Journal of Applied Logic 10 (2012) 419–430

Contents lists available at SciVerse ScienceDirect

Journal of Applied Logic

www.elsevier.com/locate/jal

A planner agent that tries its best in presence of nondeterminism

Silvio do Lago Pereira [a,*,1], Leliane Nunes de Barros [b]

[a] Information Technology Department at FATEC-SP, Praça Cel. Fernando Prestes 30, 01124-060, São Paulo, Brazil
[b] Computer Science Department at IME-USP, Rua do Matão 1010, 04717-020, São Paulo, Brazil

ARTICLE INFO

Article history:
Received 21 November 2011
Accepted 20 June 2012
Available online 30 June 2012

Keywords:
Automated planning
Planning under uncertainty
Planning as model checking
Temporal logic

ABSTRACT

In many nondeterministic planning domains, an agent whose goal is to achieve φ may not succeed in finding a policy that can guarantee that all paths from the initial state lead to a final state where φ holds (*strong solution*). Nevertheless, if the agent is trying its best to achieve φ, it cannot give up. Instead, it may be inclined to accept weaker guarantees, such as having a path leading to φ from any intermediate state reached by the policy (*strong-cyclic solution*), or even less, such as having at least one path leading to φ from the initial state (*weak solution*). But the agent should choose among such different options based on their availability in each situation. Although the specification of this type of goal has been addressed before, in this paper we show how a planner based on the branching time temporal logic α-CTL can be used to plan for intuitive and useful goals of the form "*try your best to achieve* φ".

1. Introduction

Classical AI planning assumes that agent's actions have deterministic effects and that its goal is to achieve φ (i.e., to reach a state where the property φ holds) [12]. Although this assumption can simplify the planning task [5], it almost never corresponds to the reality. In practical planning applications, actions have *nondeterministic effects* and we often need to specify *extended goals*, i.e., conditions that must be satisfied not only in the reached final state, but also in each intermediate state visited by the agent to achieve its goal.

To deal with extended goals, many works in the planning literature [6,7,10] have proposed the use of the branching time temporal logic CTL [8]. However, as first pointed in [9], there are many intuitive and useful extended goals that cannot be expressed in CTL. For instance, goals of the form "*try your best to achieve* φ", where φ is a property describing desired final states. In many nondeterministic planning domains, an agent whose goal is to achieve φ may not succeed in finding a policy that can guarantee that all paths from the initial state lead to a final state where φ holds (*strong solution*). Nevertheless, if the agent is trying its best to achieve φ, it cannot give up. Instead, it may be inclined to accept weaker guarantees, such as having a path leading to φ from any intermediate state reached by the policy (*strong-cyclic solution*), or even less, such as having at least one path leading to φ from the initial state (*weak solution*). But the agent should choose among such different options based on their availability in each situation. Example 1, adapted from [4], illustrates the difficulty associated with the specification of this kind of goal.

Example 1. Consider the nondeterministic planning domain depicted in Fig. 1. Suppose that initially the agent is in the state s_0 and that its goal is to try its best to achieve g, i.e., to reach the final state s_4. Thus, in the state s_0, since there

* Corresponding author.
E-mail addresses: slago@pq.cnpq.br (S.L. Pereira), leliane@ime.usp.br (L.N. Barros).
[1] The author would like to thank CNPq (grant 304322/2009-1) for financial support.

1570-8683/$ – see front matter © 2012 Published by Elsevier B.V.
http://dx.doi.org/10.1016/j.jal.2012.06.002

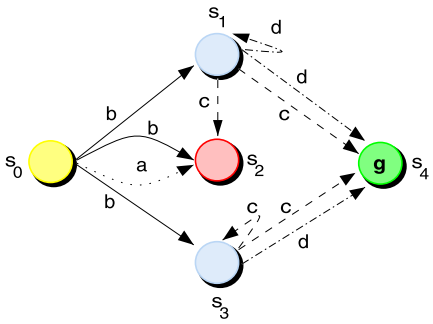

Fig. 1. The nondeterministic planning domain $\mathcal{D}^1$.

is no action that can guaranteedly lead it to s_4, the agent is content with a weak solution (action b). But after executing action b in state s_0, the agent must change its intention; otherwise it can make a poor decision in the next current state. For example, if the agent reaches state s_1 without changing its intention, it can choose action c (weak solution) instead of action d (strong-cyclic solution), which is the best choice in s_1. Analogously, in state s_3, it can choose action c (strong-cyclic solution) instead of action d (strong solution), which is the best choice in state s_3. In short, if an agent is trying its best to achieve its goal, it should be capable of changing its intention all along the planning task. □

Related works. To overcome the limitation of CTL with respect to planning for extended goals, in the last years, some new branching time temporal logic have been proposed. The logic π-CTL*, proposed in [3], extends the semantics of CTL by introducing quantification over paths of a given policy π (operators $\forall_\pi$ and $\forall_\pi$). An algorithm based on Horn SAT encoding and logic programming, capable of finding strong, strong-cyclic and weak solutions for goals specified in π-CTL* was presented in [2]. However, as pointed in [4], the semantics of π-CTL* is yet not expressive enough to specify goals of the kind "*try your best to achieve* φ". Thus, a new logic called P-CTL*, which extends π-CTL* with quantification over policies (operators $\mathcal{EP}$ and $\mathcal{AP}$) was proposed in [4]. Although the semantics of P-CTL* allows the specification of very complex extended goals, to the best of our knowledge, no planning algorithm based on its semantics has been presented so far. Other recent works that address nondeterministic planning for extended goals are [18] and [19]. The first proposes a new variant of CTL that uses quantification over immediate successor states (operators $\forall\odot$ and $\exists\odot$), called α-CTL, and shows how to specify extended reachability goals in this new proposed logic, as well as to plan for them by using slightly modified versions of the standard CTL model checking algorithms [15]. The second shows how α-CTL can be used for the specification of some other complex planning goals.

