

A Framework for Planning with Extended Goals under Partial Observability

Piergiorgio Bertoli and Alessandro Cimatti and Marco Pistore and Paolo Traverso

ITC-IRST, Via Sommarive 18, 38050 Povo, Trento, Italy

{bertoli,cimatti,pistore,traverso}@irst.itc.it

Abstract

Planning in nondeterministic domains with temporally extended goals under partial observability is one of the most challenging problems in planning. Subsets of this problem have been already addressed in the literature. For instance, planning for extended goals has been developed under the simplifying hypothesis of full observability. And the problem of a partial observability has been tackled in the case of simple reachability goals. The general combination of extended goals and partial observability is, to the best of our knowledge, still an open problem, whose solution turns out to be by no means trivial.

In this paper we do not solve the problem in its generality, but we perform a significant step in this direction by providing a solid basis for tackling it. Our first contribution is the definition of a general framework that encompasses both partial observability and temporally extended goals, and that allows for describing complex, realistic domains and significant goals over them. A second contribution is the definition of the K-CTL goal language, that extends CTL (a classical language for expressing temporal requirements) with a knowledge operator that allows to reason about the information that can be acquired at run-time. This is necessary to deal with partially observable domains, where only limited run-time “knowledge” about the domain state is available. A general mechanism for plan validation with K-CTL goals is also defined. This mechanism is based on a monitor, that plays the role of evaluating the truth of knowledge predicates.

Introduction

Planning in nondeterministic domains has been devoted increasing interest, and different research lines have been developed. On one side, planning algorithms for tackling temporally extended goals have been proposed in (Kabanza, Barbeau, & St-Denis 1997; Pistore & Traverso 2001; Dal Lago, Pistore, & Traverso 2002), motivated by the fact that many real-life problems require temporal operators for expressing complex goals. This research line is carried out under the assumption that the planning domain is fully observable. On the other side, in (Bertoli *et al.* 2001; Weld, Anderson, & Smith 1998; Bonet & Geffner 2000; Rintanen 1999) the hypothesis of full observability is relaxed in order to deal with realistic situations, where the plan

executor cannot access the whole status of the domain. The key difficulty is in dealing with the uncertainty arising from the inability to determine precisely at run-time what is the current status of the domain. These approaches are however limited to the case of simple reachability goals.

Tackling the problem of planning for temporally extended goals under the assumption of partial observability is not trivial. The goal of this paper is to settle a general framework that encompasses all the aspects that are relevant to deal with real-world domains and problems which feature partial observability and extended goals. This framework gives a precise definition of the problem, and will be a basis for solving it in its full complexity.

The framework we propose is based on the Planning as Model Checking paradigm. We give a general notion of planning domain, in terms of finite state machine, where actions can be nondeterministic, and different forms of sensing can be captured. We define a general notion of plan, that is also seen as a finite state machine, with internal control points that allow to encode sequential, conditional, and iterative behaviors. The conditional behavior is based on sensed information, i.e., information that becomes available during plan execution. By connecting a plan and a domain, we obtain a closed system, that induces a (possibly infinite) computation tree, representing all the possible executions. Temporally extended goals are defined as CTL formulas. In this framework, the standard machinery of model checking for CTL temporal logic defines when a plan satisfies a temporally extended goal under partial observability. As a side result, this shows that a standard model checking tool can be applied as a black box to the validation of complex plans even in the presence of limited observability.

Unfortunately, CTL is not adequate to express goals in presence of partial observability. Even in the simple case of conformant planning, i.e., when a reachability goal has to be achieved with no information available at run-time, CTL is not expressive enough. This is due to the fact that the basic propositions in CTL only refer to the status of the world, but do not take into account the aspects related to “knowledge”, i.e., what is known at run-time. In fact, conformant planning is the problem of finding a plan after which we *know* that a certain condition is achieved. In order to overcome this limitation, we define the K-CTL goal language, obtained by extending CTL with a knowledge operator, that allows to

express knowledge atoms, i.e., what is known at a certain point in the execution. Then, we provide a first practical solution to the problem of checking if a plan satisfies a K-CTL goal. This is done by associating a given K-CTL goal with a suitable monitor, i.e., an observer system that is able to recognize the truth of knowledge atoms. Standard model checking techniques can be then applied to the domain-plan system enriched with the monitor.

The work presented in this paper focuses on setting the framework and defining plan validation procedures, and does not tackle the problem of plan synthesis. Still, the basic concepts presented in this paper formally distinguish what is known at planning time versus what is known at run time, and provide a solid basis for tackling the problem of plan synthesis for extended goals under partial observability.

The paper is structured as follows. First we provide a formal framework for partially observable, nondeterministic domains, and for plans over them. Then we incrementally define CTL goals and K-CTL goals; for each of those classes of goals, we describe a plan validation procedure. We wrap up with some concluding remarks and future and related work.

The framework

In our framework, a domain is a model of a generic system, such as a power plant or an aircraft, with its own dynamics. The plan can control the evolutions of the domain by triggering *actions*. We assume that, at execution time, the state of the domain is only partially visible to the plan; the part of a domain state that is visible is called the *observation* of the state. In essence, planning is building a suitable plan that can guide the evolutions of the domain in order to achieve the specified goals.

Planning domains

A planning domain is defined in terms of its *states*, of the *actions* it accepts, and of the possible *observations* that the domain can exhibit. Some of the states are marked as valid *initial states* for the domain. A *transition function* describes how (the execution of) an action leads from one state to possibly many different states. Finally, an *observation function* defines what observations are associated to each state of the domain.

Definition 1 (planning domain) A nondeterministic planning domain with partial observability is a tuple $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$, where:

- \mathcal{S} is the set of states.
- \mathcal{A} is the set of actions.
- \mathcal{U} is the set of observations.
- $\mathcal{I} \subseteq \mathcal{S}$ is the set of initial states; we require $\mathcal{I} \neq \emptyset$.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{S}}$ is the transition function; it associates to each current state $s \in \mathcal{S}$ and to each action $a \in \mathcal{A}$ the set $\mathcal{T}(s, a) \subseteq \mathcal{S}$ of next states.
- $\mathcal{X} : \mathcal{S} \rightarrow 2^{\mathcal{U}}$ is the observation function; it associates to each state s the set of possible observations $\mathcal{X}(s) \subseteq \mathcal{U}$.

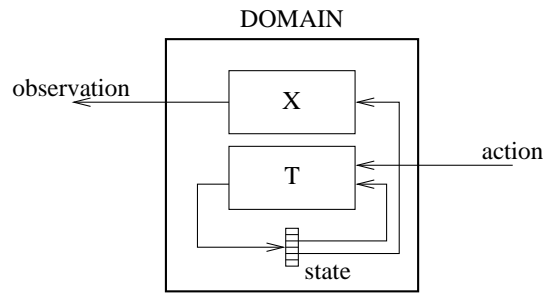


Figure 1: The model of the domain.

