Auton Agent Multi-Agent Syst (2008) 16:327–344
DOI 10.1007/s10458-008-9034-0

A logic-based agent that plans for extended reachability goals

Silvio Lago Pereira · Leliane Nunes de Barros

Published online: 14 March 2008

Abstract Planning to reach a goal is an essential capability for rational agents. In general, a goal specifies a condition to be achieved at the end of the plan execution. In this article, we introduce nondeterministic planning for *extended reachability goals* (*i.e.*, goals that also specify a condition to be preserved during the plan execution). We show that, when this kind of goal is considered, the temporal logic CTL turns out to be inadequate to formalize plan synthesis and plan validation algorithms. This is mainly due to the fact that the CTL's semantics cannot discern among the various actions that produce state transitions. To overcome this limitation, we propose a new temporal logic called α-CTL. Then, based on this new logic, we implement a planner capable of synthesizing reliable plans for extended reachability goals, as a side effect of model checking.

Keywords Automated planning · Model checking · Temporal logic

1 Introduction

In the last few years, *automated planning* [12] has being increasingly demanded for practical applications in several areas that require solutions for complex goals, including autonomous agents [11]. In this setting, a formal method based approach [8,23] is very attractive to guarantee the reliability of the solutions. In spite of this, the use of formal methods in the automated planning area has received relatively little attention. The few related works [5,6,9] are almost always based on model checking [16], a research area that has been called *planning based on model checking* [13]. In this approach, planning goals are often specified by formulas of the branching time temporal logic CTL [7], a formalism that is only appropriate to deal with

S. L. Pereira (✉) · L. N. de Barros
Institute of Mathematics and Statistics, University of São Paulo, Sao Paulo, Brazil
e-mail: slago@ime.usp.br

L. N. de Barros
e-mail: leliane@ime.usp.br

planning problems for simple reachability goals and for some very specific kinds of more complex goals.

An interesting kind of more complex goal, which has not been treated yet by the planning based on model checking community, is given in Example 1.

Example 1 *Roomba* (Fig. 1) is a very popular vacuum cleaner robot. To date, more than 2 million of such robots have been sold worldwide. While this robot moves to clean a room, it is capable of detecting when its battery is weak, driving itself to a recharging station, and returning to its original location in order to continue its cleaning task. In the future, a project to integrate this vacuum cleaner with an intelligent carpet, capable of mapping the environment and communicating the dirt and the robot locations, will allow for the robot to plan its cleaning route so that whenever its battery is weak, it would be next to a recharging station. Observe that the recharging stations do not need to be in the planned route, they only need to be reachable from it. As we can see, this goal cannot be specified by a condition to be achieved only at the end of the plan execution. □

In this article, we introduce *extended reachability goals*, a class of planning goals that has simple reachability goals as subclass. We show that, when this wider class of planning goals is considered, the temporal logic CTL becomes inadequate to specify goals (as well as solution quality requirements) and to formalize plan synthesis and plan validation algorithms. This happens because the CTL's semantics cannot distinguish among the different actions that produce state transitions. To overcome this limitation, we propose a new branching time temporal logic called α-CTL. Then, based on this new logic, we implement a planner capable of synthesizing reliable plans for extended reachability goals (as a side effect of the model checking for α-CTL formulas expressing such planning goals).

We must emphasize that the proposed logic differs from others action logics found in literature, where formulas impose constraints over states and also over *actions* [17,18]. Although actions play an important role in the α-CTL's semantics, they are not used in the formula's composition. Indeed, when we specify an extended reachability goal, we want to impose constraints only over the states visited during plan execution and not over the actions used to compose the plan.

The remainder of this article is organized as follows: in Sect. 2, we present the background on automated planning in nondeterministic environments and define the class of extended reachability goals; in Sect. 3, we discuss how the model checking framework can be adapted

Fig. 1 Roomba: a vacuum cleaner robot

for automated planning and why the CTL's semantics is not appropriate to deal with extended reachability goals; in Sect. 4, we define the new logic α-CTL and present a model checker based on its semantics; in Sect. 5, we implement a planner based on the α-CTL model checker; and finally, in Sect. 6, we present our conclusions.

2 Automated planning

Automated planning [12] is the field of the artificial intelligence that studies the deliberative process involved in the planning task, seeking for the implementation of planners. Essentially, a *planner* is an algorithm that synthesizes a plan of actions, by analyzing a formal description of the environment's dynamics and of the agent's goal. A *plan* defines the agent behavior pattern: at each instant, it observes the environment's current state and executes the corresponding action, as specified in the plan. Behaving in this manner, the agent must be capable of conducting the environment's evolution, in spite of exogenous events (i.e., events over which the agent has no control), still making sure that its goal can be achieved. The interaction among these components can be seen in Fig. 2.

2.1 Nondeterministic environments

There are several factors that challenge the automation of the planning task [12]. In order to simplify this process, the classical planning approach [10] assumes that the planning environment evolves deterministically, i.e.: (*i*) there is no uncertainty regarding the effects of the agent's actions; (*ii*) the current state of the environment changes only due to the actions executed by the agent; and (*iii*) the agent executes actions until a desired state is finally reached (simple reachability goal). Although these assumptions can really simplify the automation of the planning task, deterministic planning still belongs to the PSPACE-complete complexity class [1,4]. In addition, simple reachability goal and deterministic environment assumptions can indeed be inappropriate in various practical situations [22]. Hence, in this work, we consider planning for extended reachability goals in (completely observable) nondeterministic environments.

2.2 Domains, problems and solutions

Let $\mathbb{P} \neq \emptyset$ be a finite set of atomic propositions, denoting states properties of an environment, and $\mathbb{A} \neq \emptyset$ be a finite set of actions, representing the agent's abilities in this environment. A *planning domain* is a formal model of the environment's dynamics and, since the sets $\mathbb{P}$ and $\mathbb{A}$ are dependent of the specific environment considered, the pair $(\mathbb{P}, \mathbb{A})$ is called the *signature* of the planning domain.

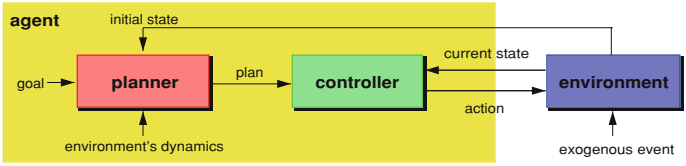


Fig. 2 The components involved in automated nondeterministic planning

Definition 1 A planning domain with signature $(\mathbb{P}, \mathbb{A})$ is defined by a structure $\mathcal{D} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$, where:

- $\mathcal{S} \neq \emptyset$ is a finite set of states;
- $\mathcal{L} : \mathcal{S} \mapsto 2^{\mathbb{P}}$ is a state labeling function;
- $\mathcal{T} : \mathcal{S} \times \mathbb{A} \mapsto 2^{\mathcal{S}}$ is a state transition function.

We assume that $\top \in \mathcal{L}(s)$, for every state $s \in \mathcal{S}$. We also assume that the set $\mathbb{A}$ contains the *trivial* action τ and that $\mathcal{T}(s, \tau) = \{s\}$, for every state $s \in \mathcal{S}$. When the agent executes the action τ, the current state remains the same. Intuitively, this action represents the fact that, in any state, the agent may choose to do nothing. Given a state $s \in \mathcal{S}$ and an action $a \in \mathbb{A}$, the set of *a-successors* of s, denoted by $\mathcal{T}(s, a)$, is the set of states that can be directly reached by the execution of a in s.

A planning domain with signature $(\mathbb{P}, \mathbb{A})$ can be represented as a *transition graph*, where states are labeled with subsets of $\mathbb{P}$ and transitions are labeled with elements of $\mathbb{A}$. For example, in the transition graph for the planning domain $\mathcal{D}^1$, depicted in Fig. 3, the states are labeled with subsets of the set $\mathbb{P} = \{r, g\}$ and the transitions are labeled with elements of the set $\mathbb{A} = \{a, b, c\}$.