In this paper we show how the logic α-CTL can be used to plan for intuitive and useful extended goals of the form "*try your best to achieve* φ". Although the specification of this kind of extended goal in P-CTL* has been addressed before in [4], we claim that a planning algorithm based on α-CTL is simpler and demands less computational effort than one based on P-CTL*, since the universal quantification over policies required in P-CTL* is a very costly operation.

The remainder of this paper is organized as follows. First, we briefly present the background on planning based on model checking and emphasize the frailty of CTL to formalize plan synthesis algorithms. Next, we present the temporal logic α-CTL and show how a plan synthesis algorithm for extended reachability goals can be implemented based on its semantics. Then, using this plan synthesis algorithm, we show how to synthesize policies for goals of the form "*try your best to achieve* φ". Finally, we present our conclusions.

2. Planning based on model checking

Model checking consists of deciding if $\mathcal{K} \models \varphi$, 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. 2) is an algorithm that receives a pair $(\mathcal{K}, \varphi)$ as input and systematically visits the states of the model $\mathcal{K}$, verifying if the property φ holds. When all states in $\mathcal{K}$ satisfy φ, the model checker returns *success*; otherwise, it returns a *counter-example* (e.g., a state in $\mathcal{K}$ where φ does not hold).

When applying the model checking framework to automated planning [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 behavioral policy that allows for the agent to achieve its goal); otherwise, the planner returns *failure* (Fig. 3).

2.1. Domains, problems and solutions

Let $\mathbb{P} \neq \emptyset$ be a finite set of atomic propositions denoting state properties and $\mathbb{A} \neq \emptyset$ a finite set of actions representing the agent's abilities. A *planning domain* over $(\mathbb{P}, \mathbb{A})$ is a state transition graph $\mathcal{D} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{L} : \mathcal{S} \mapsto 2^{\mathbb{P}}$ is a state labeling function, and $\mathcal{T} : \mathcal{S} \times \mathbb{A} \mapsto 2^{\mathcal{S}}$ is a nondeterministic transition function. A *planning problem* $\mathcal{P}$ in a domain $\mathcal{D}$ is defined by an *initial state* $s_0 \in \mathcal{S}$ and by a goal formula φ, describing a set of *goal states* $\mathcal{G} = \{s \in \mathcal{S}:$

Fig. 2. The model checking framework.

Fig. 3. Planning as model checking.

$(\mathcal{D}, s) \models \varphi\}$. A *policy* $\pi : \mathcal{S} \mapsto \mathbb{A}$ is a partial function mapping from states to actions. The set $\mathcal{S}_\pi$ of states reached by a policy π is $\{s\colon\ (s,a) \in \pi\} \cup \{s'\colon\ (s,a) \in \pi$ and $s' \in \mathcal{T}(s,a)\}$. Given a policy π, the corresponding *execution structure* $\mathcal{D}_\pi$ is the subgraph of $\mathcal{D}$ that has $\mathcal{S}_\pi$ as set of states and contains all transitions induced by the actions in policy π. Let $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$ be a planning problem, and π a policy in $\mathcal{D}$ (with execution structure $\mathcal{D}_\pi$). We can distinguish three different qualities of solutions. We say that π is a:

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

Intuitively, a *weak solution* is a policy that can achieve a goal state; but due to the nondeterminism, it does not guarantee to do so [6]; a *strong-cyclic solution* is a policy that always achieves the goal, under the fairness assumption that its execution will eventually exit from all existing cycles [10]; and a *strong solution* is a policy that always achieves a goal state, in spite of nondeterminism [7].

2.2. Goal specification in CTL

In a nondeterministic environment, an agent is not always capable of knowing exactly what is the next state of the environment after the execution of an action. Therefore, when a planner 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*) [8] 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 this logic. Indeed, CTL is the main formalism that has been used to specify nondeterministic planning problems and related algorithms based on model checking [6,7,10].

The CTL formulas are composed by atomic propositions, propositional operators, and temporal operators. The symbols $\circ$ (*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:

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

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 total state transition relation [14]. A *path* in $\mathcal{K}$ is a sequence of states $\langle s_0, s_1, s_2, \ldots \rangle$ such that $s_i \in \mathcal{S}$ and $(s_i, s_{i+1}) \in \mathcal{T}$, for all $i \geqslant 0$. Given a Kripke structure $\mathcal{K}$ and a state $s_0 \in \mathcal{S}$, the semantics of CTL 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 $(\mathcal{K}, s_0) \models \varphi'$;
$(\mathcal{K}, s_0) \models \exists \circ \varphi$	iff for some path $\langle s_0, s_1, \ldots \rangle$ in $\mathcal{K}$, $(\mathcal{K}, s_1) \models \varphi$;
$(\mathcal{K}, s_0) \models \forall \circ \varphi$	iff for every path $\langle s_0, s_1, \ldots \rangle$ in $\mathcal{K}$, $(\mathcal{K}, s_1) \models \varphi$;
$(\mathcal{K}, s_0) \models \exists \Box \varphi$	iff for some path $\langle s_0, s_1, \ldots \rangle$ in $\mathcal{K}$, for $i \geqslant 0$, $(\mathcal{K}, s_i) \models \varphi$;
$(\mathcal{K}, s_0) \models \forall \Box \varphi$	iff for every path $\langle s_0, s_1, \ldots \rangle$ in $\mathcal{K}$, for $i \geqslant 0$, $(\mathcal{K}, s_i) \models \varphi$;
$(\mathcal{K}, s_0) \models \exists(\varphi \sqcup \varphi')$	iff for some path $\langle s_0, s_1, \ldots \rangle$ in $\mathcal{K}$, there exists $i \geqslant 0$ such that $(\mathcal{K}, s_i) \models \varphi'$ and, for $0 \leqslant j < i$, $(\mathcal{K}, s_j) \models \varphi$;
$(\mathcal{K}, s_0) \models \forall(\varphi \sqcup \varphi')$	iff for every path $\langle s_0, s_1, \ldots \rangle$ in $\mathcal{K}$, there exists $i \geqslant 0$ such that $(\mathcal{K}, s_i) \models \varphi'$ and, for $0 \leqslant j < i$, $(\mathcal{K}, s_j) \models \varphi$.

