttt{danger}$, $obs(\texttt{dead}) = \texttt{dead}$, and $obs(\texttt{nil}) = obs(\texttt{posB}) = \texttt{nil}$. Since Ag is filtering away perception $\texttt{posB}$, the existence of a strategy does not imply that agent Ag is actually able to recognize that it has achieved its goal. Notice that this does not affect the solution, nor its executability, in any way. The goal is achieved irrespective of what the agent can observe.

Moreover, Ag has a "safe" protocol function $poss$ that allows all moving actions to be executed in any possible local state, but prohibits it to perform $wait$ if the agent is receiving the observation $\texttt{danger}$: $poss(\langle c_0, o\rangle) = Act_a$ if $o \neq \texttt{danger}$, $Act_a \setminus \{wait\}$ otherwise.

Finally, let $\mathcal{G}$ be the automaton corresponding to the LTL formula $\phi_{\mathcal{G}} = (\Box\Diamond\texttt{posA}) \wedge (\Box\Diamond\texttt{posB}) \wedge \Box\neg\texttt{dead}$ over the perception alphabet, constraining perceivable traces such that the controlled objects (the car / the hero) visit positions

A and B infinitely often. ■

State-Based Solutions

To solve the synthesis problem in the context above, the first solution that we analyze is based on customizing the agent to achieve the LTL goal, by *restricting the agent protocol* while keeping the same set of local states. We do this by considering so called state-based strategies (or plans) to achieve the goal. We call a strategy for an agent Ag *state-based* if it can be expressed as a (partial) function

$$\sigma_p : (Conf \times Obs) \to Act_a$$

For it to be acceptable, a strategy also needs to be *allowed* by the protocol: it can only select available actions, i.e., for each local state $l = \langle c, o\rangle$ we have to have $\sigma_p(l) \in poss(l)$.

State-based strategies do not exploit an agent's memory, which, in principle, could be used to reduce its uncertainty about the environment by remembering observations from the past. Exploiting this memory requires having the ability of extending its configuration space, which at the moment we do not allow (see later). In return, these state-based strategies can be synthesized by just taking into account all allowed choices the agent has in each local state (e.g., by exhaustive search, possibly guided by heuristics). The advantage is that to meet its goal, the agent Ag does not need any modification to its configurations, local states and transition function, since only the protocol is affected. In fact, we can see a strategy σ_p as a restriction of an agent's protocol yielding an action-deterministic protocol $\overline{poss}$ derived as follows:

$$\overline{poss}(\langle c, o\rangle) = \begin{cases} \{\alpha\}, & \text{iff } \sigma_p(c, o) = \alpha \\ \emptyset, & \text{if } \sigma_p(c, o) \text{ is undefined} \end{cases}$$

Notice that $\overline{poss}$ is then a total function. Notice also that agent $\overline{Ag}$ obtained by substituting the protocol function maintains a protocol compatible with the agent transition function. Indeed, the new allowed behaviours are a subset of those permitted by original agent protocol.

Example 2. Consider again Example 1. No state-based solution exists for this problem, since selecting the same action every time the agent is in a given local state does not solve the problem. Indeed, just by considering the local states we can not get any information about the train's position, and we would be also bound to move the car (the hero) in the same direction every time we get the same observation (agent Ag has only one configuration c_0). Nonetheless, observe that if we could keep track of past observations when selecting actions, then a solution can be found.

History-Based Solutions

We turn to strategies that can take into account past observations. Specifically, we focus on strategies (plans) that may depend on the entire unbounded observation history. These are often called perfect-recall strategies.

A *(perfect-recall) strategy* for $\mathcal{P}$ is a (partial) function

$$\sigma : Obs^* \to Act_a$$

that, given a sequence of observations (the *observation history*), returns an action. A strategy σ is *allowed* by Ag's protocol iff, given any observation history $h \in Obs^*$, $\sigma(h) = \alpha$ implies $\alpha \in poss(\langle c, last(h)\rangle)$, where c is the current configuration of the agent. Notice that, given an observation history, the current configuration can be always reconstructed by applying the transition function of Ag, starting from initial configuration c_0. Hence, a strategy σ is a solution of the problem $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G}\rangle$ iff it is allowed by Ag and it generates, for each environment Env_i, an infinite trace $\tau = e_{0i}\alpha_1 e_{1i}\alpha_2 \ldots$ such that the corresponding perceivable trace $perc(\tau)$ satisfies the LTL goal, i.e., it is accepted by the corresponding Büchi automaton.

The technique we propose is based on previous automata theoretic approaches. In particular, we extend the technique for automata on infinite strings presented in (De Giacomo and Vardi 1999) for partial observability, to the case of a finite set of environments, along the lines of (Hu and De Giacomo 2011). The crucial point is that we need both the ability of simultaneously dealing with LTL goals and with multiple environments. We build a generalized Büchi automaton that returns sequences of *action vectors* with one component for each environment in the environment set $\mathcal{E}$. Assuming $|\mathcal{E}| = k$, we arbitrarily order the k environments, and consider sequences of action vectors of the form $\vec{a} = \langle a_1, \ldots, a_k\rangle$, where each component specifies one operation for each environment. Such sequences of action vectors correspond to a strategy σ, which, however, must be *executable*: for any pair $i, j \in \{1, \ldots, k\}$ and observation history $h \in Obs^*$ such that both σ_i and σ_j are defined, then $\sigma_i(h) = \sigma_j(h)$. In other words, if we received the same observation history, the function select the same action. In order to achieve this, we keep an *equivalence relation* $\equiv \subseteq \{1, \ldots, k\} \times \{1, \ldots, k\}$ in the states of our automaton. Observe that this equivalence relation has correspondences with the epistemic relations considered in epistemic approaches (Jamroga and van der Hoek 2004; Lomuscio and Raimondi 2006; Pacuit and Simon 2011).

