Strong Planning in Non-Deterministic Domains via Model Checking

Alessandro Cimatti and Marco Roveri and Paolo Traverso
IRST, Povo, 38100 Trento, Italy
{cimatti,roveri,leaf}@irst.itc.it

Abstract

Most real world domains are non-deterministic: the state of the world can be incompletely known, the effect of actions can not be completely foreseen, and the environment can change in unpredictable ways. Automatic plan formation in non-deterministic domains is, however, still an open problem. In this paper we show how to do strong planning in non-deterministic domains, i.e. finding automatically plans which are guaranteed to achieve the goal regardless of non-determinism. We define a notion of planning solution which is guaranteed to achieve the goal independently of non-determinism, a notion of plan including conditionals and iterations, and an automatic decision procedure for strong planning based on model checking techniques. The procedure is correct, complete and returns optimal plans. The work has been implemented in MBP, a planner based on model checking techniques.

Introduction

A fundamental assumption underlying most of the work in classical planning (see e.g. (Fikes & Nilsson 1971; Penberthy & Weld 1992)) is that the planning problem is deterministic: executing an action in a given state always leads to a unique state, environmental changes are all predictable and well known, and the initial state is completely specified. In real world domains, however, planners have often to deal with non-deterministic problems. This is mainly due to the intrinsic complexity of the planning domain, and to the fact that the external environment is highly dynamic, incompletely known and unpredictable. Examples of non-deterministic domains are many robotics, control and scheduling domains. In these domains, both the executions of actions and the external environment are often non-deterministic. The execution of an action in the same state may have - non-deterministically - possibly many different effects. For instance a robot may fail to pick up an object. The external environment may change in a way which is unpredictable. For instance, while a mobile robot is navigating, doors might be closed/opened, e.g. by external agents. Moreover, the initial state of a planning problem may be partially specified. For instance, a mobile robot should be able to reach a given location starting from different locations and situations.

The problem of dealing with non-deterministic domains has been tackled in reactive planning systems (e.g. (Beetz & McDermott 1994; Firby 1987; Georgeff & Lansky 1986; Simons 1990)) and in deductive planning (e.g. (Steel 1994; Stephan & Biundo 1993)). Nevertheless, the automatic generation of plans in non-deterministic domains is still an open problem. This paper shows how to do strong planning, i.e. finding automatically plans which are guaranteed to achieve the goal regardless of non-determinism. Our starting point is the framework of planning via model checking, together with the related system MBP, first presented in (Cimatti et al. 1997). We use AR (Giunchiglia, Kartha, & Lifschitz 1997), an expressive language to describe actions which, among other things, allows for non-deterministic action effects and environmental changes. We give semantics to domain descriptions in terms of finite state automata. Plans are generated by a completely automatic procedure, using OBDD-based model checking techniques, known as symbolic model checking (see, e.g., (Bryant 1992; Clarke, Grunberg, & Long 1994; Burch et al. 1992)), which allow for a compact representation and efficient exploration of finite state automata. This paper builds on (Cimatti et al. 1997) making the following contributions.

- We define a formal notion of strong planning, which captures the intuitive requirement that a plan should be able to solve a goal regardless of non-determinism.
- We enrich the notion of classical plans with more complex constructs including conditionals and iterations. This captures the intuition that simple sequences of actions, which may be successful for a given behavior of the environment, can fail for others, and therefore more complex planning constructs are required.
- We define a planning decision procedure for strong planning in non-deterministic domains. The procedure constructs universal plans (Schoppers 1987),
We have implemented this procedure in MBP (Model Based Planner) (Cimatti et al. 1997), a planner based on symbolic model checking. We have extended MBP with the ability to generate and execute plans containing the usual programming logic constructs, including conditionals, iterations and non-deterministic choice. The use of model checking techniques is of crucial importance for the compact representation of conditional plans.

This paper is structured as follows. We review the planning language 𝒜𝑹, and its semantics given in terms of finite state automata. Then we define the notion of "strong planning solution" and the extended deterministic plan executions is of minimal length. We prove that the procedure always terminates with a strong solution, i.e. a plan which guarantees goal achievement regardless of non-determinism, or with failure if no strong solution exists. Moreover, the solution is guaranteed to be "optimal", in the sense that the "worst" execution among the possible non-deterministic plan executions is of minimal length.

