Learning and Nonlinear Models (L&NLM) - Journal of the Brazilian Computational Intelligence Society, Vol. 12, Iss. 2, pp. 98-114, 2014
© Brazilian Computational Intelligence Society (SBIC)

# Symbolic Regression for Non-Deterministic Actions

**Maria Viviane de Menezes**
Federal University of Ceara
vivianemenezes@ufc.br

**Leliane Nunes de Barros**
University of Sao Paulo
leliane@ime.usp.br

**Silvio do Lago Pereira**
Faculty of Technology of Sao Paulo
slago@fatecsp.br

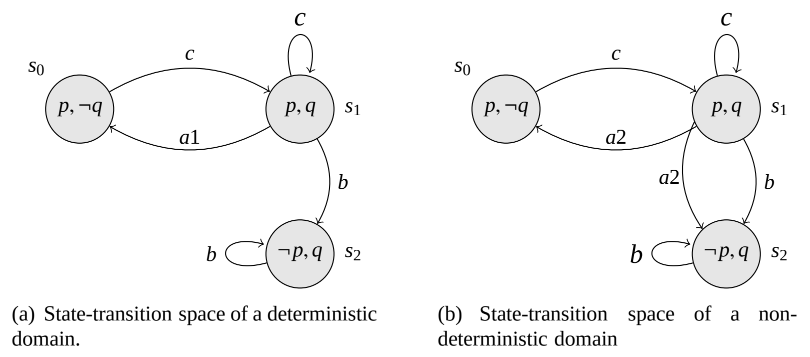
**Abstract –** Model checking (MC) is a widely used approach for verifying if a formal model of a system satisfies a particular temporal logic formula. Symbolic model checking has been largely applied to solve non-deterministic planning problems, an area called *planning as model checking*. In this approach, the planning domain is the system to be verified and the planning goal is the formula that must be satisfied. In general, the planning domain is given by a set of action specifications given in terms of preconditions and effects formulas and the MC pre-image computation performs some kind of translation of the actions specification into a symbolic representation of the whole state-transition space. However, the symbolic representation of the entire state transition space is a very expensive operation and, in some cases, even using the symbolic representation it is impossible to come up with a plan for large problems. In order to overcome this limitation, one can compute the pre-image of $X$ by using directly the action specification, without representing the whole state-transition space, an operation called *symbolic regression*. In this paper, we propose new symbolic non-deterministic regression operations based on *Quantified Boolean Formulas* (QBF) inference, as an extension of previous work on deterministic symbolic planning. In addition, we prove that the regression operations are equivalent to existing pre-image operations.

**Keywords –** Model Checking, Automated Planning, Non-Deterministic Actions, Temporal Logic.

## 1. INTRODUCTION

Automated planning [1] is the field of the artificial intelligence that studies the deliberative process involved in the planning task, aiming to the implementation of automated planning systems. In this context, a *planning domain* $\mathcal{M}$ is a specification of the environment dynamics and is modeled as a direct graph where: (*i*) the vertices represent the states of the environment and; (*ii*) the edges represent the transitions caused by actions (Figure 1), which are performed by an agent. A planning problem is defined by a specification of a planning domain $\mathcal{M}$, an *initial state* $s_0$ and a *goal* formula $\varphi$ that must be satisfied in a state reached by the agent.

*Classical planning* [2] assumes that the environment evolves deterministically, i.e., there is no uncertainty regarding the action effects. Figure 1(a) shows a planning domain where the actions are deterministic, i.e. when an action is applied to a state $s$ it leads to a unique successor state. For instance, in Figure 1(a), the execution of action $a1$ in the state $s_1$ leads to a successor state $s_0$; the execution of $b$ in $s_1$ (or $s_2$) leads to a state $s_2$ and; the execution of $c$ in $s_0$ (or $s_1$) leads to a state $s_1$. Deterministic assumptions can indeed be inappropriate in various practical situations. Figure 1(b) shows a non-deterministic action $a2$ that when executed in the state $s_1$ can lead to the state $s_0$ or to the state $s_2$. A planning domain with at least one action with non-deterministic effect is called a *non-deterministic* planning domain.



(a) State-transition space of a deterministic domain.
(b) State-transition space of a non-deterministic domain

Figure 1: (Explicit) State-transition space representation of two planning domains.

Basically, a planning algorithm performs a search in the explicit representation of the domain (e.g., the graphs on Figure 1). The search can be in a forward (progressive search) or backward way (regressive search). To find a solution, the progressive search starts from the initial state and generates successor states, until it reaches a goal state. In a reverse way, the regressive search starts from the set of states that satisfy a goal formula $\varphi$ and generates the predecessor states, until it reaches the initial state $s_0$. A solution for a deterministic planning problem is a *plan*, i.e., a path in the domain graph, starting from $s_0$ and finishing in a final state satisfying $\varphi$. A solution for a non-deterministic planning problem is a *policy*, i.e., a mapping from states to actions,

| | *Graph Transitions* | *Action language* |
|---|---|---|
| Set Theory | Explicit + Enumerative (model checking) | Implicit + Enumerative (LAO*) |
| Logic | Explicit + Symbolic (symbolic model checking) | Implicit + Symbolic |

Figure 2: Different ways to compute the regression step for non-deterministic planning laid out along two directions: explicit or implicit versus enumerative or symbolic representation.

that can induce at least one path starting from the initial state and finishing in a state satisfying the planning goal. A policy is a sub-graph of the domain graph, that contains the initial state $s_0$, in which at least one path starting from $s_0$ leads to a final state satisfying $\varphi$. A solution of a non-deterministic planning problem can be classified as *weak*, *strong* or *strong-cyclic* [3]. Intuitively, in a weak solution the agent can achieve a goal state; but due to the non-determinism, it does not guarantee to do so; in a strong solution the agent always achieves a goal state, in spite of non-determinism; and in a strong-cyclic solution the agent always achieves the goal, under the fairness assumption that execution will eventually exit from all existing cycles [4].

Efficient deterministic planning techniques, in general, do not work with explicit domain graphs as input but works with a formal language (e.g., STRIPS [2]) to represent the actions that can be used to induce the graph. This is called an *implicit representation* of the planning domain model, in opposition to the complete graph, i.e., an *explicit representation* of the domain model. With such language to specify the actions many efficient heuristic search planning algorithms for deterministic planning have been proposed [5–7].

A common approach used to solve non-deterministic planning is *model checking* [8–10]. In this case, given an initial state $s_0$ and a goal formula $\varphi$, specified in a temporal logic (e.g., *Computational Tree Logic* - CTL [11]), a model checker is used to verify if a state $s_0$ of the domain model satisfies $\varphi$ and, in the affirmative case, returns a corresponding policy. CTL allows the specification of complex temporal goal formulas, for instance: $\exists \diamond \varphi$ (exists a path where the goal $\varphi$ is finally reached) to represent a weak solution; $\forall \diamond \varphi$ (for all paths the goal $\varphi$ is finally reached) to represent a strong solution and; $\forall \diamond \exists \varphi$ (for all paths the goal $\varphi$ is finally reached) to represent a strong cyclic solution. The main computation step in the planning as model checking is the computation of predecessor states of a set of states: an operation called *pre-image of a set of states*. In this paper we focus on the *regression step of non-deterministic planning*; however, for didactic reasons we also describe some progressive computations.

An important aspect of planning as model checking when reasoning over the explicit graph is its performance: it can not solve planning problems with large state space due to the *enumerative* representation of the state transitions, i.e., based on set theory. To overcome this limitation planning as *symbolic model checking* represents the set of states and the transition relations of the explicit domain graph as propositional logic formulas and verify the satisfaction of the goal formula $\varphi$ by manipulating formulas, using Binary Decision Diagrams [12] (BDDs) to allow efficient computation. Another way to improve the efficiency of the regression step in non-deterministic planning is to use symbolic representation and computation (also using BDDs) together with the implicit representation of the domain, i.e., using STRIPS [2] action description.

Thus, given the symbolic representation of a set of states $X$, this paper focus on how to compute the predecessor states of $X$ using a symbolic representation of actions (i.e., an implicit and symbolic representation of non-deterministic planning domains), instead of the transition relation (the explicit representation). For this, we define two new operations for non-deterministic planning: *symbolic weak regression* and *symbolic strong regression* and show how it can be implemented in an efficient way using BDDs.

Figure 2 shows four different ways to represent and compute the regression step for non-deterministic planning problems. The definitions on the left refer to the *explicit representation* of the state transitions (graph transitions), whereas the ones on the right refers to the *implicit representation* of the state transitions (action language). The definitions on top refers to the enumerative representation of the state transitions (set theory representation), whereas the ones on the bottom refers to the symbolic representation of the state transitions (logic formulas). This paper contribution is defining the regression operation for the bottom right category: implicit and symbolic representation.

In order to illustrate the importance of the implicit representation for planning problems, consider the benchmark domain Logistics, used in the International Planning Competition (IPC [13]). The task in this domain is to transport several packages from their initial location to their desired destinations, using trucks for transporting packages in the same city and planes to transport packages among different cities. In this domain, a package is transported from one location to another by loading it into a truck (or plane), driving the truck (or flying a plane) to the destination, and unloading the truck (or plane). A truck (or plane) can load any number of packages. The easiest problem in this domain has 2 cities, 2 trucks, 1 plane and 4 packages to be delivered. The implicit representation for this simple problem takes 80 propositions and 212 actions. If we have to work with the explicit representation, we have to construct from the actions specification the entire transition relation for a state-space with $2^{80}$ states. This is a difficult task, even using symbolic representation. MIPS [14] is a planner based on model checking that computes the transition relation from the action specification. However, the transition relation is not computed from the *original* description of the domain: there is a precompiling phase, which infers a concise state representation by exhibiting knowledge that is implicit in the description of the planning domain [14]. This analysis uses structures called causal graphs [15] and drastically reduce the length of the state space. As mentioned by Edelkamp and Helmert [16], building the BDD for the transition relation is critical in both time and space.

This paper is organized as follows. In Section 2, we show how to reason about deterministic and non-deterministic actions in a progressive and regressive way. In Section 3, we present the planning as model checking framework. In Section 4, we show how to represent symbolically states and transitions of the planning domain and how to perform symbolic pre-image computation in a model that does not consider the actions behind the state transitions (e.g., the graphs of Figure 6). In Section 5, we describe the symbolic pre-image computation in a model that consider the actions behind the state transitions (e.g., the graphs of Figure 1). In Section 6, we show how to represent deterministic and non-deterministic actions as propositional formulas; for didactic reasons, we first present a previous work on regression for deterministic actions and; we show the *symbolic regression for non-deterministic actions*. Finally, in Section 7, we draw some conclusions.