We say that action a is executable in state s if $\mathcal{T}(s, a) \neq \emptyset$. We require that in each state $s \in \mathcal{S}$ there is some executable action, that is some $a \in \mathcal{A}$ such that $\mathcal{T}(s, a) \neq \emptyset$. We also require that some observation is associated to each state $s \in \mathcal{S}$, that is, $\mathcal{X}(s) \neq \emptyset$.

A picture of the model of the domain corresponding to this definition is given in Figure 1. Technically, a domain is described as a nondeterministic Moore machine, whose outputs (i.e., the observations) depend only on the current state of the machine, not on the input action. Uncertainty is allowed in the initial state and in the outcome of action execution. Also, the observation associated to a given state is not unique. This allows modeling noisy sensing and lack of information.

Notice that the definition provides a general notion of domain, abstracting away from the language that is used to describe it. For instance, a planning domain is usually defined in terms of a set of *fluents* (or state variables), and each state corresponds to an assignment to the fluents. Similarly, the possible observations of the domain, that are primitive entities in the definition, can be presented by means of a set of *observation variables*, as in (Bertoli *et al.* 2001): each observation variable can be seen as an input port in the plan, while an observation is defined as a valuation to all the observation variables. The definition of planning domain does not allow for a direct representation of *action-dependent* observations, that is, observations that depend on the last executed action. However, these observations can be easily modeled by representing explicitly in the state of the domain (the relevant information on) the last executed action.

In the following example, that will be used throughout the paper, we will outline the different aspects of the defined framework.

Example 2 Consider the domain represented in Figure 2. It consists of a ring of N rooms. Each room contains a light that can be on or off, and a button that, when pressed, switches the status of the light. A robot may move between adjacent rooms (actions *go-right* and *go-left*) and switch the lights (action *switch-light*). Uncertainty in the domain is due to an unknown initial room and initial status of the lights. Moreover, the lights in the rooms not occupied by the robot may be nondeterministically switched on without the direct intervention of the robot (if a light is already on, instead, it can be turned off only by the robot). The domain is

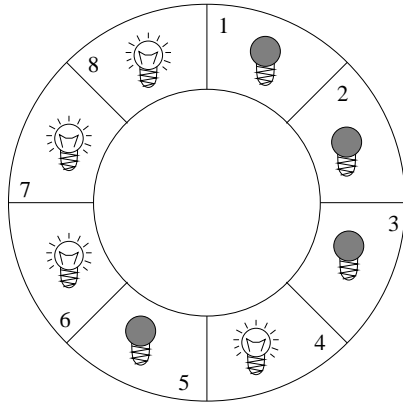


Figure 2: A simple domain.

only partially observable: the rooms are indistinguishable, and, in order to know the status of the light in the current room, the robot must perform a **sense** action.

A state of the domain is defined in terms of the following fluents:

- fluent **room**, that ranges from 1 to N , describes in which room the robot is currently in;
- boolean fluents **light-on** $[i]$, for $i \in \{1, \dots, N\}$, describe whether the light in room i is on;
- boolean fluent **sensed**, describes whether last action was a **sense** action.

Any state with fluent **sensed** false is a possible initial state.

The actions are **go-left**, **go-right**, **switch-light**, **sense**, and **wait**. Action **wait** corresponds to the robot doing nothing during a transition (the state of the domain may change only due to the lights that may be turned on without the intervention of the robot). The effects of the other actions have been already described.

The observation is defined in terms of observation variable **light**. If fluent **sensed** is true, then observation variable **light** is true if and only if the light is on in the current room. If fluent **sensed** is false (no sensing has been done in the last action), then observation **light** may be nondeterministically true or false.

The mechanism of observations allowed by the model presented in Definition 1 is very general. It can model *no observability* and *full observability* as special cases. *No observability* (conformant planning) is represented by defining $\mathcal{U} = \{\bullet\}$ and $\mathcal{X}(s) = \{\bullet\}$ for each $s \in \mathcal{S}$. That is, observation \bullet is associated to all states, thus conveying no information. *Full observability* is represented by defining $\mathcal{U} = \mathcal{S}$ and $\mathcal{X}(s) = \{s\}$. That is, the observation carries all the information contained in the state of the domain.

Plans

Now we present a general definition of plans, that encode sequential, conditional and iterative behaviors, and are expressive enough for dealing with partial observability and with extended goals. In particular, we need plans where the

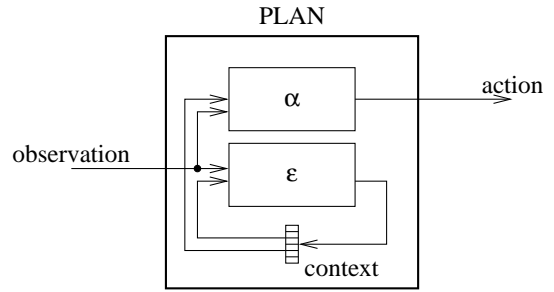


Figure 3: The model of the plan.

selection of the action to be executed depends on the observations, and on an “internal state” of the executor, that can take into account, e.g., the knowledge gathered during the previous execution steps. A plan is defined in terms of an *action function* that, given an observation and a *context* encoding the internal state of the executor, specifies the action to be executed, and in terms of a *context function* that evolves the context.

Definition 3 (plan) A plan for planning domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ is a tuple $\Pi = \langle \Sigma, \sigma_0, \alpha, \epsilon \rangle$, where:

- Σ is the set of plan contexts.
- $\sigma_0 \in \Sigma$ is the initial context.
- $\alpha : \Sigma \times \mathcal{U} \rightarrow \mathcal{A}$ is the action function; it associates to a plan context c and an observation o an action $a = \alpha(c, o)$ to be executed.
- $\epsilon : \Sigma \times \mathcal{U} \rightarrow \Sigma$ is the context evolutions function; it associates to a plan context c and an observation o a new plan context $c' = \epsilon(c, o)$.

A picture of the model of plans is given in Figure 3. Technically, a plan is described as a Mealy machine, whose outputs (the action) depends in general on the inputs (the current observation). Functions α and ϵ are deterministic (we do not consider nondeterministic plans), and can be partial, since a plan may be undefined on the context-observation pairs that are never reached during execution.