We have implemented this procedure in MBP (Model Based Planner) (Cimatti et al. 1997), a planner based on symbolic model checking. We have extended MBP with the ability to generate and execute plans containing the usual programming logic constructs, including conditionals, iterations and non-deterministic choice. The use of model checking techniques is of crucial importance for the compact representation of conditional plans.

Our approach is restricted to the finite case. This hypothesis, though quite strong, is acceptable for many practical cases.
drive-train has preconditions \( pos = \text{train-station} \lor (pos = \text{Victoria-station} \land light = \text{green}) \) \hspace{1cm} (6)

\( \text{drive-train causes } pos = \text{Victoria-station} \text{ if } pos = \text{train-station} \) \hspace{1cm} (7)

\( \text{drive-train causes } pos = \text{Gatwick} \text{ if } pos = \text{Victoria-station} \land light = \text{green} \) \hspace{1cm} (8)

\( \text{wait-at-light causes } light = \text{green} \text{ if } light = \text{red} \) \hspace{1cm} (9)

\( \text{wait-at-light causes } light = \text{red} \text{ if } light = \text{green} \) \hspace{1cm} (10)

\( \text{drive-truck has preconditions } pos = (\text{truck-station}, \text{city-center}) \land fuel \) \hspace{1cm} (11)

\( \text{drive-truck causes } pos = \text{city-center} \text{ if } pos = \text{truck-station} \) \hspace{1cm} (12)

\( \text{drive-truck causes } pos = \text{Gatwick} \text{ if } pos = \text{city-center} \) \hspace{1cm} (13)

\( \text{drive-truck possibly changes } fuel \text{ if } fuel \) \hspace{1cm} (14)

\( \text{make-fuel has preconditions } \neg fuel \) \hspace{1cm} (15)

\( \text{make-fuel causes } fuel \) \hspace{1cm} (16)

\( \text{fly has preconditions } pos = \text{air-station} \) \hspace{1cm} (17)

\( \text{fly causes } pos = \text{Gatwick} \text{ if } \neg fog \) \hspace{1cm} (18)

\( \text{fly causes } pos = \text{Luton} \text{ if } fog \) \hspace{1cm} (19)

\( \text{air-truck-transit has preconditions } pos = \text{air-station} \) \hspace{1cm} (20)

\( \text{air-truck-transit causes } pos = \text{truck-station} \) \hspace{1cm} (21)

\( \text{initially } (pos = (\text{train-station}, \text{air-station}) \lor (pos = \text{truck-station} \land fuel)) \)

\( \text{goal } pos = \text{Gatwick} \) \hspace{1cm} (22)

Figure 1: An example of non-deterministic planning problem.

\[ \text{in } \mathcal{F}, \text{we introduce a variable } F \text{ ranging over } \text{Rng}_F. \]

The intuition is that a formula represents a set of valuations to fluents, namely the set of valuations which satisfy the formula. For instance, the atomic formula \( pos = \text{Gatwick} \) represents the set of all possible valuations such that the fluent \( pos \) has value \( \text{Gatwick} \), and any other fluent \( F \) has any value in \( \text{Rng}_F \). Propositional connectives (conjunction, disjunction, negation) represent set-theoretic operations over sets of valuations (union, intersection, complement). For instance, the formula \( (pos = \text{Gatwick}) \lor fog \) represents the union of the sets of valuations where \( pos \) has value \( \text{Gatwick} \), and the set of valuations where \( fog \) has value \( \top \). In the following we will not distinguish formulae from the set of valuations they represent.

We construct now the automaton codifying the domain description. The states of the automaton are the valuations which satisfy every proposition of the form (always \( P \)). The initial states and the goal states are the states which satisfy every proposition of the form (initially \( P \)) and (goal \( G \)), respectively. The corresponding formulae are defined as follows.

\[
\begin{align*}
\text{State} & \equiv \bigwedge_{P; (\text{always } P) \in \mathcal{D}} P; \\
\text{Init} & \equiv \text{State} \land \bigwedge_{P; (\text{initially } P) \in \mathcal{D}} P; \\
\text{Goal} & \equiv \text{State} \land \bigwedge_{G; (\text{goal } G) \in \mathcal{D}} G.
\end{align*}
\]