A reachability goal with built-in desired solution quality can be expressed in CTL [8] by the formula $\exists \Diamond \varphi$, $\forall \Box \exists \Diamond \varphi$ or $\forall \Diamond \varphi$, respectively, for weak, strong-cyclic and strong solution.

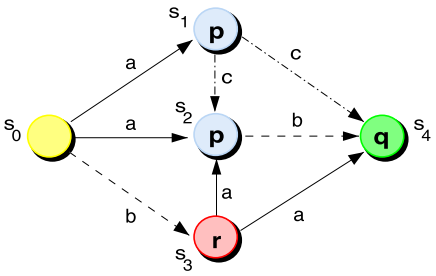

Fig. 4. The nondeterministic planning domain $\mathcal{D}^2$.

Let $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$ be a planning problem, where $\mathcal{D}$ is a planning domain, s_0 is an initial state and γ is a reachability goal with built-in desired solution quality specified in CTL. Let π be a policy in $\mathcal{D}$. We say that π is a *solution* for $\mathcal{P}$ if and only if $(\mathcal{K}(\mathcal{D}_\pi), s_0) \models \gamma$, where $\mathcal{K}(\mathcal{D}_\pi)$ is the Kripke structure corresponding to the 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 be used to formalize the synthesis of a policy π from a CTL formula γ.

3. The branching time temporal logic α-CTL

In this section, we present the branching time temporal logic α-CTL [18]. Then, based on its semantics, we implement a planning algorithm for extended reachability goals with built-in desired solution quality.

3.1. 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 $\sqcup$ (*until*).

Definition 1. Let $p \in \mathbb{P}$ be an atomic proposition. The **syntax** of α-CTL is inductively defined as $\varphi ::= \top \mid p \mid \neg p \mid (\varphi \wedge \varphi') \mid (\varphi \vee \varphi') \mid \exists \odot \varphi \mid \forall \odot \varphi \mid \exists \boxdot \varphi \mid \forall \boxdot \varphi \mid \exists(\varphi \sqcup \varphi') \mid \forall(\varphi \sqcup \varphi')$. □

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 $\diamondsuit$ are defined as $\exists \diamondsuit \varphi \doteq \exists(\top \sqcup \varphi)$ and $\forall \diamondsuit \varphi \doteq \forall(\top \sqcup \varphi)$.

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). For this reason, existing actions logics [11,16], which allow constraints over actions, are inadequate for our purpose.

3.2. 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 state 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 the trivial action 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 an α-CTL 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 that 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 $\exists$ requires that *some* of these α-*successors* satisfy φ.

For instance, consider the nondeterministic planning domain $\mathcal{D}^2$, depicted in Fig. 4. 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] $(\mathcal{D}^2, s_0) \models \forall \odot p$. Furthermore, it also follows that $(\mathcal{D}^2, s_0) \models \forall \odot \neg p$ (by choosing action b in 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 $(\mathcal{D}^2, s_0) \models \forall \odot p \wedge \forall \odot \neg p$ does not mean that there exists a policy to achieve both

[2] According to CTL's semantics, it follows that $(\mathcal{K}(\mathcal{D}^2), s_0) \not\models \forall \bigcirc p$.

subgoals p and $\neg p$ at the same time[3]; it only means that, from state s_0, the agent can choose to reach p or $\neg p$ in the next state. This possibility of choosing is very important: if the agent cannot make choices, there is no need of planning.

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 $\mathcal{D}^2$ (Fig. 4), the strong preimage of the set $Y = \{s_4\}$ is the set $X = \{s_2\}$, since s_2 is the only state in $\mathcal{D}^2$ from which we can necessarily reach s_4 after one step.

Definition 2. Let $Y \subseteq \mathcal{S}$ be a set of states. The **weak preimage** of Y, denoted by $\mathcal{T}_{\exists}^{-}(Y)$, is the set $\{s \in \mathcal{S}\colon\ \exists a \in \mathbb{A}\,.\,\mathcal{T}(s,a) \cap Y \neq \emptyset\}$, and the **strong preimage** of Y, denoted by $\mathcal{T}_{\forall}^{-}(Y)$, is the set $\{s \in \mathcal{S}\colon\ \exists a \in \mathbb{A}\,.\,\emptyset \neq \mathcal{T}(s,a) \subseteq Y\}$. □

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 3. 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:

- $[\![p]\!]_{\mathcal{D}} = \{s\colon\ p \in \mathcal{L}(s)\}$ ($[\![\top]\!]_{\mathcal{D}} = \mathcal{S}$ 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 4. 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. The α-CTL planning algorithm

Formally, an extended reachability goal is a pair of formulas (ϕ, φ), where ϕ is a condition to be *preserved* during the policy execution and φ is a condition to be *achieved* at the end of the policy execution. Using α-CTL, an extended reachability goal with built-in desired solution quality γ can be specified as follows:

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

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

```
α-PLANNER(P)
1 M ← MODEL(lfp, γ, D)
2 C ← STATESCOVEREDBY(M)
3 if s0 ∈ 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 γ, 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 (see next subsection), 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 ∈ S: s ∈ M} ∪ {s ∈ S: (s, a) ∈ M}
```

[3] An impossible goal that would be specified as $\forall \odot (p \wedge \neg p)$.

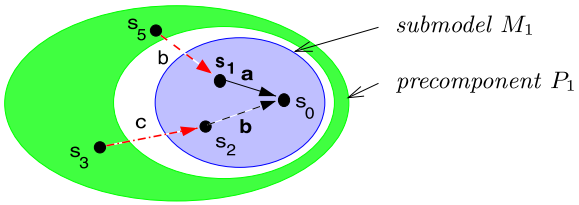

Fig. 5. Model synthesis for local temporal operators.