Example 4 We consider two plans for the domain of Figure 2. According to plan Π_1 , the robot moves cyclically through the rooms, and turns off the lights whenever they are on. The plan is cyclic, that is, it never ends. The plan has three contexts E , S , and L , corresponding to the robot having just entered a room (E), the robot having sensed the light (S), and the robot being about to leave the room after switching the light (L). The initial context is E . Functions α and ϵ for Π_1 are defined by the following table:

c	o	$\alpha(c, o)$	$\epsilon(c, o)$
E	any	sense	S
S	$light = \top$	switch-light	L
S	$light = \perp$	go-right	E
L	any	go-right	E

In plan Π_2 , the robot traverses all the rooms and turns on the lights; the robot stops once all the rooms have been visited. The plan has contexts of the form (E, i) , (S, i) , and

(L, i) , where i represents the number of rooms to be visited. The initial context is $(E, N-1)$, where N is the number of rooms. Functions α and ϵ for Π_2 are defined by the following table:

c	o	$\alpha(c, o)$	$\epsilon(c, o)$
(E, i)	any	sense	(S, i)
(S, i)	light = \perp	switch-light	(L, i)
$(S, 0)$	light = \top	wait	$(L, 0)$
$(S, i+1)$	light = \top	go-right	(E, i)
$(L, 0)$	any	wait	$(L, 0)$
$(L, i+1)$	any	go-right	(E, i)

Plan execution

Now we discuss plan execution, that is, the effects of running a plan on the corresponding planning domain. Since both the plan and the domain are finite state machines, we can use the standard techniques for synchronous compositions defined in model checking. That is, we can describe the execution of a plan over a domain in terms of transitions between configurations that describe the state of the domain and of the plan. This idea is formalized in the following definition.

Definition 5 (configuration) A configuration for domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ and plan $\Pi = \langle \Sigma, \sigma_0, \alpha, \epsilon \rangle$ is a tuple (s, o, c, a) such that:

- $s \in \mathcal{S}$,
- $o \in \mathcal{X}(s)$,
- $c \in \Sigma$, and
- $a = \alpha(c, o)$.

Configuration (s, o, c, a) may evolve into configuration (s', o', c', a') , written $(s, o, c, a) \rightarrow (s', o', c', a')$, if $s' \in \mathcal{T}(s, a)$, $o' \in \mathcal{X}(s')$, $c' = \epsilon(c, o)$, and $a' = \alpha(c', o')$. Configuration (s, o, c, a) is initial if $s \in \mathcal{I}$ and $c = \sigma_0$. The reachable configurations for domain \mathcal{D} and plan Π are defined by the following inductive rules:

- if (s, o, c, a) is initial, then it is reachable;
- if (s, o, c, a) is reachable and $(s, o, c, a) \rightarrow (s', o', c', a')$, then (s', o', c', a') is also reachable.

Notice that we include the observations and the actions in the configurations. In this way, not only the current states of the two finite states machines, but also the information exchanged by these machines are explicitly represented. In the case of the observations, this explicit representation is necessary in order to take into account that more than one observation may correspond to the same state.

We are interested in plans that define an action to be executed for each reachable configuration. These plans are called *executable*.

Definition 6 (executable plan) Plan Π is executable on domain \mathcal{D} if:

1. if $s \in \mathcal{I}$ and $o \in \mathcal{X}(s)$ then $\alpha(\sigma_0, o)$ is defined;
- and if for all the reachable configurations (s, o, c, a) :
2. $\mathcal{T}(s, a) \neq \emptyset$;
 3. $\epsilon(c, o)$ is defined;

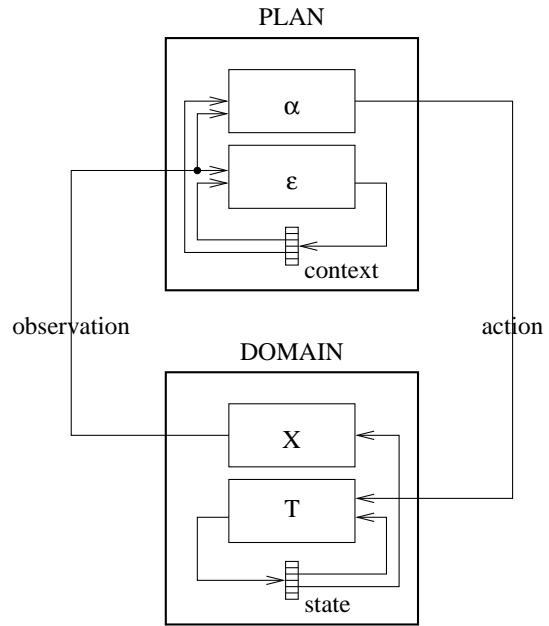


Figure 4: Plan execution.

4. if $s' \in \mathcal{T}(s, a)$, $o' \in \mathcal{X}(s')$, and $c' = \epsilon(c, o)$, then $\alpha(c', o')$ is defined.

Condition 1 guarantees that the plan defines an action for all the initial states (and observations) of the domain. The other conditions guarantee that, during plan execution, a configuration is never reached where the execution cannot proceed. More precisely, condition 2 guarantees that the action selected by the plan is executable on the current state. Condition 3 guarantees that the plan defines a next context for each reachable configuration. Condition 4 is similar to condition 1 and guarantees that the plan defines an action for all the states and observations of the domain that can be reached from the current configuration.

The executions of a plan on a domain correspond to the synchronous executions of the two machines corresponding to the domain and the plan, as shown in Figure 4. At each time step, the flow of execution proceeds as follows. The execution starts from a configuration that defines the current domain state, observation, context, and action. The new state of the domain is determined by function \mathcal{T} from the current state and action. The new observation is then determined by applying nondeterministic function \mathcal{X} to the new state. Based on the current context and observation, the plan determines the next context applying function ϵ . And, finally, the plan determines the new action to be executed by applying function α to the new context and observation. At the end of the cycle, the newly computed values for the domain state, the observation, the context, and the action define the value of the new configuration.

An execution of the plan is basically a sequence of subsequent configurations. Due to the nondeterminism in the domain, we may have an infinite number of different executions of a plan. We provide a finite presentation of these

executions with an *execution structure*, i.e., a Kripke Structure (Emerson 1990) whose set of states is the set of reachable configurations of the plan, and whose transition relation corresponds to the transitions between configurations.

Definition 7 (execution structure) *The execution structure corresponding to domain \mathcal{D} and plan Π is the Kripke structure $K = \langle Q, Q_0, R \rangle$, where:*

- Q is the set of reachable configurations;
- $Q_0 = \{(s, o, \sigma_0, a) \in Q : s \in \mathcal{I} \wedge o \in \mathcal{X}(s) \wedge a = \alpha(\sigma_0, o)\}$ are the initial configurations;
- $R = \{((s, o, c, a), (s', o', c', a')) \in Q \times Q : (s, o, c, a) \rightarrow (s', o', c', a')\}$.

Temporally extended goals: CTL

Extended goals are expressed with temporal logic formulas. In most of the works on planning with extended goals (see, e.g., (Kabanza, Barbeau, & St-Denis 1997; de Giacomo & Vardi 1999; Bacchus & Kabanza 2000)), Linear Time Logic (LTL) is used as goal language. LTL provides temporal operators that allow one to define complex conditions on the sequences of states that are possible outcomes of plan execution. Following (Pistore & Traverso 2001), we use Computational Tree Logic (CTL) instead. CTL provides the same temporal operators of LTL, but extends them with universal and existential path quantifiers that provide the ability to take into account the non-determinism of the domain.