We are now ready to give the details of the automata construction. Given a set of k environments $\mathcal{E}$ with $Env_i = \langle E_i, Act_{ei}, Perc, perc_i, \delta_i, e_{0i}\rangle$, an agent $Ag = \langle Conf, Act_a, Obs, obs, L, poss, \delta_a, c_0\rangle$ and goal $\mathcal{G} = \langle Perc, G, g_0, \gamma, G^{acc}\rangle$, we build the generalized Büchi automaton $\mathcal{A}_\mathcal{P} = \langle Act_a^k, W, w_0, \rho, W^{acc}\rangle$ as follows:

- $Act_a^k = (Act_a)^k$ is the set of k-vectors of actions;
- $W = E^k \times Conf^k \times G^k \times \wp(\equiv)$;[2]
- $w_0 = \langle e_{10}, \ldots, e_{k0}, c_0, \ldots, c_0, g_0, \ldots, g_0, \equiv_0\rangle$ where
  $i \equiv_0 j$ iff $obs(perc_i(e_{i0})) = obs(perc_j(e_{j0}))$;
- $\langle \vec{e}', \vec{c}', \vec{g}', \equiv'\rangle \in \rho(\langle \vec{e}, \vec{c}, \vec{g}, \equiv\rangle, \vec{\alpha})$ iff
  - if $i \equiv j$ then $\alpha_i = \alpha_j$;
  - $e'_i = \delta_i(e_i, \alpha_i)$;
  - $c'_i = \delta_a(l_i, \alpha_i) \wedge \alpha_i \in poss(l_i)$
    where $l_i = \langle c_i, obs(perc_i(e_i))\rangle$;
  - $g'_i = \gamma(g_i, perc_i(e_i))$;

[2]We denote by $\wp(\equiv)$ the set of all possible equivalence relations $\equiv \subseteq \{1, \ldots, k\} \times \{1, \ldots, k\}$.

- $i \equiv' j$ iff $i \equiv j \wedge obs(perc_i(e'_i)) = obs(perc_j(e'_j))$.

- $W^{acc} = \{$
  $E^k \times Conf^k \times G^{acc} \times G \times \ldots \times G \times \equiv$, ... ,
  $E^k \times Conf^k \times G \times \ldots \times G \times G^{acc} \times \equiv \}$

Each automaton state $w \in W$ holds information about the internal state of each environment, the corresponding current goal state, the current configuration of the agent for each environment, and the equivalence relation. Notice that, even with fixed agent and goal, we need to duplicate their corresponding components in each state of $\mathcal{A}_\mathcal{P}$ in order to consider all possibilities for the k environments. In the initial state w_0, the environments, the agent and the goal automaton are in their respective initial state. The initial equivalence relation $\equiv_0$ is computed according to the observation provided by environments. The transition relation ρ is built by suitably composing the transition function of each state component, namely δ_a for agent, δ_i for the environments, and γ for the goal. Notice that we do not build transitions in which an action α is assigned to the agent when either it is in a configuration from which a transition with action α is not defined, or α is not allowed by the protocol *poss* for the current local state. The equivalence relation is updated at each step by considering the observations taken from each environment. Finally, each member of the accepting set W^{acc} contains a goal accepting state, in some environment.

Once this automaton is constructed, we check it for non-emptiness. If it is not empty, i.e., there exists a infinite sequence of action vectors accepted by the automaton, then from such an infinite sequence it is possible to build a strategy realizing the LTL goal. The non-emptiness check is done by resolving polynomially transforming the generalized Büchi automaton into standard Büchi one and solving *fair reachability* over the graph of the automaton, which (as standard reachability) can be solved in NLOGSPACE (Vardi 1996). The non-emptiness algorithm itself can also be used to return a strategy, if it exists.

The following result guarantees that not only the technique is sound (the perfect-recall strategies do realize the LTL specification), but it is also complete (if a perfect-recall strategy exists, it will be constructed by the technique).

this technique is correct, in the sense that if a perfect-recall strategy exists then it will return one.

Theorem 1 (Soundness and Completeness). *A strategy σ that is a solution for problem $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G}\rangle$ exists iff $\mathcal{L}(\mathcal{A}_\mathcal{P}) \neq \emptyset$.*