A *policy* (or plan) for a planning domain $\mathcal{D}$ with signature $(\mathbb{P}, \mathbb{A})$ is a partial function $\pi : \mathcal{S} \mapsto \mathbb{A}$, that maps states to actions. The set $\mathcal{S}_\pi$ of states reachable by a policy π is $\{s : (s, a) \in \pi\} \cup \{s' : (s, a) \in \pi \text{ and } s' \in \mathcal{T}(s, a)\}$. The *execution structure* of π, denoted by $\mathcal{D}_\pi$, is the subgraph of $\mathcal{D}$ that has $\mathcal{S}_\pi$ as set of states and that contains all transitions induced by the actions in π. For instance, the execution structure $\mathcal{D}^1_{\pi_1}$ of the policy $\pi_1 = \{(s_0, a), (s_1, b), (s_2, c)\}$, in the planning domain $\mathcal{D}^1$, can be seen in Fig. 4. During the execution of a policy π, if the agent reaches a state not covered by π, it continues executing the action τ.

Definition 2 A planning problem is defined by a structure $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$, where:

- $\mathcal{D}$ is a planning domain with signature $(\mathbb{P}, \mathbb{A})$;
- $s_0 \in \mathcal{S}$ is the initial state of the environment;
- φ is a propositional formula over $\mathbb{P}$, specifying a simple reachability goal.

Given a planning problem, we can distinguish three classes of solutions: *weak*, *strong* and *strong-cyclic*; each one indicating a different quality of policies.

Definition 3 Let $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$ be a planning problem, and π be a policy in $\mathcal{D}$ (with execution structure $\mathcal{D}_\pi$). We say that π is a:

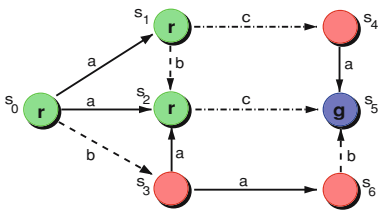


Fig. 3 The transition graph for the planning domain $\mathcal{D}^1$

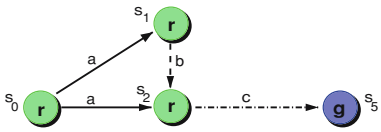


Fig. 4 The execution structure $\mathcal{D}^1_{\pi_1}$ of policy $\pi_1 = \{(s_0, a), (s_1, b), (s_2, c)\}$

- weak solution for $\mathcal{P}$, if some path starting from s_0 in $\mathcal{D}_\pi$ reaches a state where φ holds;
- strong solution for $\mathcal{P}$, if every path starting from s_0 in $\mathcal{D}_\pi$ is acyclic and reaches a state where φ holds;
- strong-cyclic solution for $\mathcal{P}$, if every path starting in s_0 in $\mathcal{D}_\pi$ reaches a state where φ holds.

Intuitively, a *weak solution* is a policy that can allow an agent to achieve a goal state; but due to the nondeterminism, it does not guarantee to do so; a *strong solution* is a policy that always achieves a goal state, in spite of nondeterminism; and a *strong-cyclic solution* is a policy that always achieves the goal, under the fairness assumption that execution will eventually exit from all existing cycles.

Note that, according to Definition 2, a planning problem specifies only the planning goal (through the formula φ). It is up to the agent to decide the quality of the solution that will achieve this planning goal. Indeed, there are specialized algorithms for each one of these classes of solutions [5,6,9].

2.3 Extended reachability goals

Extended planning goals (i.e., a goal formulation that extends the expressiveness of the simple reachability goals from classical planning) can be seen as the more general class of goals that includes, for instance, extended goals for search control [14], extended goals for process control [21] and extended goals for best policy choice [2]. In this article, we are interested in a particular kind of extended goal, named *extended reachability goal*. Formally, an extended reachability goal is a pair of formulas (φ_1, φ_2), where φ_1 is a condition to be *preserved* during the policy execution and φ_2 is a condition to be *achieved* at the end of the policy execution. For instance, the policy π_1, depicted in Fig. 4, is a solution for a planning problem where the extended reachability goal is (r, g), i.e., the goal is to achieve a state that satisfies the property g, by preserving the property r in every state visited during the policy execution.

Extended reachability goals provides a significant improvement on expressivity to specify planning problems. Through this kind of goal, besides specifying the desired final states, we can also establish preferences on the possible intermediate states (i.e., we can impose constraints on the plan trajectories).

Some interesting variations of extended reachability goals are:

- $(\top, \varphi_2)$: achieves property φ_2 (*simple reachability goal*);
- (φ_1, φ_2): achieves property φ_2, by preserving property φ_1;
- $(\neg\varphi_1, \varphi_2)$: achieves property φ_2, by avoiding property φ_1;
- $(\varphi_1 \wedge \neg\varphi_1', \varphi_2)$: achieves property φ_2, by preserving φ_1 and avoiding φ_1'.

Definition 4 Let $\mathcal{P} = \langle \mathcal{D}, s_0, (\varphi_1, \varphi_2) \rangle$ be an extended planning problem and π be a policy in $\mathcal{D}$ (with execution structure $\mathcal{D}_\pi$). We say that π is a:

- weak solution for $\mathcal{P}$, if some path starting from s_0 in $\mathcal{D}_\pi$ passes only through states satisfying φ_1, and reaches a state where φ_2 holds;
- strong solution for $\mathcal{P}$, if every path starting from s_0 in $\mathcal{D}_\pi$ is acyclic, passes only through states satisfying φ_1, and reaches a state where φ_2 holds;
- strong-cyclic solution for $\mathcal{P}$, if every path starting from s_0 in $\mathcal{D}_\pi$ passes only through states satisfying φ_1, and reaches a state where φ_2 holds.

We should emphasize that, even for planning problems with extended reachability goals, the quality of the solution is still an agent's decision. The question that arises is: *is it also*

possible to specify the desired solution quality within the goal specification? That is, *is it possible to write a formula that expresses both: the planning goal and the desired solution quality?* In the next section, we show that, by using CTL, this is possible only for a subclass of extended reachability goals. In Sect. 4, we propose a new logic, called α-CTL, that can be used to specify a larger class of extended goals (with built-in desired solution quality).

3 Planning based on model checking

In this section, we introduce the fundamentals of automated planning based on model checking and show how simple reachability goals can be specified in CTL (a branching time temporal logic traditionally used as specification language in model checking). We also show that, for this kind of simple goal, although the CTL's semantics allows for plan validation, it is inadequate for plan synthesis. Following, we show that, when dealing with extended reachability goals, CTL becomes inadequate not only for plan synthesis but for plan validation as well.

3.1 The model checking framework

Model checking consists of solving the problem $\mathcal{K} \models \varphi$, where $\mathcal{K}$ is a formal model of a system and φ is a formal specification of a property to be verified in this system. Essentially, a *model checker* (Fig. 5) is an algorithm that receives a pair $(\mathcal{K}, \varphi)$ as input and systematically visits the states of the model $\mathcal{K}$, in order to verify if the property φ holds. When all states in $\mathcal{K}$ satisfy property φ, the model checker returns *success*; otherwise, it returns a *counter-example* (e.g., a state in the model $\mathcal{K}$ where the property φ is violated).

When applying the model checking framework to automated planning (e.g., [13]), the model $\mathcal{K}$ describes the planning environment's dynamics, and the property φ describes the agent's goal in this environment. Besides the inputs $\mathcal{K}$ and φ, the planner has an extra input that is the environment initial state s_0. Thus, if $(\mathcal{K}, s_0) \models \varphi$, the planner returns a *plan* (i.e., a behavior policy that allows for the agent to achieve its goal); otherwise, the planner returns *failure* (Fig. 6).

3.2 The temporal logic CTL

In a nondeterministic environment, an agent is not always capable of knowing exactly what is the next state of the environment, after performing an action. Therefore, when an agent selects an action to compose its policy, it has to consider all possible environment changes that can result from the execution of this action. CTL (*Computation Tree Logic*) [7] is a branching time temporal logic that allows for an agent to reason about alternative time lines (i.e., alternative futures); thus, it seams very "natural" to specify planning goals by using

Fig. 5 The model checking framework

Fig. 6 Planning as model checking

this logic. Indeed, CTL is the main formalism that has been used to specify nondeterministic planning problems and related algorithms based on model checking [5,6,9].