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 $a = a'$.

4.1. Model synthesis

The notion of intension of an α-CTL formula γ (Definition 3) can be reformulated 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 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 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 synthesize a submodel from an α-CTL formula γ and a model $\mathcal{D}$, the α-CTL planner uses the function Model, whose first parameter (sc) informs the scope of computation: `lfp` (least fixpoint, which avoids cycles) or `gfp` (greatest fixpoint, which 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($sc, \gamma, \mathcal{D}$)
1 `if` $\gamma \in \mathbb{P}$ `return` $\{s \in \mathcal{S}:\ \gamma \in \mathcal{L}(s)\}$
2 `if` $\gamma = \neg\phi$ `return` $\mathcal{S} \setminus$ Model($sc, \phi, \mathcal{D}$)
3 `if` $\gamma = \phi \wedge \varphi$ `return` Model($sc, \phi, \mathcal{D}$) $\cap$ Model($sc, \varphi, \mathcal{D}$)
4 `if` $\gamma = \phi \vee \varphi$ `return` Model($sc, \phi, \mathcal{D}$) $\cup$ Model($sc, \varphi, \mathcal{D}$)
5 `if` $\gamma = \exists\odot\varphi$ `return` Model$_{\odot}$(sc, WeakPreimageMap, $\varphi, \mathcal{D}$)
6 `if` $\gamma = \forall\odot\varphi$ `return` Model$_{\odot}$(sc, StrongPreimageMap, $\varphi, \mathcal{D}$)
7 `if` $\gamma = \exists\boxdot\varphi$ `return` Model$_{\boxdot}$(`gfp`, WeakPreimageMap, $\varphi, \mathcal{D}$)
8 `if` $\gamma = \forall\boxdot\varphi$ `return` Model$_{\boxdot}$(`gfp`, StrongPreimageMap, $\varphi, \mathcal{D}$)
9 `if` $\gamma = \exists\diamondsuit\varphi$ `return` Model$_{\sqcup}$(sc, WeakPreimageMap, $\top, \varphi, \mathcal{D}$)
10 `if` $\gamma = \forall\diamondsuit\varphi$ `return` Model$_{\sqcup}$(sc, StrongPreimageMap, $\top, \varphi, \mathcal{D}$)
11 `if` $\gamma = \exists(\phi \sqcup \varphi)$ `return` Model$_{\sqcup}$(sc, WeakPreimageMap, $\phi, \varphi, \mathcal{D}$)
12 `if` $\gamma = \forall(\phi \sqcup \varphi)$ `return` Model$_{\sqcup}$(sc, StrongPreimageMap, $\phi, \varphi, \mathcal{D}$)

Propositional formulas. A submodel M synthesized from a propositional formula γ 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 $s \models \gamma$. 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$_{\odot}$(sc, *PreimageMapFunc*, $\varphi, \mathcal{D}$)
1 $M_1 \leftarrow$ Model($sc, \varphi, \mathcal{D}$)
2 $I_1 \leftarrow$ *PreimageMapFunc*(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 $\exists\odot\varphi$ or $\forall\odot\varphi$, this function starts by synthesizing a submodel M_1 from subformula φ 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 ($\exists\odot\varphi$ or $\forall\odot\varphi$), as depicted in Fig. 5.

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

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

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

STRONGPREIMAGEMAP$(C, \mathcal{D})$
1 return $\{(s,a) \colon s \in \mathcal{S},\ a \in \mathbb{A}$ 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$_{\boxdot}(sc, \mathit{PreimageMapFunc}, \varphi, \mathcal{D})$
1 $M \leftarrow$ MODEL$(sc, \varphi, \mathcal{D})$
2 $M' \leftarrow \emptyset$
3 while $M' \neq M$ do
4 $M' \leftarrow M$
5 $C \leftarrow$ STATESCOVEREDBY(M)
6 $I \leftarrow \mathit{PreimageMapFunc}(C, \mathcal{D})$
7 $M \leftarrow$ PRUNE$_{\boxdot}(I, C)$
8 return M

In order to synthesize a submodel of $\mathcal{D}$, from a formula $\exists \boxdot \varphi$ or $\forall \boxdot \varphi$, this function starts by synthesizing a submodel M from subformula φ. 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 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 the result of the function) satisfies the goal specified by the initial formula ($\exists \boxdot \varphi$ or $\forall \boxdot \varphi$). The pruning function called by MODEL$_{\boxdot}$ is defined as follows:

PRUNE$_{\boxdot}(I, C)$
1 return $\{(s,a) \in I \colon s \in C\}$

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

MODEL$_{\sqcup}(sc, \mathit{PreimageMapFunc}, \phi, \varphi, \mathcal{D})$
1 $C_1 \leftarrow$ STATESCOVEREDBY(MODEL(lfp, $\phi, \mathcal{D}$))
2 $M_2 \leftarrow$ MODEL$(sc, \varphi, \mathcal{D})$
3 $M_2' \leftarrow \emptyset$
4 while $M_2 \neq M_2'$ do
5 $M_2' \leftarrow M_2$
6 $C_2 \leftarrow$ STATESCOVEREDBY(M_2)
7 $I_2 \leftarrow \mathit{PreimageMapFunc}(C_2, \mathcal{D})$
8 $P_2 \leftarrow$ PRUNE$_{\sqcup}(\mathit{scope}, I_2, C_1, C_2)$
9 $M_2 \leftarrow M_2 \cup P_2$
10 return M_2

In order to synthesize a submodel of $\mathcal{D}$, from a formula $\exists(\phi \sqcup \varphi)$ or $\forall(\phi \sqcup \varphi)$, this function starts by computing the set C_1 of states that satisfy the subformula ϕ and synthesizing a submodel M_2 from the formula φ. 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 ($sc =$ 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 the result of the function) satisfies the goal specified by the initial formula ($\exists(\phi \sqcup \varphi)$ or $\forall(\phi \sqcup \varphi)$). The pruning function for MODEL$_{\sqcup}$ is defined as follows:

PRUNE$_{\sqcup}(sc, I_2, C_1, C_2)$
1 $P_2 \leftarrow \{(s,a) \in I_2 \colon s \in C_1\}$
2 if $sc = \texttt{lfp}$ then $P_2 \leftarrow \{(s,a) \in P_2 \colon s \notin C_2\}$
3 return P_2

4.2. Formal properties of the algorithm α-PLANNER

The following theorems, whose proofs are found in [17], establish some formal properties of the α-CTL planner.

