Planning under Uncertainty for Extended Reachability Goals

Silvio do Lago Pereira and Leliane Nunes de Barros
Institute of Mathematics and Statistics - University of São Paulo

Abstract. Planning under uncertainty has being increasingly demanded for practical applications in several areas that require reliable solutions for complex goals and, therefore, some approaches for automatic plan synthesis based on formal methods have been proposed in AI Planning. Among these approaches, planning based on model checking seems to be a very attractive one, however, it is mainly based on CTL and deals only with simple reachability goals, as those in classical planning. In this paper, we introduce a more expressive class of planning goals (*extended reachability goals*) and show that, for this class of goals, the CTL's semantics is not adequate to formalize plan synthesis (or validation) algorithms. Then, as a way to overcome this limitation, we propose a new variant of CTL, based on which we implement plan synthesis (and validation) algorithms capable of solving problems with extended reachability goals, in totally observable nondeterministic environments.

Student level: PhD
Date of conclusion: November, 2007
International publications from the thesis: [13], [14], [15] and [16].
Link: www.teses.usp.br/teses/disponiveis/45/45134/tde-09042008-105750

1 Introduction

In the last few years, *automated planning* [7] has being increasingly demanded for practical applications in several areas that require solutions for complex goals, including autonomous agents [14]. In this setting, formal method based approaches [5, 17] are very attractive to guarantee the reliability of the solutions. In spite of this, the use of formal methods in planning has received relatively little attention and the few related works [2, 3, 6] are almost all *based on model checking* [9]. In this approach, goals are often specified by formulas of the branching time temporal logic CTL [4], a formalism that is only appropriate to deal with planning problems for simple reachability goals and for very specific kinds of more complex goals. An interesting kind of more complex goal, that has not been treated yet by the planning based on model checking community, is given in Example 1.

Example 1. *Roomba is a popular vacuum cleaner robot that, while cleaning a room, is capable of detecting if its battery is weak, driving itself to a recharging station, returning to its original location in order to continue its task. In the future, a project to integrate this vacuum cleaner with an intelligent carpet, capable of mapping the environment, 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. Note that these stations do not need to be in the planned route, but only be reachable from it, and this goal cannot be specified as a simple reachability goal.* ◆

In this paper, 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 the state transitions. To overcome this limitation, we propose a new branching time temporal logic, called α-CTL, basead on which 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).

The remainder of this paper is organized as follows: in Section 2, we present the background on automated planning in nondeterministic environments and define the class of extended reachability goals; in Section 3, we discuss how the model checking framework can be adapted for automated planning and why the CTL's semantics is not appropriate to deal with extended reachability goals; in Section 4, we define the new logic α-CTL and present a model checker based on its semantics; in Section 5, we present a planner based on the α-CTL model checker; and finally, in Section 6, we present our conclusions.

2 Automated Planning

Automated planning [7] is the field of the AI that aim at 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. 1.

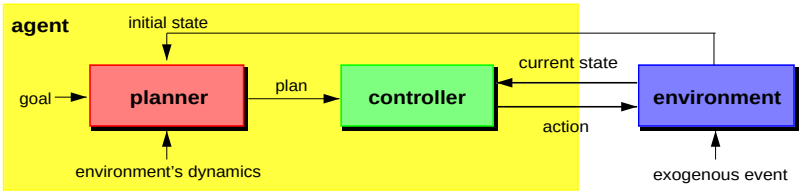

Fig. 1. The components involved in automated nondeterministic planning.

2.1 Domains, problems and solutions

Let $\mathbb{P} \neq \varnothing$ be a finite set of atomic propositions, denoting states properties of an environment, and $\mathbb{A} \neq \varnothing$ 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.

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* $\mathcal{S} \neq \varnothing$ *is a finite set of states,* $\mathcal{L} : \mathcal{S} \mapsto 2^{\mathbb{P}}$ *is a state labeling function, and* $\mathcal{T} : \mathcal{S} \times \mathbb{A} \mapsto 2^{\mathcal{S}}$ *is a state transition function.* ◆

A planning domain can be represented by a *transition graph*, as in Fig. 2-a. We assume that, for all $s \in \mathcal{S}$, we have $\top \in \mathcal{L}(s)$ and $\mathcal{T}(s, \tau) = \{s\}$, where τ is the *trivial* action. Intuitively, action τ denotes that the agent can always 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 $\mathcal{T}(s, a)$, contains all states that can be directly reached by executing a in s. A *policy* (or plan) for a planning domain $\mathcal{D}$ is a partial function $\pi : \mathcal{S} \mapsto \mathbb{A}$ that maps states to actions. The set $\mathcal{S}_\pi$ of states reachable by 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 is the subgraph $\mathcal{D}_\pi \subseteq \mathcal{D}$ that has $\mathcal{S}_\pi$ as set of states and that contains all transitions induced by the actions in π (Fig. 2-b). During the execution of a policy π, if the agent reaches a state not covered by π, it continues executing the action τ.