The CTL formulas are composed by atomic propositions, propositional operators, and temporal operators. The symbols $\bigcirc$ (*next*), $\Box$ (*invariantly*), $\Diamond$ (*finally*) and $\sqcup$ (*until*), combined with the quantifiers $\exists$ and $\forall$, are used to compose the temporal operators of this logic. The syntax of CTL is inductively defined as:

$$\varphi ::= \top \mid p \in \mathbb{P} \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \exists \bigcirc \varphi \mid \forall \bigcirc \varphi \mid \exists\Box\varphi \mid \forall\Box\varphi \mid \exists(\varphi \sqcup \varphi') \mid \forall(\varphi \sqcup \varphi'),$$

and some useful abbreviations are: $\bot \doteq \neg\top$, $\exists\Diamond\varphi \doteq \exists(\top \sqcup \varphi)$, and $\forall\Diamond\varphi \doteq \forall(\top \sqcup \varphi)$.

The semantics of CTL is defined over a *Kripke structure* $\mathcal{K} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{L}: \mathcal{S} \mapsto 2^{\mathbb{P}}$ is a state labeling function and $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{S}$ is a transition relation. A *path* in $\mathcal{K}$ is a sequence of states $s_0, s_1, \ldots$ such that $s_i \in \mathcal{S}$ and $(s_i, s_{i+1}) \in \mathcal{T}$, for all $i \geq 0$. Given a Kripke structure $\mathcal{K}$ and a state $s_0 \in \mathcal{S}$, the CTL satisfiability relation is defined as:

$(\mathcal{K}, s_0) \models \top$	for all $s_0 \in \mathcal{S}$;
$(\mathcal{K}, s_0) \models p$	iff $p \in \mathcal{L}(s_0)$;
$(\mathcal{K}, s_0) \models \neg\varphi$	iff $(\mathcal{K}, s_0) \not\models \varphi$;
$(\mathcal{K}, s_0) \models (\varphi \wedge \varphi')$	iff $(\mathcal{K}, s_0) \models \varphi$ and $(\mathcal{K}, s_0) \models \varphi'$;
$(\mathcal{K}, s_0) \models (\varphi \vee \varphi')$	iff $(\mathcal{K}, s_0) \models \varphi$ or $(\mathcal{K}, s_0) \models \varphi'$;
$(\mathcal{K}, s_0) \models \exists \bigcirc \varphi$	iff for some path $s_0, s_1, \ldots$ in $\mathcal{K}$, $(\mathcal{K}, s_1) \models \varphi$;
$(\mathcal{K}, s_0) \models \forall \bigcirc \varphi$	iff for every path $s_0, s_1, \ldots$ in $\mathcal{K}$, $(\mathcal{K}, s_1) \models \varphi$;
$(\mathcal{K}, s_0) \models \exists\Box\varphi$	iff for some path $s_0, s_1, \ldots$ in $\mathcal{K}$, for $i \geq 0$, $(\mathcal{K}, s_i) \models \varphi$;
$(\mathcal{K}, s_0) \models \forall\Box\varphi$	iff for every path $s_0, s_1, \ldots$ in $\mathcal{K}$, for $i \geq 0$, $(\mathcal{K}, s_i) \models \varphi$;
$(\mathcal{K}, s_0) \models \exists(\varphi \sqcup \varphi')$	iff for some path $s_0, s_1, \ldots$ in $\mathcal{K}$, there exists $i \geq 0$ such that $(\mathcal{K}, s_i) \models \varphi'$ and, for $0 \leq j < i$, $(\mathcal{K}, s_j) \models \varphi$.
$(\mathcal{K}, s_0) \models \forall(\varphi \sqcup \varphi')$	iff for every path $s_0, s_1, \ldots$ in $\mathcal{K}$, there exists $i \geq 0$ such that $(\mathcal{K}, s_i) \models \varphi'$ and, for $0 \leq j < i$, $(\mathcal{K}, s_j) \models \varphi$.

3.3 Inadequacy of CTL to deal with extended reachability goals

An extended reachability goal (φ_1, φ_2), where φ_1 is a preservation condition and φ_2 is an achievement condition, is a wide class of goals that can be partitioned in two distinct subclasses, according to the type[1] of φ_1: when φ_1 is a propositional formula, we have a *linear* extended reachability goal, since the validity of φ_1 depends only on the actual path that leads to the goal state; on the other hand, when φ_1 is a temporal formula, we have a *branching* extended reachability goal, since the validity of φ_1 depends not only on the actual path to the goal, but also on the possible ramifications of this path.

Following, we show that although CTL can be used to specify linear extended reachability goals, as well as to validate policies for them, it cannot deal with branching extended reachability goals (neither to specify these goals, nor to validate policies for them). In addition, we give some intuition on why this logic is also inadequate to formalize plan synthesis algorithms for both subclasses of extended reachability goals.

Linear extended reachability goals. Using CTL, a linear extended reachability goal (φ_1, φ_2) with built-in desired solution quality can be specified as following:

– $\exists(\varphi_1 \sqcup \varphi_2)$, when a weak solution is desired;

[1] Note that φ_2 can be a temporal formula, since one may want to reach a state from where all successors have a certain property. However, its type is not important for the partition of the extended reachability goals class that we propose in this work.

– $\forall(\varphi_1 \sqcup \varphi_2)$, when a strong solution is desired; or
– $\forall\Box\exists(\varphi_1 \sqcup \varphi_2)$, when a strong-cyclic solution is desired.

Thus, these formulas specify the planning goal and also the quality desired for the solution through the CTL's semantics. Moreover, if φ_1 is $\top$, a linear extended reachability goal reduces to a simple reachability goal and, then, it can be equivalently specified as $\exists \Diamond \varphi_2$, $\forall \Diamond \varphi_2$, or $\forall \Box \exists \Diamond \varphi_2$.

Let $\mathcal{P} = \langle \mathcal{D}, s_0, \phi \rangle$ be a planning problem, where $\mathcal{D}$ is a planning domain, s_0 is the initial state, and ϕ is a linear extended reachability goal specified in CTL. Let π be a policy in $\mathcal{D}$, and $\mathcal{D}_\pi$ be the execution structure of π. By deleting the transition labels in $\mathcal{D}_\pi$, we obtain a corresponding Kripke structure, denoted by $\mathcal{K}(\mathcal{D}_\pi)$. Then, the policy π is a solution (with the desired quality) for the planning problem $\mathcal{P}$ if and only if $(\mathcal{K}(\mathcal{D}_\pi), s_0) \models \phi$. As we can see, the CTL's semantics (built in the definition of the satisfiability relation $\models$) can indeed be used to formalize plan validation algorithms for the linear subclass of extended reachability goals.

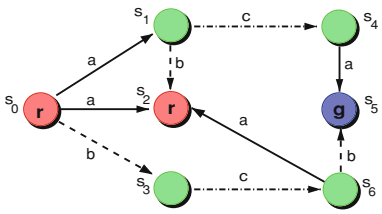
To see why CTL's semantics is not adequate to formalize plan synthesis algorithms for linear extended reachability goals, consider the planning domain $\mathcal{D}^1$ (Fig. 3). Suppose that the agent in this domain is initially at state s_0 and its goal is to necessarily reach a final state satisfying property g, passing only through states where the atomic proposition r holds. It is easy to see that, according to the CTL's semantics, this linear extended reachability goal can be specified by the formula $\forall(r \sqcup g)$. However, according to this same semantics, it is also clear that $(\mathcal{K}(\mathcal{D}^1), s_0) \not\models \forall(r \sqcup g)$ (observe that in $\mathcal{K}(\mathcal{D}^1)$ there exists an "unlabeled" transition from s_0 to state s_3, where property r does not hold). This means that, from the planning domain $\mathcal{D}^1$, a planner based on CTL (whose semantics cannot distinguish different types of transitions) would not be able to synthesize a policy that achieves the goal specified by formula $\forall(r \sqcup g)$; and, thus, such planner would stop with *failure*.