## 2. Planning Foundations

STRIPS [2] is a first-order language largely used to represent deterministic actions. In a propositional version of STRIPS language, the domain is defined by a set of atomic propositions $\mathbb{P}$, representing the properties of the world, and a set of actions $\mathbb{A}$ representing the agent abilities to change the world state. The states are subsets of $\mathbb{P}$ (assuming the closed world assumption). The initial state $s_0$ of a planning problem is given by a complete set of properties that defines a unique possible state in the world. The planning goal is given by an incomplete set of properties (i.e., a subset of $\mathbb{P}$) that defines a set of states that satisfies them. Each STRIPS action is a partial function from states to states (Definition 2.1). Notice that this representation is a set theory based representation, i.e., an enumerative representation.

**Definition 2.1.** *(STRIPS Deterministic actions) A deterministic action $\alpha$ over a set of propositions $\mathbb{P}$ is specified by $\alpha = \langle precond(\alpha); effects(\alpha)\rangle$ where $precond(\alpha)$ is a set of preconditions, representing what has to be satisfied in the current state before executing $\alpha$, and $effects(\alpha)$ represent how the state $s$ is modified with the action execution. The effects are given by the couple $effects(\alpha) = \langle add(a), del(a)\rangle$ where: $add(a)$ is a set of propositions that become true after executing $\alpha$ and $del(a)$ is a set of propositions that become false.*

In this section, we show how to compute successor and predecessor states from the STRIPS action specification for deterministic domains and for non-deterministic domains defined with a simple extension of STRIPS for non-deterministic actions. Figure 3(a) shows the STRIPS representation for the deterministic actions $a_1$, $b$ and $c$, which correspond to the explicit representation shown in Figure 1(a). Figure 3(b) shows the extension of the STRIPS notation that includes the non-deterministic action $a_2$ and corresponds to the domain depicted in Figure 1(b).

$a_1 : \langle precond(a_1) = \{p, q\};\ \langle add(a_1) = \{p\};\ del(a_1) = \{q\}\rangle\rangle$
$b :\ \langle precond(b) = \{q\};\ \langle add(b) = \{\};\ del(b) = \{p\}\rangle\rangle$
$c :\ \langle precond(c) = \{p\};\ \langle add(c) = \{q\};\ del(c) = \{\}\rangle\rangle$

(a) The actions specification for the deterministic domain depicted in Figure 1(a)

$a_2 : \langle precond(a_2) = \{p, q\};\ \{\ \langle\ add(a_2, e_1) = \{p\};\ del(a_2, e_1) = \{q\}\rangle\ ,$
$\qquad\qquad\qquad\qquad\qquad\qquad \langle add(a_2, e_2) = \{q\};\ del(a_2, e_2) = \{p\}\rangle\ \}\rangle$
$b : \langle precond(b) = \{q\};\ \langle add(b) = \{\};\ del(b) = \{p\}\rangle\rangle$
$c : \langle precond(c) = \{p\};\ \langle add(c) = \{q\};\ del(c) = \{\}\rangle\rangle$

(b) The actions specification for the non-deterministic domain depicted in Figure 1(b)

Figure 3: (Implicit) Representation of planning domains by actions specifications.

### 2.1 Reasoning about Deterministic Actions

The *progression* of an initial state $x$ by a deterministic action $a$ produces a unique successor state $progr^a(x)$ (Equation 1). For example, in the explicit representation of a planning domain of Figure 1(a), $progr^c(s_0) = s_1$. In the implicit representation, to compute $progr^a(x)$ we have first to verify if the action $a$ is *applicable* in a state $x$, i.e., if $precond(a) \subseteq x$. In this case, the successor state reached by the execution of $a$ in $x$ is obtained by adding the positive effects in $x$ and, after that, eliminating the negative effects (i.e., $(x \cup add(a)) \setminus del(a)$). If $precond(a) \nsubseteq x$, then there isn't a successor state. Formally, $progr^a(x)$ is given by:

$$progr^a(x) = \begin{cases} (x \cup del(a)) \setminus add(a) & \text{if} \qquad precond(a) \subseteq x \\ \emptyset & \text{otherwise.} \end{cases} \tag{1}$$

In Figure 1(a), the successor state of $s_0$ by the action $c$ is obtained using Equation 1 as follows: $progr^c(s_0) = progr^c(\{p\}) = (\{p\} \cup \{\}) \setminus \{q\} = \{p, q\} = s_1$. Notice that the properties not affected by the action execution maintain their values in the successor state (frame axiom [17]).

The *regression* of a goal state $x$ by a deterministic action $a$ produces a set of predecessor states $regr^a(x)$. For instance, in Figure 1(a), $regr^c(s_1) = \{s_1, s_0\}$. When reasoning on the implicit representation, i.e., on the actions, the regression generates an abstract state, that can represent various states of the domain, once the propositions not included in the abstract state are considered unknown, i.e., they can be true or false [18]. Notice the difference between the state representation used in the regression and the one used in progression: in the progression we can make the close world assumption (i.e., the propositions not included in the state are considered false) while in the regression this assumption is not allowed. Thus, regression reasons over *partial truth-assignments* (i.e., belief-state space) and progression reasons over *complete truth-assignments* [19]. For an action to be *relevant* to a current state $x$ it must contribute to the properties satisfied in $x$, i.e., at least one of the action effects must unify with an element of $x$ [20]. Furthermore, the action must not have any effect that negates a property of $x$. Formally, given a state $x$, an action can lead to $x$ if $add(a) \subseteq x$ and $del(a) \cap x = \emptyset$, in this case we say that $a$ is a *relevant* action to state $x$. Thus, the set of predecessor states, of $x$ by the action $a$ is obtained by adding the preconditions and, after that, removing the positive effects in $x$, i.e., $(x \cup precond(a)) \setminus add(a)$, as defined by Equation 2:

$$regr^a(x) = \begin{cases} (x \setminus add(a)) \cup precond(a) & \text{if} \quad add(a) \subseteq x \text{ and } del(a) \cap x = \emptyset, \\ \emptyset & \text{otherwise.} \end{cases} \tag{2}$$

For example, in Figure 1(a), the predecessor state of $s_1$ by action $c$ is obtained using Equation 2 as follows: $regr^c(s_1) = regr^c(\{p, q\}) = (\{p, q\} \setminus \{q\}) \cup \{p\} = \{p\} = \{\{p, \neg q\}, \{p, q\}\} = \{s_0, s_1\}$.

The deterministic progression and regression operations can also be extended for set of states. Let $X$ be a set of states and $a$ be an action, the Equation 3 defines the set $progr^a(X)$ of states that are successors of states in $X$, by the application of $a$. A successor is generated for each state $x \in X$ where $a$ is *applicable*, i.e.,

$$progr^a(X) = \bigcup_{x \in X} \{progr^a(x)\}. \tag{3}$$

Equation 4 defines the set $regr^a(X)$ of states that are predecessors of states in $X$, according to the action $a$. A predecessor is generated by each state $x \in X$ where the action $a$ can lead to $x$, that is:

$$regr^a(X) = \bigcup_{x \in X} \{regr^a(x)\}. \tag{4}$$

For most planning domains, regression keeps the search branching factor smaller than progression. However, the fact that regression uses set of states, rather than individual states, makes it harder to come up with good heuristics. That is the main reason why the majority of efficient planners use progression [20]. Nevertheless, most of the approaches to solve non-deterministic planning based on model checking performs regressive search.

## 2.2 Reasoning about Non-deterministic Actions

We can use an extension of STRIPS in order to describe non-deterministic actions, where we can express alternative effects.

**Definition 2.2.** *(STRIPS-like non-deterministic actions) Based on STRIPS deterministic actions notation, a non-deterministic action $\alpha$ is specified by $\alpha = \langle precond(\alpha); effects(\alpha)\rangle$ where: (i) $precond(\alpha)$ is a set of preconditions and; (ii) $effects(\alpha)$ is a set of non-deterministic effects such that $effects(a) = \{e_1, e_2, \cdots, e_k\}$ and each $e_i \in effects(a)$ is given by the pair $\langle add(a, e_i), del(a, e_i)\rangle$.*

Figure 3(b) shows an example of a non-deterministic action, according to Definition 2.2. In that figure, the non-deterministic action $a2$ has two alternative effects, namely, $e_1$ and $e_2$. The progression of an initial state $x$ by a non-deterministic action $a$ produces a set of possible successor states $progr^a(x)$ (Equation 5). For example, in the explicit representation of a planning domain of Figure 1(b), $progr^{a2}(s_1) = \{s_0, s_2\}$. In the implicit representation, an action $a$ is applied in a state $x$ if $precond(a) \subseteq x$. The set of successors states is generated by the union of the successor states computed for each possible effect of $a$. If $precond(a) \nsubseteq x$, then there is not a successor state.

$$progr^a(x) = \begin{cases} \bigcup_{e \in effects(a)} \{(x \setminus del(a, e)) \cup add(a, e)\} & \text{if} \quad precond(a) \subseteq x, \\ \emptyset & \text{otherwise.} \end{cases} \tag{5}$$

In Figure 1(b), the successor state of $s_1$ by the non-deterministic action $a2$ is obtained using Equation 5 as follows: $progr^{a2}(s_1) = \{(\{p, q\} \setminus \{q\}) \cup \{p\}\} \cup \{(\{p, q\} \setminus \{p\}) \cup \{q\}\} = \{p\} \cup \{q\} = \{s_0\} \cup \{s_2\} = \{s_0, s_2\}$.

Equation 6 extends the non-deterministic progression operation for a set of states $X = \{x_1, x_2, \cdots, x_n\}$, by computing the set of states that are successors of each state $x \in X$ by the non-deterministic action $a$.

$$progr^a(X) = \bigcup_{x \in X} progr^a(x). \tag{6}$$

As in the deterministic case, the regression of a state $x$ according to a non-deterministic action produces a set of predecessor states $regr^a(x)$. For instance, in Figure 1(b), $regr^{a2}(s_0) = \{s_1\}$. When reasoning on the implicit representation, we can use the Equation 7 to compute $regr^a(x)$ when $a$ is non-deterministic. Basically, if some effect is relevant to the state $x$, then for each $e \in$ *effects*$(a)$ the predecessor states are obtained as in the regression for deterministic actions (Equation 2) and, after that, the subsets obtained are conjoint.

$$regr^a(x) = \begin{cases} (x \setminus add(a,e)) \cup precond(a) & \text{if} \quad \text{if } \exists e \in \textit{effects(a)} \text{ such that } add(a,e) \subseteq x \text{ and} \\ del(a,e) \cap x = \emptyset, & \\ \emptyset & \quad\quad \text{otherwise.} \end{cases} \tag{7}$$

In Figure 1(b), the predecessor state of $s_0$ by the non-deterministic action $a2$ is obtained using Equation 7 as follows: $regr^{a2}(s_0) = \{(\{p\} \setminus \{p\}) \cup \{p, q\}\} = \{p, q\}$, once $add(a2, e_1) \subseteq x$ and $del(a2, e_1) \cap x = \emptyset$ for the non-deterministic effect $e_1$ of the action $a2$.

