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, exploit-
ing 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, it 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 achievement 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](image)

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 \( E_{\text{env}} = (E, Act_e, Perc, perc, \delta, e_0) \) such that:

- \( E = \{e_0, \ldots\} \) is a finite set of local states of the environ-
Notice that, as in classical planning, such an environment is deterministic. Assume that the agent is in its initial configuration and that the current environment's state is observable. The corresponding perceivable trace is the trace obtained by applying the perception function: \( \text{perc}(\tau) = \text{perc}(c_0), \ldots, \text{perc}(c_n) \).

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 \( \text{Ag} = (\text{Conf}, \text{Act}_a, \text{Obs}, \text{obs}, \text{L}, \text{poss}, \delta_a, c_0) \) where:

- \( \text{Conf} = \{c_0, \ldots\} \) is a finite set of the agent's configurations (internal states);
- \( \text{Act}_a \) is the finite set of agent's actions;
- \( \text{Obs} \) is the alphabet of observations the agent can make;
- \( \text{obs} : \text{Perc} \to \text{Obs} \) is the observation function;
- \( \text{L} = \text{Conf} \times \text{Obs} \) is the set of agent's local states;
- \( \text{poss} : \text{L} \to 2^{(\text{Act}_a)} \) is a protocol function;
- \( \delta_a : \text{L} \times \text{Act}_a \to \text{Conf} \) is the (partial) transition function;
- \( c_0 \in \text{Conf} \) is the initial configuration. A local state \( l = (c, o) \) is called initial iff \( c = c_0 \).

Agents defined as above are deterministic: given a local state \( l \) and an action \( a \), there exists a unique next configuration \( c' = \delta_a(l, a) \). Nonetheless, observations introduce nondeterminism when computing the new local state resulting from action execution: executing action \( a \) in a given local state \( l = (c, o) \) results into a new local state \( l' = (c', o') \) such that \( c' = \delta_a(l, a) \) 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 \( c_0 \). Assume also that \( \text{obs(perc}(c_0)) = o \), i.e., the agent receives observation \( o \). Hence, the current local state is \( l_0 = (c_0, o) \). If the agent performs action \( b \) (with \( b \in \text{poss}(l_0) \)), the agent moves from configuration \( c_0 \) to configuration \( c_1 = \delta_a((c_0, o), b) \).

At the same time, the environment changed its state from \( e_0 \) to \( e_1 \), so that the new local state is \( l_1 = (c_1, o') \), where \( o' = \text{obs}(\text{perc}(e_1)) \).

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 if it is compatible with its transition function, i.e., \( \alpha \in \text{poss}((c, o)) \rightarrow \exists c' \in \text{Conf} \mid \delta_a((c, o), \alpha) = c' \). Moreover, we say that a protocol \( \text{poss} \) 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 \( \text{poss} \) and for each sequence of local states \( l_0 = (c_0, o_0) \) such that \( l_i = (c_i, o_i) \) and \( o_{i+1} \in \text{poss}((c_i, o_i)) \) for each \( 0 \leq i < n \), we have \( \text{poss}(l_n) \neq \emptyset \). So, an agent is nonblocking iff it has a compatible protocol function which always provides a non-empty 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 \( \text{Env} \), the observation history of an agent \( \text{Ag} \) is defined as the sequence obtained by applying the observation function: \( \text{obs(perc}(\tau)) = \text{obs}(\text{perc}(e_0)), \ldots, \text{obs}(\text{perc}(e_n)) \). Given one such history \( h \in \text{Obs}^* \), we denote with \( \text{last}(h) \) its latest observation: \( \text{last}(h) = \text{obs}(\text{perc}(e_n)) \).

**LTL Specifications against Multiple Environments**

In this paper, we consider an agent \( \text{Ag} \) and a finite set \( \mathcal{E} \) of environments, all sharing common actions and the same perception alphabet. Such environments are indistinguishable, 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 \( \mathcal{P} = (\text{Ag}, \mathcal{E}, \mathcal{G}) \), where \( \text{Ag} \) is an agent, \( \mathcal{E} \) an environment set, and \( \mathcal{G} \) the 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 \( \mathcal{E} = \{ \text{Env}_1, \ldots, \text{Env}_k \} \), with \( \text{Env}_i = (E_i, \text{Act}_{e_i}, \text{Perc}, \text{perc}_i, \text{\delta}_i, e_i) \). Environments share the same alphabet \( \text{Perc} \) of the agent \( \text{Ag} \). Moreover the \( \text{Ag} \)’s actions must be executable in the various environments: \( \text{Act}_a \subseteq \bigcap_{i=1, \ldots, k} \text{Act}_{e_i} \). 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\(^1\) \( \mathcal{G} = (\text{Perc}, G, g_0, \gamma, C^\text{acc}) \), describing the desired behaviour of the system in terms of perceivable traces, where:

- \( \text{Perc} \) 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.
Initially, the car is in $e_{11}$ and the train is in $e_{13}$. The train is controlled by the agent, whereas the train is a moving obstacle moving from $e_{13}$ to $e_{23}$ again and so on. The set of actions is $Act_{e_1} = \{goN, goS, goE, goW, wait\}$. The train and the car cannot leave the grid, so actions are allowed only when feasible. The state space is then $E_1 = \{e_{11},...,e_{24}\} \times \{e_{13},e_{23}\}$, representing the positions of the car and the train. We consider a set of perceptions $Perc = \{posA, posB, danger, dead, nil\}$, and a function $perc_1$ defined as follows: $perc_1(e_{11}, e_i) = posA$, $perc_1(e_{21}, e_i) = posB$, $perc_1(e_{13}, e_{23}) = perc_1(e_{23}, e_{13}) = danger$ and $perc_1(e_{13}, e_{13}) = perc_1(e_{23}, e_{23}) = dead$. $perc_1(e, e_i) = nil$ for any other state.

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 $Act_{e_3} = \{goN, goS, goE, goW, wait\}$. Recall that the set of perceptions $Perc$ is instead shared with $Env_1$ and $Env_2$. The state space is $E_3 = \{e_{11},...,e_{34}\}$, and the function $perc_3$ is defined as follows: $perc_3(e_{11}) = posA$, $perc_3(e_{34}) = posB$, $perc_3(e_{23}) = perc_3(e_{13}) = dead$, $perc_3(e) = danger$ for $e \in \{e_{13},e_{21},e_{22},e_{24},e_{32},e_{33}\}$, whereas $perc_3(e) = nil$ 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 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 danger in $Env_1$ or $Env_2$ can not be used to prevent an accident.

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 $e_{23}$ again and so on. The set of actions is $Act_{e_1} = \{goN, goS, goE, goW, wait\}$. The train and the car cannot leave the grid, so actions are allowed only when feasible. The state space is then $E_1 = \{e_{11},...,e_{24}\} \times \{e_{13},e_{23}\}$, representing the positions of the car and the train. We consider a set of perceptions $Perc = \{posA, posB, danger, dead, nil\}$, and a function $perc_1$ defined as follows: $perc_1(e_{11}, e_i) = posA$, $perc_1(e_{21}, e_i) = posB$, $perc_1(e_{13}, e_{23}) = perc_1(e_{23}, e_{13}) = danger$ and $perc_1(e_{13}, e_{13}) = perc_1(e_{23}, e_{23}) = dead$. $perc_1(e, e_i) = nil$ for any other state.

Figure 2: Environments $Env_1$ and $Env_2$

![Figure 2: Environments $Env_1$ and $Env_2$](image-url)

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 $Act_a = Act_{e_i}$, $i = 1, 2, 3$. 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 $posA, danger, dead$, but it is unable to sense $posB$, 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 $Obs = \{posA, danger, dead, nil\}$ and $obs(posA) = posA, obs(danger) = danger, obs(dead) = dead,$ and $obs(nil) = obs(posB) = nil$. Since $Ag$ is filtering away perception $posB$, 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 danger: $poss((e_0, 0)) = Act_a$ if $0 \neq danger, Act_a \setminus \{wait\}$ otherwise.