Proof. ($\Leftarrow$) The proof is based on the fact that $\mathcal{L}(\mathcal{A}_\mathcal{P}) = \emptyset$ implies that it holds the following persistence property: for all runs in $\mathcal{A}_\mathcal{P}$ of the form $r_\omega = w_0\vec{\alpha_1}w_1\vec{\alpha}_2w_2 \ldots \in W^\omega$ there exists an index $i \geq 0$ such that $w_j \notin W^{acc}$ for any $j \geq i$. Conversely, if $\mathcal{L}(\mathcal{A}_\mathcal{P}) \neq \emptyset$, there exists an infinite run $r_\omega = w_0\vec{\alpha_1}w_1\vec{\alpha}_2w_2 \ldots$ on $\mathcal{A}_\mathcal{P}$ visiting at least one state for each accepting set in W^{acc} infinitely often (as required by its acceptance condition), thus satisfying the goal in each environment. First, we notice that such an infinite run r_ω is the form $r_\omega = r'(r'')^\omega$ where both r' and r'' are finite sequences. Hence such a run can be represented with a finite *lazo* shape representation: $r = w_0\vec{a}_1w_1 \ldots \vec{a}_nw_n\vec{a}_{n+1}w_m$ with $m < n$ (Vardi 1996). Hence we can synthesize the corresponding partial function σ by unpacking r (see later). Essentially, given one such r and any observable history $h = o_0, \ldots, o_\ell$ and denoting with α_{ji} the i-th component of $\vec{\alpha}_j$, σ is inductively defined as follows:

- if $\ell = 0$ then $\sigma(o_0) = \alpha_{1i}$ iff $o_0 = obs(perc_i(e_{0i}))$ in w_0.
- if $\sigma(o_0, \ldots, o_{\ell-1}) = \alpha$ then $\sigma(h) = \alpha_{ji}$ iff $o_\ell = obs(perc_i(e_{ji}))$ in $w_j = \langle \vec{e}_j, \vec{c}_j, \vec{g}_j, \equiv_j\rangle$ and α is such that $\alpha = \alpha_{\ell z}$ with $i \equiv_j z$ for some z, where $j = \ell$ if $\ell \leq m$, otherwise $j = m + \ell$ mod(n-m). If instead $o_\ell \neq obs(perc_i(e_{ji}))$ for any e_{ji} then $\sigma(h)$ is undefined.

Indeed, σ is a prefix-closed function of the whole history: we need to look at the choice made at previous step to keep track of it. In fact, we will see later how unpacking r will result into a sort of tree-structure representation. Moreover, it is trivial to notice that any strategy σ synthesized by emptiness checking $\mathcal{A}_\mathcal{P}$ is allowed by agent Ag. In fact, transition relation ρ is defined according to the agent's protocol function $poss$.

($\Rightarrow$) Assume that a strategy σ for $\mathcal{P}$ does exist. We prove that, given such σ, there exists in $\mathcal{A}_\mathcal{P}$ a corresponding accepting run r_ω as before. We prove that there exists in $\mathcal{A}_\mathcal{P}$ a run $r_\omega = w_0\vec{\alpha}_1w_1\vec{\alpha}_2w_2 \ldots$, with $w_\ell = \langle \vec{e}_\ell, \vec{c}_\ell, \vec{g}_\ell, \equiv_\ell\rangle \in W$, such that:

1. ($\ell = 0$) $\sigma(obs(perc_i(e_{0i}))) = \alpha_{1i}$ for all $0 < i \leq k$;
2. ($\ell > 0$) if $\sigma(obs(perc_i(e_{(\ell-1)i}))) = \alpha$ for some $0 < i \leq k$, then $\alpha = \alpha_{\ell i}$ and $e_{\ell i} = \delta_i(e_{(\ell-1)i}, \alpha_{\ell i})$ and $\sigma(obs(perc_i(e_{\ell i})))$ is defined;
3. r_ω is accepting.

In other words, there exists in $\mathcal{A}_\mathcal{P}$ an accepting run that is induced by executing σ on $\mathcal{A}_\mathcal{P}$ itself. Point 1 holds since, in order to be a solution for $\mathcal{P}$, the function σ has to be defined for histories constituted by the sole observation $obs(perc_i(e_{0i}))$ of any environment initial state. According to the definition of the transition relation ρ, there exists in $\mathcal{A}_\mathcal{P}$ a transition from each e_{0i} for all available actions α such that $\delta_i(e_{0i}, \alpha)$ is defined for Env_i. In particular, the transition $\langle w, \vec{\alpha}, w'\rangle$ is not permitted in $\mathcal{A}_\mathcal{P}$ iff either some action component α_i is not allowed by agent's protocol $poss$ or it is not available in the environment Env_i, $0 < i \leq k$. Since σ is a solution of $\mathcal{P}$ (and thus allowed by Ag) it cannot be one of such cases. Point 2 holds for the same reason: there exists a transition in ρ for all available actions of each environment. Point 3 is just a consequence of σ being a solution of $\mathcal{P}$. $\square$

Checking wether $\mathcal{L}(\mathcal{A}_\mathcal{P}) \neq \emptyset$ can be done NLOGSPACE in the size of the automaton. Our automaton is exponential in the number of environments in $\mathcal{E}$, but its construction can be done on-the-fly while checking for non-emptiness. This give us a PSPACE upperbound in the size of the original specification with explicit states. If we have a compact representation of those, then we get an EXPSPACE upperbound. Considering that even the simple case of generalized planning for reachability goals in (Hu and De Giacomo 2011) is PSPACE-complete (EXPSPACE-complete considering compact representation), we obtain a worst case complexity characterization of the problem at hand.