To overcome this limitation on the CTL's semantics, planners based on model checking often use specialized algorithms [15] to construct a policy (i.e., a subgraph of the planning domain) and, then, use the CTL's semantics only to guarantee that the execution structure of this policy satisfies the goal specification in CTL. For instance, considering policy $\pi_1 = \{(s_0, a), (s_1, b), (s_2, c)\}$ (Fig. 4), it is clear that $(\mathcal{K}(\mathcal{D}^1_{\pi_1}), s_0) \models \forall(r \sqcup g)$.

Thus, although CTL can be used to specify goals in the linear subclass of extended reachability goals, as well as to formalize plan validation algorithms for them, it cannot be used to formalize plan synthesis algorithms for such goals.

Branching extended reachability goals. The branching subclass of extended reachability goals comprises those goals where the preserving condition φ_1 is a temporal formula. For instance, consider the planning domain $\mathcal{D}^2$, depicted in Fig. 7. In this domain, the agent could be a mobile robot, the proposition r could describe the property of the states where there exists a battery recharging station and the property g could describe the property of the final state that the robot wants to reach. In this context, suppose that the agent's goal is, starting from the state s_0, *necessarily to reach a state that satisfies property g, passing only*

Fig. 7 The planning domain $\mathcal{D}^2$



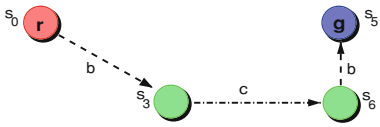


Fig. 8 The execution structure $\mathcal{D}^2_{\pi_2}$ of policy $\pi_2 = \{(s_0, b), (s_3, c), (s_6, b)\}$

through states from which a battery recharging station can be necessarily reached in at most two steps. This extended reachability goal could be specified by the following CTL formula: $\forall((r \vee \forall\bigcirc r \vee \forall\bigcirc\forall\bigcirc r) \sqcup g)$. However, there are two problems with this formulation that we need to highlight:

- First of all, since CTL's semantics cannot distinguish among different types of transitions, it does not allow reasoning about alternative ramifications induced by actions that will not be actually executed. However, the preserving condition $(r \vee \forall\bigcirc r \vee \forall\bigcirc\forall\bigcirc r)$ is only *contingent*. It does not require that the agent really reaches a battery recharging station, unless this turn out to be strictly necessary. Thus, it should be clear that the semantics of the formula $\forall((r \vee \forall\bigcirc r \vee \forall\bigcirc\forall\bigcirc r) \sqcup g)$ does not specify exactly what we need.
- Second, even if this formula could be used to specify the desired goal, we would have that $(\mathcal{K}(\mathcal{D}^2), s_0) \not\models \forall((r \vee \forall\bigcirc r \vee \forall\bigcirc\forall\bigcirc r) \sqcup g)$. However, as we can easily see in Fig. 7:
  - there exists a battery recharging station in s_0;
  - from s_3, the battery recharging station in s_2 can be reached in two steps;
  - from s_6, the battery recharging station in s_2 can be reached in one step.

  Clearly, by following the policy $\pi_2 = \{(s_0, b), (s_3, c), (s_6, b)\}$, the agent would achieve its goal and, therefore, π_2 is a solution for the proposed planning problem. Regardless of this fact, the execution structure $\mathcal{D}^2_{\pi_2}$ (Fig. 8) *does not* satisfy the goal specified by the CTL formula $\forall((r \vee \forall\bigcirc r \vee \forall\bigcirc\forall\bigcirc r) \sqcup g)$.

Thus, with this example, we show that CTL is not adequate to deal with branching extended reachability goals.

4 The new temporal logic $\alpha-$ CTL

In this section, we present the branching time temporal logic α-CTL. Based on this new logic, we implement a model checker that, in Sect. 5, is adapted for automated planning for extended reachability goals.

4.1 The syntax of α-CTL

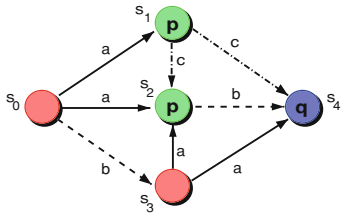
In CTL, a formula $\forall\bigcirc\varphi$ holds on a state s if and only if it holds on *all* successors of s, independently of the actions labeling the transitions from s to its successors. In α-CTL, to enforce that actions play an important role in its semantics, we use a different set of "dotted" symbols to represent temporal operators: $\odot$ (*next*), $\boxdot$ (*invariantly*), $\diamondsuit$ (*finally*) and $\sqcup\!\!\cdot$ (*until*).

Definition 5 Let $p \in \mathbb{P}$ be an atomic proposition. The syntax of the logic α-CTL is inductively defined as:

$$\varphi ::= p \mid \neg p \mid (\varphi \wedge \varphi') \mid (\varphi \vee \varphi') \mid \exists\odot\varphi \mid \forall\odot\varphi \mid \exists\boxdot\varphi \mid \forall\boxdot\varphi \mid \exists(\varphi \sqcup\!\!\cdot\, \varphi') \mid \forall(\varphi \sqcup\!\!\cdot\, \varphi')$$

According to the α-CTL's syntax, well-formed formulas are in *negative normal form*, where the scope of negation is restricted to the atomic propositions (this allows to easily define a

Fig. 9 The planning domain $\mathcal{D}^3$



fixpoint semantics for the formulas). Furthermore, all temporal operators are prefixed by a path quantifier ($\exists$ or $\forall$). The temporal operators derived from $\Diamond$ are defined as:

$$\begin{array}{lcl} \exists \Diamond \varphi & \doteq & \exists(\top \sqcup \varphi) \\ \forall \Diamond \varphi & \doteq & \forall(\top \sqcup \varphi) \end{array}$$

Although actions are essential in the semantics of α-CTL, note that they are not used to compose α-CTL formulas. Indeed, when we specify a planning goal, we wish to impose constraints only over the states visited during the execution of the plan. In general, constraints over the actions that will be used to compose a plan are not relevant when we specify the planning goal. For this reason, we claim that existing actions logics [17,18], which allow formulas with constraints over actions, are also inadequate to formalize planning algorithms.

4.2 The semantics of α-CTL

Let $\mathbb{P} \neq \emptyset$ be a finite set of atomic propositions and $\mathbb{A} \neq \emptyset$ be a finite set of actions. An α-CTL temporal model over $(\mathbb{P}, \mathbb{A})$ is a transition graph where states are labeled with subsets of $\mathbb{P}$ and transitions are labeled with elements of $\mathbb{A}$. In this temporal model, *terminal states* (i.e., states where the only executable action is τ) persist infinitely in time. In other words, a temporal model for α-CTL is a planning domain or a policy execution structure.

Intuitively, a state s in a temporal model $\mathcal{D}$ satisfies a formula $\forall \odot \varphi$ (or $\exists \odot \varphi$) if there *exists* an action α that, when executed in s, *necessarily* (or *possibly*) reaches an immediate successor of s which satisfies the formula φ. In other words, the modality $\odot$ represents the set of α-*successors* of s, *for some particular action* $\alpha \in \mathbb{A}$ (denoted by $\mathcal{T}(s, \alpha)$); the quantifier $\forall$ requires that *all* these α-*successors* satisfy φ; and quantifier $\exists$ requires that *some* of these α-*successors* satisfy φ.

For instance, consider the domain $\mathcal{D}^3$, depicted in Fig. 9. In this domain, $\mathcal{T}(s_0, a) = \{s_1, s_2\}$ and both states s_1 and s_2 satisfy p. Thus, by the α-CTL's semantics, it follows that[2] $(\mathcal{D}^3, s_0) \models \forall \odot p$. Furthermore, it also follows that $(\mathcal{D}^3, s_0) \models \forall \odot \neg p$ (by choosing action b in the state s_0). This is due to the fact that each occurrence of the modality $\odot$ can instantiate a different action $\alpha \in \mathbb{A}$ and, consequently, the quantification can be made over different sets of α-*successors* of the state s_0. However, the fact that $(\mathcal{D}^3, s_0) \models \forall \odot p \wedge \forall \odot \neg p$ does not mean that there exists a policy to achieve both subgoals p and $\neg p$ at the same time[3], from state s_0; it only means that, from this state, the agent can choose to reach p or $\neg p$ in the next current state. This possibility of choosing is very important in planning. In fact, if the agent cannot make choices, there is no need of planning.