We assume that a set \mathcal{B} of basic propositions is defined on domain \mathcal{D} . Moreover, we assume that for each $b \in \mathcal{B}$ and $s \in \mathcal{S}$, predicate $s \models_0 b$ holds if and only if basic proposition b is true on state s . In the case of the domain of Figure 2, possible basic propositions are $\text{light-on}[i]$, that is true in those states where the light is on in room i , or $\text{room} = i$, that is true if the robot is in room i .

Definition 8 (CTL) *The goal language CTL is defined by the following grammar, where b is a basic proposition:*

$$\begin{aligned} g &::= p \mid g \wedge g \mid g \vee g \mid \text{AX } g \mid \text{EX } g \\ &\quad \text{A}(g \text{ U } g) \mid \text{E}(g \text{ U } g) \mid \text{A}(g \text{ W } g) \mid \text{E}(g \text{ W } g) \\ p &::= b \mid \neg p \mid p \wedge p \end{aligned}$$

CTL combines temporal operators and path quantifiers. “X”, “U”, and “W” are the “next time”, “(strong) until”, and “weak until” temporal operators, respectively. “A” and “E” are the universal and existential path quantifiers, where a path is an infinite sequence of states. They allow us to specify requirements that take into account nondeterminism. Intuitively, the formula $\text{AX } g$ means that g holds in every immediate successor of the current state, while the formula $\text{EX } g$ means that g holds in some immediate successor. The formula $\text{A}(g_1 \text{ U } g_2)$ means that for every path there exists an initial prefix of the path such that g_2 holds at the last state of the prefix and g_1 holds at all the other states along the prefix. The formula $\text{E}(g_1 \text{ U } g_2)$ expresses the same condition, but only on some of the paths. The formulas $\text{A}(g_1 \text{ W } g_2)$ and $\text{E}(g_1 \text{ W } g_2)$ are similar to $\text{A}(g_1 \text{ U } g_2)$ and $\text{E}(g_1 \text{ U } g_2)$, but allow for paths where g_1 holds in all the states and g_2 never holds. Formulas $\text{AF } g$ and $\text{EF } g$ (where the temporal

operator “F” stands for “future” or “eventually”) are abbreviations of $\text{A}(\text{T U } g)$ and $\text{E}(\text{T U } g)$, respectively. $\text{AG } g$ and $\text{EG } g$ (where “G” stands for “globally” or “always”) are abbreviations of $\text{A}(g \text{ W } \perp)$ and $\text{E}(g \text{ W } \perp)$, respectively.

A remark is in order. Even if negation \neg is allowed only in front of basic propositions, it is easy to define $\neg g$ for a generic CTL formula g , by “pushing down” the negations: for instance $\neg \text{AX } g \equiv \text{EX } \neg g$ and $\neg \text{A}(g_1 \text{ W } g_2) \equiv \text{E}(\neg g_2 \text{ U } (\neg g_1 \wedge \neg g_2))$.

Goals as CTL formulas allow us to specify different classes of requirements on plans. Let us consider first some examples of *reachability goals*. $\text{AF } g$ (“reach g ”) states that a condition should be guaranteed to be reached by the plan, in spite of nondeterminism. $\text{EF } g$ (“try to reach g ”) states that a condition might possibly be reached, i.e., there exists at least one execution that achieves the goal. A reasonable reachability requirement that is stronger than $\text{EF } g$ is $\text{A}(\text{EF } g \text{ W } g)$: it allows for those execution loops that have always a possibility of terminating, and when they do, the goal g is guaranteed to be achieved.

We can distinguish also among different kinds of *maintainability goals*, e.g., $\text{AG } g$ (“maintain g ”), $\text{AG } \neg g$ (“avoid g ”), $\text{EG } g$ (“try to maintain g ”), and $\text{EG } \neg g$ (“try to avoid g ”). The “until” operators $\text{A}(g_1 \text{ U } g_2)$ and $\text{E}(g_1 \text{ U } g_2)$ can be used to express the reachability goals g_2 with the additional requirement that property g_1 must be maintained until the desired condition is reached.

We can also compose reachability and maintainability goals in arbitrary ways. For instance, $\text{AF AG } g$ states that a plan should guarantee that all executions reach eventually a set of states where g can be maintained. The weaker goal $\text{EF AG } g$ requires that there exists a possibility to reach a set of states where g can be maintained. As a further example, the goal $\text{AG EF } g$ intuitively means “maintain the possibility of reaching g ”.

Notice that in all examples above, the ability of composing formulas with universal and existential path quantifiers is essential. Logics like LTL that do not provide this ability cannot express these kinds of goals.

Given an execution structure K and an extended goal g , we now define when a goal g is true in (s, o, c, a) , written $K, (s, o, c, a) \models g$ by using the standard semantics for CTL formulas over the Kripke Structure K .

Definition 9 (semantics of CTL) *Let K be a Kripke structure with configurations as states. We extend \models_0 to propositions as follows:*

- $s \models_0 \neg p$ if not $s \models_0 p$;
- $s \models_0 p \wedge p'$ if $s \models_0 p$ and $s \models_0 p'$.

We define $K, q \models g$ as follows:

- $K, q \models p$ if $q = (s, o, c, a)$ and $s \models_0 p$.
- $K, q \models g \wedge g'$ if $K, q \models g$ and $K, q \models g'$.
- $K, q \models g \vee g'$ if $K, q \models g$ or $K, q \models g'$.
- $K, q \models \text{AX } g$ if for all q' , if $q \rightarrow q'$ then $K, q' \models g$.
- $K, q \models \text{EX } g$ if there is some q' such that $q \rightarrow q'$ and $K, q' \models g$.

- $K, q \models A(g \cup g')$ if for all $q = q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \dots$ there is some $i \geq 0$ such that $K, q_i \models g'$ and $K, q_j \models g$ for all $0 \leq j < i$.
- $K, q \models E(g \cup g')$ if there is some $q = q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \dots$ and some $i \geq 0$ such that $K, q_i \models g'$ and $K, q_j \models g$ for all $0 \leq j < i$.
- $K, q \models A(g \cup W g')$ if for all $q = q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \dots$, either $K, q_j \models g$ for all $j \geq 0$, or there is some $i \geq 0$ such that $K, q_i \models g'$ and $K, q_j \models g$ for all $0 \leq j < i$.
- $K, q \models E(g \cup W g')$ if there is some $q = q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \dots$ such that either $K, q_j \models g$ for all $j \geq 0$, or there is some $i \geq 0$ such that $K, q_i \models g'$ and $K, q_j \models g$ for all $0 \leq j < i$.

We define $K \models g$ if $K, q_0 \models g$ for all the initial configurations $q_0 \in Q_0$ of K .

Plan validation

The definition of when a plan satisfies a goal follows.

Definition 10 (plan validation for CTL goals) Plan Π satisfies CTL goal g on domain \mathcal{D} , written $\Pi \models_{\mathcal{D}} g$, if $K \models g$, where K is the execution structure corresponding to \mathcal{D} and Π .