To compute the non-deterministic regression for a set of states, let $X = \{x_1, \cdots, x_n\}$ be a set of states and $a$ a non-deterministic action relevant to at least one of the states in $X$. When $a$ is executed in a state $x$, it can lead *necessarily* to a state in $X$ or *possibly* to a state in $X$ (and *possibly* to a state not in $X$). Given a non-deterministic action $a$, the set of predecessors states that lead *necessarily* to states in $X$ is computed by the *strong regression* of $X$ according to the action $a$. The set of states that lead *possibly* to states in $X$ is computed by the *weak regression* of $X$ according to $a$.

**Definition 2.3.** *(Weak regression of a set of states) Let $S$ be the set of states of the planning domain, $X$ be a subset of $S$ and, $\alpha$ be a non-deterministic action. The weak regression of $X$ according to $\alpha$ (denoted by weakRegr$^\alpha(X)$) is the set of states from which a successor state in $X$ is possibly reached after the execution of $\alpha$ and is given by:*

$$weakRegr^\alpha(X) = \{y \in S \ : \ progr^\alpha(y) \cap X \neq \emptyset\}. \tag{8}$$

**Definition 2.4.** *(Strong regression of a set of states) Let $S$ be the set of states of the planning domain, $X$ be a subset of $S$, and $\alpha$ be a non-deterministic action. The strong regression of $X$ according to $\alpha$ (denoted by strongRegr$^\alpha(X)$) is the set of states from which a successor state in $X$ is necessarily reached after the execution of $\alpha$ and is given by:*

$$strongRegr^\alpha(X) = \{y \in S \ : \ \emptyset \neq progr^\alpha(y) \subseteq X\}. \tag{9}$$

The weak regression of a set of states $X$ according to a set of actions $\mathbb{A}$ can be obtained in terms of the weak regression according to each action $a \in \mathbb{A}$, i.e.:

$$weakRegr(X) = \bigcup_{a \in \mathbb{A}} weakRegr^a(X). \tag{10}$$

The strong regression of a set of states $X$ according to a set of actions $\mathbb{A}$ can also be obtained in terms of the strong regression according to each action $a \in \mathbb{A}$, that is:

$$strongRegr(X) = \bigcup_{a \in \mathbb{A}} strongRegr^a(X). \tag{11}$$

In Section 6, we show how the regression operations can be computed symbolically, i.e., by using an extension of propositional logic with quantified formulas (*Quantified Boolean Formulas* [21]). Henceforth, we will only discuss regressive approaches for planning.

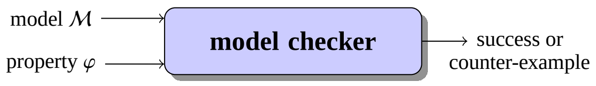


Figure 4: The model checking verifies if a system model $\mathcal{M}$ satisfies a formula $\varphi$.

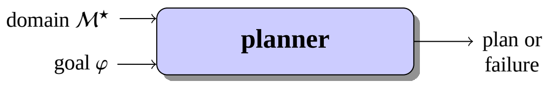


Figure 5: The model checking framework applied to solve planning problems.

## 3. Planning as Model Checking

Model checking consists of solving the problem $(\mathcal{M}, s) \models \varphi$, where $\mathcal{M}$ is a formal model of a system, $s$ is a state of the world and $\varphi$ is a formal specification of a property to be verified in this system. Essentially, a model checker (Figure 3) is an algorithm that receives $(\mathcal{M}, s, \varphi)$ and systematically visits the states of the model $\mathcal{M}$ in order to verify if the property $\varphi$ holds from the state $s$. If $(\mathcal{M}, s) \models \varphi$, then the model checker returns *success*; otherwise, it returns a counter-example, e.g., a state where $\varphi$ is violated.

When applying model checking framework to solve planning problems (Figure 3), the model $\mathcal{M}$ describes the planning domain, $s$ is the initial state $s_0$ of the problem and the property $\varphi$ specifies the planning goal. Thus, if $(\mathcal{M}, s_0) \models \varphi$, the planner based on model checking returns a plan; otherwise, it returns failure.

The reasoning that happens using model checking is done based on temporal logics over an *explicit representation* of the state transition space. In Section 3.1 we describe how to compute the predecessor states of a set of states $X$, using CTL, which represents the state transition space as a Kripke structure (graphs in Figure 1 without action labels). In Section 3.2 we describe how to perform this computation using the logic $\alpha$-CTL, which reason over an *action labeled transition system* (as the graphs in Figure 1). Notice that these representations are based on set theory, i.e., an enumerative representation.

### 3.1 Planning as Model Checking based on CTL

CTL (*Computation Tree Logic*) is a branching time temporal logic that allows for reasoning about alternative time lines (i.e., alternative futures) and it has been used to specify non-deterministic planning problems and related algorithms based on model checking [3, 22–24]. CTL formulas are composed by atomic propositions, propositional operators and temporal operators. The symbols $\bigcirc$ (*next*), $\diamond$ (*finally*), $\square$ (*globally*) 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:

$$\varphi \doteq p \in \mathbb{P} \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists \bigcirc \varphi_1 \mid \forall \bigcirc \varphi_1 \mid \exists\square\varphi_1 \mid \forall\square\varphi_1 \mid \exists(\varphi_1 \sqcup \varphi_2) \mid \forall(\varphi_1 \sqcup \varphi_2)$$

The semantics of CTL is defined over a Kripke structure $\mathcal{M} = \langle S, L, T \rangle$ , where: (*i*) $S$ is a set of states; (*ii*) $L : S \mapsto 2^P$ is a state labeling function and; (*iii*) $T \subseteq S \times S$ is a transition relation. A path in $\mathcal{M}$ is a sequence of states $s_0, s_1, \cdots$ such that $s_0 \in S$ and $(s_i, s_{i+1}) \in T$, for all $i \geq 0$. Figure 6(a) shows a Kripke structure corresponding to the graph in Figure 1(b). Notice that in a Kripke structure there is no label distinguishing the actions responsible for the transitions.

Model checking algorithms based on CTL [25] have a fundamental operation which is the computation of the pre-image of a set of states. Given a set of states $X$, the pre-image computes the predecessor states of $X$. Pre-image operations can be classified as: *strong pre-image* and *weak pre-image*. The weak pre-image computes the set of states from which a state in $X$ is *possibly* reached in one step and the strong pre-image computes the set of states from which a state in $X$ is *necessarily* reached in one step.

**Definition 3.1.** *(CTL pre-image of a set of states) Let $M = \langle S, L, T \rangle$ be a Kripke structure over a set of propositions $\mathbb{P}$ and $X \subseteq S$ be a set of states. The function weakPre$_{ctl}$($X$) returns the set of states from which some transitions lead to states in $X$. The function strongPre$_{ctl}$($X$) returns the set of states from which all transitions lead to states in $X$. Formally:*

- *weakPre$_{ctl}$($X$) $= \{s \in S \mid T(s) \cap X \neq \emptyset\}$;*
- *strongPre$_{ctl}$($X$) $= \{s \in S \mid \emptyset \neq T(s) \subseteq X\}$.*

**Example 3.1.** *(CTL pre-image computation) Let $M = \langle S, L, T \rangle$ be the Kripke structure depicted in Figure 6(a). Let $X = \{s_2\}$ be a subset of $S$. The weak pre-image of $X$ is:*

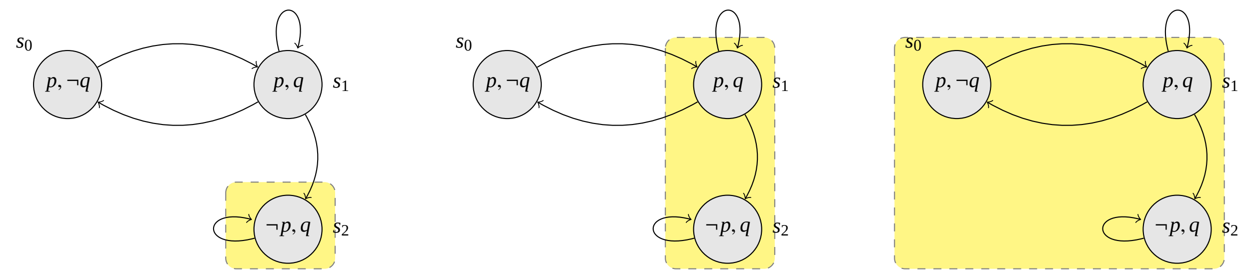
$$weakPre_{ctl}(X) = \{s_1, s_2\}$$

*, because from these states it is possible to reach $s_2$ in one step. Furthermore, the strong pre-image of $X$ is:*

$$strongPre_{ctl}(X) = \{s_2\}$$

*, because all transitions starting in $s_2$ leads only to states into $X$. Considering the state $s_1$, for instance, although there is a transition leading to the state $s_2 \in X$, there is another transition leading to $s_0 \notin X$. Thus, $s_1 \notin$ strongPre$_{ctl}$($X$).*

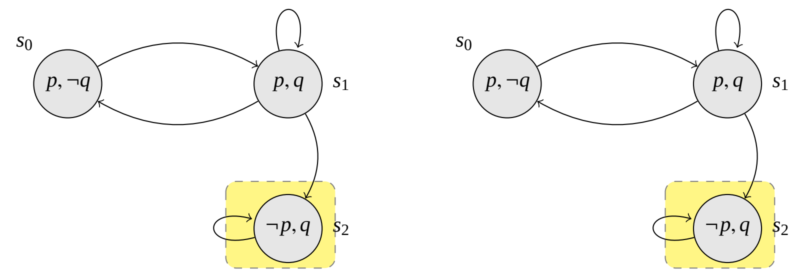
Figure 6 shows how the CTL model checking algorithm computes the set of states satisfying the CTL formula $\exists \diamond (\neg p \wedge q)$. The Figure 6(a) depicts the Kripke structure with three states: $s_0$, $s_1$ and $s_2$. In order to compute the set of states from which *there is a path where the formula* $(\neg p \wedge q)$ *is finally satisfied*, the model checking algorithm starts by computing the set $X$ of states satisfying the formula $(\neg p \wedge q)$. In this case, $X = \{s_2\}$. After that, in each iteration, the algorithm computes the *weak pre-image* of the set of states obtained in the previous iteration. This iterative process stops when a fix-point is achieved, i.e., when no new state is reached by the pre-image computation. In the first iteration of the algorithm, the weak pre-image of $X$ is computed, obtaining $weakPre_{ctl}(X) = \{s_1, s_2\}$ (Figure 6(b)). In the second iteration, the weak pre-image of the set of states obtained in the previous iteration is computed, this is: $weakPre_{ctl}(weakPre_{ctl}(X)) = \{s_0, s_1, s_2\}$. In the fourth iteration, the algorithm computes the weak pre-image of the set of states obtained in the previous iteration and verifies that no new state was reached (i.e., $weakPre_{ctl}(weakPre_{ctl}(X)) = weakPre(weakPre_{ctl}(weakPre_{ctl}(X)))$), meaning that a fix-point was reached. Then, the algorithm returns the set of states $\{s_0, s_1, s_2\}$ where each state satisfies the formula $\exists \diamond (\neg p \wedge q)$.