[2] Inversely, according to CTL's semantics, we would have that $(\mathcal{D}^3, s_0) \not\models \forall \bigcirc p$.

[3] Such policy would exist only if it is guaranteed to reach both p and $\neg p$ with only one step (the goal specified by $\forall \odot (p \wedge \neg p)$); but, clearly, this goal cannot be satisfied.

Before we can give a formal definition of the α-CTL's semantics, we need to define the concept of *preimage* of a set of states. Intuitively, the strong (weak) preimage of a set Y of states is the set X of those states from which a state in Y can necessarily (possibly) be reached with one step. For instance, in domain $\mathcal{D}^3$ (Fig. 9), the strong preimage of the set $Y = \{s_4\}$ is the set $X = \{s_2\}$, since s_2 is the only state in $\mathcal{D}^3$ from which we can necessarily reach s_4 after one step.

Definition 6 Let $Y \subseteq \mathcal{S}$ be a set of states. The *weak preimage* of Y, denoted by $\mathcal{T}_{\exists}^{-}(Y)$, is the set $\{s \in \mathcal{S} : \exists a \in \mathbb{A} \,.\, \mathcal{T}(s,a) \cap Y \neq \emptyset\}$, and the *strong preimage* of Y, denoted by $\mathcal{T}_{\forall}^{-}(Y)$, is the set $\{s \in \mathcal{S} : \exists a \in \mathbb{A} \,.\, \emptyset \neq \mathcal{T}(s,a) \subseteq Y\}$.

The semantics of the global temporal operators ($\exists\boxdot$, $\forall\boxdot$, $\exists\sqcup$ and $\forall\sqcup$) is derived from the semantics of the local temporal operators ($\exists\odot$ and $\forall\odot$), by using least (μ) and greatest (ν) fixpoint operations.

Definition 7 Let $\mathcal{D} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ be a temporal model with signature $(\mathbb{P}, \mathbb{A})$ and $p \in \mathbb{P}$ be an atomic proposition. The intension of an α-CTL formula φ in $\mathcal{D}$ (or the set of states satisfying φ in $\mathcal{D}$), denoted by $[\![\varphi]\!]_{\mathcal{D}}$, is defined as:

- $[\![p]\!]_{\mathcal{D}} = \{s : p \in \mathcal{L}(s)\}$ (by definition, $[\![\top]\!]_{\mathcal{D}} = \mathcal{S}$ and $[\![\bot]\!]_{\mathcal{D}} = \emptyset$)
- $[\![\neg p]\!]_{\mathcal{D}} = \mathcal{S} \setminus [\![p]\!]_{\mathcal{D}}$
- $[\![(\varphi \wedge \varphi')]\!]_{\mathcal{D}} = [\![\varphi]\!]_{\mathcal{D}} \cap [\![\varphi']\!]_{\mathcal{D}}$
- $[\![(\varphi \vee \varphi')]\!]_{\mathcal{D}} = [\![\varphi]\!]_{\mathcal{D}} \cup [\![\varphi']\!]_{\mathcal{D}}$
- $[\![\exists \odot \varphi]\!]_{\mathcal{D}} = \mathcal{T}_{\exists}^{-}([\![\varphi]\!]_{\mathcal{D}})$
- $[\![\forall \odot \varphi]\!]_{\mathcal{D}} = \mathcal{T}_{\forall}^{-}([\![\varphi]\!]_{\mathcal{D}})$
- $[\![\exists \boxdot \varphi]\!]_{\mathcal{D}} = \nu Y.([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\exists}^{-}(Y))$
- $[\![\forall \boxdot \varphi]\!]_{\mathcal{D}} = \nu Y.([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\forall}^{-}(Y))$
- $[\![\exists(\varphi \sqcup \varphi')]\!]_{\mathcal{D}} = \mu Y.([\![\varphi']\!]_{\mathcal{D}} \cup ([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\exists}^{-}(Y)))$
- $[\![\forall(\varphi \sqcup \varphi')]\!]_{\mathcal{D}} = \mu Y.([\![\varphi']\!]_{\mathcal{D}} \cup ([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\forall}^{-}(Y)))$

Definition 8 Let $\mathcal{D}$ be a temporal model, s be a state in $\mathcal{D}$, and φ be an α-CTL formula. The α-CTL's satisfiability relation is defined as: $(\mathcal{D}, s) \models \varphi \Leftrightarrow s \in [\![\varphi]\!]_{\mathcal{D}}$

4.3 A model checker for α-CTL

A model checker for α-CTL can be directly implemented from its semantics. Given a planning domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ and an α-CTL formula φ, the model checker computes the set C of states that do not satisfy the formula φ in $\mathcal{D}$; then, if C is the empty set, it returns *success*; otherwise, it returns C as counter-example.

```
α-ModelChecker(φ, D)
1 C ← S \ Intension(φ, D)
2 if C = ∅ then return success
3 else return C
```

The basic operation on this model checker is implemented by the function Intension, that inductively computes the intension of the formula φ in the model $\mathcal{D}$ (see Definition 7) as following:

```
Intension(φ, D)
 1 if φ ∈ P then return {s ∈ S : φ ∈ L(s)}
 2 case φ of
 3   ¬φ1        : return S \ Intension(φ1, D)
 4   φ1 ∧ φ2    : return Intension(φ1, D) ∩ Intension(φ2, D)
```

```
 5  φ1 ∨ φ2      : return INTENSION(φ1, D) ∪ INTENSION(φ2, D)
 6  ∃⊙ φ1        : return WEAKPREIMAGE(INTENSION(φ1, D), D)
 7  ∀⊙ φ1        : return STRONGPREIMAGE(INTENSION(φ1, D), D)
 8  ∃⊡ φ1        : return INTENSION⊡(WEAKPREIMAGE, φ1, D)
 9  ∀⊡ φ1        : return INTENSION⊡(STRONGPREIMAGE, φ1, D)
10  ∃(φ1 ⊔ φ2)   : return INTENSION⊔(WEAKPREIMAGE, φ1, φ2, D)
11  ∀(φ1 ⊔ φ2)   : return INTENSION⊔(STRONGPREIMAGE, φ1, φ2, D)
```

In this algorithm, propositional operators are treated in a straightforward way (lines 1–5); while temporal operators are treated by specialized auxiliary functions (lines 6–11).

Preimage computations To treat local temporal operators ($\exists\odot$ and $\forall\odot$), the algorithm INTENSION calls the following auxiliary functions:

```
WEAKPREIMAGE(Y, D)
1 return {s ∈ S : ∃a ∈ A . T(s, a) ∩ Y ≠ ∅}

STRONGPREIMAGE(Y, D)
1 return {s ∈ S : ∃a ∈ A . ∅ ≠ T(s, a) ⊆ Y}
```

Given a set of states $Y \subseteq \mathcal{S}$ and a temporal model $\mathcal{D}$, the function WEAKPREIMAGE returns a maximal set of states $X \subseteq \mathcal{S}$ such that, for each $s \in X$, there exists an action $a \in \mathbb{A}$ whose execution in state s leads to at least one state inside the set Y. Analogously, the function STRONGPREIMAGE returns a maximal set of states $X \subseteq \mathcal{S}$ such that, for each $s \in X$, there exists an action $a \in \mathbb{A}$ whose execution in state s leads only to states inside the set Y.

Fixpoint computations To treat global temporal operators, the algorithm INTENSION calls auxiliary functions for fixpoint computations.

The global temporal operators $\exists\sqcup$ and $\forall\sqcup$, which require greatest fixpoint computations, are treated by the following function:

```
INTENSION⊡(PreimageFunction, φ1, D)
1 I ← INTENSION(φ1, D)
2 I' ← ∅
3 while I ≠ I' do
4    I' ← I
5    I ← I ∩ PreimageFunction(I, D)
6 return I
```

This function starts by computing the set I of states which satisfy the formula φ_1 in $\mathcal{D}$; afterward, iteratively, it computes the preimage of the set I and, by taking the intersection between this set and its preimage, the function eliminates from I all states which have all transitions (for $PreimageFunction =$ WEAKPREIMAGE), or some transition (for $PreimageFunction =$ STRONGPREIMAGE), leading to states outside of I. This iterative process stops only when a greatest fixpoint is reached. In this case, the final set I, returned as solution by the function INTENSION$_{\boxdot}$, is the semantics of the initial temporal formula ($\exists\boxdot\varphi_1$ or $\forall\boxdot\varphi_1$).