In the case of CTL goals, the *plan validation* task amounts to CTL model checking. Given a domain \mathcal{D} and a plan Π , the corresponding execution structure K is built as described in Definition 7 and standard model checking algorithms are run on K in order to check whether it satisfies goal g . This simple consideration has an important consequence: the problem of plan validation under partial observability can be tackled with standard model checking machinery, and with existing model checking tools.

We describe now some goals for the domain of Figure 2. We recall that the initial room of the robot is uncertain, and that light can be turned on (but not off) without the intervention of the robot.

Example 11 The first goal we consider is

$$\text{AF } (\neg \text{light-on}[3]),$$

which requires that the light of room 3 is eventually off. Plan Π_1 satisfies this goal: eventually, the robot will be in room 3 and will turn out the light if it is on.

There is no plan that satisfies to following goal:

$$\text{AF AG } (\neg \text{light-on}[3]),$$

which requires that the light in room 3 is turned off and stays then off forever. This can be only guaranteed if the robot stays in room 3 forever, and it is impossible to guarantee this condition in this domain: due to the partial observability of the domain, the robot does never know it is in room 3.

Plan Π_1 satisfies the following goal

$$\bigwedge_{i \in \{1, \dots, N\}} \text{AG AF } (\neg \text{light-on}[i]),$$

which requires that the light in every room is turned off infinitely often. On the other hand, it does not satisfy the following goal

$$\text{AG AF } \bigwedge_{i \in \{1, \dots, N\}} (\neg \text{light-on}[i]),$$

which requires that the lights in all the rooms are off at the same time infinitely often. Indeed, the nondeterminism in the domain may cause lights to turn on at any time.

While plan Π_1 does not guarantee that all the lights will be eventually off, it always leaves open the possibility that such a configuration will be eventually reached. That is, plan Π_1 satisfies the following goal

$$\text{AG EF } \bigwedge_{i \in \{1, \dots, N\}} (\neg \text{light-on}[i]),$$

which asserts that in each moment (AG) there is the possibility of reaching (EF) the desired configuration.

Finally, consider the goal

$$\text{AG AF } \bigwedge_{i \in \{1, \dots, N\}} (\text{light-on}[i]),$$

which requires that the lights in all the rooms are on at the same time infinitely often. It is satisfied by plan Π_2 : once all the rooms have been explored, and the lights have been turned on, they will stay on forever.

Goals over actions and observations

The CTL formulas that we have considered so far can only express properties on the evolution of the states of the domains. The proposed approach, however, can be easily extended in order to allow for formulas that express properties also on the observations and on the executed actions. We have only to extend to observations and actions the interpretation of basic propositions b . This allows for basic propositions like *light*, that is true if the robot observes that the light is on in the current room, or like *switch-light*, that is true if the robot is going to switch the light.

Formally, we assume that predicate $s, o, a \models_0 b$ holds if and only if basic proposition b is true on state s , observation o , and action a ; moreover, we define:

- $K, q \models p$ if $q = (s, o, c, a)$ and $s, o, a \models_0 p$

with \models_0 extended to generic propositions as in Definition 9.

As shown by the following example, the possibility of speaking of actions and observations in the goals is very useful in planning, in particular for expressing conditions on the valid behaviors of the plan.

Example 12 Goal

$$\text{AG } (\text{sensed} \wedge \text{light} \rightarrow \text{switch-light})$$

expresses the requirement that, if the robot has just sensed the current status of the light, and if it is on, then the robot has to turn it immediately off.

As a further example, goal

$$\text{AG } (\text{go-right} \vee \text{go-left} \rightarrow \text{AX sense})$$

requires that, as soon as the robot moves into a new room (performing actions *go-right* or *go-left*), then it has to sense the status of the light.

Goals over knowledge: K-CTL

Under the hypothesis of partial observability, CTL is not adequate to express many interesting goals. Consider for instance the first goal in Example 11. Notice that the robot will never “know” when condition $\neg\text{light-on}[3]$ holds. In fact, the robot cannot detect when it is in room 3, and once that room is left, the light can be turned on again. The inadequacy of CTL is related with the limited knowledge that the plan execution has to face at run-time, because of different forms of uncertainty (e.g., in the initial condition, and in the execution of actions) that cannot be ruled out by the partial observability. In order to tackle this problem, in this section we extend CTL with a knowledge operator $\mathbf{K}p$. Goal $\mathbf{K}p$ expresses the fact that the executor *knows*, or *believes*, that all the possible current states of the domain, that are compatible with the past history and the past observations, satisfy condition p . This allows, for instance, for expressing reachability under partial observability, by stating a goal of the kind $\text{AF } \mathbf{K}g$.

Definition 13 (K-CTL) *The goal language K-CTL is defined by the following grammar:*

$$\begin{aligned} g &::= p \mid \mathbf{K}p \mid g \wedge g \mid g \vee g \mid \text{AX}g \mid \text{EX}g \\ &\quad \text{A}(g \text{ U } g) \mid \text{E}(g \text{ U } g) \mid \text{A}(g \text{ W } g) \mid \text{E}(g \text{ W } g) \\ p &::= b \mid \neg p \mid p \wedge p \end{aligned}$$

In order to define when a plan satisfies a given K-CTL goal, we have to extend the execution structure with an additional piece of information, called *belief state*. A belief state is a set of possible candidate states of the domain that we cannot distinguish given the past actions and the observations collected so far.

Definition 14 (bs-configuration) *A bs-configuration for domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ and plan $\Pi = \langle \Sigma, \sigma_0, \alpha, \epsilon \rangle$ is a tuple (s, o, c, a, bs) such that (s, o, c, a) is a configuration for \mathcal{D} and Π and $bs \in 2^{\mathcal{S}}$. We require:*

- $s \in bs$ (the current state must belong to the belief state);
- if $\bar{s} \in bs$ then $o \in \mathcal{X}(\bar{s})$ (the states in the belief state must be compatible with the observed output).

Bs-configuration (s, o, c, a, bs) may evolve into bs-configuration (s', o', c', a', bs') , written $(s, o, c, a, bs) \rightarrow (s', o', c', a', bs')$, if $(s, o, c, a) \rightarrow (s', o', c', a')$ and $bs' = \{\bar{s}' : \exists \bar{s} \in bs. \bar{s}' \in \mathcal{T}(\bar{s}, a) \wedge o' \in \mathcal{X}(\bar{s}')\}$. Bs-configuration (s, o, c, a, bs) is initial if $s \in \mathcal{I}$, $c = \sigma_0$, and $bs = \{\bar{s} \in \mathcal{I} : o \in \mathcal{X}(\bar{s})\}$. The reachable bs-configurations are defined by trivially extending Definition 5.

Definition 15 (semantics of K-CTL) *Let K be a bs-execution structure, namely a Kripke structures with bs-configurations as states. We define $K, q \models g$ by extending Definition 9 as follows:*

- $K, q \models \mathbf{K}p$ if $q = (s, o, c, a, bs)$ and $\bar{s} \models_0 p$ for all $\bar{s} \in bs$.

We define $K \models g$ if $K, q_0 \models g$ for all the initial configurations q_0 of K .