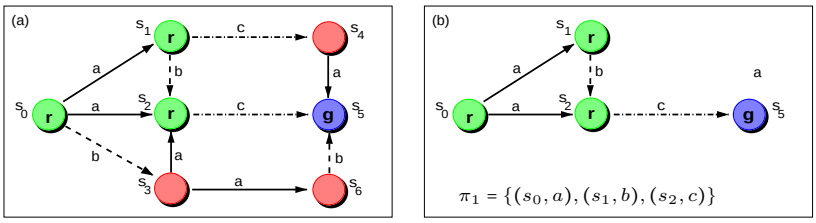

Fig. 2. The planning domain $\mathcal{D}^1$ and the execution structure $\mathcal{D}^1_{\pi_1}$.

Definition 2. *A planning problem is defined by a structure* $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$*, where* $\mathcal{D}$ *is a domain with signature* $(\mathbb{P}, \mathbb{A})$*,* $s_0 \in \mathcal{S}$ *is the initial state of the environment, and* φ *is a propositional formula over* $\mathbb{P}$*, specifying a simple reachability goal.* ◆

There are three classes of solutions for planning problems with simple reachability goals. Intuitively, a *weak solution* is a policy that can lead to a goal state but, due to the nondeterminism, does not guarantee this [2]; a *strong solution* is a policy that always leads to a goal state, in spite of nondeterminism [3]; and a *strong-cyclic solution* is a policy that always leads to a goal, under the fairness assumption that execution will eventually exit from all existing cycles [6].

2.2 Extended reachability goals

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. Extended reachability goals provide 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. For instance, in Fig. 2-b, π_1 is a solution for the planning problem which extended reachability goal is (r, g). Some interesting variations of extended reachability goals are $(\top, \varphi_2)$, that achieves φ_2 (i.e., a *simple reachability goal*); (φ_1, φ_2), that achieves φ_2, while preserving φ_1; and $(\neg\varphi_1, \varphi_2)$, that achieves φ_2, while avoiding φ_1.

Definition 3. *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 a planning problem specifies the agent's goal, but it is up to the agent to decide the quality of the problem solution. The question that arises is: *is it also possible to specify the desired solution quality within the goal specification?* In Section 4, we propose a new logic (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

Model checking consists of deciding if $\mathcal{K} \vDash \varphi$, 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. When applying the model checking framework to automated planning, 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 $(\mathcal{K}, s_0) \vDash \varphi$, 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. 3).

Fig. 3. Planning as model checking.

3.1 Goal Specification in CTL

The semantics of CTL (Computation Tree Logic) [4] 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 total state transition relation. A *path* in $\mathcal{K}$ is a sequence $\langle s_0, s_1, \dots \rangle$ 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 $s_0 \in \mathcal{S}$, the semantics of CTL is defined as following[1]:

$(\mathcal{K}, s_0) \vDash \varphi$ iff $\varphi \in \mathcal{L}(s_0)$;
$(\mathcal{K}, s_0) \vDash \exists \Diamond \varphi$ iff for some path $\langle s_0, s_1, \dots \rangle$ in $\mathcal{K}$, $\exists i \geq 0$ such that $(\mathcal{K}, s_i) \vDash \varphi$;
$(\mathcal{K}, s_0) \vDash \forall \Box \varphi$ iff for every path $\langle s_0, s_1, \dots \rangle$ in $\mathcal{K}$, $\forall i \geq 0$, $(\mathcal{K}, s_i) \vDash \varphi$;
$(\mathcal{K}, s_0) \vDash \forall \Diamond \varphi$ iff for every path $\langle s_0, s_1, \dots \rangle$ in $\mathcal{K}$, $\exists i \geq 0$ such that $(\mathcal{K}, s_i) \vDash \varphi$.

Simple reachability goals with built-in desired solution quality are specified by the formulas $\exists \Diamond \varphi$, $\forall \Box \exists \Diamond \varphi$ and $\forall \Diamond \varphi$, resp., for weak, strong-cyclic and strong solution. Let $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$ be a planning problem, where φ is a goal specification in CTL. Let π be a policy in $\mathcal{D}$. We say that π is a *solution* for $\mathcal{P}$ iff $(\mathcal{K}(\mathcal{D}_\pi), s_0) \vDash \varphi$, where $\mathcal{K}(\mathcal{D}_\pi)$ is the Kripke structure corresponding to the

[1] Due to space limitation, we consider only some CTL operators.

execution structure of π. As we can see, CTL's semantics can easily be used to formalize plan validation algorithms. However, it is not clear how it can also be used to formalize the synthesis of a policy π from a formula φ.