Theorem 2 (Complexity). *Solving the problem $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G}\rangle$ admitting perfect-recall solutions is* PSPACE-*complete (*EXPSPACE-*complete considering compact representation).*

We conclude this section by remarking that, since the agent gets different observation histories from two environments Env_i and Env_j, then from that point on it will be always possible to distinguish these. More formally, denoting with r a run in $\mathcal{A}_P$ and with $r_\ell = \langle \vec{e}_\ell, \vec{c}_\ell, \vec{g}_\ell, \equiv_\ell\rangle$ its ℓ-th state, if $i \equiv_\ell j$, then $i \equiv_{\ell'} j$ for every state $r_{\ell'<\ell}$. Hence it follows that the equivalence relation $\equiv$ is indentical for each state belonging to the same strongly connected component of $\mathcal{A}_\mathcal{P}$. Indeed, assume by contradiction that there exists some index ℓ' violating the assumption above. This implies that $\equiv_{\ell'} \subset \equiv_{\ell'+1}$. So, there exists a tuple in $\equiv_{\ell'+1}$ that is not in $\equiv_{\ell'}$. But this is impossible since, by definition, we have that $i \equiv_{\ell'+1} j$ implies $i \equiv_{\ell'} j$.

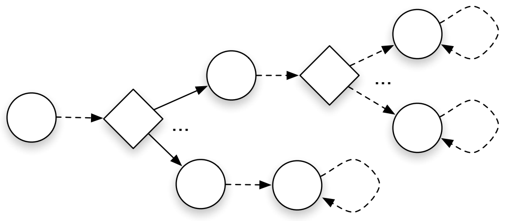


Figure 4: Decision-tree like representation of a strategy.

Figure 4 shows a decision-tree like representation of a strategy. The diamond represents a decision point where the agent reduces its uncertainty about the environment. Each path ends in a loop thereby satisfying the automaton acceptance condition. The loop, which has no more decision point, represents also that the agent cannot reduce its uncertainty anymore and hence it has to act blindly as in conformant planning. Notice that if our environment set includes only one environment, or if we have no observations to use to reduce uncertainty, then the strategy reduces to the structure in Figure 5, which reflects directly the general *lazo* shape of runs satisfying LTL properties: a sequence reaching a certain state and a second sequence consisting in a loop over that state.

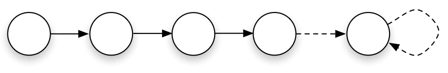


Figure 5: A resulting strategy execution.

Representing Strategies

The technique described in the previous section provides, if a solution does exist, the run of $\mathcal{A}_\mathcal{P}$ satisfying the given LTL specification. As discussed above such a run can be represented finitely. In this section, we exploit this possibility to generate a finite representation of the run that can be used directly as the strategy σ for the agent Ag. The strategy σ can be represented as a finite-state structure with nodes labeled with agent's configuration and edges labeled with a *condition-action* rule $[o]\alpha$, where $o \in Obs$ and $\alpha \in Act_a$. The intended semantics is that a transition can be chosen to be carried on for environment Env_i only if the current observation of its internal state e_i is o, i.e. $o = obs(perc_i(e_i))$. Hence, notice that a strategy σ can be expressed by means of sequences, *case*-conditions and infinite iterations.

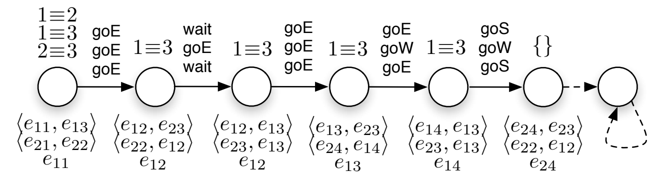

$1\equiv2$
$1\equiv3$
$2\equiv3$
goE
goE
goE
$1\equiv3$
wait
goE
wait
$1\equiv3$
goE
goE
goE
$1\equiv3$
goE
goW
goE
$1\equiv3$
goS
goW
goS
$\{\}$
$\langle e_{11}, e_{13}\rangle$
$\langle e_{21}, e_{22}\rangle$
e_{11}
$\langle e_{12}, e_{23}\rangle$
$\langle e_{22}, e_{12}\rangle$
e_{12}
$\langle e_{12}, e_{13}\rangle$
$\langle e_{23}, e_{13}\rangle$
e_{12}
$\langle e_{13}, e_{23}\rangle$
$\langle e_{24}, e_{14}\rangle$
e_{13}
$\langle e_{14}, e_{13}\rangle$
$\langle e_{23}, e_{13}\rangle$
e_{14}
$\langle e_{24}, e_{23}\rangle$
$\langle e_{22}, e_{12}\rangle$
e_{24}

Figure 6: Accepting run r for Example 1.

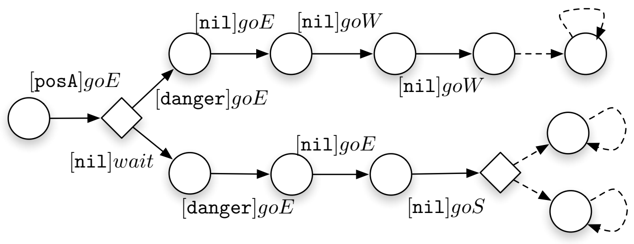


Figure 7: Corresponding G_r

In other words, it can be represented as a graph $G_r = \langle N, E\rangle$ where N is a set of nodes, $\lambda : N \to Conf$ its labeling, and $E \subseteq N \times \Phi \times N$ is the transition relation. G_r can be obtained by unpacking the run found as witness, exploiting the equivalence relation $\equiv$. More in details, let $r = w_0\vec{a}_1 w_1 \ldots \vec{a}_n w_n \vec{a}_{n+1} w_m$ with $m \le n$ be a finite representation of the infinite run returned as witness. Let r_ℓ be the ℓ-th state in r, whereas we denote with $r|_\ell$ the sub-run of r starting from state r_ℓ. A projection of r over a set $X \subseteq \{1, \ldots, k\}$ is the run $r(X)$ obtained by projecting away from each w_i all vector components and indexes not in X. We mark state w_m: $loop(w_\ell) =$ *true* if $\ell = m$, *false* otherwise.