(a) Kripke structure where the set of states $X = \{s_2\}$ satisfies $\neg p \wedge q$ (b) 1st iteration: $weakPre_{ctl}(X) = \{s_1, s_2\}$ (c) 2nd iteration: $weakPre_{ctl}(weakPre_{ctl}(X)) = \{s_0, s_1, s_2\}$

Figure 6: CTL model checking algorithm computing the set of states that satisfies the formula $\exists \diamond (\neg p \wedge q)$.

Figure 7 shows how the CTL model checking algorithm computes the set of states satisfying the CTL formula $\forall \diamond (\neg p \wedge q)$, i.e., the states from which *all paths finally reach a state satisfying* $(\neg p \wedge q)$. Figure 7(a) highlights the set $X = \{s_2\}$ of states satisfying the formula $(\neg p \wedge q)$. In the first iteration of the algorithm (Figure 7(b)), the strong pre-image of $X$ is computed, obtaining $strongPre_{ctl}(X) = \{s_2\}$. As no new state was computed by the pre-image operation, a fix-point was reached. Then, the algorithm returns the set of states $\{s_2\}$ that satisfies the formula $\forall \diamond (\neg p \wedge q)$.



(a) Kripke structure where the set of states $X = \{s_2\}$ satisfies $\neg p \wedge q$ (b) 1st iteration: $strongPre_{ctl}(X) = \{s_2\}$

Figure 7: CTL model checking algorithm computing the set of states that satisfies the formula $\forall \diamond (\neg p \wedge q)$.

Model checking algorithms have been largely used to solve non-deterministic planning problems [3, 23]; however, these algorithms have one limitation: the Kripke structure does not consider the actions responsible for the state transitions. In order to overcome this limitation, an extension of CTL, called $\alpha$-CTL [4], was proposed. The $\alpha$-CTL semantics is based on *action labeled transition systems*, i.e., Kripke structures where the transitions are labeled with actions (as the graphs showed in Figure 1).

### 3.2 Planning as Model Checking based on $\alpha$-CTL

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

**Definition 3.2.** *($\alpha$-CTL Syntax) Let $p \in \mathbb{P}$ be an atomic proposition, the syntax of $\alpha$-CTL is defined inductively as:*

$$\varphi ::= p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists \odot \varphi_1 \mid \forall \odot \varphi_1 \mid \exists \boxdot \varphi_1 \mid \forall \boxdot \varphi_1 \mid \exists \diamonddot \varphi_1 \mid \forall \diamonddot \varphi_1 \mid \exists(\varphi_1 \sqcup\!\!\!\cdot\, \varphi_2) \mid \forall(\varphi_1 \sqcup\!\!\!\cdot\, \varphi_2)$$

Let $\mathbb{P} \neq \emptyset$ be a set of atomic propositions and $\mathbb{A}$ a finite set of actions. The semantics of $\alpha$-CTL is defined over an action labeled transition system (ALT system, for short) $\mathcal{M} = \langle S, L, \mathcal{T} \rangle$ with signature $(\mathbb{P}, \mathbb{A})$, where: (*i*) $S$ is a set of states; (*ii*)

$L : S \mapsto 2^P$ is a state labeling function and; (*iii*) $\mathcal{T} \subseteq S \times \mathbb{A} \times S$ is a transition relation where each transition is labeled by an action. Figure 1(a) and (d) depicts ALT systems. Notice that, in an ALT system, the states are labeled by elements of $\mathbb{P}$ and the transitions are labeled by elements of $\mathbb{A}$. Intuitively, a state $s$ in an ALT system satisfies a formula $\forall \odot \varphi$ (or $\exists \odot \varphi$) if there exists an action $\alpha \in \mathbb{A}$ that, when executed in $s$, necessarily (or possibly) reaches an immediate successor of $s$ which satisfies the formula $\varphi$. In other words, the modality $\odot$ represents the set of $\alpha$-successors of $s$, for some particular action $\alpha \in \mathbb{A}$; the quantifier $\forall$ requires that all these $\alpha$-successors satisfy $\varphi$; and quantifier $\exists$ requires that some of these $\alpha$-successors satisfy $\varphi$.

Although actions are essential in the semantics of $\alpha$-CTL, note that they are not used to compose $\alpha$-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, the existing actions logics [26, 27], which allow formulas with constraints over actions, are also inadequate to formalize planning algorithms.

A model checker for $\alpha$-CTL can be directly implemented from its semantics [4]. This algorithm is similar to the CTL model checking algorithm, having three fundamental differences: (i) the system to be verified is an ALT system; (ii) the formula is specified in $\alpha$-CTL and; (iii) the pre-image computation is modified in order to consider the actions labelling the transitions, as showed in the Definitions 3.3.

**Definition 3.3.** *($\alpha$-CTL pre-image of a set of states) Let $M = \langle S, L, \mathcal{T} \rangle$ be an action labeled state transition system with signature $(\mathbb{P}, \mathbb{A})$ and $X \subseteq S$ be a set of states. The function $weakPre_{\alpha ctl}(X)$ computes the maximal set $Y \subseteq S$ such that, for each state $s \in Y$, there is an action $a \in \mathbb{A}$, whose execution in $s$ leads to a state in $X$. Analogously, the function $strongPre_{\alpha ctl}(X)$ computes the maximal set $Y \subseteq S$ such that, for each state $s \in Y$, there is an action $a \in \mathbb{A}$, whose execution in $s$ leads only to states in $X$.*

$$weakPre_{\alpha ctl}(X) = \{s \in S \mid \exists a \in \mathbb{A} \text{ and } \mathcal{T}(s, a) \cap X \neq \emptyset\}; \tag{12}$$

$$strongPre_{\alpha ctl}(X) = \{s \in S \mid \exists a \in \mathbb{A} \text{ and } \emptyset \neq \mathcal{T}(s, a) \subseteq X\}. \tag{13}$$

**Example 3.2.** *($\alpha$-CTL pre-image computation) Let $M = \langle S, L, \mathcal{T} \rangle$ be the ALT system of Figure 1(b) and $X = \{s_2\}$ be a subset of $S$. The weak pre-image of $X$ is:*

$$weakPre_{\alpha ctl}(X) = \{s_1, s_2\}$$

*, because from $s_1$ it is possible to reach $s_2$ (by following the deterministic action b or the non-deterministic action a2) and from $s_2$ it is possible to reach $s_2$ (by following the action b). Furthermore, the strong pre-image of $X$ is:*

$$strongPre_{\alpha ctl}(X) = \{s_1, s_2\}$$

*, because all transitions caused by the execution of the action b in $s_1$ (and $s_2$) lead to $s_2$. Notice that, since we are working with action-labeled transitions, we can distinguish the transitions starting from $s_1$ and consider only those transitions which are labeled by the action b. This is not possible in a Kripke structure because the transitions starting from a specific state cannot be distinguished.*

In $\alpha$-CTL, we can also define the weak and strong pre-images of a set of states, according to each action $a \in \mathbb{A}$, as show in Equations 14 and 15, respectively.

$$weakPre^{a}_{\alpha ctl}(X) = \{s \in S \mid \mathcal{T}(s, a) \cap X \neq \emptyset\}; \tag{14}$$

$$strongPre^{a}_{\alpha ctl}(X) = \{s \in S \mid \emptyset \neq \mathcal{T}(s, a) \subseteq X\}. \tag{15}$$

Taking the union of the states computed by strong and weak pre-images, according to each action $a \in \mathbb{A}$, we obtain $weakPre_{\alpha ctl}(X)$ and $strongPre_{\alpha ctl}(X)$, respectively:

$$weakPre_{\alpha ctl}(X) = \bigcup_{a \in \mathbb{A}} weakPre^{a}_{\alpha ctl}(X); \tag{16}$$

$$strongPre_{\alpha ctl}(X) = \bigcup_{a \in \mathbb{A}} strongPre^{a}_{\alpha ctl}(X). \tag{17}$$

**Example 3.3.** *($\alpha$-CTL pre-image computation according to each action) Let $M = \langle S, L, T \rangle$ be the ALT system with signature $(\mathbb{P}, \mathbb{A})$ depicted in the Figure 1(b) and $X = \{s_2\}$ be a subset of $S$. The weak pre-image of $X$ according to each action $a \in \mathbb{A}$ is:*

$$weakPre^{a2}_{\alpha ctl}(X) = \{s_1\}; \quad weakPre^{b}_{\alpha ctl}(X) = \{s_1, s_2\}; \quad weakPre^{c}_{\alpha ctl}(X) = \emptyset.$$

*The strong pre-image of $X$ according to each action $a \in \mathbb{A}$ is:*

$$strongPre^{a2}_{\alpha ctl}(X) = \emptyset; \quad strongPre^{b}_{\alpha ctl}(X) = \{s_1, s_2\}; \quad strongPre^{c}_{\alpha ctl}(X) = \emptyset.$$

Following, we show theorems confirming that the set of states obtained by the regression is the same obtained by the $\alpha$-CTL pre-image computation.