Finally, let $G$ be the automaton corresponding to the LTL formula $\phi_G = (\Box \Diamond posA) \land (\Box \Diamond posB) \land \neg \Diamond dead$ over the perception alphabet, constraining perceivable traces such that the controlled objects (the car / the hero) visit positions...
that, given a sequence of observations (the observation history), returns an action. A strategy \( \sigma \) is allowed by \( A \)'s protocol iff, given any observation history \( h \in \text{Obs}^* \), \( \sigma(h) = \alpha \) implies \( \alpha \in \text{poss}(\langle c, \text{last}(h) \rangle) \), 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 \( A \), starting from initial configuration \( c_0 \). Hence, a strategy \( \sigma \) is a solution of the problem \( \mathcal{P} = \langle A, \mathcal{E}, \mathcal{G} \rangle \) iff it is allowed by \( A \) and it generates, for each environment \( E_{\mathcal{V}_i} \), an infinite trace \( \tau = e_0\alpha_1\alpha_2\ldots \) such that the corresponding perceivable trace \( \text{perc}(\tau) \) 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 \( d = \langle a_1, \ldots, a_k \rangle \), where each component specifies one operation for each environment. Such sequences of action vectors correspond to a strategy \( \sigma \), which, however, must be executable: for any pair \( i,j \in \{1, \ldots, k\} \) and observation history \( h \in \text{Obs}^* \) such that both \( \sigma_i \) and \( \sigma_j \) 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 \( E_{\mathcal{V}_i} = \langle E_i, \text{Act}_{E_i}, \text{Perc}, \text{perc}, \delta_i, e_0 \rangle \), an agent \( A \) is \( \langle \text{Conf}, \text{Act}_E, \text{Obs}, \text{L}, \text{poss}, \delta_0, c_0 \rangle \) and goal \( \mathcal{G} = \langle \text{Perc}, \text{G}, \text{g}_0, \gamma, \text{Gacc} \rangle \), we build the generalized Büchi automaton \( \mathcal{A}_\mathcal{P} = \langle \text{Act}_E^k, W, w_0, \rho, \mathcal{W}^{\text{acc}} \rangle \) as follows:

- \( \text{Act}_E^k = (\text{Act}_E)^k \) is the set of \( k \)-vectors of actions; 
- \( W = E^{k} \times \text{Conf}^k \times G^k \times \varphi(\equiv) \);\(^2\)
- \( w_0 = \langle e_{10}, \ldots, e_{k0}, c_0, \ldots, c_0, g_0, \ldots, g_0, \equiv_0 \rangle \) where \( i \equiv_0 j \) iff \( \text{obs} (\text{perc}_i(e_{i0})) = \text{obs} (\text{perc}_j(e_{j0})) \);
- \( \langle \bar{e}', \bar{c}', \bar{g}', \equiv' \rangle \in \rho((\bar{e}, \bar{c}, \bar{g}, \equiv), \equiv) \) iff
  - if \( i \equiv j \) then \( \alpha_i = \alpha_j \); 
  - \( e'_i = \delta_i(e_i, \alpha_i) \); 
  - \( e'_i = \delta_i(l_i, \alpha_i) \land \alpha_i \in \text{poss}(l_i) \) where \( l_i = \langle \bar{e}_i, \text{obs} (\text{perc}_i(e_i)) \rangle \); 
  - \( g'_i = \gamma(g_i, \text{perc}_i(e_i)) \);

\(^2\)We denote by \( \varphi(\equiv) \) the set of all possible equivalence relations \( \equiv \subseteq \{1, \ldots, k\} \times \{1, \ldots, k\} \).
\[ i \equiv j \ \text{iff} \ i \equiv j \land \text{obs}(\text{perc}_i(e'_i)) = \text{obs}(\text{perc}_j(e'_j)). \]

- \( W^\text{acc} = \{ \)
  \[ E^k \times \text{Conf}^k \times G^\text{acc} \times G \times \ldots \times G \times \equiv, \ldots, \]
  \[ E^k \times \text{Conf}^k \times G \times \ldots \times G \times G^\text{acc} \times \equiv \} \]

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 \( A_P \) 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 \( \rho \) is built by suitably composing the transition function of each state component, namely \( \delta_i \) for agent, \( \delta_i \) for the environments, and \( \gamma \) for the goal. Notice that we do not build transitions in which an action \( \alpha \) is assigned to the agent when either it is in a configuration from which a transition with action \( \alpha \) is not defined, or \( \alpha \) is not allowed by the protocol \( \text{pos} \) 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 \( W^\text{acc} \) 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 \textit{fair reachability} over the graph of the automaton, which (as standard reachability) can be solved in \textsc{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.