Theorem 1. *Given an α-CTL formula γ and a temporal model $\mathcal{D}$, the set of states of the submodel returned by the function* MODEL *is $[\![\gamma]\!]_{\mathcal{D}}$.*

Theorem 2. *Let $\mathcal{P} = \langle \mathcal{D}, s_0, \exists(\phi \sqcup \varphi)\rangle$ 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 $\mathcal{P} = \langle \mathcal{D}, s_0, \forall \boxdot \exists(\phi \sqcup \varphi)\rangle$ 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 4. *Let $\mathcal{P} = \langle \mathcal{D}, s_0, \forall(\phi \sqcup \varphi)\rangle$ 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 5. *Let $\mathcal{P} = \langle \mathcal{D}, s_0, \gamma\rangle$ be a planning problem for an extended reachability goal γ. Then, α-PLANNER$(\mathcal{P})$ fails if and only if $\mathcal{P}$ has no solution.*

5. An agent that tries its best

In this section, we show how the α-CTL planning algorithm can be used to implement a planner for reachability goals of the form "*try your best to achieve φ*", where the propositional formula φ describes goal states.

5.1. Why we need a specialized planning algorithm?

To better understand why we need a specialized planning algorithm to deal with *try-your-best* goals, consider the planning domain $\mathcal{D}^1$ (Fig. 6(a)) and the possible policies in this domain (Figs. 6(b) to 6(f)):

$$\pi_1 = \{(s_0, a)\}$$
$$\pi_2 = \{(s_0, b), (s_1, c), (s_3, c)\}$$
$$\pi_3 = \{(s_0, b), (s_1, c), (s_3, d)\}$$
$$\pi_4 = \{(s_0, b), (s_1, d), (s_3, c)\}$$
$$\pi_5 = \{(s_0, b), (s_1, d), (s_3, d)\}$$

Suppose that the agent is initially in state s_0 and that its goal is to try its best to reach a final state satisfying the property g. Clearly, if we specify this goal by the formula $\forall \diamondsuit g$ (*strong solution*), it turns out to be unachievable from the state s_0 and, then, the α-CTL planner returns failure. Due to nondeterminism, after the execution of action a in the state s_0, we cannot guarantee that a goal state still can be reached (since action a can lead to the dead-end s_2). The same happen if we specify the goal by the formula $\forall \boxdot \exists \diamondsuit g$ (*strong-cyclic solution*). On the other hand, if the goal is specified by the formula $\exists \diamondsuit g$ (*weak solution*), the policy $\pi_2 = \{(s_0, b), (s_1, c), (s_3, c)\}$ can be returned as solution by the α-CTL planner. However, by following this policy, the agent is not trying its best (see the preference relation for policies in domain $\mathcal{D}^1$, depicted in Fig. 7). As we can see, the formulas $\forall \diamondsuit g$, $\forall \boxdot \exists \diamondsuit p$ and $\exists \diamondsuit g$ are inappropriate to express the goal "*try your best to achieve g*".

5.2. How an agent that tries its best should proceed?

Starting from the initial state s_0 in domain $\mathcal{D}^1$ (Fig. 6(a)), the agent has to choose between action a, that can lead it to a dead-end, or action b, that can lead it to a state from where it can still achieve its goal. Clearly, if the agent is trying its best, it should choose action b. With this choice, the agent does not guarantee to achieve its goal; however, at this point, this is the best that it can do (there is no strong nor strong-cyclic solution from state s_0 and the agent must be "happy" with a weak solution). After executing action b in state s_0, the agent can reach one of the following states:

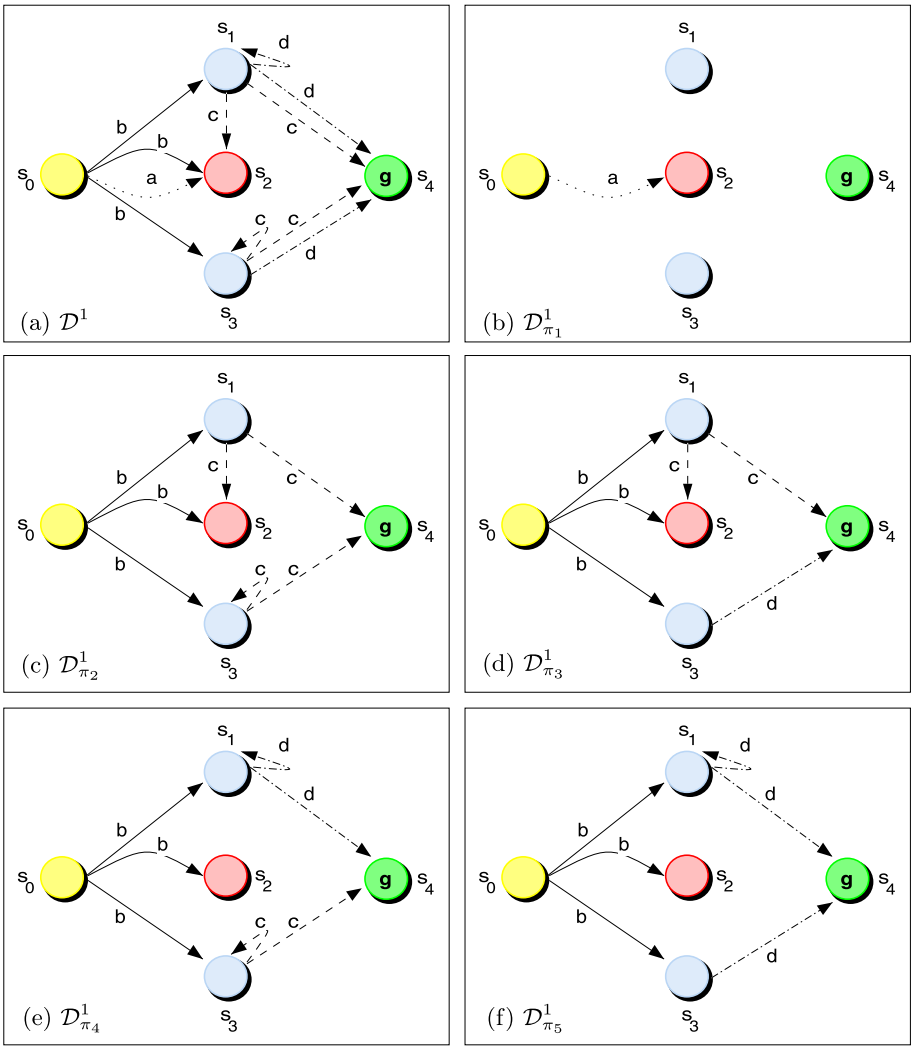

Fig. 6. Execution structures for policies in domain $\mathcal{D}^1$.

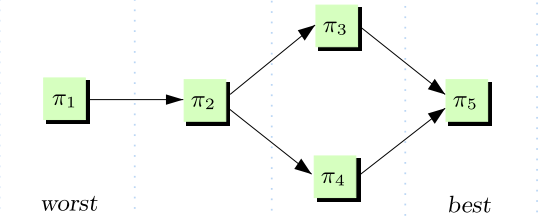

Fig. 7. The preference relation for policies in $\mathcal{D}^1$.

- s_1: from this state, the best choice is action d, that always maintains the possibility of reaching the goal (i.e., from state s_1, the agent must prefer a strong-cyclic solution);
- s_2: from this state, the agent can no longer reach the goal;
- s_3: from this state, the best choice is action d, which will necessarily reach the goal (i.e., from state s_3, the agent must prefer a strong solution).

In short, if the agent is trying its best, it should consider policy $\pi_5 = \{(s_0, b), (s_1, d), (s_3, d)\}$ as the only possible solution to the goal "*try your best to achieve g*".

As we can see, when a planner agent is trying its best to achieve a goal, it should alter its intention all along the planning task. The question that arises is "*how we can implement a planning algorithm so that the agent's intention can be altered during*

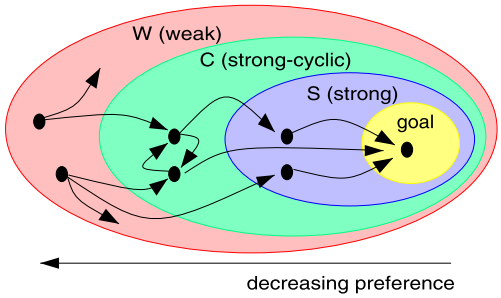

Fig. 8. Policy synthesis process.

the planning task?" By expressing the goal as $\exists \diamond g$, we do not allow that the agent alters its expectation: it will be always content with a weak solution, even when a strong or a strong-cyclic solution exists. On the other hand, by expressing the goal as $\forall \diamond g$ (or $\forall \boxdot \exists \diamond g$), if the planning problem has no strong (nor strong-cyclic) solution, the agent can desist of solving the problem (even when a weak solution exists). In the next subsection, we present a planning algorithm that is the answer to that question.

5.3. The algorithm α-BestPlanner

The algorithm α-BestPlanner for reachability goals is defined as follows:

α-BestPlanner$(\mathcal{D}, s_0, \varphi)$
1 $S \leftarrow$ Model$(\texttt{lfp}, \forall \diamond \varphi, \mathcal{D})$ // *strong*
2 `if` $s_0 \in$ StatesCoveredBy(S) `return` Policy(S)
3 $C \leftarrow$ Join$(S,$ Model$(\texttt{gfp}, \forall \boxdot \exists \diamond \varphi, \mathcal{D}))$ // *strong-cyclic*
4 `if` $s_0 \in$ StatesCoveredBy(C) `return` Policy(C)
5 $W \leftarrow$ Join$(C,$ Model$(\texttt{lfp}, \exists \diamond \varphi, \mathcal{D}))$ // *weak*
6 `if` $s_0 \in$ StatesCoveredBy(W) `return` Policy(W)
7 `return failure`

The correctness of this algorithm follows from the theorems presented in Section 4.2. An example of the synthesis process implemented by this algorithm is illustrated in Fig. 8.

First, the function Model (the same used in the α-Planner) is used to synthesize a strong model S from the formula $\forall \diamond \varphi$, where φ is a formula describing goal states. Then, if the initial state s_0 is covered by S, α-BestPlanner returns a strong solution. Otherwise, the function Model is used to synthesize a strong-cyclic model from the formula $\forall \boxdot \exists \diamond \varphi$ and its output is joined with the strong model S, resulting in a "mixed" model C. Then, if s_0 is covered by C, α-BestPlanner returns the "best" strong-cyclic solution. Otherwise, the function Model is used to synthesize a weak model from the formula $\exists \diamond \varphi$ and its output is joined with the mixed model C, resulting in a new mixed model W. Then, if s_0 is covered by W, α-BestPlanner returns the "best" weak solution.