The global temporal operators $\exists\sqcup$ and $\forall\sqcup$, which require least fixpoint computations, are treated by the following function:

```
INTENSION⊔(PreimageFunction, φ1, φ2, D)
1 I1 ← INTENSION(φ1, D)
2 I2 ← INTENSION(φ2, D)
3 I2' ← ∅
4 while I2 ≠ I2' do
5    I2' ← I2
6    I2 ← (I1 ∩ PreimageFunction(I2, D)) ∪ I2
7 return  I2
```

This function starts by computing the sets I_1 and I_2 of states satisfying formulas φ_1 and φ_2, respectively; afterward, iteratively, it computes the intersection between the set I_1 and the preimage of the set I_2 (to guarantee that φ_1 can be maintained through the paths to states where φ_2 holds); and, finally, the function takes the union of the resulting intersection and the set I_2 (to guarantee that states which satisfy φ_2, but not φ_1, can still be considered in the final solution). This iterative process stops only when a least fixpoint is reached. In this case, the final set I_2, returned as solution by the function $\textsc{Intension}_{\sqcup}$, is the semantics of the initial temporal formula ($\exists(\varphi_1 \sqcup \varphi_2)$ or $\forall(\varphi_1 \sqcup \varphi_2)$).

Time complexity Each fixpoint computation, performed by the function $\textsc{Intension}_{\boxdot}$ or $\textsc{Intension}_{\sqcup}$, costs $O(|\mathcal{S}|)$ steps (each one of them involving only propositional and local temporal operators). Since the number of fixpoint computations to verify a formula φ is equal to the number of global temporal operators that appear in it, denoted by $|\varphi|$, we conclude that the α-CTL model checking complexity is $O(|\varphi| \times |\mathcal{S}|)$.

Formal properties The following results establish the correctness and the completeness of the α-CTL model checker.

Theorem 1 *Given an α-CTL formula φ and a temporal model $\mathcal{D}$ with signature $(\mathbb{P}, \mathbb{A})$, the function* $\textsc{Intension}(\varphi, \mathcal{D})$ *returns the set $[\![\varphi]\!]_{\mathcal{D}}$.*

Proof (sketch) The result follows by induction on the structure of φ. □

Corollary 1 *Given an α-CTL formula φ and a temporal model $\mathcal{D}$ with signature $(\mathbb{P}, \mathbb{A})$, the algorithm α-*$\textsc{ModelChecker}$ *succeeds if and only if every state in $\mathcal{D}$ satisfies the formula φ.*

Proof (sketch) The result follows directly from Theorem 1. □

5 Planning with $\alpha-$ CTL

The main works on planning for extended goals either (*i*) propose an *ad hoc* plan synthesis algorithm [15], without proving its correctness through formal analysis; or (*ii*) propose a new logic that can be used to specify extended goals and to do plan validation [2], without presenting any plan synthesis algorithm (making the assumption that plans are given a priori). In this section, we implement a framework for planning for extended reachability goals, using as basis the α-CTL's model checker presented in the last section. In this framework, a solution for a planning problem is obtained as a side effect of the verification of a model $\mathcal{D}$, specifying a planning environment, with respect to an α-CTL formula φ, specifying a planning goal.

5.1 The planning algorithm

Using α-CTL, an extended reachability goal (φ_1, φ_2) with built-in desired solution quality can be specified as following:

- $\exists(\varphi_1 \sqcup \varphi_2)$, when a weak solution is desired;
- $\forall(\varphi_1 \sqcup \varphi_2)$, when a strong solution is desired; or
- $\forall \boxdot \exists(\varphi_1 \sqcup \varphi_2)$, when a strong-cyclic solution is desired.

Given a planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$, where φ is an extended reachability goal specified in α-CTL, a solution for $\mathcal{P}$ can be obtained by the following algorithm:

α-PLANNER($\mathcal{P}$)
1 $M \leftarrow$ MODEL(lfp, $\varphi, \mathcal{D}$)
2 $C \leftarrow$ STATESCOVEREDBY(M)
3 if $s_0 \in C$ then return POLICY(M)
4 else return failure

This algorithm starts by synthesizing a submodel $M \subseteq \mathcal{D}$ from φ and computing the set C of states covered by this submodel. Then, if $s_0 \in C$, it returns a policy π extracted from M, whose execution allows the agent to reach the goal φ, from s_0, in the domain $\mathcal{D}$; otherwise, it returns *failure*.

To synthesize the submodel M, the algorithm α-PLANNER calls the function MODEL (*vide* Subsect. 5.2), that returns a set M containing states and pairs of states and actions (*i.e.*, a submodel). To obtain the covering set of this submodel, α-PLANNER calls the following function:

STATESCOVEREDBY(M)
1 return $\{s \in \mathcal{S} : s \in M\} \cup \{s \in \mathcal{S} : (s, a) \in M\}$

that returns the union of the sets of *terminals* and *non-terminals* states of M. Finally, to extract a policy from M, the algorithm calls the function POLICY(M), that returns a policy π such that:

– STATESCOVEREDBY(π) = STATESCOVEREDBY(M), and
– for every pair $(s, a), (s', a') \in \pi$, if $s = s'$, then $a = a'$.

5.2 Model synthesis

The notion of intension of a formula φ can be reformulated such that $[\![\varphi]\!]_{\mathcal{D}}$ turns out to be a subgraph of $\mathcal{D}$ containing all the states satisfying φ, as well as all the transitions considered during the selection of these states (we need essentially to redefine preimage functions such that they collect the pair (s, a), whenever the action a is considered to show that s satisfies the property φ). With this reformulation we can synthesize plans as a collateral effect of the verification of property φ in the temporal model $\mathcal{D}$ and it is the main contribution of this work.

To synthesize a submodel from an α-CTL formula φ and a model $\mathcal{D}$, the α-CTL planner uses the function MODEL. This function is very similar to the function INTENSION, used by the α-CTL model checker presented in the last section. The main difference between these functions is that the function MODEL requires an additional parameter that informs the scope of computation: lfp (that avoids cycles) or gfp (that allows cycles). For instance, to synthesize a submodel for a formula containing a global temporal operator specifying an invariant property ($\exists\boxdot$ or $\forall\boxdot$), this parameter should be defined as gfp.

MODEL($scope, \varphi, \mathcal{D}$)
1 if $\varphi \in \mathbb{P}$ then return $\{s \in \mathcal{S} : \varphi \in \mathcal{L}(s)\}$
2 case φ of
3 $\neg\varphi_1$: return $\mathcal{S} \setminus$ MODEL($scope, \varphi_1, \mathcal{D}$)
4 $\varphi_1 \wedge \varphi_2$: return MODEL($scope, \varphi_1, \mathcal{D}$) $\cap$ MODEL($scope, \varphi_2, \mathcal{D}$)
5 $\varphi_1 \vee \varphi_2$: return MODEL($scope, \varphi_1, \mathcal{D}$) $\cup$ MODEL($scope, \varphi_2, \mathcal{D}$)
6 $\exists\odot\varphi_1$: return MODEL$_{\odot}$($scope$, WEAKPREIMAGEMAP, $\varphi_1, \mathcal{D}$)
7 $\forall\odot\varphi_1$: return MODEL$_{\odot}$($scope$, STRONGPREIMAGEMAP, $\varphi_1, \mathcal{D}$)
8 $\exists\boxdot\varphi_1$: return MODEL$_{\boxdot}$(gfp, WEAKPREIMAGEMAP, $\varphi_1, \mathcal{D}$)
9 $\forall\boxdot\varphi_1$: return MODEL$_{\boxdot}$(gfp, STRONGPREIMAGEMAP, $\varphi_1, \mathcal{D}$)
10 $\exists\diamondsuit\varphi_2$: return MODEL$_{\sqcup}$($scope$, WEAKPREIMAGEMAP, $\top, \varphi_2, \mathcal{D}$)
11 $\forall\diamondsuit\varphi_2$: return MODEL$_{\sqcup}$($scope$, STRONGPREIMAGEMAP, $\top, \varphi_2, \mathcal{D}$)
12 $\exists(\varphi_1 \sqcup \varphi_2)$: return MODEL$_{\sqcup}$($scope$, WEAKPREIMAGEMAP, $\varphi_1, \varphi_2, \mathcal{D}$)
13 $\forall(\varphi_1 \sqcup \varphi_2)$: return MODEL$_{\sqcup}$($scope$, STRONGPREIMAGEMAP, $\varphi_1, \varphi_2, \mathcal{D}$)