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

\textbf{Proof.} \((\Leftarrow)\) The proof is based on the fact that \( \mathcal{L}(A_P) = \emptyset \) implies that it holds the following persistence property: for all runs in \( A_P \) of the form \( r_w = w_0 a_1 w_1 a_2 w_2 \ldots \in W^w \) there exists an index \( j \geq 0 \) such that \( w_j \notin W^\text{acc} \) for any \( j \geq i \). Conversely, if \( \mathcal{L}(A_P) \neq \emptyset \), there exists an infinite run \( r_w = w_0 a_1 w_1 a_2 w_2 \ldots \) on \( A_P \) visiting at least one state for each accepting set in \( W^\text{acc} \) infinitely often (as required by its acceptance condition), thus satisfying the goal in each environment. First, we notice that such an infinite run \( r_w \) is the form \( r_w = r'(r^m) \) where both \( r' \) and \( r^m \) are finite sequences. Hence such a run can be represented with a finite \textit{lazo} shape representation: \( r = w_0 a_1 w_1 \ldots a_n w_n a_{n+1} w_m \)

with \( m < n \) (Vardi 1996). Hence we can synthesize the corresponding partial function \( \sigma \) by unpacking \( r \) (see later). Essentially, given one such \( r \) and any observable history \( h = o_0, \ldots, o_t \) and denoting with \( \alpha_j \) the \( i \)-th component of \( \vec{\alpha}_j \), \( \sigma \) is inductively defined as follows:

- if \( \ell = 0 \) then \( \sigma(o_0) = \alpha_{i_0} \) iff \( o_0 = \text{obs}(\text{perc}_i(e'_i)) \) in \( w_0 \).
- if \( \sigma(o_0, \ldots, o_{\ell-1}) = \alpha \) then \( \sigma(h) = \alpha_{i_\ell} \) iff \( o_\ell = \text{obs}(\text{perc}_i(e_{\ell_\ell})) \) in \( w_\ell = (e_{\ell_1}, e_{\ell_2}, \ldots, e_{\ell_\ell}) \) and \( \alpha \) is such that \( \alpha = \alpha_{i_\ell} \equiv_\ell z \) for some \( z \), where \( j = \ell \)
- if \( \ell < m \), otherwise \( j = m + \ell \mod(n-m) \). If instead \( o_\ell \not\equiv \text{obs}(\text{perc}_i(e_{\ell_\ell})) \) for any \( e_{\ell_\ell} \) then \( \sigma(h) \) is undefined.

Indeed, \( \sigma \) 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 \( \sigma \) synthesized by emptiness checking \( A_P \) is allowed by agent \( Ag \). In fact, transition relation \( \rho \) is defined according to the agent’s protocol function \( \text{pos} \).

\((\Rightarrow)\) Assume that a strategy \( \sigma \) for \( P \) does exist. We prove that, given such \( \sigma \), there exists in \( A_P \) a corresponding accepting run \( r_w \) as before. We prove that there exists in \( A_P \) a run \( r_w = w_0 a_1 w_1 a_2 w_2 \ldots \) with \( w_l = (e_{\ell_1}, e_{\ell_2}, \ldots, e_{\ell_\ell}) \in W \), such that:

1. \( (\ell = 0) \) \( \sigma(\text{obs}(\text{perc}_i(e'_i))) = \alpha_{i_0} \) for all \( 0 < i \leq k \);
2. \( (\ell > 0) \) if \( \sigma(\text{obs}(\text{perc}_i(e_{(\ell-1)_i}))) = \alpha \) for some \( 0 < i \leq k \), then \( \alpha = \alpha_{i_\ell} \) and \( e_{i_\ell} = \delta_i(e_{(\ell-1)_i}, \alpha_{i_\ell}) \) and \( \sigma(\text{obs}(\text{perc}_i(e_{i_\ell}))) \) is defined;
3. \( r_w \) is accepting.