The α-BestPlanner returns the "best" solution in the following sense: *for each state covered by the model, the associated actions are the best choice for the agent*. This property is guaranteed by construction, thanks to the use of the function Join. When a model M_1 is joined with a model M_2, every pair $(s, a) \in M_1$ is maintained and, on the other hand, every pair $(s, a) \in M_2$ such that s is already covered by M_1 is discarded. In other words, the function Join gives preference to the actions chosen during the construction of M_1.

Join(M_1, M_2)
1 `return` $M_1 \cup \{(s, a) \in M_2 \colon s \notin$ StatesCoveredBy$(M_1)\}$

Therefore, if there not exists a strong solution for the planning problem, the call of Join in line 3 of the algorithm α-BestPlanner gives preference to actions in the strong component S (constructed from the goal states) and extends it with new pairs of states and actions that, even leading to cycles, always maintain the possibility of reaching a goal state (Fig. 8). Analogously, if there not exists a strong-cyclic solution for the planning problem, the call of Join in line 5 of the algorithm α-BestPlanner preserves the actions in the strong-cyclic component C and extends it with new pairs of states and actions that, even leading to dead-ends, maintain the possibility of reaching a goal state. Finally, if there not exists a weak solution for the planning problem, the algorithm α-BestPlanner stops with failure.

Fig. 9 shows the steps performed by the algorithm α-BestPlanner to find a solution for the problem discussed in Example 1, whose planning domain $\mathcal{D}$ is represented in Fig. 1. The planning problem is to find the best policy to achieve $\varphi = g$ (i.e., to reach state s_4), starting from s_0. The strong model S synthesized in line 1 of the algorithm, as the resulting fixpoint computation of $\forall \diamond g$, is shown in Fig. 9(a). Since s_0 is not covered by S, line 3 computes the strong-cyclic model C,

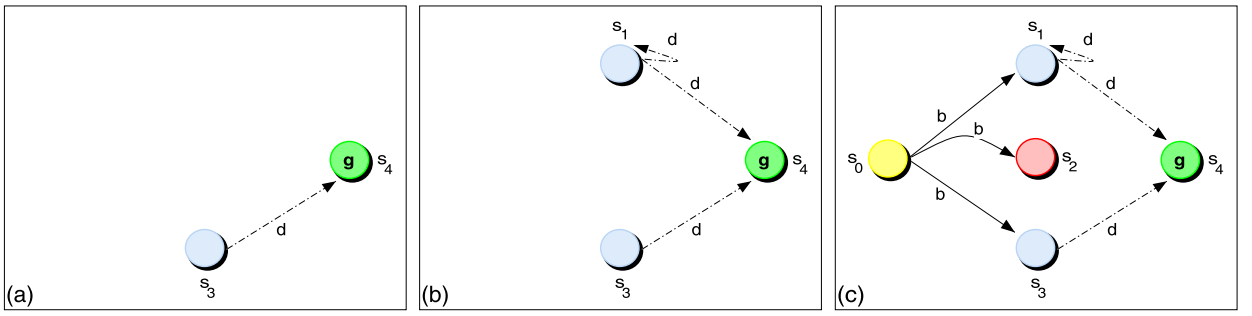

Fig. 9. How α-BestPlanner finds a solution for the problem in Example 1.

as the resulting fixpoint computation of $\forall \boxdot \exists \diamondsuit g$, shown in Fig. 9(b). The Join function preserves the strong model S as a submodel of model C. Again, the initial state s_0 is not yet covered by C. Thus, line 5 computes the weak model W, as the resulting fixpoint computation of $\exists \diamondsuit g$ and joins it with model C (Fig. 9(c)). Finally, since s_0 is covered by W, the algorithm returns a policy from W.

Theorem 6. *The algorithm α-BestPlanner has **time complexity** of $O(|\gamma| \times |\mathcal{S}|)$, where $|\gamma|$ is the size of the extended goal formula ($\forall \diamondsuit \varphi$, for strong; $\forall \boxdot \exists \diamondsuit \varphi$, for strong-cyclic; or $\exists \diamondsuit \varphi$, for weak) and $|\mathcal{S}|$ is size of the planning domain.*

Proof. The complexity of the algorithm α-BestPlanner depends on the complexity of the function Model, which computes models for formulas γ involving propositional operators, local temporal operators and global temporal operators. Only global temporal operators require fixpoint computations, performed by functions $\text{Model}_{\boxdot}$ and $\text{Model}_{\sqcup}$. Each fixpoint computation costs $O(|\mathcal{S}|)$ steps (each step involving only propositional and local temporal operators) [1]. Since the number of fixpoint computations needed to synthesize a model for a formula γ is equal to the number of global temporal operators that appear in it, denoted by $|\gamma|$, we conclude that the time complexity of the algorithm α-BestPlanner is $O(|\gamma| \times |\mathcal{S}|)$.

Remark 1. The algorithm α-BestPlanner can be easily modified to deal with extended reachability goals of the form (ϕ, φ), where the formula φ is a property to be achieved in the final state and the formula ϕ is a property to be maintained through the path to this final state. That can be done by using the formulas $\forall(\phi \sqcup \varphi)$, $\forall \boxdot \exists(\phi \sqcup \varphi)$ and $\exists(\phi \sqcup \varphi)$, respectively, instead of the formulas $\forall \diamondsuit \varphi$, $\forall \boxdot \exists \diamondsuit \varphi$ and $\exists \diamondsuit \varphi$.

Remark 2. The algorithm α-BestPlanner was implemented in SWI-Prolog, directly following the α-CTL's semantic specification (Definition 3). This implementation was used to verify if it is capable of finding solutions of appropriated quality (strong, strong-cyclic or weak), according to their availability in each situation. We have verified that whenever the planning problem solved by the algorithm had a strong solution, the algorithm really found such solution. The same was observed with respect to problems with strong-cyclic and weak solutions. Furthermore, we have also observed that the weak and strong-cyclic solutions found by the α-BestPlanner have a better quality compared which those found by the α-Planner, with predefined goal formulas $\forall \diamondsuit \varphi$, $\forall \boxdot \exists \diamondsuit \varphi$ and $\exists \diamondsuit \varphi$. This is due to the fact that the weak and strong-cyclic solutions found by the α-BestPlanner have strong substructures, whenever they are available in the planning domain. The same cannot be guaranteed for the policies returned by the α-Planner.