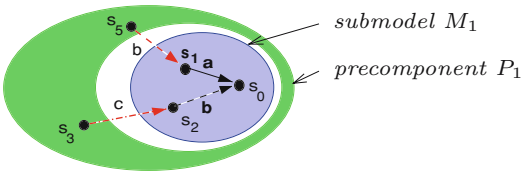


Fig. 10 Model synthesis mechanism for the local temporal operators ($\exists\odot$ and $\forall\odot$)

Propositional formulas A submodel M synthesized from a propositional formula φ and a model $\mathcal{D}$ is the maximal set of states $\mathcal{Y} \subseteq \mathcal{S}$ such that, for all $s \in \mathcal{Y}$, we have that $s \models \varphi$. Propositional formulas are treated directly by the function MODEL, through structural induction.

Local temporal operators To treat the local temporal operators $\exists\odot$ and $\forall\odot$, the function MODEL calls the following auxiliary function:

MODEL$_\odot$($scope$, $PreimageMapFunction$, φ_1, $\mathcal{D}$)
1 $M_1 \leftarrow$ MODEL($scope$, φ_1, $\mathcal{D}$)
2 $I_1 \leftarrow PreimageMapFunction$(STATESCOVEREDBY($M_1$), $\mathcal{D}$)
3 $P_1 \leftarrow$ PRUNE$_\odot$(I_1, M_1)
4 return $M_1 \cup P_1$

In order to synthesize a submodel of $\mathcal{D}$, from a formula of the form $\exists \odot \varphi_1$ or $\forall \odot \varphi_1$, this function starts by synthesizing a submodel M_1 from subformula φ_1 and by computing the preimage I_1 of the set of states covered by this submodel; afterwards, it prunes the set I_1 to avoid that new actions could be assigned to states already covered by M_1. Finally, the union of the submodel M_1 with the precomponent P_1 is returned as the final result. The pairs $(s, a) \in P_1$ guarantee that a policy extracted from $M_1 \cup P_1$ satisfies the goal specified by the initial formula ($\exists \odot \varphi_1$ or $\forall \odot \varphi_1$), as depicted in Fig. 10.

The pruning and preimage computations are performed by the following functions:

PRUNE$_\odot$(I_1, M_1)
1 return $\{(s, a) \in I_1 : s \notin M_1\}$

WEAKPREIMAGEMAP(C, $\mathcal{D}$)
1 return $\{(s, a) : s \in \mathcal{S}, a \in \mathbb{A}$ and $\mathcal{T}(s, a) \cap C \neq \emptyset\}$

STRONGPREIMAGEMAP(C, $\mathcal{D}$)
1 return $\{(s, a) : s \in \mathcal{S}, a \in \mathbb{A}$ and $\emptyset \neq \mathcal{T}(s, a) \subseteq C\}$

Global temporal operators To treat the global temporal operators $\exists\boxdot$ and $\forall\boxdot$, the function MODEL calls the following greatest fixpoint function:

MODEL$_\boxdot$($scope$, $PreimageMapFunction$, φ_1, $\mathcal{D}$)
1 $M \leftarrow$ MODEL($scope$, φ_1, $\mathcal{D}$)
2 $M' \leftarrow \emptyset$
3 while $M' \neq M$ do
4 $\quad M' \leftarrow M$
5 $\quad C \leftarrow$ STATESCOVEREDBY(M)
6 $\quad I \leftarrow PreimageMapFunction(C, \mathcal{D})$
7 $\quad M \leftarrow$ PRUNE$_\boxdot$(I, C)
8 return M

In order to synthesize a submodel of $\mathcal{D}$, from a formula of the form $\exists \boxdot \varphi_1$ or $\forall \boxdot \varphi_1$, this function starts by synthesizing a submodel M from subformula φ_1. Afterwards, iteratively,

this submodel is reduced in the following way: first, the preimage I of the set of states covered by the current submodel M is computed; next, the set I is pruned such that only states covered by the submodel M are maintained. Then, in the next iteration, the result of this pruning is taken as the new current submodel M. Proceeding in this way, at each iteration, states that do not satisfy the invariant property specified by φ_1 are discarded. Thus, when the greatest fixpoint is reached, we can guarantee that a policy extracted from the submodel M computed in the last iteration (and returned as final result of the function) satisfies the goal specified by the initial formula ($\exists \boxdot \varphi_1$ or $\forall \boxdot \varphi_1$).

The pruning function called by $\textsc{Model}_{\boxdot}$ is defined as following:

$\textsc{Prune}_{\boxdot}(I, C)$
1 return $\{(s, a) \in I : s \in C\}$

Finally, to treat the global temporal operators $\exists \sqcup$ and $\forall \sqcup$, the function Model calls the following least fixpoint function:

$\textsc{Model}_{\sqcup}(scope, PreimageMapFunction, \varphi_1, \varphi_2, \mathcal{D})$
1 $C_1 \leftarrow \textsc{StatesCoveredBy}(\textsc{Model}(\texttt{lfp}, \varphi_1, \mathcal{D}))$
2 $M_2 \leftarrow \textsc{Model}(scope, \varphi_2, \mathcal{D})$
3 $M_2' \leftarrow \emptyset$
4 while $M_2 \neq M_2'$ do
5 $\quad M_2' \leftarrow M_2$
6 $\quad C_2 \leftarrow \textsc{StatesCoveredBy}(M_2)$
7 $\quad I_2 \leftarrow PreimageMapFunction(C_2, \mathcal{D})$
8 $\quad P_2 \leftarrow \textsc{Prune}_{\sqcup}(scope, I_2, C_1, C_2)$
9 $\quad M_2 \leftarrow M_2 \cup P_2$
10 return M_2

In order to synthesize a submodel of $\mathcal{D}$, from a formula of the form $\exists(\varphi_1 \sqcup \varphi_2)$ or $\forall(\varphi_1 \sqcup \varphi_2)$, this function starts by computing the set C_1 of states that satisfy the subformula φ_1 and synthesizing a submodel M_2 from the formula φ_2. Afterwards, iteratively, the submodel M_2 is expanded in the following way: first, the preimage I_2 of the set of states covered by the current submodel M_2 is computed; next, the set I_2 is pruned such that only states that also belong to the set C_1 are maintained in the precomponent P_2 (if cycles need to be avoided ($scope = \texttt{lfp}$), the pruning function also deletes from P_2 all states already covered by M_2). Then, the union of the submodel M_2 with its precomponent P_2 is taken as the new submodel M_2 to be considered in the next iteration. In this way, at each iteration, the initial submodel M_2 is expanded with new pairs (s, a), such that $s \in C_1$ and the execution of the action a in the state s leads to states in M_2. Thus, when the least fixpoint is reached, we can guarantee that a policy extracted from the submodel M_2 computed in the last iteration (and returned as final result) satisfies the goal specified by the initial formula ($\exists(\varphi_1 \sqcup \varphi_2)$ or $\forall(\varphi_1 \sqcup \varphi_2)$).

The pruning function called by $\textsc{Model}_{\sqcup}$ is defined as following:

$\textsc{Prune}_{\sqcup}(scope, I_2, C_1, C_2)$
1 $P_2 \leftarrow \{(s, a) \in I_2 : s \in C_1\}$
2 if $scope = \texttt{lfp}$ then $P_2 \leftarrow \{(s, a) \in P_2 : s \notin C_2\}$
3 return P_2

5.3 Formal properties

The following theorems, whose proofs are presented in [19,20], establish some formal properties of the α-CTL planner.