**Theorem 3.1.** *If $a \in \mathbb{A}$ is an action and $X \subseteq S$ is a set of states, then $weakRegr^a(X) = weakPre^a_{\alpha ctl}(X)$.*

*Proof.* Part (I): Let $s \in weakRegr^a(X)$ be a state and suppose that $s \notin weakPre^a_{\alpha ctl}(X)$. Thus, according to Equation 8, it follows that $progr^a(s) \cap X \neq \emptyset$; and, according to Equation 14, it follows that $\mathcal{T}(s,a) \cap X = \emptyset$. However, this is a contradiction (if $\mathcal{T}(s,a) \cap X = \emptyset$, then the state $s$ has no successor in the set $X$ that can be reached by a transition labeled by the action $a$ and, consequently, the progression of $s$ through the action $a$ cannot lead to a state inside $X$). Thus, it follows that $s \in weakRegr^a(X) \rightarrow s \in weakPre^a_{\alpha ctl}(X)$, i.e., $weakRegr^a(X) \subseteq weakPre^a_{\alpha ctl}(X)$. Part (II): Analogously, we can show that $s \in weakPre^a_{\alpha ctl}(X) \rightarrow s \in weakRegr^a(X)$, i.e., $weakPre^a_{\alpha ctl}(X) \subseteq weakRegr^a(X)$. Therefore, we can conclude that $weakRegr^a(X) = weakPre^a_{\alpha ctl}(X)$. □

**Theorem 3.2.** *$weakPre_{\alpha ctl}(X) = weakRegr(X)$.*

*Proof.*

$$\begin{aligned} weakPre_{\alpha ctl}(X) &= \bigcup_{a\in\mathbb{A}} weakPre^a(X) && \text{(by Equation 16)} \\ &= \bigcup_{a\in\mathbb{A}} weakRegr^a(X) && \text{(by Theorem 3.1)} \\ &= weakRegr(X) && \text{(by Equation 10)} \end{aligned}$$

□

**Theorem 3.3.** *If $a \in \mathbb{A}$ is an action and $X \subseteq S$ is a set of states, then $strongRegr^a(X) = strongPre^a_{\alpha ctl}(X)$.*

*Proof.* Part (I): Let $s \in strongRegr^a(X)$ be a state and suppose that $s \notin strongPre^a_{\alpha ctl}(X)$. Thus, according to Equation 9, it follows that $\emptyset \neq progr^a(s) \subseteq X$; and, according to Equation 15, it follows that $\emptyset \neq \mathcal{T}(s,a) \nsubseteq X$. However, this is a contradiction (if $\emptyset \neq \mathcal{T}(s,a) \nsubseteq X$, then executing the action $a$ in the state $s$ a state $s' \notin X$ can be achieved, consequently, the progression of $s$ through the action $a$ cannot lead to only states inside $X$). Thus, it follows that $s \in strongRegr^a(X) \rightarrow s \in strongPre^a_{\alpha ctl}(X)$, i.e., $strongRegr^a(X) \subseteq strongPre^a_{\alpha ctl}(X)$. Part (II): Analogously, we can show that $s \in strongPre^a_{\alpha ctl}(X) \rightarrow s \in strongRegr^a(X)$, i.e., $strongPre^a_{\alpha ctl}(X) \subseteq strongRegr^a(X)$. Therefore, we can conclude that $strongRegr^a(X) = strongPre^a_{\alpha ctl}(X)$.
□

**Theorem 3.4.** *$strongPre_{\alpha ctl}(X) = strongRegr(X)$.*

*Proof.*

$$\begin{aligned} strongPre_{\alpha ctl}(X) &= \bigcup_{a\in\mathbb{A}} strongPre^a_{\alpha ctl}(X) && \text{(by Equation 17)} \\ &= \bigcup_{a\in\mathbb{A}} strongRegr^a(X) && \text{(by Theorem 3.3)} \\ &= strongRegr(X) && \text{(by Equation 11)} \end{aligned}$$

□

The image and pre-image operations showed in this section are inefficient, once it is impossible to represent all transitions for large state-spaces. As we will see in the next section, the use of symbolic operations allows to apply model checking in practice.

## 4. Planning as Symbolic Model Checking

In the previous section, we describes planning as model checking using an *explicit representation* of the planning domain model (i.e., the Kripke structure or the ALT system), which can be very inefficient. We can use a symbolic representation of the explicit representation of $M$, i.e., a symbolic representation of $T$ (or $\mathcal{T}$) and apply efficient BDD operations allowing the verification of large state-space systems [3, 14, 28]. The symbolic model checking requires the following steps: (i) codifying states and transitions as propositional formulas; (ii) representing these formulas as BDDs and; (iii) performing efficient BDDs operations. In this section, we show a symbolic representation of the Kripke structure and the ALT system and how to symbolically compute the CTL and $\alpha$-CTL pre-image (weak and strong).

## 4.1 Symbolic Representation for States and Transitions

In this section we show how to represent states as formulas and also the two transition functions $T$ (CTL model) and $\mathcal{T}$ ($\alpha$-CTL model) represented as logical formulas.

**Definition 4.1.** *(Propositional representation for states and set of states) Let $M = \langle S, L, T\rangle$ be a state transition system over a set of propositions $\mathbb{P}$. The propositional codification of a state $s \in S$, denoted by $\xi(s)$, is the formula*

$$\xi(s) = \bigwedge_{p \in L(s)} p, \tag{18}$$

*and the propositional codification of a set of states $X \subseteq S$ is the formula*

$$\xi(X) = \bigvee_{s \in X} \xi(s). \tag{19}$$

**Example 4.1.** *(Symbolic representation of a set of states) The set of states $S = \{s_0, s_1, s_2\}$ of the state transition system depicted in Figure 6(a) can be represented by the formula:*

$$\xi(S) = (p \wedge \neg q) \vee (p \wedge q) \vee (\neg p \wedge q).$$

In a Kripke structure (depicted in Figure 6), a transition is a subset of $S \times S$. In order to represent the states before and after a transition, we generate an extra copy of the propositions $\mathbb{P}$ and prime all variables. For instance, in Figure 8, $\xi(s_0) = p \wedge \neg q$ is a formula representing the current state in the transition and $\xi(s_1') = p' \wedge q'$ is a formula representing the next state in the transition.

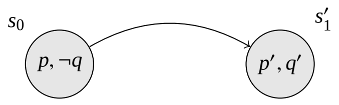


Figure 8: Representation of the states before and after a transition in a Kripke structure.

**Definition 4.2.** *(Propositional representation for transition and transition relation in a Kripke structure) Let $M = \langle S, L, T\rangle$ be a Kripke structure over a set of propositions $\mathbb{P}$. The propositional codification for a transition $(s, s') \in T$, denoted by $\xi(t)$ is a formula*

$$\xi(t) = \xi(s) \wedge \xi(s') \tag{20}$$

*and the propositional codification for the transition relation $T$ is a formula $\xi(T)$*

$$\xi(T) = \bigvee_{t \in T} \xi(t). \tag{21}$$

**Example 4.2.** *(Symbolic representation for the transition relation in a Kripke structure) The transition relation $T = \{(s_0, s_1), (s_1, s_1), (s_1, s_0), (s_1, s_2), (s_2, s_2)\}$ of the Kripke structure depicted in Figure 6(a) can be represented by the formula:*

$$\xi(T) = (p \wedge \neg q \wedge p' \wedge q') \vee (p \wedge q \wedge p' \wedge q') \vee (p \wedge q \wedge p' \wedge \neg q') \vee (p \wedge q \wedge \neg p' \wedge q') \vee (\neg p \wedge q \wedge p' \wedge \neg q').$$

However, when applying model checking to solve planning problems, the Kripke structure must be enhanced by the actions in the label of the transitions. Thus, in an ALT system, a transition is a subset of $S \times \mathbb{A} \times S$. In order to consider actions, planning as symbolic model checking approaches [3] use another set $\mathcal{A}$ of propositional variables, called *action variables*. Thus, each action $a \in \mathbb{A}$ has a corresponding action variable $\alpha \in \mathcal{A}$.

**Definition 4.3.** *(Propositional representation for transition and transition relation in an ALT system) Let $M = \langle S, L, T\rangle$ be an ALT system with signature $(\mathbb{P}, \mathbb{A})$. The propositional codification for a transition $(s, a, s') \in T$, denoted by $\xi(t)$ is a formula*

$$\xi(t) = \xi(s) \wedge \alpha \wedge \xi(s') \tag{22}$$

*and the propositional codification for the transition relation $T$ is a formula $\xi(T)$*

$$\xi(T) = \bigvee_{t \in T} \xi(t). \tag{23}$$

**Example 4.3.** *(Symbolic representation for the transition relation in an ALT system) The transition relation $T = \{(s_0, c, s_1), (s_1, c, s_1), (s_1, a_2, s_0), (s_1, a_2, s_2), (s_1, b, s_2), (s_2, b, s_2)\}$ of the ALT system depicted in Figure 1(b) is:*

$$\xi(T) = (p \wedge \neg q \wedge c \wedge p' \wedge q') \;\vee\; (p \wedge q \wedge c \wedge p' \wedge q') \;\vee\; (p \wedge q \wedge a_2 \wedge p' \wedge \neg q') \;\vee$$
$$(p \wedge q \wedge a_2 \wedge \neg p' \wedge q') \;\vee\; (p \wedge q \wedge b \wedge \neg p' \wedge q') \;\vee\; (\neg p \wedge q \wedge b \wedge p' \wedge \neg q')).$$

**Quantified Boolean Formulas** In order to perform an efficient computation of regression of pre-image and regression, it is necessary to use an extension of proposition logic which allows the quantification over the propositions values [21] (*Quantified Boolean Formulas*). For instance, in pre-image computation, the most important symbolic operations between sets of states and transitions represented as propositional formulas are based on existential and universal abstraction [29]. Given a propositional formula $\varphi$ and a proposition $p$ occurring in $\varphi$, the existential abstraction is defined as:

$$\exists p.\, \varphi \equiv \varphi[\top/p] \vee \varphi[\bot/p] \tag{24}$$

where $\varphi[\top/p]$ is obtained by replacing the proposition $p$ by the value $\top$ in the formula $\varphi$ and $\varphi[\bot\ /p]$ is obtained by replacing $p$ by $\bot$ in the formula $\varphi$. The universal abstraction is a formula:

$$\forall p.\, \varphi \equiv \varphi[\top/p] \wedge \varphi[\bot/p] \tag{25}$$

The quantifications also can be defined for a set of variables. Let $B = \{b_1, b_2, \cdots, b_n\}$ be a subset of propositional variables occurring in $\varphi$. Thus, $\exists B.[\varphi] \equiv \exists b_1.(\cdots \exists b_n.\varphi)$ e $\forall B.[\varphi] \equiv \forall b_1.(\cdots \forall b_n.\varphi)$.

**Binary Decision Diagrams** Binary Decision Diagrams (BDDs) are a canonical representation for boolean functions. A BDD is similar to a decision tree: an acyclic graph where non-terminal nodes are labelled with boolean variables and terminal nodes are labelled with 0 or 1. In order to allow a compact representation, the following optimizations are performed: (i) removal of duplicate terminals and non-terminals nodes and; (ii) removal of redundant tests. Furthermore, it is imposed an ordering on the variables occurring along any path in order to improve efficiency. Applying these removals and fixing the ordering on the variables, we obtain an *Reduced and Ordering BDD* (ROBDD). For simplicity, when we mention a BDD actually we are mentioning a ROBDD. BDDs allow compact representations for boolean functions which only have exponential representations in other systems, such as truth tables and conjunctive normal forms [25].

### 4.2 Symbolic CTL Pre-Image Computation

Let $M = \langle S, L, T\rangle$ be a Kripke structure over a set of propositions $\mathbb{P}$, $\xi(X)$ (Definition 4.1) be the propositional representation of a set of states $X \subseteq S$ and $\xi(T)$ be the propositional representation of the transition relation, symbolic model checking tools [30,31] computes the pre-image of $X$ using QBF as showed in the Definitions 4.4 and 4.5.