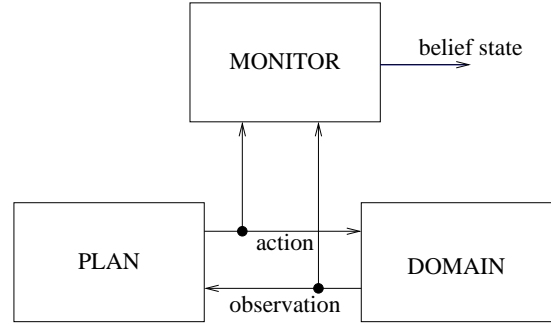


Figure 5: Monitor.

Plan validation for K-CTL goals

Also in the case of K-CTL, the definition of when a plan satisfies a goal is reduced to model checking.

Definition 16 (plan validation for K-CTL goals) *Plan Π satisfies K-CTL goal g on domain \mathcal{D} , written $\Pi \models_{\mathcal{D}} g$, if $K \models g$, where K is the bs-execution structure corresponding to \mathcal{D} and Π .*

We consider now an additional set of K-CTL goals.

Example 17 *In Example 11 we have seen that plan Π_1 satisfies goal $\text{AF}(\neg\text{light-on}[3])$. However, it does not satisfy goal*

$$\text{AF } \mathbf{K}(\neg\text{light-on}[3]).$$

In fact, this goal cannot be satisfied by any plan: due to the uncertainty on the room occupied by the robot, there is no way to “know” when the light in room 3 is turned off.

Goal

$$\text{AF } \mathbf{K}(\text{light-on}[3]),$$

instead, is satisfied by Π_2 . Even if it is not possible to know when the robot is turning on the light in room 3, we “know” for sure that the light is on once the robot has visited all the rooms. Plan Π_2 satisfies also the more complex goal

$$\bigwedge_{i \in \{1, \dots, N\}} \text{AF } \mathbf{K}(\text{light-on}[i]).$$

According to Definition 16, the problem of checking whether a plan satisfies a K-CTL goal g is reduced to model checking formula g on the bs-execution structure corresponding to the plan. While theoretically sound, this approach is not practical, since the number of possible belief states for a given planning domain is exponential in the number of its states. This makes the exploration of a bs-execution structure infeasible for non-trivial domains.

In order to overcome this limitation, in this section we introduce a different approach for plan validation. This approach is based on the concept of *monitor*. A monitor is a machine that observes the execution of the plan on the domain and reports a belief state, i.e., a set of possible current states of the domain (see Figure 5). Differently from the belief states that appear in a bs-configuration, the belief states reported by the monitor may be a super-set of the states that are compatible with the past history. As we will see, it is this possibility of approximating the possible current states that makes monitors usable in practice for validating plans.

Definition 18 (monitor) A monitor for a domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ is a tuple $\mathcal{M} = \langle \mathcal{MS}, m_0, \mathcal{MT}, \mathcal{MO} \rangle$, where:

- \mathcal{MS} is the set of states of the monitor.
- $m_0 \in \mathcal{MS}$ is the initial state of the monitor.
- $\mathcal{MT} : \mathcal{MS} \times \mathcal{U} \times \mathcal{A} \rightarrow \mathcal{MS}$ is the transition function of the monitor; it associates to state m of the monitor, observation o , and action a , an updated state of the monitor $m' = \mathcal{MT}(m, o, a)$.
- $\mathcal{MO} : \mathcal{MS} \times \mathcal{U} \rightarrow 2^{\mathcal{S}}$ is the output function of the monitor; it associates to each state m of the monitor and observation o the corresponding belief state $\mathcal{MO}(m, o)$.

Definition 19 (m-configuration) A m-configuration for domain \mathcal{D} , plan Π and monitor \mathcal{M} is a tuple (s, o, c, a, m) such that $s \in \mathcal{S}$, $o \in \mathcal{X}(s)$, $c \in \Sigma$, and $m \in \mathcal{MS}$. M-configuration (s, o, c, a, m) may evolve into m-configuration (s', o', c', a', m') , written $(s, o, c, a, m) \rightarrow (s', o', c', a', m')$, if: $(s, o, c, a) \rightarrow (s', o', c', a')$ and $m' = \mathcal{MT}(m, o, a)$. M-configuration (s, o, c, a, m) is initial if $s \in \mathcal{I}$, $c = \sigma_0$, and $m = m_0$. The reachable m-configurations are defined by trivially extending Definition 5.

We say that a monitor is *correct* for a given domain and plan if the belief state reported by the monitor after a certain evolution contains *all* the states that are compatible with the observation gathered during the evolution. In the following definition, this property is expressed by requiring that there are no computations along which a state of the domain is reached that is not contained in the belief state reported by the monitor.

Definition 20 (correct monitor) Monitor \mathcal{M} is correct for domain \mathcal{D} and plan Π if the following conditions holds for all the reachable m-configurations (s, o, c, a, m) :

- $s \in \mathcal{MO}(m, o)$;
- $\mathcal{MT}(m, o, a)$ is defined.

From now on we consider only correct monitors.

We now define when a triple domain-plan-monitor satisfies a given K-CTL goal g . We start by defining the Kripke structure corresponding by the synchronous execution of the machines corresponding to domain, plan, and monitor.

Definition 21 (m-execution structure) The m-execution structure corresponding to domain \mathcal{D} , plan Π , and monitor \mathcal{M} is the Kripke structure $K = \langle Q, Q_0, R \rangle$, where:

- Q is the set of reachable m-configurations;
- Q_0 are the initial m-configurations;
- $R = \{((s, o, c, a, m), (s', o', c', a', m')) \in Q \times Q : (s, o, c, a, m) \rightarrow (s', o', c', a', m')\}$.

The validity of a K-CTL formula on a m-execution structure K is defined as in Definition 15, with the exception of the case of goals $\mathbf{K}p$, where:

- $K, q \models \mathbf{K}p$ if $q = (s, o, c, a, m)$ and $\bar{s} \models_0 p$ for all $\bar{s} \in \mathcal{MO}(m, o)$.

Definition 22 (plan validation using monitors) Plan Π satisfies K-CTL goal g on domain \mathcal{D} according to monitor \mathcal{M} , written $\Pi \models_{\mathcal{D}, \mathcal{M}} g$, if $K \models g$, where K is the m-execution structure corresponding to \mathcal{D} , Π , and \mathcal{M} .

The possibility of using monitors for plan validation is guaranteed by the following theorem.

Theorem 23 Plan Π satisfies K-CTL goal g on domain \mathcal{D} if and only if there is a correct monitor \mathcal{M} for \mathcal{D} and Π such that $\Pi \models_{\mathcal{D}, \mathcal{M}} g$.