Theorem 2 *Let $\mathcal{P} = \langle \mathcal{D}, s_0, \exists(\varphi_1 \sqcup \varphi_2)\rangle$ be a planning problem. If $\mathcal{P}$ has a solution, then the policy returned by α-Planner$(\mathcal{P})$ is a weak solution for $\mathcal{P}$.*

Theorem 3 *Let* $\mathcal{P} = \langle \mathcal{D}, s_0, \forall(\varphi_1 \sqcup \varphi_2)\rangle$ *be a planning problem. If* $\mathcal{P}$ *has a solution, then the policy returned by* α-PLANNER$(\mathcal{P})$ *is a strong solution for* $\mathcal{P}$.

Theorem 4 *Let* $\mathcal{P} = \langle \mathcal{D}, s_0, \forall \boxdot \exists(\varphi_1 \sqcup \varphi_2)\rangle$ *be a planning problem. If* $\mathcal{P}$ *has a solution, then the policy returned by* α-PLANNER$(\mathcal{P})$ *is a strong-cyclic solution for* $\mathcal{P}$.

Theorem 5 *Let* $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi\rangle$ *be a planning problem for an extended reachability goal* φ. *Then,* α-PLANNER$(\mathcal{P})$ *fails if and only if* $\mathcal{P}$ *has no solution.*

Theorem 6 *The shortest execution path of a policy* π, *returned by the call to* α-PLANNER $(\langle \mathcal{D}, s_0, \exists(\varphi_1 \sqcup \varphi_2)\rangle)$, *is minimum in the best case.*

Theorem 7 *The longest execution path of a policy* π, *returned by the call to* α-PLANNER $(\langle \mathcal{D}, s_0, \forall(\varphi_1 \sqcup \varphi_2)\rangle)$, *is minimum in the worst case.*

6 Conclusion

Practical applications for automated planning require reliable plans for complex goals [12]. However, although such requirement can only be guaranteed by mean of formal specification and analysis, few works in the planning literature make use of formal methods for plan synthesis and plan validation [5,6,9]. Besides, in general, those works are related to planning based on model checking techniques, for simple reachability goals in nondeterministic environments [13].

In this work, we introduce the class of *extended reachability goals* and, through examples, we evince that the CTL's semantics is inadequate to specify goals in this class (with built-in desired solution quality: *weak*, *strong* or *strong-cyclic*) and to formalize plan synthesis and validation as well. Motivated by this scenario, we have proposed a new temporal logic, named α-CTL. Unlike other existing action logics found in literature [17,18], the proposed logic does not make use of actions to compose formulas. Nevertheless, the actions play an important role in α-CTL's semantics by allowing the definition of special purpose temporal operators. Based on this new logic, we also implement a planning framework capable of synthesizing policies for extended reachability goals with built-in desired solution quality. By proceeding in this way, instead of constructing plans in an *ad hoc* fashion to be later validated, *we can synthesize plans whose validity is an immediate consequence of a well formalized synthesis process*.

It is important to note that the existing works on planning for extended goals either propose an *ad hoc* planning algorithm [15], without proving its validity through formal analysis; or propose a new logic that can be used to specify extended goals and do plan validation [2], without presenting any planning algorithm (making the assumption that policies are given a priori). In the present work, we provide both: a logic that can be used as a formal language to specify extended reachability goals and a planning framework based on this logic. Hence, this work presents important contributions for the planning community as well for the logic-based agents community. The first one now acquires reliable plans for more complex type of goals; while the second can now make use of a new formal mechanism capable of verifying properties that, as far as we know, could not have been verified by any mechanisms previously proposed.

Finally, we need stress that the main purpose of this work was not to offer an efficient implementation of a planner, but to show that *it is possible to implement a logic-based agent that guarantees the quality of automated synthesized plans for extended reachability goals*.

However, the efficiency of our planner can be highly improved with the use of BDDs [3], resulting in an extremely efficient symbolic version of the α-CTL planner proposed in this paper.

Acknowledgements We thank FAPESP (grant 04/09568-0) for financial support and the three anonymous reviewers for the suggestions and comments.

References

1. Backstrom, C. (1995). Expressive equivalence of planning formalisms. *Artificial Intelligence, 76*(1-2), 17–34.
2. Baral, C., & Zhao, J. (2006). Goal specification, non-determinism and quantifying over policies. In: *AAAI*-2006, pp. 231–237.
3. Bryant, R. E. (1992). Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys, 24*(3), 293–318.
4. Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence, 69*(1-2), 165–204.
5. Cimatti, A., Giunchiglia, F., Giunchiglia, E., & Traverso, P. (1997). Planning via model checking: A decision procedure for $\mathcal{AR}$. In *4th European conference on planning (ECP'99)*, vol. 1348, pp. 130–142.
6. Cimatti, A., Roveri, M., & Traverso, P. (1998). Strong planning in non-deterministic domains via model checking. In *AIPS*, pp. 36–43.
7. Clarke, E. M., & Emerson, E. A. (1982). Design and synthesis of synchronization skeletons using branching-time temporal logic. In: *Logic of programs, workshop* (pp. 52–71). London, UK: Springer-Verlag.
8. Clarke, E. M., & Wing, J. (1996). Formal methods: state of the art and future directions. In *ACM Computing Systems Surveys*, vol. 28.
9. Daniele, M., Traverso, P., & Vardi, M. Y. (1999). Strong cyclic planning revisited. In *5th European conference on planning (ECP'99)*, vol. 1809, pp. 35–48.
10. Fikes, R. E., & Nilsson, N. J. (1990). STRIPS: A new approach to the application of theorem proving to problem solving. In J. Allen, J. Hendler, & A. Tate (Eds.), *Readings in planning* (pp. 88–97). San Mateo, CA: Kaufmann.
11. Franklin, S., & Graesser, A. (1996). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent agents III. agent theories, architectures and languages (ATAL'96)*, vol 1193, Berlin, Germany, Springer-Verlag.
12. Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning: Theory and practice*. USA: Morgan Kaufmann Publishers Inc.
13. Giunchiglia, F., & Traverso, P. (1999). Planning as model checking. In *5th European conference on planning (ECP'99)*, vol. 1809, pp. 1–20.
14. Kabanza, F., Barbeau, M., & St.-Denis, R. (1997). Planning control rules for reactive agents. *Artificial Intelligence, 95*(1), 67–11.
15. Dal Lago, U., Pistore, M., & Traverso, P. (2002). Planning with a language for extended goals. In *Eighteenth National Conference on Artificial Intelligence* (pp. 447–454). CA, USA: Menlo Park.
16. Müller-Olm, M., Schimidt, D., & Steffen, B. (1999). Model checking: A tutorial introduction. In *SAS'99*, LNCS 1694, pp. 330–354.
17. De Nicola, R., & Vaandrager, F. (1990). Action versus state based logics for transition systems. In *Proceedings of the LITP spring school on theoretical computer science on Semantics of systems of concurrent processes*. New York, NY: Springer-Verlag, pp. 407–419.
18. Pecheur, C., & Raimondi, F. (2006). Symbolic model checking of logics with actions. In *MoChArt 2006* (pp. 1215–1222). Springer Verlag.
19. Pereira, S. L. (2007). Planning under uncertainty for extended reachability goals. PhD thesis, IME-USP.
20. Pereira, S. L., & Barros, L. N. (2007). Nondeterministic planning based on α-CTL: implementation and formal properties. Technical Report RT-MAC-2007-11, IME-USP, São Paulo, Brasil.
21. Ramadge, P. J. G., & Wonham, W. M. (1989). The control of discrete event systems. *Proceedings of the IEEE, 77*(1), 81–98.
22. Russell, S., & Norvig, P. (2002). *Artificial Intelligence: a modern approach* (2nd ed.). New Jersey, USA: Prentice-Hall.
23. Saiedian, H. (1996). An invitation to formal methods. *IEEE Computer, 29*(4), 16–30.