$$G_r = \text{UNPACK}(r, nil);$$

```
UNPACK($r$, $loopnode$):
1: $N = E = \emptyset$;
2: be $r_0 = \langle \vec{e}, \vec{c}, \vec{g}, \equiv\rangle$;
3: **if** $loop(r_0) \wedge loopnode \neq nil$ **then**
4:     **return** $\langle\{loopnode\}, E\rangle$;
5: **end if**
6: be $\vec{a}_1 = \langle \alpha_1, \ldots, \alpha_k\rangle$;
7: Let $X = \{X_1, \ldots, X_b\}$ be the partition induced by $\equiv$;
8: $node$ = new node;
9: **if** $loop(r_0)$ **then**
```

```
10:     $loopnode = node$;
11: end if
12: for ($j = 1$; $j \leq b$; $j$++) do
13:     $G' = $ UNPACK$(r(X_j)|_1, loopnode)$;
14:     choose $i$ in $X_j$;
15:     $\lambda(node) = c_i$;
16:     $E = E' \cup \langle node, [obs(perc_i(e_i))]\alpha_i, root(G')\rangle$;
17:     $N = N \cup N'$;
18: end for
19: return $\langle N \cup \{node\}, E\rangle$;
```

The algorithm above, presented in pseudocode, recursively processes run r until it completes the loop on w_m, then it returns. For each state, it computes the partition induced by relation $\equiv$ and, for each element in it, generates an edge in G_r labeled with the corresponding action α taken from the current action vector.

From G_r we can derive *finite state strategy* $\sigma_f = \langle N, succ, act, n_0\rangle$. where:

- $succ : N \times Obs \times Act_a \to N$ such that $succ(n, o, a) = n'$ iff $\langle n, [o]\alpha, n'\rangle \in E$;
- $act : N \times Conf \times Obs \to Act_a$ such that $\alpha = act(n, c, o)$ iff $\langle n, [o]\alpha, n'\rangle \in E$ for some $n' \in N$ and $c = \lambda(n)$;
- $n_0 = root(G_r)$, i.e., the initial node of G_r.

From σ_f we can derive an actual perfect-recall strategy $\sigma : Obs^* \to Act_a$ as follows. We extend the deterministic function $succ$ to observation histories $h \in Obs^*$ of length ℓ in the obvious manner. Then we define σ as the function: $\sigma(h) = act(n, c, last(h))$, where $n = succ(root(G_r), h^{n-1})$, $h^{\ell-1}$ is the prefix of h of length ℓ-1 and $c = \lambda(n)$ is the current configuration. Notice that such strategy is a partial function, dependent on the environment set $\mathcal{E}$: it is only defined for observation histories embodied by the run r.

It can be show that the procedure above, based on the algorithm UNPACK, is correct, in the sense that the executions of the strategy it produces are the same as those of the strategy generated by the automaton constructed for Theorem 1.

Example 3. Let us consider again the three environments, the agent and goal as in Example 1. Several strategies do exist. In particular, an accepting run r for $\mathcal{A}_\mathcal{P}$ is depicted in Figure 6, from which a strategy σ can be unpacked. Strategy σ can be equivalently represented as in Figure 7 as a function from observation histories to actions. For instance, $\sigma(c_0, \{\texttt{posA}, \texttt{nil}, \texttt{danger}, \texttt{nil}\}) = goE$. In particular, being all environments indistinguishable in the initial state (the agent receives the same observation `posA`), this strategy prescribes action goE for the three of them. Resulting states are such that both Env_1 and Env_3 provide perception `nil`, whereas Env_2 provides perception `danger`. Having received different observation histories so far, strategy σ is allowed to select different action for Env_2: goE for Env_2 and $wait$ for Env_1 and Env_3. In fact, according to protocol $poss$, action $wait$ is not an option for Env_2, whereas action goE is not significant for Env_3, though it avoids an accident in Env_1. In this example, by executing the strategy, the agent eventually receives different observation histories from each environment, but this does not necessary hold in general: different environments could also remain indistinguishable forever. ■

There is still no link between synthesized strategies and agents. The main idea is that a strategy can be easily seen as a sort of an agents' protocol refinement where the states used by the agents are extended to store the (part of the) relevant history. This is done in the next section.

Embedding Strategies into Protocols

We have seen how it is possible to synthesize perfect-recall strategies that are function of the observation history the agent received from the environment. Computing such strategies in general results into a function that requires an unbounded amount of memory. Nonetheless, the technique used to solve the problem shows that *(i)* if a strategy does exist, there exists a bound on the information actually required to compute and execute it and *(ii)* such strategies are finite-state. More precisely, from the run satisfying the LTL specification, it is possible to build the finite-state strategy $\sigma_f = \langle N, succ, act, n_0\rangle$. We now incorporate such a finite-state strategy into the agent protocol, by suitably expanding the configuration space of the agent to store in the configuration information needed to execute the finite state strategy. This amounts to define a new configuration space $\overline{Conf} = Conf \times N$ (hence a new local state space $\overline{L}$).