The proof of this theorem is simple. For the *if* implication, it is sufficient to notice that, for any reachable m-configuration (s, o, c, a, m) , the output $\mathcal{MO}(m, o)$ of a correct monitor is a super-set of the belief states that are compatible with the evolutions leading to the m-configuration. The condition for the validity of knowledge goals given in Definition 21 is stronger than the condition given in Definition 15. Therefore, if $\Pi \models_{\mathcal{D}, \mathcal{M}} g$, then $\Pi \models_{\mathcal{D}} g$. In order to prove the *only if* implication, we introduce *universal monitors*.

Definition 24 (universal monitor) The universal monitor $\mathcal{M}_{\mathcal{D}}$ for domain \mathcal{D} is defined as follows:

- $\mathcal{MS} = 2^{\mathcal{S}}$ are the belief states of \mathcal{D} .
- $m_0 = \mathcal{I}$.
- $\mathcal{MT}(bs, o, a) = \{\bar{s}' : \exists \bar{s} \in bs. o = \mathcal{X}(\bar{s}) \wedge \bar{s}' = \mathcal{T}(\bar{s}, a)\}$.
- $\mathcal{MO}(bs, o) = \{\bar{s} \in bs : o \in \mathcal{X}(\bar{s})\}$.

The universal monitor of a domain traces the precise evolution of the belief states, that is, it does not lose any information. One can check that the belief state reported by this monitor for a given m-configuration coincides with the belief state of the bs-configuration corresponding to the same computation. Therefore, $\Pi \models_{\mathcal{D}} g$ if and only if $\Pi \models_{\mathcal{D}, \mathcal{M}_{\mathcal{D}}} g$. Since the universal monitor is correct, this is sufficient to prove the *only if* implication of Theorem 23.

The possibility of losing some of the information on the current belief state makes monitors very convenient for plan validation. In many practical cases, monitors are able to represent in a very compact way the aspects of the evolution of belief states that are relevant to the goal being analyzed. Consider for instance the extreme case of a K-CTL goal g that does not contain any $\mathbf{K}p$ sub-goal — so it is in fact a CTL goal. In order to apply Definition 16, we should trace the exact evolution of belief states, which may lead to an exponential blowup w.r.t. the size of the domain. Theorem 23, on the other hand, allows us to prove a plan correct against a very simple monitor: the monitor with a single state that is associated to the belief state $bs = \mathcal{S}$, independently from the observation. This monitor traces no information at all on the belief states, which is possible since no knowledge goal appears in g (compare with Definition 9).

Another, less extreme example of the advantages of using monitors for plan validation is the following.

Example 25 Consider the execution of Π_2 on the domain of Figure 2. The belief states corresponding to the different steps of the execution are rather complex. They have to take into account that, after i rooms have been visited by the robot, we know that there are i consecutive rooms with the light on, but that we do not know which are these rooms. For instance, after two rooms have been visited, the belief state

is the following:

$$\begin{aligned} & (\text{room} = 1 \wedge \text{light-on}[1] \wedge \text{light-on}[N]) \vee \\ & (\text{room} = 2 \wedge \text{light-on}[2] \wedge \text{light-on}[1]) \vee \\ & (\text{room} = 3 \wedge \text{light-on}[3] \wedge \text{light-on}[2]) \vee \dots \end{aligned}$$

Most of the information of these belief states is useless for most of the goals. Consider for instance goals

$$\text{AF } \mathbf{K} (\text{light-on}[3])$$

or

$$\bigwedge_{i \in \{1, \dots, N\}} \text{AF } \mathbf{K} (\text{light-on}[i]).$$

The only relevant information for proving that these goals are satisfied by plan Π_2 is that, once the robot has visited all the rooms, all the lights are on. A suitable monitor for these goals is the following. It has two states, m_0 and m_1 , with m_0 the initial state, and m_1 corresponding to the termination of the exploration. The transition function and the output of the monitor are defined by the following table:

m	o	a	$\mathcal{MT}(m, o, a)$	$\mathcal{MO}(m, o)$
m_0	any	wait	m_1	\top
m_0	any	\neq wait	m_0	\top
m_1	any	wait	m_1	$\bigwedge_{i \in \{1, \dots, N\}} (\text{light-on}[i])$

According to Definition 22, the problem of proving that plan Π satisfies K-CTL goal g on domain \mathcal{D} according to monitor \mathcal{M} is reduced to model checking goal g on the m-execution structure corresponding to the synchronous execution of \mathcal{D} , Π , and \mathcal{M} . In order to conclude that Π satisfies goal g , however, we have to prove that monitor \mathcal{M} is correct. The correctness of a monitor can also be proved using model checking techniques. Indeed, it corresponds to prove that the following formula is true on the m-execution structure:

$$\text{AG} (s \in \mathcal{MO}(m, o)).$$

In this paper we do not address the problem of defining a suitable monitor for checking plan validation. In practice, it may be very difficult to decide what information on the belief states has to be traced by the monitor. Intuitively, the problem amounts to identifying an abstraction of the universal monitor that is sufficient for proving that the plan satisfies a given the goal. Although the use of incremental abstraction refinement techniques can be envisaged, this is currently an open problem. In the case the plan is synthesized by an algorithm, however, a proof of the correctness of the plan is built implicitly during the search. In this case, a monitor could be produced by the algorithm itself, by generating a sort of *proof-carrying plan*.

K-CTL and negation

A limit of K-CTL is that it does not allow to express the fact that a given property is *not known*. For instance, in the domain of Figure 2 we would like to express the fact that, if the robot is in a given room and the current status of the light is not known, then a given goal g has to be achieved

(for instance, the robot has to sense the light status). This could be expressed by the following formula:

$$\text{AG} (\neg \mathbf{K} \text{light-on}[\text{room}] \wedge \neg \mathbf{K} \neg \text{light-on}[\text{room}] \rightarrow g).$$

We remark that property $\neg \mathbf{K} p$ is weaker than property $\mathbf{K} \neg p$. The second property states that executor knows that property p is false, while the first property only states that the executor does not know for sure that p is true.

We now extend K-CTL with the possibility of expressing that some property is not known.

Definition 26 (K-CTL with negation) *The goal language K-CTL with negation is defined by the following grammar:*

$$\begin{aligned} g & ::= p \mid \mathbf{K} p \mid \neg \mathbf{K} p \mid g \wedge g \mid g \vee g \mid \text{AX } g \mid \text{EX } g \\ & \quad \text{A}(g \text{ U } g) \mid \text{E}(g \text{ U } g) \mid \text{A}(g \text{ W } g) \mid \text{E}(g \text{ W } g) \\ p & ::= b \mid \neg p \mid p \wedge p \end{aligned}$$

The semantics of K-CTL with negation is defined by extending Definition 15 with the following case:

- $K, q \models \neg \mathbf{K} p$ if $q = (s, o, c, a, bs)$ and $\bar{s} \not\models_0 p$ for some $\bar{s} \in bs$.