**Definition 4.4.** *(CTL symbolic weak pre-image) Let $M = \langle S, L, T\rangle$ be a Kripke structure over a set of propositions $\mathbb{P}$, the CTL symbolic weak pre-image of a set of states $X \subseteq S$ is given by [25]:*

$$symbWeakPre_{ctl}(X) = \exists \mathbb{P}'.(\xi(T) \wedge \xi(X')) \tag{26}$$

**Example 4.4.** *(CTL symbolic weak pre-image) Let $M = \langle S, L, T\rangle$ be the Kripke structure depicted in Figure 6(a); $X = \{s_2\} \subseteq S$ be a set of states, whose representation using primed variables is given by the formula $\xi(X') = \neg p' \wedge q'$ and; $\xi(T)$ be the propositional representation of the transition relation (Example 4.2). The CTL symbolic weak pre-image of X is given by:*

$$\begin{aligned}
symbWeakPre_{ctl}(X) &= \exists \mathbb{P}'.(\xi(T) \wedge \xi(X)') \quad (by\ Equation\ 26)\\
&= \exists\{p', q'\}.(((p \wedge \neg q \wedge p' \wedge q') \ \vee \ (p \wedge q \wedge p' \wedge q') \ \vee \ (p \wedge q \wedge p' \wedge \neg q')\\
&\quad \vee (p \wedge q \wedge \neg p' \wedge q') \ \vee \ (\neg p \wedge q \wedge p' \wedge \neg q')) \wedge (\neg p' \wedge q'))\\
&= \exists p'.\exists q'.(q \wedge \neg p' \wedge q')\\
&= \exists p'.((q \wedge \neg p' \wedge \bot) \vee (q \wedge \neg p' \wedge \top))\\
&= \exists p'.(q \wedge \neg p')\\
&= q \quad (representing\ the\ set\ of\ states\ \{s_1, s_2\})
\end{aligned}$$

**Definition 4.5.** *(CTL symbolic strong pre-image) Let $M = \langle S, L, T\rangle$ be a Kripke structure over a set of propositions $\mathbb{P}$, the symbolic strong pre-image of a set of states $X \subseteq S$ is computed by [3]:*

$$symbStrongPre_{ctl}(X) = \forall \mathbb{P}'.(\xi(T) \rightarrow \xi(X')) \wedge \exists \mathbb{P}'.\xi(T) \tag{27}$$

**Example 4.5.** *Let $M = \langle S, L, T\rangle$ be the Kripke structure depicted in Figure 6, $\xi(X')$ be the propositional representation of $X = \{s_2\} \subseteq S$ using primed variables and; $\xi(T)$ the propositional representation of the transition relation (Example 4.2). The*

*CTL symbolic strong pre-image of X is given by:*

$$\begin{aligned}
symbStrongPre_{ctl}(X) &= \forall\mathbb{P}'.(\xi(T) \rightarrow \xi(X')) \wedge \exists\mathbb{P}'.\xi(T) \quad (\textit{by Equation 27})\\
&= \forall\{p',q'\}.((p \wedge \neg q \wedge p' \wedge q') \vee (p \wedge q \wedge p' \wedge q') \vee (p \wedge q \wedge p' \wedge \neg q')\\
&\quad \vee (p \wedge q \wedge \neg p' \wedge q') \vee (\neg p \wedge q \wedge \neg p' \wedge q') \rightarrow (\neg p' \wedge q')) \wedge \exists\{p',q'\}.\xi(T)\\
&= \forall\{p',q'\}.((\neg p \vee \neg q \vee \neg p') \wedge (\neg p \vee \neg p' \vee \neg q')) \wedge ((p \wedge \neg q) \vee (p \wedge q) \vee (\neg p \wedge q))\\
&= \forall p'.\forall q'.((\neg p \vee \neg q \vee \neg p') \wedge (\neg p \vee \neg p' \vee \neg q')) \wedge ((p \wedge \neg q) \vee (p \wedge q) \vee (\neg p \wedge q))\\
&= (\neg p \vee \neg q) \wedge \neg p \wedge ((p \wedge \neg q) \vee (p \wedge q) \vee (\neg p \wedge q))\\
&= \neg p \wedge q \quad (\textit{representing the state } s_2)
\end{aligned}$$

In this section, we showed how image and pre-image operations for CTL are defined based on a symbolic representation of the state-space. The symbolic representation using Binary Decision Diagrams are the state-of-art in the model checking area. However, when apply model checking for automated planning, CTL is not adequate to represent the actions labelling the transition relation. For this reason, in the next section we show how the symbolic model checking operations can be extended to represent the actions.

## 5. Symbolic $\alpha$-CTL Pre-Image Computation

When symbolic model checking is applied to solve planning problems, the Kripke structure have to be enhanced by the actions that labels the transition relation. In the traditional model checking approaches, each transition is a pair (*state,successor state'*); however, in non-deterministic planning as model checking, each transition is a triple (*state,action, successor state'*). For instance, in the Figure 1(b), there are two transitions between the states $s_1$ and $s_2$: $(s_1, a_2, s_2)$ produced by action $a_2$ and $(s_1, b, s_2)$ caused by action $b$. In a Kripke structure, these two transitions are the same transition, once the actions labeling the transitions are not represented.

In this section we extend the work in [4] to define the first $\alpha$-CTL symbolic pre-image computation for models with transitions labeled with actions, i.e., the first $\alpha$-CTL symbolic model checking. Therefore, we represent the actions by using a set of propositional variables called *action variables* based on the work of *Cimatti, et al. (2003)* [3]. Intuitively, an action variable is true if and only if the corresponding action is being executed. Thus, a transition to a state $s$ to a successor state $s'$ produced by the action $a$ is represented as:

$$T_a = \xi(s) \wedge \alpha \wedge \xi(s'),$$

where $\alpha$ is a proposition representing an action $a \in \mathbb{A}$. Each action $a \in \mathbb{A}$ has a corresponding action variable $\alpha \in \mathcal{A}$. The transition relation $T$ of the graph corresponding to a planning domain is represented as a disjunction of all $T_a$ (as in the Definition 4.2). The Definition 5.1 shows how to compute weak pre-image when the transitions are labeled with actions and the Definition 5.2 shows how to compute strong pre-image. In both definitions, it is necessary to include the elimination of the actions variables ($\exists\mathcal{A}$).

**Definition 5.1.** *($\alpha$-CTL symbolic weak pre-image) Let $M = \langle S, L, \mathcal{T}\rangle$ be an ALT system with signature $(\mathbb{P}, \mathbb{A})$; $\xi(X')$ be the propositional representation using primed variables of $X \subseteq S$; $\xi(\mathcal{T})$ be the propositional representation of the transition relation $\mathcal{T}$ and; $\mathcal{A}$ be the set of propositional actions variables set. The symbolic weak pre-image of X is:*

$$symbWeakPre_{\alpha ctl}(X) = \exists\mathcal{A}.\exists\mathbb{P}'.(\xi(\mathcal{T}) \wedge \xi(X')). \tag{28}$$

**Example 5.1.** *($\alpha$-CTL symbolic weak pre-image) Let $M = \langle S, L, \mathcal{T}\rangle$ be the ALT system depicted in Figure 1(b); $\xi(X') = \neg p' \wedge q'$ be the primed representation of the set $X = \{s_2\} \subseteq S$ and; $\xi(\mathcal{T})$ be the symbolic representation of the transition relation (Example 4.3). The symbolic weak pre-image of X is given by:*

$$\begin{aligned}
symbWeakPre_{\alpha ctl}(X) &= \exists\mathcal{A}.\exists\mathbb{P}'.(\xi(T) \wedge \xi(X')) \quad (\textit{Equation } 28)\\
&= \exists\mathcal{A}.\exists\mathbb{P}'.(((p \wedge \neg q \wedge c \wedge p' \wedge q') \vee (p \wedge q \wedge c \wedge p' \wedge q') \vee (p \wedge q \wedge a_2 \wedge p' \wedge \neg q') \vee\\
&\quad (p \wedge q \wedge a_2 \wedge \neg p' \wedge q') \vee (p \wedge q \wedge b \wedge \neg p' \wedge q') \vee (\neg p \wedge q \wedge b \wedge \neg p' \wedge q')) \wedge (\neg p' \wedge q'))\\
&= \exists a2.\exists b.\exists c.\exists p'.\exists q'.((p \wedge q \wedge a2 \wedge \neg p' \wedge q') \vee (q \wedge b \wedge \neg p' \wedge q'))\\
&= (p \wedge q) \vee q\\
&= q \quad (\textit{representing the set of states}\{s_1, s_2\}).
\end{aligned}$$

**Definition 5.2.** *($\alpha$-CTL symbolic strong pre-image) Let $M = \langle S, L, \mathcal{T}\rangle$ be an ALT system with signature $(\mathbb{P}, \mathbb{A})$; $\xi(X')$ be the propositional representation using primed variables of $X \subseteq S$; $\xi(\mathcal{T})$ be the propositional representation of the transition relation $\mathcal{T}$ and; $\mathcal{A}$ be the the propositional actions variables set. The symbolic strong pre-image of X is:*

$$symbStrongPre_{\alpha ctl}(X) = \exists\mathcal{A}.(\forall\mathbb{P}'.(\xi(\mathcal{T}) \rightarrow \xi(X')) \wedge \exists\mathbb{P}'.\xi(\mathcal{T})). \tag{29}$$

**Example 5.2.** *Let $M = \langle S, L, \mathcal{T}\rangle$ be the ALT system depicted in Figure 1(b), $\xi(X')$ be the primed representation of the set of states $X = \{s_2\} \subseteq S$; and $\xi(\mathcal{T})$ be the symbolic representation of the transition relation (Example 4.3). The symbolic strong pre-image of X is given by:*

$$\begin{aligned}
symbStrongPre_{\alpha ctl}(X) &= \exists\mathcal{A}.(\forall\mathbb{P}'.(\xi(\mathcal{T}) \rightarrow \xi(X')) \wedge \exists\mathbb{P}'.\xi(\mathcal{T}))\\
&= \exists\mathcal{A}.(\forall\mathbb{P}'.(((p \wedge \neg q \wedge c \wedge p' \wedge q') \vee (p \wedge q \wedge c \wedge p' \wedge q') \vee (p \wedge q \wedge a2 \wedge p' \wedge \neg q') \vee (p \wedge q \wedge a2\\
&\quad \wedge \neg p' \wedge q') \vee (p \wedge q \wedge b \wedge \neg p' \wedge q') \ \vee\ (\neg p \wedge q \wedge b \wedge \neg p' \wedge q')) \rightarrow (\neg p' \wedge q')) \ \wedge\ \exists\mathbb{P}'.\xi(\mathcal{T}))\\
&= \exists\mathcal{A}.((\forall\mathbb{P}'.((\neg p \vee \neg q \vee \neg a2 \vee \neg p' \vee q') \wedge (\neg p \vee \neg c \vee \neg p' \vee \neg q')) \wedge ((p \wedge q \wedge a2) \vee (q \wedge b) \vee (p \wedge c)))\\
&= \exists\mathcal{A}.((\neg p \vee \neg c) \wedge (\neg p \vee \neg q \vee \neg a2) \wedge ((p \wedge q \wedge a2) \vee (q \wedge b) \vee (p \wedge c)))\\
&= \exists\mathcal{A}.((q \wedge \neg a2 \wedge b \wedge \neg c) \vee (\neg p \wedge q \wedge b))\\
&= q \vee (\neg p \wedge q)\\
&= q \quad \textit{(representing the set of states}\{s_1, s_2\}).
\end{aligned}$$

In this section we showed that symbolic image and pre-image operations for $\alpha$-CTL are more adequate for planning as model checking, since they consider the actions behind the state-transitions. In the next section, we show how to symbolically compute the set of predecessors and successors states directly from the action specification. In this case, given a planning problem described in terms of the actions, it is not necessary to translate the actions into the transition relation to apply model checking algorithms.