The transition relation of an automaton maps an action and a given state into the set of possible states resulting from the execution. We introduce a variable \( Act \), ranging over \( \mathcal{A} \), representing the action to be executed. To represent the value of fluents after the execution of the action we introduce a variable \( F' \) for each fluent \( F \) in \( \mathcal{F} \). (In the following we call \( F \) and \( F' \) current and next fluent variables, respectively.) The transition relation of the automaton is represented by the formula \( Res \), in the current and next fluent variables, and in the variable \( Act \). The assignments to these variables which satisfy \( Res \) represent the possible transitions in the automaton. The transition relation for the automaton incorporates the solution to the frame problem in presence of non-deterministic actions. We first define the formula \( Res^0 \), which, intuitively, states what will necessarily happen in correspondence of actions. \( Res^0 \) relates a "current" state and an action \( A \) to the set of "next" states which satisfy the effects stated by all propositions in \( A \) of the form \( (A \text{ causes } P \text{ if } Q) \), which are active in the current state (i.e. \( Q \) holds). This is expressed formally by the following definition.

\[
\begin{align*}
\text{Res}^0 & \equiv \text{State} \land \text{State}[F_1/F'_1, \ldots, F_n/F'_n] \land \\
& \land ((\text{Act}=A\land Q) \lor P[F_1/F'_1, \ldots, F_n/F'_n]) \land \\
& \land (A \text{ causes } P \text{ if } Q) \in \mathcal{A}
\end{align*}
\]
eliminating all those valuations where variations of fluents are not necessary. In the following we assume that $F_1, \ldots, F_m$ $[F_{m+1}, \ldots, F_n, \text{resp.}]$ are the inertial [not inertial, resp.] fluents of $A$, listed according to a fixed enumeration. The formula $Res$ is defined as follows.

$$Res \triangleq Res^0 \land 
eg \exists v_1 \ldots v_n, (Res^0[F'_1/v_1, \ldots, F'_n/v_n] \land \bigwedge_{1 \leq i \leq m}(F_i = v_i \lor F'_i = v_i) \land \bigvee_{1 \leq i \leq m}(F'_i \neq v_i) \land (\text{Act} = A \land Q) \supset F'_h = v_h)$$

where $Q \in \mathcal{D}$. (25)

The informal reading of the above definition is that, given an action $A$, a valuation to next variables $F'_1, \ldots, F'_n$ is compatible with a valuation to current variables $F_1, \ldots, F_m$ if and only if it satisfies the effect conditions (i.e. $Res^0$), and there is no other valuation $\neg \exists v_1 \ldots v_n$ which also satisfies the effect conditions ($Res^0[F'_1/v_1, \ldots, F'_n/v_n]$), it agrees with the valuation to current variables on the fluents affected by $(\text{Act} = A \land Q) \supset F'_h = v_h$, and is closer to the current state (big disjunction and conjunction). Notice that inertial fluents which are affected by propositions of the form (2) are allowed to change freely (within the constraints imposed by $Res^0$), but only when the conditions (i.e. $Q$) are satisfied in the starting state. Non-inertial fluents are allowed to change freely, within the constraints imposed by $Res^0$.

A (non-deterministic) planning problem is the triple $<Res, Init, Goal>$. We say that a state $s$ satisfies a goal $Goal$ if $s \in Goal$. In the rest of this paper we assume a given (non-deterministic) planning problem $<Res, Init, Goal>$, where $Init$ is non-empty.

**Strong Solutions to Non-Deterministic Problems**

Intuitively, a strong solution to a non-deterministic planning problem is a plan which achieves the goal regardless of the non-determinism of the domain. This means that every possible state resulting from the execution of the plan is a goal state (in non-deterministic domains a plan may result in several possible states). This is a substantial extension of the definition of "planning solution" given in (Cimatti et al. 1997) (which, from now on, we will refer to as weak solution). A weak solution may be such that for some state resulting from the execution of the plan the goal is not satisfied, while a strong solution always guarantees the achievement of the goal even in non-deterministic domains.

Searching for strong solutions in the space of classical plans, i.e. sequences of basic actions, is bound to failure. Even for a simple problem as the example in figure 1, there is no sequence of actions which is a strong solution to the goal. Non-determinism must be tackled by planning a conditional behavior, which depends on the (sensing) information which can be gathered at execution time. For instance, we would expect to decide what to do according to the presence of fog in the initial state, or according to the status of the traffic light. Therefore, we extend the (classical) notion of plan (i.e. sequence of actions) to include non-deterministic choice, conditional branching, and iterative plans.