Formally, given the original agent $Ag = \langle Conf, Act_a, Obs, L, poss, \delta_a, c_0\rangle$ and the finite state strategy $\sigma_f = \langle N, succ, act, n_0\rangle$, we construct a new agent $\overline{Ag} = \langle\overline{Conf}, Act_a, Obs, \overline{L}, \overline{poss}, \overline{\delta}_a, \overline{c}_0\rangle$ where :

- Act_a and Obs are as in Ag;
- $\overline{Conf} = Conf \times N$ is the new set of configurations;
- $\overline{L} = \overline{Conf} \times Obs$ is the new local state space;
- $\overline{poss} : \overline{L} \to Act_a$ is an action-deterministic protocol defined as:

$$\overline{poss}(\langle c, n\rangle, o) = \begin{cases} \{\alpha\}, & \text{iff } act(n, c, o) = \alpha \\ \emptyset, & \text{if } act(n, c, o) \text{ is undefined;} \end{cases}$$

- $\overline{\delta}_a : \overline{L} \times Act_a \to \overline{Conf}$ is the transition function, defined as:

$$\overline{\delta}_a(\langle\langle c, n\rangle, o\rangle, a) = \langle\delta_a(c, o), succ(n, o, a)\rangle;$$

- $\overline{c_0} = \langle c_0, n_0\rangle$.

On this new agent the original strategy can be phrased as a state-base strategy:

$$\overline{\sigma} : \overline{Conf} \times Obs \to Act_a$$

simply defined as: $\overline{\sigma}(\langle c, n\rangle, o) = \overline{poss}(\langle c, n\rangle, o)$.

It remains to understand in what sense we can think the agent $\overline{Ag}$ as a refinement or customization of the agent Ag. To do so we need to show that the executions allowed by the new protocol are also allowed by the original protocol, in spite of the fact that the configuration spaces of the two agents are different. We show this by relaying on the theoretical notion of *simulation*, which formally captures the ability of one agent (Ag) to simulate, i.e., copy move by move, another agent ($\overline{Ag}$).

Given the two agents Ag_1 and Ag_2, a *simulation relation* is a relation $\mathcal{S} \subseteq L_1 \times L_2$ such that $\langle l_1, l_2 \rangle \in \mathcal{S}$ implies that:

if $l_2 \xrightarrow{\alpha} l_2'$ and $\alpha \in poss_2(l_2)$ then there exists l_1' such that $l_1 \xrightarrow{\alpha} l_1'$ and $\alpha \in poss_1(l_1)$ and $\langle l_1', l_2' \rangle \in \mathcal{S}$.

where $l_i \xrightarrow{\alpha} l_i'$ iff c_i' is the agent configuration in l_i' and $c_i' = \delta_a(l_i, \alpha)$. We say that agent Ag_1 *simulates* agent Ag_2 iff there exists a simulation relation $\mathcal{S}$ such that $\langle l_1^0, l_0^2 \rangle \in \mathcal{S}$ for each couple of initial local states $\langle l_1^0, l_2^0 \rangle$ with the same initial observation.

Theorem 3. *Agent Ag simulates $\overline{Ag}$.*

Proof. First, we notice that $\overline{poss}(\langle c, n \rangle, o) \subseteq poss(\langle c, o \rangle)$ for any $c \in Conf, o \in Obs$. In fact, since we are only considering allowed strategies, the resulting protocol $\overline{poss}$ is compatible with agent Ag. The result follows from the fact that original configurations are kept as fragment of both L and $\overline{L}$. Second, being both the agent and environments deterministic, the result of applying the same action α from states $\langle \langle c, n \rangle, o \rangle \in \overline{L}$ and $\langle c, o \rangle \in L$ are states $\langle \langle c', n' \rangle, o' \rangle$ and $\langle c', o' \rangle$, respectively.

Finally, assume towards contradiction that $\overline{Ag}$ is not simulated by Ag. This implies that there exists a sequence of length $n \geq 0$ of local states $l_0 \xrightarrow{\alpha_1} l_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_k} l_k$ of Ag, where l_0 is some initial local state, and a corresponding sequence $\overline{l}_0 \xrightarrow{\alpha_1} \overline{l}_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_k} \overline{l}_k$ of $\overline{Ag}$, starting from a local state $\overline{l}_0$ sharing the same observation of l_0, such that $\alpha \in \overline{poss}(\overline{l}_k)$ but $\alpha \notin poss(l_k)$ for some α. For what observed before, l_k and $\overline{l}_k$ share the same agent's configuration; in particular, they are of the form $\overline{l}_k = \langle \langle c_k, n_k \rangle, o_k \rangle$ and $l_k = \langle c_k, o_k \rangle$. Hence $\overline{poss}(\langle \langle c_k, n_k \rangle, o_k \rangle) \subseteq poss(\langle c_k, o_k \rangle)$ and we get a contradiction. □

Theorem 4. *$\overline{Ag}$ is nonblocking.*

It follows from the fact that a strategy σ that is a solution for problem $\mathcal{P}$ is a prefix-closed function and it is allowed by Ag. Hence, for any $\overline{l} \in \overline{L}$ reachable from any initial local state by applying σ, we have $\overline{poss}(\overline{l}) \neq \emptyset$.