In other words, there exists in \( A_P \) an accepting run that is induced by executing \( \sigma \) on \( A_P \) itself. Point 1 holds since, in order to be a solution for \( P \), the function \( \sigma \) has to be defined for histories constituted by the sole observation \( \text{obs}(\text{perc}_i(e'_i)) \) of any environment initial state. According to the definition of the transition relation \( \rho \), there exists in \( A_P \) a transition from each \( e_0 \) for all available actions \( \alpha \) such that \( \delta_i(e_0, \alpha) \) is defined for \( Env_0 \). In particular, the transition \( (w, \vec{a}, w') \) is not permitted in \( A_P \) if either some action component \( \alpha_j \) is not allowed by agent’s protocol \( \text{pos} \) or it is not available in the environment \( Env_0, 0 < i \leq k \). Since \( \sigma \) is a solution of \( P \) (and thus allowed by \( Ag \)) it cannot be one of such cases. Point 2 holds for the same reason: there exists a transition in \( \rho \) for all available actions of each environment. Point 3 is just a consequence of \( \sigma \) being a solution of \( P \).

\[ \square \]

Checking whether \( \mathcal{L}(A_P) \neq \emptyset \) can be done \textsc{nlogspace} in the size of the automaton. Our automaton is exponential in the number of environments in \( E \), but its construction can be done on-the-fly while checking for non-emptiness. This gives us a \textsc{pspace} upperbound in the size of the original specification with explicit states. If we have a compact representation of those, then we get an \textsc{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 $\mathcal{P} = \langle A, E, G \rangle$ 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 $A$ and with $r_\ell = \langle \vec{e}_\ell, \vec{c}_\ell, \vec{g}_\ell, \equiv \rangle$ its $\ell$-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 $A$. Indeed, assume by contradiction that there exists some index $\ell'$ violating the assumption above. This implies that $\equiv_\ell \subseteq \equiv_{\ell'+1}$. 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 $\ell' \equiv_{\ell'+1}$ implies $i \equiv_\ell j$.

![Figure 4: Decision-tree like representation of a strategy.](image)

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 acceptence 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 features and hence it has to act blindly as in conformant planning. Notice that if our environment set includes only features and hence it has to act blindly as in conformant planning. Notice that if our environment set includes only features and hence it has to act blindly as in conformant planning. Notice that if our environment set includes only features and hence it has to act blindly as in conformant planning. Notice that if our environment set includes only features and hence it has to act blindly as in conformant planning.

![Figure 5: A resulting strategy execution.](image)

**Representing Strategies**

The technique described in the previous section provides, if a solution does exist, the run of $A$ 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 $\sigma$ for the agent $Ag$. The strategy $\sigma$ can be represented as a finite-state structure with nodes labeled with agent’s configuration and edges labeled with a condition-action rule $[\sigma | a]$, where $\sigma \in Obs$ and $a \in Act_\sigma$.

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 $a$, i.e. $a = obs(perc_i(e_i))$.

Hence, notice that a strategy $\sigma$ can be expressed by means of sequences, case-conditions and infinite iterations.

![Figure 6: Accepting run $r$ for Example 1.](image)

![Figure 7: Corresponding $G_r$.](image)

In other words, it can be represented as a graph $G_r = \langle N, E \rangle$ where $N$ is a set of nodes, $\lambda : N \rightarrow Conf$ 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 $r = w_0a_1w_1 \ldots a_nw_n$ with $m \leq n$ be a finite representation of the infinite run returned as witness. Let $r_\ell$ be the $\ell$-th state in $r$, whereas we denote with $r\mid_\ell$ the sub-run of $r$ starting from state $r_\ell$. 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 $w_m$; $loop(r_\ell) = true$ if $\ell = m$, false otherwise.

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

**UNPACK($r$, loopnode):**

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

463
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 \( \alpha \) taken from the current action vector.

From \( G_r \) we can derive a finite state strategy \( \sigma_f = \langle N, \text{succ}, \text{act}, n_0 \rangle \), where:

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

From \( \sigma_f \) we can derive an actual perfect-recall strategy \( \sigma : \text{Obs}^* \to \text{Act}_a \) as follows. We extend the deterministic function \( \text{succ} \) to observation histories \( h \in \text{Obs}^* \) of length \( \ell \) in the obvious manner. Then we define \( \alpha \) as the function: \( \sigma(h) = \text{act}(n, c, \text{last}(h)) \), where \( n = \text{succ}(\text{root}(G_r), h^{\ell-1}) \) is the prefix of \( h \) of length \( \ell-1 \) and \( c = \lambda(n) \) is the current configuration. Notice that such a 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 shown that the procedure above, based on the algorithm \( \text{UNPACK} \), 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 \( \mathcal{A}_P \) is depicted in Figure 6, from which a strategy \( \sigma \) can be unpacked. Strategy \( \sigma \) can be equivalently represented as in Figure 7 as a function from observation histories to actions. For instance, \( \sigma(c_0, \{\text{posA}, \text{nil}, \text{danger}, \text{nil}\}) = \text{goE} \). In particular, being all environments indistinguishable in the initial state (the agent receives the same observation \( \text{posA} \)), this strategy prescribes action \( \text{goE} \) for the three of them. Resulting states are such that both \( \text{Env}_1 \) and \( \text{Env}_3 \) provide perception \( \text{nil} \), whereas \( \text{Env}_2 \) provides perception \( \text{danger} \). Having received different observation histories so far, strategy \( \sigma \) is allowed to select different action for \( \text{Env}_2 \): \( \text{goE} \) for \( \text{Env}_2 \) and \( \text{wait} \) for \( \text{Env}_1 \) and \( \text{Env}_3 \). In fact, according to protocol \( \text{posA} \), action \( \text{wait} \) is not an option for \( \text{Env}_2 \), whereas action \( \text{goE} \) is not significant for \( \text{Env}_3 \), though it avoids an accident in \( \text{Env}_1 \). 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 \( \sigma_f = \langle N, \text{succ}, \text{act}, n_0 \rangle \). 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 \( \text{Conf} = \text{Conf} \times N \) (hence a new local state space \( \mathcal{L} \)).

Formally, given the original agent \( \mathcal{A}_g = \langle \text{Conf}, \text{Act}_a, \text{Obs}, \mathcal{L}, \text{poss}, \delta_a, c_0 \rangle \) and the finite state strategy \( \sigma_f = \langle N, \text{succ}, \text{act}, n_0 \rangle \), we construct a new agent \( \mathcal{A}_g' = \langle \text{Conf}, \text{Act}_a, \text{Obs}, \mathcal{L}, \text{poss}, \delta_a, c_0 \rangle \) where:

- \( \text{Act}_a \) and \( \text{Obs} \) are as in \( \mathcal{A}_g \);
- \( \text{Conf} = \text{Conf} \times N \) is the new set of configurations;
- \( \mathcal{L} = \text{Conf} \times \text{Obs} \) is the new local state space;
- \( \text{poss} : \mathcal{L} \to \text{Act}_a \) is an action-deterministic protocol defined as:
  \[
  \text{poss}((c, n), o) = \begin{cases} 
  \{\alpha\}, & \text{iff } \text{act}(n, c, o) = \alpha \\
  \emptyset, & \text{if } \text{act}(n, c, o) \text{ is undefined}; 
  \end{cases}
  \]
- \( \delta_a : \mathcal{L} \times \text{Act}_a \to \text{Conf} \) is the transition function, defined as:
  \[
  \delta_a((\langle c, n \rangle, o), a) = \langle \delta_a(c, o), \text{succ}(n, o, a) \rangle;
  \]
- \( c_0 = \langle c_0, n_0 \rangle \).

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

\[
\sigma : \text{Conf} \times \text{Obs} \to \text{Act}_a
\]

simply defined as: \( \sigma((c, n), o) = \text{poss}((c, n), o) \).

It remains to understand in what sense we can think the agent \( \mathcal{A}_g' \) as a refinement or customization of the agent \( \mathcal{A}_g \). 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 \((A_g)\) to simulate, i.e., copy move by move, another agent \((A_g')\).

Given the two agents \(A_g\) and \(A_g'\), a simulation relation is a relation \(S \subseteq L_1 \times L_2\) such that \((l_1, l_2) \in S\) implies that:

\[
\text{if } l_2 \xrightarrow{\alpha} l'_2 \text{ and } \alpha \in \text{poss}_2(l_2) \text{ then there exists } l'_1 \text{ such that } l_1 \xrightarrow{\alpha} l'_1 \text{ and } \alpha \in \text{poss}_1(l_1) \text{ and } (l'_1, l'_2) \in S.
\]

where \(l_i \xrightarrow{\alpha} l'_i\) iff \(l'_i\) is the agent configuration in \(l_i\) and \(c'_i = \delta_{\alpha}(l_i, \alpha)\). We say that agent \(A_g\) simulates agent \(A_g'\) iff there exists a simulation relation \(S\) such that \((l'_1, l'_2) \in S\) for each couple of initial local states \((l'_1, l'_2)\) with the same initial observation.