**Definition 0.1 ((Extended) Plan)** Let $\Phi$ be the set of propositions constructed from fluents in $F$. The set of (Extended) Plans $P$ for $D$ is the least set such that

1. if $a \in A$, then $a \in P$;
The Strong Planning Procedure

In this section we describe the planning procedure StrongPlan, which looks for a strong solution to a non-deterministic planning problem. The basic idea underlying the planning via model checking approach is the manipulation of sets of states. In the following we will present the routines by using standard set operators (e.g., \( \subseteq \)).

The basic planning step is the StrongPreimage function, defined as

\[
\text{StrongPreimage}(S) = \{<s, a>: s \in \text{State}, a \in \mathcal{A}, \emptyset \neq \text{Exec}[a](s) \subseteq S\}
\]

\(\text{StrongPreimage}(S)\) returns the set of pairs state-action \(<s, a>\) such that action \(a\) is applicable in state \(s\) \((\emptyset \neq \text{Exec}[a](s))\) and the (possibly non-deterministic) execution of \(a\) in \(s\) necessarily leads inside \(S\) \((\text{Exec}[a](s) \subseteq S)\). Intuitively, \(\text{StrongPreimage}(S)\) contains all the states from which \(S\) can be reached with a one-step plan regardless of non-determinism. We call state-action table a set of state-action pairs. In example in figure 2, the state-action table \(\text{StrongPreimage}(\text{Gatwick})\) is

\[
\begin{align*}
&\text{StrongPreimage}(\text{Gatwick}) = \\
&\{<\text{pos=Victoria-station, light=green}, \text{drive-train}>, \\
&\text{pos=city-center, fuel}, \text{drive-truck}>, \\
&\text{pos=air-station, \text{\text{-fog}}, fly}\} \\
\end{align*}
\]

The planning algorithm \text{StrongPlan}, shown in figure 3, takes as input the (data structure representing the) set of states satisfying \(G\). In case of success, it returns a state-action table \(SA\), which compactly encodes an extended plan which is a strong solution. In order to construct \(SA\), the algorithm loops backwards, from the goal towards the initial states, by repeatedly applying the \(\text{StrongPreimage}\) routine to the increasing set of states visited so far, i.e., \(Acc\). At each cycle, from step 5 to 10, \(Acc\) contains the set of states which have been previously accumulated, and for which we have a solution; \(SA\) contains the set of state-action pairs which are guaranteed to take us to \(Acc\). \text{PRUNESTATES(Preimage, Acc)} eliminates from \(Preimage\) the state-action pairs the states of which are already in \(Acc\), and thus have already been visited. This guarantees that we associate to a given state actions which construct only optimal plans. Then \(Acc\) is updated with the new states in \(Preimage\) (step 11). \text{PROJECTACTIONS}, given a set of state-action pairs, returns the corresponding set of states. We iterate this process, at each step testing the inclusion of the set of initial states in the currently accumulated states. The algorithm loops until either the set of initial states is completely included in the accumulated states (in which case we are guaranteed that a strong solution exists), or a fix point is found, i.e., there are no new accumulated states (in which case there is no strong solution and a \text{Fail} is returned).

The state-action table returned by \text{StrongPlan} is a compact encoding of an iterative conditional plan.
while (pos = \{Victoria-station, city-center, air-station, train-station, truck-station\}) do

1. if (pos = Victoria-station ∧ light = green) then drive-train
2. if (pos = city-center ∧ fuel) then make-fuel
3. if (pos = air-station ∧ ¬fog) then fly
4. if (pos = truck-station ∧ fuel) then drive-truck
5. if (pos = train-station) then drive-train
6. if (pos = train-station) then drive-train

Figure 4: A strong solution to the example of figure 1.

Technical Results

STRONGPLAN is guaranteed to terminate (theorem 0.1), is correct and complete (theorem 0.2), and returns optimal strong solutions (theorem 0.3). The proofs can be found in (Cimatti, Roveri, & Traverso 1998b).

Theorem 0.1 STRONGPLAN terminates.