3.2 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 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.

Linear extended reachability goals. In CTL, a linear extended reachability goal with built-in desired solution quality can be specified as $\exists(\varphi_1 \sqcup \varphi_2)$, $\forall(\varphi_1 \sqcup \varphi_2)$ or $\forall \Box \exists(\varphi_1 \sqcup \varphi_2)$, resp., for *weak*, *strong-cyclic* or *strong* solution. Moreover, if φ_1 is $\top$, a linear extended reachability goal reduces to a simple reachability goal and, then, 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 ϕ is a linear extended reachability goal specified in CTL. Let $\mathcal{D}_\pi \subseteq \mathcal{D}$ 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 $\mathcal{P}$ if and only if $(\mathcal{K}(\mathcal{D}_\pi), s_0) \vDash \phi$. As we can see, the CTL's semantics (built in the definition of the satisfiability relation $\vDash$) can indeed be used to formalize plan validation algorithms for the linear subclass of extended reachability goals.

Suppose that the agent in domain $\mathcal{D}^1$ (Fig. 2-a) is at s_0 and its goal is to necessarily reach a state where g holds, passing only through states where r holds. According to the CTL's semantics, this linear extended reachability goal can be specified by the formula $\forall(r \sqcup g)$, but $(\mathcal{K}(\mathcal{D}^1), s_0) \nvDash \forall(r \sqcup g)$, since in $\mathcal{K}(\mathcal{D}^1)$ there is a transition from s_0 to s_3, where 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 [8] to construct a policy and, then, use the CTL's semantics only to guarantee that its execution structure satisfies the goal specification in CTL. For instance, considering policy $\pi_1 = \{(s_0, a), (s_1, b), (s_2, c)\}$, in Fig. 2-b, it is clear that $(\mathcal{K}(\mathcal{D}^1_{\pi_1}), s_0) \vDash \forall(r \sqcup g)$.

Therefore, 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 domain $\mathcal{D}^2$ in Fig. 4-a. In this domain,

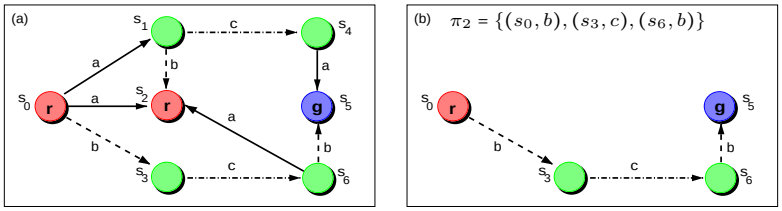

Fig. 4. The planning domain $\mathcal{D}^2$ and the execution structure $\mathcal{D}^2_{\pi_2}$.

the agent could be a mobile robot, r could describe states where there exists a battery recharging station and g could describe the final state that the robot wants to reach. Suppose that the agent's goal is, starting from s_0, *necessarily to reach a state satisfying g, passing only through states from which a 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. 4-a:
  - 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. 4-b) *does not* satisfy the goal specified by the CTL formula $\forall((r \vee \forall\bigcirc r \vee \forall\bigcirc\forall\bigcirc r) \sqcup g)$.

Therefore, we must conclude that CTL cannot deal with branching extended reachability goals.

4 The new temporal logic α-CTL

In this section, we present the logic α-CTL, based on which we implement a model checker that, in Section 5, is adapted for 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 φ holds on *all* immediate successors of s, independently of the actions labeling the transitions from s to them. In α-CTL, to enforce that actions play an important role in its semantics, we use a different set of dotted symbols to denote temporal operators: $\odot$ (*next*), $\boxdot$ (*invariantly*), $\diamondsuit$ (*finally*) and $\sqcup\!\!\cdot$ (*until*).

Definition 4. *Let $p \in \mathbb{P}$ be a proposition. The syntax of α-CTL is defined as* $\varphi ::= p \mid \neg p \mid (\varphi \wedge \varphi') \mid (\varphi \vee \varphi') \mid \exists \odot \varphi \mid \forall \odot \varphi \mid \exists \boxdot \varphi \mid \forall \boxdot \varphi \mid \exists(\varphi \sqcup \varphi') \mid \forall(\varphi \sqcup \varphi')$ ◆

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 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 $\exists \diamond \varphi \doteq \exists(\top \sqcup \varphi)$ and $\forall \diamond \varphi \doteq \forall(\top \sqcup \varphi)$.