The approach based on monitor for proving the validity of a plan has to be extended in order to deal with K-CTL with negation. In particular, in order to conclude that $\neg \mathbf{K} p$ is true in m-configuration (s, o, c, a, m) , it is not sufficient to check that there is some state $\bar{s} \in \mathcal{MO}(m, o)$ such that $\bar{s} \not\models_0 p$ (see Definition 22). Indeed, $\mathcal{MO}(m, o)$ is an over-approximation of the current belief state, and it might be the case that \bar{s} is not part of the belief state. In order to probe that $\neg \mathbf{K} p$ is true, we need an under-approximation of the current belief state and a state \bar{s} in this under-approximation such that $\bar{s} \not\models_0 p$. Exploiting this idea, we define an extended version of monitors, that provide both an over- and an under-approximation of the current belief state.

Definition 27 (extended monitor) *An extended monitor for a domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ is a tuple $\mathcal{M} = \langle \mathcal{MS}, m_0, \mathcal{MT}, \mathcal{MO}^+, \mathcal{MO}^- \rangle$, where:*

- \mathcal{MS} is the set of states of the monitor.
- $m_0 \in \mathcal{MS}$ is the initial state of the monitor.
- $\mathcal{MT} : \mathcal{MS} \times \mathcal{U} \times \mathcal{A} \rightarrow \mathcal{MS}$ is the transition function of the monitor.
- $\mathcal{MO}^+ : \mathcal{MS} \times \mathcal{U} \rightarrow 2^{\mathcal{S}}$ provides an over-approximation of the current belief state.
- $\mathcal{MO}^- : \mathcal{MS} \times \mathcal{U} \rightarrow 2^{\mathcal{S}}$ provides an under-approximation of the current belief state.

The validity of a K-CTL formula on a m-execution structure K is defined as in Definition 15, with the exception of the case of goals $\mathbf{K} p$, where:

- $K, q \models \mathbf{K} p$ if $q = (s, o, c, a, m)$ and $\bar{s} \models_0 p$ for all $\bar{s} \in \mathcal{MO}^+(m, o)$.
- $K, q \models \neg \mathbf{K} p$ if $q = (s, o, c, a, m)$ and $\bar{s} \not\models_0 p$ for some $\bar{s} \in \mathcal{MO}^-(m, o)$.

Extended monitors are also useful for proving that a given plan does *not* satisfy a given K-CTL goal. In order to prove that goal g is not satisfied by plan Π , it is not sufficient to

exhibit a monitor for which the goal is not satisfied. Indeed, it might be the case that the goal is satisfied, but that the monitor is not powerful enough for proving the correctness of the plan. To prove that goal g is not satisfied from state s_0 , one can prove instead that formula “not g ” is satisfied, that is, one can exhibit a monitor for “not g ”. In general, if g is a K-CTL formula, then one needs $\neg\mathbf{K}$ atoms in order to represent formula “not g ”, as shown by the following example, thus extended monitors are necessary.

Example 28 To prove that plan Π_1 does not satisfy goal

$$\text{AF } \mathbf{K} (\neg\text{light-on}[3])$$

one can show that the negation of this goal is satisfied instead. That is, we should show that the plan satisfies formula

$$\text{EG } \neg\mathbf{K} (\neg\text{light-on}[3]).$$

Concluding remarks

This paper is a first step towards planning for temporally extended goals under the hypothesis of partial observability. We defined the basic framework and introduced the K-CTL language, that combines the ability of expressing temporally extended constraints with the ability to predicate over uncertainty aspects. Then, we introduced the notion of monitor, and defined correctness criteria that can be used in practice to validate plans against K-CTL goals.

The issue of “temporally extended goals”, within the simplified assumption of full observability, is certainly not new. However, most of the works in this direction restrict to deterministic domains, see for instance (de Giacomo & Vardi 1999; Bacchus & Kabanza 2000). A work that considers extended goals in nondeterministic domains is described in (Kabanza, Barbeau, & St-Denis 1997). Extended goals make the planning problem close to that of automatic synthesis of controllers (see, e.g., (Kupferman, Vardi, & Wolper 1997)). However, most of the work in this area focuses on the theoretical foundations, without providing practical implementations. Moreover, it is based on rather different technical assumptions on actions and on the interaction with the environment.

On the other side, partially observable domains have been tackled either using a probabilistic Markov-based approach (see (Bonet & Geffner 2000)), or within a framework of possible-world semantics (see, e.g., (Bertoli *et al.* 2001; Weld, Anderson, & Smith 1998; Rintanen 1999)). These works do not go beyond the possibility of expressing more than simple reachability goals. An exception is (Karlsson 2001), where a linear-time temporal logics with a knowledge operator is used to define search control strategies in a progressive probabilistic planner. The usage of a linear-time temporal logics and of a progressive planning algorithm makes the approach of (Karlsson 2001) quite different in aims and techniques from the one discussed in this paper.

Future steps of this work will include the definition of a procedure for synthesizing monitors from K-CTL goals, and the investigation of planning procedures for planning for extended goals under partial observability. The synthesis of monitors appears to be related to the problem

of generating supervisory controllers for the diagnosis of failures. We are investigating whether the techniques developed by the diagnosis community (see, e.g., (Sampath *et al.* 1996)) can be applied to the synthesis of monitors. The main challenge for obtaining a planning procedure appears to be the effective integration of the techniques in (Bertoli, Cimatti, & Roveri 2001; Bertoli *et al.* 2001; Pistore & Traverso 2001) that make effective use of (extensions of) symbolic model checking techniques, thus obtaining a practical implementation based on symbolic model checking techniques.

References

- Bacchus, F., and Kabanza, F. 2000. Using Temporal Logic to Express Search Control Knowledge for Planning. *Artificial Intelligence* 116(1-2):123–191.
- Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In *Proc. IJCAI'01*.
- Bertoli, P.; Cimatti, A.; and Roveri, M. 2001. Heuristic Search + Symbolic Model Checking = Efficient Conformant Planning. In *Proc. IJCAI'01*.
- Bonet, B., and Geffner, H. 2000. Planning with Incomplete Information as Heuristic Search in Belief Space. In *Proc. AIPS 2000*.
- Dal Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Proc. AAAI'02*.
- de Giacomo, G., and Vardi, M. 1999. Automata-theoretic approach to Planning with Temporally Extended Goals. In *Proc. ECP'99*.
- Emerson, E. A. 1990. Temporal and modal logic. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier.
- Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning Control Rules for Reactive Agents. *Artificial Intelligence* 95(1):67–113.
- Karlsson, L. 2001. Conditional progressive planning under uncertainty. In *Proc. IJCAI'01*.
- Kupferman, O.; Vardi, M.; and Wolper, P. 1997. Synthesis with incomplete information. In *Proc. ICTL'97*.
- Pistore, M., and Traverso, P. 2001. Planning as Model Checking for Extended Goals in Non-deterministic Domains. In *Proc. IJCAI'01*.
- Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research* 10:323–352.
- Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1996. Failure Diagnosis Using Discrete-Event Models. *IEEE Transactions on Control Systems Technology* 4(2):105–124.
- Weld, D.; Anderson, C.; and Smith, D. 1998. Extending Graphplan to Handle Uncertainty and Sensing Actions. In *Proc. AAAI'98*.