## 6. Symbolic Regression of Actions

Representing symbolically states and actions, we can compute the predecessor states of a set of states without representing explicitly the transitions of the planning domain (i.e., the tuple (*state, action, successor state*)) as it is done in the traditional symbolic model checking approaches. We call this operation as: *symbolic regression of actions*. Notice that the symbolic regression reasons over the implicit model of the planning domain (as those presented in Figure 3) while the symbolic pre-image reasons over the explicit model of the planning domain (as those depicted in Figure 1). In the Section 6.1 we present a previous work on symbolic regression for deterministic actions and in Section 6.2 we show the main contribution of this paper, which is the *symbolic regression for non-deterministic actions*.

### 6.1 Symbolic Regression for Deterministic Actions

First, we will show how to represent STRIPS deterministic actions as propositional formulas and, after that, how to perform symbolic regression computation.

**Definition 6.1.** *(Propositional representation for deterministic actions) The propositional representation for a STRIPS deterministic action $\alpha = \langle precond(\alpha); effects(\alpha)\rangle$ (Definition 6.1) is a pair of formulas $\alpha = \langle \xi(precond(\alpha)); \xi(effects(\alpha))\rangle$ such that:*

- *$\xi(precond(\alpha))$ is a literal or a conjunction of literals representing the preconditions of $\alpha$, i.e.,*

$$\xi(precond(\alpha)) = \bigwedge_{p\,\in\,precond(\alpha)} p \qquad \textit{and,} \tag{30}$$

- *$\xi(effects(\alpha))$ is a literal or a conjunction of literals representing the effects of $\alpha$, i.e.,*

$$\xi(effects(\alpha)) = \bigwedge_{d\,\in\,add(\alpha)} d \ \wedge \bigwedge_{r\,\in\,del(\alpha)} \neg r. \tag{31}$$

**Example 6.1.** *The deterministic actions $a1$, b and c described in Figure 3(a) can be represented as:*

$$a1 = \langle p \wedge q; p \wedge \neg q\rangle; b = \langle q; \neg p\rangle; c = \langle p; q\rangle.$$

Given a set of states $X$ represented in a symbolic way, the propositional planner PropPlan [32] computes the set of predecessor states of $X$ using directly the deterministic action specification. We call this operation as *symbolic regression for deterministic actions*. In order to perform the regression operation, it is defined for each action $a \in \mathbb{A}$ the set *changes*$(e)$ as the set of propositions occurring in the effect $e$ of the action $a$. For instance, $changes(e_{a1}) = \{p, q\}$, changes$(e_b) = \{p\}$ and changes$(e_c) = \{q\}$ are, respectively, the change set for the actions $a_1$, $b$ and $c$ (Figure 3(a)).

**Definition 6.2.** *(Symbolic regression of a set of states by a deterministic action) Let $\alpha$ be a deterministic action represented by a pair of proposition formulas such that $\alpha = \langle \xi(precond(\alpha)); \xi(effects(\alpha)) \rangle$ and let $\xi(X)$ be the propositional representation of a set of states $X$ (Equation 23), the symbolic regression of $X$ by the deterministic action $\alpha$ is [32]:*

$$symRegr^{\alpha}(\xi(X)) = \xi(precond(\alpha)) \wedge \exists changes(e).(\xi(e) \wedge \xi(X)). \tag{32}$$

**Example 6.2.** *Given the set of actions $\mathbb{A} = \{a1, b, c\}$ (Figure 3(a)), $symRegr^{a1}(X)$ , $symRegr^{b}(X)$ and $symRegr^{c}(X)$ for $\xi(X) = \neg p \wedge q$ where $X = \{s_2\}$ is computed as follows:*

$$\begin{aligned} symRegr^{a1}(\xi(X)) &= \xi(precond(a_1)) \wedge \exists changes(a_1).(\xi(e(a_1)) \wedge \xi(X)) \quad (by\ Equation\ 32)\\ &= (p \wedge q) \wedge \exists\{p,q\}.((p \wedge \neg q) \wedge (\neg p \wedge q))\\ &= (p \wedge q) \wedge \exists\{p,q\}.(\bot)\\ &= \bot; \end{aligned}$$

$$\begin{aligned} symRegr^{b}(\xi(X)) &= \xi(precond(b)) \wedge \exists changes(b).(\xi(e(b)) \wedge \xi(X)) \quad (by\ Equation\ 32)\\ &= q \wedge \exists p.(\neg p \wedge (\neg p \wedge q))\\ &= q \wedge \exists p.(\neg p \wedge q)\\ &= q; \end{aligned}$$

$$\begin{aligned} symRegr^{c}(\xi(X)) &= \xi(precond(c)) \wedge \exists changes(c).(\xi(e(c)) \wedge \xi(X)) \quad (by\ Equation\ 32)\\ &= p \wedge \exists q.(q \wedge (\neg p \wedge q))\\ &= p \wedge \exists q.(\neg p \wedge q)\\ &= p \wedge \neg p\\ &= \bot\ . \end{aligned}$$

Symbolic regression can be also computed for a set of actions $\mathbb{A}$ taking the disjunction of the symbolic regression according to each action $a \in \mathbb{A}$ as we can see in the Definition 6.3.

**Definition 6.3.** *(Symbolic regression of a set of states by a set of actions) Let $\mathbb{A}$ be a set of actions and let $\xi(X)$ be a propositional representation of a set of states $X$, the symbolic regression of $X$ is given by:*

$$symbRegr(\xi(X)) = \bigvee_{a \in \mathbb{A}} symbRegr^{a}(\xi(X)). \tag{33}$$

Notice that, symbolic regression reasons about actions (implicit representation of the planning domain) instead of the transition relation (explicit representation) and, furthermore, does not need an extra set of primed propositions neither an extra set of propositional variables as in the symbolic pre-image.

## 6.2 Symbolic Regression for Non-deterministic Actions

In this paper, we propose how to perform symbolic regression when the actions are non-deterministic. The operations *symbolic weak regression* and *symbolic strong regression* are able to compute the set of predecessor states using directly the non-deterministic actions specification instead of the transition relation. First, we show how to represent non-deterministic actions as propositional formulas (Definition 6.4) and, after that, how to compute symbolic weak regression (Definition 6.7) and symbolic strong regression (Definition 6.8).

**Definition 6.4.** *(Propositional representation for non-deterministic actions) The propositional representation for a non-deterministic action $\alpha = \langle precond(a); effects(\alpha) \rangle$ is a pair $\alpha = \langle \xi(precond(\alpha); \xi(effects(\alpha)) \rangle$ such that:*

- *$\xi(precond(\alpha))$ is a literal or a conjunction of literals representing the preconditions of $\alpha$, i.e.,*

$$\xi(precond(\alpha)) = \bigwedge_{p \,\in\, precond(\alpha)} p \quad and, \tag{34}$$

- *$\xi(effects(\alpha))$ is a disjunction of all non-deterministic effects of $\alpha$, i.e.,*

$$\bigvee_{e_i \,\in\, effects(\alpha)} ( \bigwedge_{d \,\in\, add(\alpha,e_i)} d \ \wedge \bigwedge_{r \,\in\, del(\alpha,e_i)} \neg r). \tag{35}$$

**Example 6.3.** *The non-deterministic action $a2$ in the Figure 3(b) can be represented as:*

$$a2 = \langle p \wedge q; (p \wedge \neg q) \vee (\neg p \wedge q)\rangle$$

In the symbolic non-deterministic regression, each effect $e_i$ of the action $\alpha$ has a corresponding change set given by $changes(\alpha, e_i)$. The change set of the action $\alpha$ is given by taking the union of the each $change(\alpha, e_i)$, i.e.:

$$changes(\alpha) = \bigcup_{e_i \in effects(\alpha)} changes(a, e_i). \tag{36}$$

For instance, the change set of the action $a2$ (Figure 3(b)) that has two non-deterministic effects $e_1$ and $e_2$ is given by: $changes(e_1) \cup\ changes(e_2) = \{p, q\}\ \cup\ \{p, q\} = \{p, q\}$.

The symbolic weak regression of a set of states $X$ by a non-deterministic action $\alpha$ (Definition 6.5) computes the set of states from which some effect of $\alpha$ reaches a state in $X$ while the symbolic strong regression of $X$ (Definition 6.6) computes the set of states from which all the non-deterministic effects of $a$ reach a state in $X$.

**Definition 6.5.** *(Weak symbolic regression of a set of states by a non-deterministic action) Let $\alpha$ be a non-deterministic action represented by a pair of proposition formulas $\langle \xi(precond(\alpha)); \xi(effects(\alpha))\rangle$ (Definition 6.1) and let $\xi(X)$ be the propositional representation of a set of states $X$ (Equation 23), the symbolic weak regression of $X$ is given by:*

$$symbWeakRegr^{\alpha}(\xi(X)) = \xi(precond(\alpha)) \wedge \exists changes(\alpha).(\xi(effects(\alpha)) \wedge \xi(X)). \tag{37}$$

We can analyse this equation from right to left. First, the conjunction $\xi(effects(\alpha)) \wedge \xi(X)$ selects the subset of states in $X$ reached by the effects of $\alpha$. If none effect of $\alpha$ is relevant to $X$, then $\xi(effects(\alpha)) \wedge \xi(X) = \bot$. However, if some effect of $a$ is *relevant* to $X$, then $\xi(effects(\alpha)) \wedge \xi(X) \neq \bot$ (notice that a similar analysis is done in the Equation 7 when it is verified if $\exists e \in effects(a)$ such that $e$ is relevant to the state $x$). After this conjunction, the existential quantification using the set changes($\alpha$) eliminates one by one the effects variables in the relevant states. A similar elimination also is done in the Equation 7, but in this case the positive effects of the action are eliminated by the difference set operation. Finally, the precondition is conjoint, obtaining the formula representing the set of predecessor states (as can also be verified in the Equation 7 by taking the union of the preconditions).