Although actions are essential in the semantics of α-CTL, they are not used to compose α-CTL's formulas. In general, when we specify a planning goal, we wish to impose constraints only over the states visited during the plan execution (constraints over actions used in the plan are not relevant).

4.2 The semantics of α-CTL

Let $\mathbb{P} \neq \varnothing$ be a finite set of atomic propositions and $\mathbb{A} \neq \varnothing$ 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 a temporal model, *terminal states* (i.e., states where the only executable action is τ) persist infinitely in time. In short, a temporal model for α-CTL is a nondeterministic 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.

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 $\mathcal{D}^1$ (Fig. 2-a), the strong preimage of the set $Y = \{s_2\}$ is the set $X = \{s_1\}$, since s_1 is the only state in $\mathcal{D}^1$ from which we can necessarily reach s_2 after one step.

Definition 5. *Let $Y \subseteq \mathcal{S}$ be a set of states. The weak preimage of Y, denoted by $\mathcal{T}^-_\exists(Y)$, is the set $\{s \in \mathcal{S} : \exists a \in \mathbb{A} \,.\, \mathcal{T}(s,a) \cap Y \neq \varnothing\}$, and the strong preimage of Y, denoted by $\mathcal{T}^-_\forall(Y)$, is the set $\{s \in \mathcal{S} : \exists a \in \mathbb{A} \,.\, \varnothing \neq \mathcal{T}(s,a) \subseteq Y\}$.* ◆

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 6. *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 $[\![\bot]\!]_\mathcal{D} = \varnothing$)*
- $[\![\neg p]\!]_\mathcal{D} = \mathcal{S} \setminus [\![p]\!]_\mathcal{D}$
- $[\![(\varphi \wedge \varphi')]\!]_\mathcal{D} = [\![\varphi]\!]_\mathcal{D} \cap [\![\varphi']\!]_\mathcal{D}$

- $[\![(\varphi \vee \varphi')]\!]_{\mathcal{D}} = [\![\varphi]\!]_{\mathcal{D}} \cup [\![\varphi']\!]_{\mathcal{D}}$
- $[\![\exists \odot \varphi]\!]_{\mathcal{D}} = \mathcal{T}_{\exists}^{-}([\![\varphi]\!]_{\mathcal{D}})$
- $[\![\forall \odot \varphi]\!]_{\mathcal{D}} = \mathcal{T}_{\forall}^{-}([\![\varphi]\!]_{\mathcal{D}})$
- $[\![\exists \boxdot \varphi]\!]_{\mathcal{D}} = \nu Y.([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\exists}^{-}(Y))$
- $[\![\forall \boxdot \varphi]\!]_{\mathcal{D}} = \nu Y.([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\forall}^{-}(Y))$
- $[\![\exists(\varphi \sqcup \varphi')]\!]_{\mathcal{D}} = \mu Y.([\![\varphi']\!]_{\mathcal{D}} \cup ([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\exists}^{-}(Y)))$
- $[\![\forall(\varphi \sqcup \varphi')]\!]_{\mathcal{D}} = \mu Y.([\![\varphi']\!]_{\mathcal{D}} \cup ([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\forall}^{-}(Y)))$ ◆

4.3 A Model Checker for α-CTL

We have implemented a model checker (Chap. 5 in [12]), directly from the α-CTL's semantics. Given a 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 φ in $\mathcal{D}$; then, if C is the empty set, it returns *success*; otherwise, it returns C as counter-example.

α-ModelChecker$(\varphi, \mathcal{D})$
1 $C \leftarrow \mathcal{S} \setminus$ Intension$(\varphi, \mathcal{D})$
2 **if** $C = \varnothing$ **then return success**
3 **else return** C

The basic operation on this model checker is performed by the function Intension, that inductively computes the intension of the formula φ in the model $\mathcal{D}$ (see Definition 6). The following results [12] 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* Intension$(\varphi, \mathcal{D})$ *returns the set $[\![\varphi]\!]_{\mathcal{D}}$.*

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

5 Planning with α-CTL

In α-CTL, an extended reachability goal with built-in desired solution quality can be specified as $\exists(\phi \sqcup \varphi)$, $\forall \boxdot \exists(\phi \sqcup \varphi)$ and $\forall(\phi \sqcup \varphi)$, respectively, for *weak*, *strong-cyclic* and *strong* solution. Given a planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, \gamma \rangle$, where γ is an extended reachability goal specified in α-CTL, a solution for $\mathcal{P}$ can be obtained by the following planning algorithm:

α-Planner$(\mathcal{P})$
1 $M \leftarrow$ Model$(\texttt{lfp}, \gamma, \mathcal{D})$
2 $C \leftarrow$ StatesCoveredBy(M)
3 **if** $s_0 \in C$ **return** Policy(M)
4 **else return failure**

This algorithm (see details in [12]) 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 γ, from s_0, in the domain $\mathcal{D}$; otherwise, it returns *failure*.