**Theorem 3.** Agent \(A_g\) simulates \(\overline{A_g}\).

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

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

**Theorem 4.** \(\overline{A_g}\) is nonblocking.

It follows from the fact that a strategy \(\sigma\) that is a solution for problem \(P\) is a prefix-closed function and it is allowed by \(A_g\). Hence, for any \(\bar{l} \in L\) reachable from any initial local state by applying \(\sigma\), we have \(\text{pos}(\bar{l}) \neq \emptyset\).

From Theorem 1 and results in previous sections we have:

**Theorem 5.** Any execution of agent \(A_g\) over each environment \(\text{Env}_i\) satisfies the original agent specification \(A_g\) 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 \(\text{Act}_{\text{env}} \subseteq \bigcap_{i=1, \ldots, k} \text{Act}_{\text{env}_i}\) and that all environment share the same alphabet of perceptions \(\text{Perc}\). Differently from before, we associate a distinct goal to each environment. We take as input alphabet of each goal specification \(G\), the set of environment’s state \(E_i\), i.e., \(G_i = \langle E_i, G_i, \gamma_i, \phi_{\text{acc}} \rangle\). All goals are thus intimately different, as they are strictly related to the specific environment. Intuitively, we require that a strategy for agent \(A_g\) satisfies, in all environments \(\text{Env}_i\), its corresponding goal \(G_i\).

In other words, \(\sigma\) is a solution of the generalized planning problem \(P = \langle A_g, E, G \rangle\) iff it is allowed by \(A_g\) and it generates, for each environment \(\text{Env}_i\), an infinite trace \(\tau_i\) that is accepted by \(G_i\).

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 \(A_g\) and \(E\) as defined before, and \(k\) goals \(G_i = \langle E_i, G_i, \gamma_i, \phi_{\text{acc}} \rangle\) we build the generalized Büchi automaton \(\mathcal{A}_P = \langle \mathcal{A}_{\text{act}}(W, w_0, \rho, W_{\text{acc}}) \rangle\) as follows. Notice that each automaton \(G_i\) has \(E_i\) as input alphabet.

- \(\mathcal{A}_{\text{act}}(W) = \langle \mathcal{A}_{\text{act}}(W) \rangle^k\) is the set of \(k\)-vectors of actions;
- \(W = E^k \times L^k \times G_1 \times \ldots \times G_k \times \varnothing(\equiv)\);
- \(w_0 = \langle e_1, \ldots, e_k, o_1, \ldots, o_k, g_0, \ldots, g_k, \equiv_0 \rangle\) where \(i \equiv 0\) j iff \(\text{obs}(\text{perc}_i(e_0)) = \text{obs}(\text{perc}_j(e_0))\);
- \(\langle e', e', g', \equiv' \rangle \in \rho(\langle e', e', g', \equiv \rangle)\) iff
  - if \(i = j\) then \(\alpha_i = \alpha_j\);
  - \(e' = \delta_i(e_i, \alpha_i)\);
  - \(g' = \gamma_i(g_i, \alpha_i)\);
  - \(i \equiv j \) iff \(i \equiv j \) and \(\text{obs}(\text{perc}_i(e_i)) = \text{obs}(\text{perc}_j(e_j))\).

- \(W_{\text{acc}} = \{E^k \times \text{Conf}^k \times \text{G}^k \times G_2 \times \ldots \times G_k \times \equiv , \ldots , E^k \times \text{Conf}^k \times G_1 \times \ldots \times G_{k-1} \times \text{G}^{\equiv} \times \equiv \}\)

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 \(\varnothing_G\), three distinct goals over the three environments. In particular for the car-train environments \(\text{Env}_1\) and \(\text{Env}_2\), we adopt the same kind of goal as before, but avoiding certain cells for environments, e.g., \(\varnothing_{G_1} = \varnothing \cup (e_{c1} \land (\varnothing \cup e_{c4}) \land \varnothing \cup (e_{c31} \land e_{c1}))\); \(\varnothing_{G_2} = \varnothing \cup (e_{c2} \land (\varnothing \cup e_{c4}) \land \varnothing \cup (e_{c23} \land e_{c2}))\). 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 \(\text{Env}_1\) and \(\text{Env}_2\) states mentioned above) that satisfies goal \(\varnothing_G\) over all environments as well as these three new goals (in particular, if a strategy satisfies \(\varnothing_G\) then it satisfies \(\varnothing_{G_2}\) 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 MC- MAS (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