6. Conclusion

By using the branching time temporal logic α-CTL [18] we can synthesize plans as a collateral effect of the verification of a temporal formula γ (specifying a *planning goal* with built-in desired solution quality) in a temporal model $\mathcal{D}$ (specifying a nondeterministic *planning domain*). Thus a planning algorithm for extended reachability goals expressed in α-CTL can be directly formalized by the α-CTL's semantics.

In this paper we have shown how to specify and solve some extended goals using α-CTL, which takes into account nondeterministic actions. In particular, we have considered reachability goals of the form "*try your best to achieve φ*", where φ is a property describing goal states, and also extended reachability goals of the form "*try your best to achieve φ, maintaining the property ϕ*", where ϕ is a property to be maintained all along the path to the goal state.

Extended reachability goals have been addressed before in π-CTL* [3,2] and P-CTL* [4], other two logics which extend the semantics of CTL to specify extended planning goals. However, it is important to notice that our work cannot be directly compared with those works. As mentioned by the authors in [4], the logic π-CTL* proposed in [3,2] cannot be used to express the same goals we can deal with. Moreover, in [4] the authors only use the logic P-CTL* to specify intuitive goals which cannot be automatically generated by a planning algorithm, i.e., they do not propose an algorithm for the goals expressed in their logic. In short, the logic π-CTL* lacks the expressivity needed to deal with the kind of goals addressed in this work and, hitherto, no planning algorithm based on the P-CTL*'s semantics was presented.

Furthermore, we believe that our approach based on α-CTL is simpler and demands less computational effort than that proposed in [4], since the quantification over policies required in P-CTL* is a very costly operation, compared with the quantification over immediate successor states required in α-CTL.

References

[1] A. Arnold, P. Crubillé, A linear algorithm to solve fixed-point equations on transition systems, Information Processing Letters 29 (1988) 57–66.
[2] C. Baral, T. Eiter, J. Zhao, Using sat and logic programming to design polynomial-time algorithms for planning in non-deterministic domains, in: AAAI 2005, AAAI, Menlo Park, CA, USA, 2005, pp. 578–583.
[3] C. Baral, J. Zhao, Goal specification in presence of non-deterministic action, in: R.L. Mántaras, L. Saitta (Eds.), 16th European Conference on Artificial Intelligence, 2004, pp. 273–277.
[4] C. Baral, J. Zhao, Goal specification, non-determinism and quantifying over policies, in: 21st National Conference on Artificial Intelligence, AAAI Press, Boston, MA, USA, 2006, pp. 1–7.
[5] T. Bylander, The computational complexity of propositional STRIPS planning, Journal of Artificial Intelligence 69 (1–2) (1994) 165–204.
[6] A. Cimatti, F. Giunchiglia, E. Giunchiglia, P. Traverso, Planning via model checking: A decision procedure for $\mathcal{AR}$, in: 4th European Conference on Planning (ECP-97), Springer-Verlag, London, UK, 1997, pp. 130–142.
[7] A. Cimatti, M. Roveri, P. Traverso, Strong planning in non-deterministic domains via model checking, in: R.G. Simmons, M.M. Veloso, S. Smith (Eds.), International Conference on Artificial Intelligence Planning Systems (AIPS-98), AAAI, Pennsylvania, USA, 1998, pp. 36–43.
[8] E.M. Clarke, E.A. Emerson, Design and synthesis of synchronization skeletons using branching-time temporal logic, in: Logic of Programs, Workshop, Springer-Verlag, London, UK, 1982, pp. 52–71.
[9] U. Dal Lago, M. Pistore, P. Traverso, Planning with a language for extended goals, in: 8th National Conference on Artificial intelligence (NCAI-2002), AAAI, Menlo Park, CA, USA, 2002, pp. 447–454.
[10] M. Daniele, P. Traverso, M.Y. Vardi, Strong cyclic planning revisited, in: 5th European Conference on Planning (ECP-99), Springer-Verlag, London, UK, 2000, pp. 35–48.
[11] R. De Nicola, F. Vaandrager, 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, Springer-Verlag New York, Inc., New York, NY, USA, 1990, pp. 407–419.
[12] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
[13] F. Giunchiglia, P. Traverso, Planning as model checking, in: European Conference on Planning, 1999, pp. 1–20.
[14] S. Kripke, Semantical considerations on modal logic, Acta Philosophica Fennica 16 (1963) 83–94.
[15] M. Muller-Olm, D. Schimidt, B. Steffen, Model checking: A tutorial introduction, in: SAS'99, in: Lecture Notes in Computer Science, vol. 1694, Springer, 1999, pp. 330–354.
[16] C. Pecheur, F. Raimondi, Symbolic model checking of logics with actions, in: MoChArt 2006, Springer-Verlag, 2006, pp. 1215–1222.
[17] S.L. Pereira, Planning under uncertainty for extended reachability goals, PhD thesis, Institute of Mathematics and Statistics, University of São Paulo, São Paulo, Brazil, 2007.
[18] S.L. Pereira, L.N. Barros, A logic-based agent that plans for extended reachability goals, Autonomous Agents and Multi-Agent Systems 16 (3) (2008) 327–344.
[19] S.L. Pereira, L.N. Barros, Using α-CTL to specify complex planning goals, in: W. Hodges, R.J.G.B. Queiroz (Eds.), WoLLIC, in: Lecture Notes in Computer Science, vol. 5110, Springer, 2008, pp. 260–271.