From Theorem 1 and results in previous sections we have:

Theorem 5. *Any execution of agent $\overline{Ag}$ over each environment Env_i satisfies the original agent specification Ag and the goal specification.*

A Notable Variant

Finally, we consider a variant of our problem where we specify LTL goals directly in terms of states of each environment in the environment set. In other words, instead of having a single goal specified over the percepts of the environments we have one LTL goal for each environment. More precisely, as before, we assume that a single strategy has to work on the whole set of deterministic environments. As previously, we require that $Act_a \subseteq \bigcap_{i=1,\dots,k} Act_{ei}$ and that all environment share the same alphabet of perceptions $Perc$. Differently from before, we associate a distinct goal to each environment. We take as input alphabet of each goal specification $\mathcal{G}_i$ the set of environment's state E_i, i.e., $\mathcal{G}_i = \langle E_i, G_i, g_{i0}, \gamma_i, G_i^{acc} \rangle$. All goals are thus intimately different, as they are strictly related to the specific environment. Intuitively, we require that a strategy for agent Ag satisfies, in all environments Env_i, its corresponding goal $\mathcal{G}_i$. In other words, σ is a solution of the generalized planning problem $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G} \rangle$ iff it is allowed by Ag and it generates, for each environment Env_i, an infinite trace τ_i that is accepted by $\mathcal{G}_i$.

Devising perfect-recall strategies now requires only minimal changes to our original automata-based technique to take into account that we have now k distinct goals one for each environment. Given Ag and $\mathcal{E}$ as defined before, and k goals $\mathcal{G}_i = \langle E_i, G_i, g_{i0}, \gamma_i, G_i^{acc} \rangle$, we build the generalized Büchi automaton $\mathcal{A}_\mathcal{P} = \langle Act_a^k, W, w_0, \rho, W^{acc} \rangle$ as follows. Notice that each automaton $\mathcal{G}_i$ has E_i as input alphabet.

- $Act_a^k = (Act_a)^k$ is the set of k-vectors of actions;
- $W = E^k \times L^k \times G_1 \times \dots \times G_k \times \wp(\equiv)$;
- $w_0 = \langle e_{i0}, \dots, e_{k0}, c_{i0}, \dots c_{k0}, g_{i0}, \dots, g_{k0}, \equiv_0 \rangle$ where
  $i \equiv_0 j$ iff $obs(perc_i(e_{i0})) = obs(perc_j(e_{j0}))$;
- $\langle \vec{e}', \vec{c}', \vec{g}', \equiv' \rangle \in \rho(\langle \vec{e}, \vec{c}, \vec{g}, \equiv \rangle, \vec{\alpha})$ iff
  - if $i \equiv j$ then $\alpha_i = \alpha_j$;
  - $e_i' = \delta_i(e_i, \alpha_i)$;
  - $c_i' = \delta_a(l_i, \alpha_i) \wedge \alpha_i \in poss(l_i)$ with $l_i = \langle c_i, obs(perc_i(e_i)) \rangle$;
  - $g_i' = \gamma_i(g_i, e_i)$;
  - $i \equiv' j$ iff $i \equiv j \wedge obs(perc_i(e_i')) = obs(perc_j(e_j'))$.
- $W^{acc} = \{$
  $E^k \times Conf^k \times G_1^{acc} \times G_2 \times \dots \times G_k \times \equiv$, ... ,
  $E^k \times Conf^k \times G_1 \times \dots \times G_{k-1} \times G_k^{acc} \times \equiv \}$

The resulting automaton is similar to the previous one, and the same synthesis procedures apply, including the embedding of the strategy into an agent protocol. We also get the analogous soundness and completeness result and complexity characterization as before.

Example 4. Consider again Example 1 but now we require, instead of having a single goal $\phi_\mathcal{G}$, three distinct goals over the three environments. In particular for the car-train environments Env_1 and Env_2, we adopt the same kind of goal as before, but avoiding certain cells for environments, e.g., $\phi_{\mathcal{G}_1} = (\Box\Diamond e_{11}) \wedge (\Box\Diamond e_{24}) \wedge \Box\neg(\langle c_{13}, c_{13} \rangle \vee \langle c_{23}, c_{23} \rangle) \wedge \Box\neg\langle e_{22}, e_t \rangle$ and $\phi_{\mathcal{G}_2} = (\Box\Diamond e_{21}) \wedge (\Box\Diamond e_{24}) \wedge \Box\neg(\langle c_{23}, c_{23} \rangle \vee \dots \vee \langle c_{14}, c_{14} \rangle) \wedge \Box\neg\langle e_{11}, e_{14} \rangle$, whereas in the Wumpus world we only require to reach the gold after visiting initial position: $\phi_{\mathcal{G}_3} = \Box(e_{11} \rightarrow \Diamond e_{34}) \wedge \Box\neg(c_{23} \vee c_{31})$. It can be shown that a (perfect-recall) strategy for achieving such goals exists. In fact, there exists at least one strategy (e.g., one extending the prefix depicted in Figure 7 avoiding in Env_1 and Env_2 states mentioned above) that satisfies goal $\phi_\mathcal{G}$ over all environments as well as these three new goals (in particular, if a strategy satisfies $\phi_\mathcal{G}$ then it satisfies $\phi_{\mathcal{G}_3}$ too). Such a strategy can be transformed into an agent protocol, by enlarging the configuration space of the agent, as discussed in the previous section.