**Example 6.4.** *(Symbolic weak regression of a set of states) Given the set of states $X = \{s_2\}$ (Figure 1(b)) represented by the propositional formula $\xi(X) = \neg p \wedge q$ and the non-deterministic action $a2 = \langle precond(a2); effects(a2)\rangle = \langle (p \wedge q); (p \wedge q) \vee (\neg p \wedge q)\rangle$, then $symbWeakRegr^{a2}(X)$ is:*

$$\begin{aligned} sWeakRegr^{a2}(\xi(X)) &= \xi(precond(a2)) \wedge \exists changes(a2).(\xi(effects(a2)) \wedge \xi(X))) \quad (by\ Equation\ 37)\\ &= (p \wedge q) \wedge (\exists\{p, q\}.(((p \wedge \neg q) \vee (\neg p \wedge q)) \wedge (\neg p \wedge q))\\ &= (p \wedge q) \wedge (\exists p.\exists q.(\neg p \wedge q))\\ &= (p \wedge q) \wedge \top\\ &= (p \wedge q). \end{aligned}$$

**Definition 6.6.** *(Symbolic strong regression of a set of states by an action) Let $\alpha$ be a non-deterministic action represented by a pair of proposition formulas $\langle \xi(precond(\alpha)); \xi(effects(\alpha))\rangle$ (Definition 6.1) and let $X$ be a set of states represented by a propositional formula $\xi(X)$. The symbolic strong regression of $X$ is:*

$$symbStrongRegr^{a}(\xi(X)) = \xi(precond(a)) \wedge \forall changes(\alpha).(effects(a) \rightarrow \xi(X)).$$

**Example 6.5.** *(Symbolic strong regression of a set of states) Given the set of states $X = \{s_2\}$ (Figure 1(b)) represented by the propositional formula $\xi(X) = \neg p \wedge q$ and the non-deterministic action $a2 = \langle precond(a2); effects(a2)\rangle = \langle (p \wedge q); (p \wedge q) \vee (\neg p \wedge q)\rangle$, then symbolic strong regression of $X$ according to the action $a2$ is:*

$$\begin{aligned} strongRegr^{a2}(\xi(X)) &= \xi(precond(a2)) \wedge \forall changes(a2).(\xi(effects(a2)) \wedge \xi(X))) \quad (by\ Equation\ 37)\\ &= (p \wedge q) \wedge (\forall p.\forall q.(((p \wedge \neg q) \vee (\neg p \wedge q)) \rightarrow (\neg p \wedge q))\\ &= (p \wedge q) \wedge (\forall p.\forall q.(\neg p \vee q))\\ &= (p \wedge q) \wedge \forall p.(\neg p)\\ &= (p \wedge q) \wedge (\top \wedge \bot)\\ &= (p \wedge q) \wedge (\bot)\\ &= \bot\ . \end{aligned}$$

Symbolic weak (and strong) regression can be also computed for a set of actions $\mathbb{A}$ taking the disjunction of the symbolic weak (and strong) regression according to each action $a \in \mathbb{A}$ as we can see in the Definition 6.7 (and in Definition 6.8).

**Definition 6.7.** *(Symbolic weak regression of a set of states by a set of actions) Let $\mathbb{A}$ be a set of actions and let $\xi(X)$ be a propositional representation of a set of states $X$, the symbolic weak regression of $X$ is given by:*

$$symbWeakRegr(\xi(X)) = \bigvee_{a \in \mathbb{A}} symbWeakRegr^{a}(\xi(X)). \tag{38}$$

**Definition 6.8.** *(Symbolic strong regression of a set of states by a set of actions) Let $\mathbb{A}$ be a set of actions and let $X$ be a set of states, represented by a propositional formula, the symbolic strong regression of $X$ is given by:*

$$symbStrongRegr(\xi(X)) = \bigvee_{a \in \mathbb{A}} symbStrongRegr^{a}(\xi(X)). \tag{39}$$

The symbolic progression and regression operations presented in this section are able to compute, respectively, the set of predecessor and successor states from the implicit representation of the planning domain. We claim that these operations are more appropriate when applying model checking algorithms for planning, once it is not necessary translate the actions specification into the transition relation. However, both formalisms have the same complexity since they are based on QBF inference.

## Acknowledgement

We thank Sao Paulo Research Foundation (FAPESP), grant 2010/10845-0, for financial support.

## 7. Conclusion

Symbolic model checking is a largely applied technique to solve non-deterministic planning problems. In this approach, called planning as symbolic model checking, the fundamental operation is the computation of the pre-image of a set of states, which is performed by using the symbolic representation of the entire transition relation representing the planning domain graph. However, translating the set of actions into the entire transition relation is a very expensive operation and, in some cases, even using the symbolic representation it is impossible to come up with a plan for huge domains. In order to overcome this limitation, we propose two new operations called: symbolic weak regression and symbolic strong regression. The proposed operations compute the predecessors of a set of states using directly the actions specifications (pre-conditions and effects) and, thus, it is not necessary to construct the entire transition relation.

As future work, we intend to implement the non-deterministic regression operations using BDDs and compare their performance against to symbolic pre-image operations, using the non-deterministic planning domains from the International Planning Competition.

## REFERENCES

[1] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 74. Prentice hall Englewood Cliffs, 1995.

[2] R. E. Fikes and N. J. Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". *Artificial intelligence*, vol. 2, no. 3, pp. 189–208, 1972.

[3] A. Cimatti, M. Pistore, M. Roveri and P. Traverso. "Weak, strong, and strong cyclic planning via symbolic model checking". *Artificial Intelligence*, vol. 147, no. 1, pp. 35–84, 2003.

[4] S. L. Pereira and L. N. Barros. "A logic-based agent that plans for extended reachability goals". *Autonomous Agents and Multi-Agent Systems*, vol. 16, no. 3, pp. 327–344, June 2008.

[5] J. Hoffmann. "FF: The fast-forward planning system". *AI magazine*, vol. 22, no. 3, pp. 57, 2001.

[6] M. Helmert. "The Fast Downward Planning System." *J. Artif. Intell. Res.(JAIR)*, vol. 26, pp. 191–246, 2006.

[7] A. Gerevini, A. Saetti and M. Vallati. "An Automatically Configurable Portfolio-based Planner with Macro-actions: PbP." In *ICAPS*, 2009.

[8] P. D. Clarke E., Grumberg O. *Model Checking*. MIT Press, San Francisco, 1999.

[9] F. Buccafurri, T. Eiter, G. Gottlob and N. Leone. "Enhancing model checking in verification by AI techniques". *Artif. Intell.*, vol. 112, no. 1-2, pp. 57–104, 1999.

[10] J. M. Wing and M. Vaziri-Farahani. “A case study in model checking software systems”. *Sci. Comput. Program.*, vol. 28, no. 2-3, pp. 273–299, 1997.

[11] E. M. Clarke and E. A. Emerson. “Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic”. In *Logic of Programs, Workshop*, 1982.

[12] R. E. Bryant. “Graph-Based Algorithms for Boolean Function Manipulation”. *IEEE Trans. Comput.*, vol. 35, no. 8, pp. 677–691, 1986.

[13] S. Edelkamp, J. Hoffman, M. Littman and H. Younes. “International planning competition”. In *Proceedings of Fourteenth International Conference on Automated Planning & Scheduling (ICAPS-04)*, 2004.

[14] S. Edelkamp and M. Helmert. “MIPS: The model-checking integrated planning system”. *AI magazine*, vol. 22, no. 3, pp. 67, 2001.

[15] M. Helmert. “A Planning Heuristic Based on Causal Graph Analysis.” In *ICAPS*, volume 16, pp. 161–170, 2004.

[16] S. Edelkamp and M. Helmert. “On the implementation of MIPS”. In *AIPS-Workshop on Model-Theoretic Approaches to Planning*, pp. 18–25. Citeseer, 2000.

[17] M. Shanahan. *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. MIT press, 1997.

[18] V. Alcázar, D. Borrajo, S. Fernández and R. Fuentetaja. “Revisiting regression in planning”. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pp. 2254–2260. AAAI Press, 2013.

[19] H. Geffner and B. Bonet. “A Concise Introduction to Models and Methods for Automated Planning”. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 1, pp. 1–141, 2013.

[20] P. N. Russell, Stuart. “Artificial intelligence: A modern approach”. 2009.

[21] H. K. Buning, M. Karpinski and A. Flogel. “Resolution for quantified Boolean formulas”. *Information and computation*, vol. 117, no. 1, pp. 12–18, 1995.

[22] M. Pistore and P. Traverso. “Planning as model checking for extended goals in non-deterministic domains”. In *International Joint Conference on Artificial Intelligence*, volume 17, pp. 479–486. Citeseer, 2001.

[23] F. Giunchiglia and P. Traverso. “Planning as model checking”. *Recent Advances in AI Planning*, pp. 1–20, 2000.

[24] A. Cimatti, E. Giunchiglia, F. Giunchiglia and P. Traverso. “Planning via model checking: A decision procedure for AR”. *Recent Advances in AI planning*, pp. 130–142, 1997.

[25] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge Univ Pr, 2004.

[26] R. De Nicola and F. Vaandrager. “Action versus state based logics for transition systems”. In *Semantics of Systems of Concurrent Processes*, pp. 407–419. Springer, 1990.

[27] C. Pecheur and F. Raimondi. “Symbolic model checking of logics with actions”. In *Model Checking and Artificial Intelligence*, pp. 113–128. Springer, 2007.

[28] A. Cimatti and M. Roveri. “Conformant planning via model checking”. *Recent Advances in AI Planning*, pp. 21–34, 2000.

[29] J. Rintanen. “Regression for classical and nondeterministic planning”. In *ECAI*, pp. 568–571, 2008.

[30] E. Clarke, K. McMillan, S. Campos and V. Hartonas-Garmhausen. “Symbolic model checking”. In *Computer Aided Verification*, pp. 419–422. Springer, 1996.

[31] K. L. McMillan. “Symbolic Model Checking: An Approach to the State Explosion Problem”. In *PhD thesis, CMU*, 1992.

[32] M. P. Fourman. “Propositional planning”. In *Proceedings of AIPS-00 Workshop on Model-Theoretic Approaches to Planning*, pp. 10–17, 2000.

[33] J. Rintanen. “Introduction to automated planning”, 2004.