The basic operation on the α-CTL planner is performed by function MODEL. To implement this function, we have reformulated the notion of intension of a formula γ (Definition 6) such that $[\![\gamma]\!]_{\mathcal{D}}$ turns out to be a subgraph of $\mathcal{D}$ containing all the states satisfying γ, as well as all the transitions considered during the selection of these states (we have essentially redefined preimage functions such that they collect the pair (s, a), whenever the action a is considered to show that s satisfies the property γ). With this reformulation we can synthesize plans as a collateral effect of the verification of property γ in the temporal model $\mathcal{D}$.

To extract a policy from a submodel M synthesized by the function MODEL, the planner calls the function POLICY(M), that returns a policy π such that: (i) STATESCOVEREDBY(π) = STATESCOVEREDBY(M), and (ii), for all pairs (s, a), $(s', a') \in \pi$, if $s = s'$, then $a = a'$.

The following theorems, whose proofs are presented in [12], 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, 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 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 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 [7]. 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 [2, 3, 6]. Besides, in general, those works are related to planning based on model checking techniques, for simple reachability goals in nondeterministic environments.

In this work, we introduce the class of *extended reachability goals* and show 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 [10, 11], 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 implemented a sound and complete planning algorithm 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 emphasize that the existing works on planning for extended goals either propose an *ad hoc* planning algorithm [8], without proving its validity through formal analysis; or propose a new logic that can be used to specify extended goals and do plan validation [1], without presenting any plan synthesis algorithm (making the assumption that policies are given *a priori*). With this work, we have provided both: a logic that can be used as a formal language to specify extended reachability goals and a planning system based on this logic, whose relevance is attested by the international publications [13, 14, 16, 15].

References

1. C. Baral and J. Zhao. Goal specification, non-determinism and quantifying over policies. In *AAAI*, 2006.
2. A. Cimatti, F. Giunchiglia, E. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for $\mathcal{AR}$. In *ECP*, pages 130–142, 1997.
3. A. Cimatti, M. Roveri, and P. Traverso. Strong planning in non-deterministic domains via model checking. In *AIPS*, pages 36–43, 1998.
4. E. M. Clarke. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, pages 52–71. Springer, 1982.
5. E. M. Clarke and J. Wing. Formal methods: state of the art and future directions. In *ACM Computing Systems Surveys*, volume 28, 1996.
6. M. Daniele, P. Traverso, and M. Y. Vardi. Strong cyclic planning revisited. In *ECP*, pages 35–48, 1999.
7. M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers Inc., USA, 2004.
8. U. Dal Lago, M. Pistore, and P. Traverso. Planning with a language for extended goals. In *8th NCAI*, pages 447–454, Menlo Park, CA, USA, 2002.
9. M. Muller-Olm, D. Schimidt, and B. Steffen. Model checking: A tutorial introduction. In *SAS'99, LNCS 1694*, pages 330–354, 1999.
10. R. De Nicola and F. Vaandrager. Action versus state based logics for transition systems. In *LITP*, pages 407–419, New York, NY, USA, 1990. Springer.
11. C. Pecheur and F. Raimondi. Symbolic model checking of logics with actions. In *MoChArt 2006*, pages 1215–1222. Springer Verlag, 2006.
12. S. L. Pereira. Planning under uncertainty for extended reachability goals. In *PhD thesis, IME-USP*, 2007.
13. S. L. Pereira and L. N. Barros. Using α-CTL to specify complex planning goals. In *WoLLIC'2008*, volume 5110 of *LNAI*, pages 250–260.
14. S. L. Pereira and L. N. Barros. A logic-based agent that plans for extended reachability goals. *Autonomous Agents and Multi-Agent Systems*, 9034:327–344, 2008.
15. S. L. Pereira, L. N. Barros, and F. G. Cozman. Strong probaliblistic planning (accepted). In *MICAI'2008*, pages 1–8.
16. S. L. Pereira, L. N. Barros, and F. G. Cozman. Strong probaliblistic planning (poster presentation). In *ICAPS'2008*, pages 1–8.
17. H. Saiedian. An invitation to formal methods. *IEEE Computer*, 29(4):16–30, 1996.