Conclusions

In this paper we investigated the synthesis of an agent's protocol to satisfy LTL specifications while dealing with multiple, partially-observable environments. In addition to the computationally optimal procedure here introduced, we explored an automata-based protocol refinement for a perfect-recall strategy that requires only finite states.

There are several lines we wish to pursue in the future. Firstly, we would like to implement the procedure here described and benchmark the results obtained in explicit and symbolic settings against planning problems from the literature. We note that current model checkers such as MCMAS (Lomuscio, Qu, and Raimondi 2009) and MCK (Gammie and van der Meyden 2004) support interpreted systems, the semantics here employed.

It is also of interest to explore whether the procedures here discussed can be adapted to other agent-inspired logics, such as epistemic logic (Ronald Fagin and Vardi 1995). Epistemic planning (van der Hoek and Wooldridge 2002), i.e., planning for epistemic goals, has been previously discussed in the agents-literature before, but synthesis methodologies have not, to our knowledge, been used in this context.

When dealing with LTL goals we need to consider that the agent cannot really monitor the achievement of the specification. Indeed every linear temporal specification can be split into a "liveness" part which can be checked only considering the entire run and a "safety" part that can be checked on finite prefixes of such runs (Baier and Katoen 2008). Obviously the agent can look only at the finite history of observations it got so far, so being aware of achievement of LTL properties is quite problematic in general. This issue is related to runtime verification and monitoring (Eisner et al. 2003; Bauer, Leucker, and Schallhart 2011), and in the context of AI, it makes particularly attractive to include in the specification of the dynamic property aspects related to the knowledge that the agent acquires, as allowed by interpreted systems.

Acknowledgments. We thank the anonymous reviewers for their comments. We acknowledge the support of EU Project FP7-ICT ACSI (257593).

References

Baier, C., and Katoen, J.-P. 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.

Bauer, A.; Leucker, M.; and Schallhart, C. 2011. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology* 20(4):14.

Bonet, B., and Geffner, H. 2009. Solving pomdps: Rtdp-bel vs. point-based algorithms. In *IJCAI*, 1641–1646.

De Giacomo, G., and Vardi, M. Y. 1999. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, 226–238.

Eisner, C.; Fisman, D.; Havlicek, J.; Lustig, Y.; Mcisaac, A.; and Van Campenhout, D. 2003. Reasoning with temporal logic on truncated paths. 27–39.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning about Knowledge*. Cambridge: MIT Press.

Gammie, P., and van der Meyden, R. 2004. MCK: Model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, 479–483. Springer-Verlag.

Harding, A.; Ryan, M.; and Schobbens, P.-Y. 2005. A new algorithm for strategy synthesis in ltl games. In Halbwachs, N., and Zuck, L. D., eds., *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, 477–492. Springer.

Hu, Y., and De Giacomo, G. 2011. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, 918–923.

Jamroga, W., and van der Hoek, W. 2004. Agents that know how to play. *Fundam. Inform.* 63(2-3):185–219.

Kupferman, O., and Vardi, M. Y. 2000. Synthesis with incomplete informatio. In *In Advances in Temporal Logic*, 109–127. Kluwer Academic Publishers.

Lomuscio, A., and Raimondi, F. 2006. Model checking knowledge, strategies, and games in multi-agent systems. In *AAMAS*, 161–168.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2009. Mcmas: A model checker for the verification of multi-agent systems. In Bouajjani, A., and Maler, O., eds., *CAV*, volume 5643 of *Lecture Notes in Computer Science*, 682–688. Springer.

Pacuit, E., and Simon, S. 2011. Reasoning with protocols under imperfect information. *Review of Symbolic Logic* 4(3):412–444.

Parikh, R., and Ramanujam, R. 1985. Distributed processes and the logic of knowledge. In *Logic of Programs*, 256–268.

Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2006. Synthesis of reactive(1) designs. In Emerson, E. A., and Namjoshi, K. S., eds., *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, 364–380. Springer.

Pnueli, A., and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *Proc. of POPL'89*, 179–190.

Ronald Fagin, Joseph Y. Halpern, Y. M., and Vardi, M. Y. 1995. *Reasoning about Knowledge*. Cambridge, MA: MIT Press.

Russell, S. J., and Norvig, P. 2010. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

van der Hoek, W., and Wooldridge, M. 2002. Tractable multiagent planning for epistemic goals. In *AAMAS*, 1167–1174.

van der Meyden, R. 1996. Finite state implementations of knowledge-based programs. In Chandru, V., and Vinay, V., eds., *FSTTCS*, volume 1180 of *Lecture Notes in Computer Science*, 262–273. Springer.

Vardi, M. 1996. An automata-theoretic approach to linear temporal logic. In Moller, F., and Birtwistle, G., eds., *Logics for Concurrency*, volume 1043 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 238–266.

Wooldridge, M., and Lomuscio, A. 2001. A computationally grounded logic of visibility, perception, and knowledge. *Logic Journal of the IGPL* 9(2):273–288.