In order to prove the properties of STRONGPLAN we define formally the extended plan \(\Pi_{SA}\) corresponding to a given state-action table \(SA\).

\[
\Pi_{SA} = \text{while } Domain_{SA} \text{ do IfThen}_{SA} \tag{27}
\]

where \(Domain_{SA}\) represents the states in the state-action table \(SA\), and \(\text{IfThen}_{SA}\) is the plan which, given a state \(s\), executes the corresponding action \(\alpha\) such that \((s, \alpha) \in SA\) (see figure 4). (If \(SA\) is empty, then \(Domain_{SA} = \bot\) and \(\text{IfThen}_{SA}\) can be any action.)

\[
\text{Domain}_{SA} = \exists s. \langle s, \alpha \rangle \in SA
\]

\[
\text{IfThen}_{SA} = \bigcup \{ s \text{ then } \alpha \mid (s, \alpha) \in SA \}
\]

Theorem 0.2 Let \(D = <Res, Init, Goal>\). If STRONGPLAN(Goal) returns a state-action table \(SA\), then \(\Pi_{SA}\) is a strong solution to \(D\). If STRONGPLAN(Goal) returns Fail then there exists no strong solution to \(D\).

In order to show that STRONGPLAN returns optimal strong plans, the first step is to define the cost of a plan. Given the non-determinism of the domain, the cost of a plan can not be defined in terms of the number of actions it contains. The paths of the plan \(\alpha\) in the state \(s \in \text{Apply}[\alpha]\) are all the sequences of states constructed during any possible execution of \(\alpha\). We define the \textit{worst length} of a plan in a given state as the maximal length of the paths of the plan.

Given a planning problem \(<Res, Init, Goal>\), we say that the strong solution \(\pi\) is optimal for the planning problem if, for any \(s \in \text{Init}\), there is no other strong solution with lower worst length.

As explained in section, we collapse the notion of state with that of its symbolic representation as a formula.
Theorem 0.3 Let $D = \langle Res, Init, Goal \rangle$. If there exists a strong solution to $D$, then STRONGPLAN( Goal) returns a state-action table $SA$ such that $\Pi_{SA}$ is an optimal strong solution.

Implementation and experimental results

The theory presented in this paper has been implemented in MBP (Model-Based Planner (Cimatti et al. 1997)). MBP uses OBDDs (Ordered Binary Decision Diagrams) (Bryant 1992) and symbolic model checking techniques (Clarke, Grunberg, & Long 1994) to compactly encode and analyze the automata representing the planning domain. OBDDs give a very concise representation of sets of states. The dimension, i.e. the number of nodes, of an OBDD does not necessarily depend on the actual number of states. Set theoretical operations (e.g. union and intersection) are implemented as very efficient operations on OBDD (e.g. disjunction and conjunction). In this work we make substantial use of OBDDs to compactly represent state-action tables, i.e. universal plans. A state-action table is an OBDD in the variables $Act$ and $Fi$, and each possible assignment to this relates a state to the corresponding action. The compactness of the representation results from the normal form of OBDDs, which allows for a substantial sharing. Although MBP can extract and print the verbose version of the universal plan, an approach based on the explicit manipulation of such data structures would be impractical (the size and complexity of conditional plans greatly increases with the number of tests sensing the current state).

The actual implementation of STRONGPLAN is more sophisticated and powerful than the algorithm presented in this paper. It returns, in case of failure, a plan which is guaranteed to solve the goal for the possible subset of initial states from which there exists a strong solution. In the case $Init \cap Acc \neq \emptyset$, MBP contains all the possible optimal strong solutions from the subset of the initial state $Init \cap Acc$. This allows the planner to decide, at the beginning of the execution, depending on the initial state, whether it is possible to execute a plan which is a strong solution for the particular initial states, even if no strong solution exists a-priori for any possible initial state.

In MBP it is also possible to directly execute state-action tables encoded as OBDD. At each execution step, MBP encodes the information acquired from the environment in terms of OBDDs, compares it with the state-action table (this operation is performed as a conjunction of OBDDs), and retrieves and executes the corresponding action. Finally, the routine for temporal projection presented in (Cimatti et al. 1997) has been extended to deal with extended plans. Given a set of initial states and a goal, it simulates the execution of an extended plan and checks whether the goal has been achieved. The nontrivial step is the computation of the iterations, which is done by means of OBDD-based fix point computations.

The planning algorithm has been tested on some examples. For the example in figure 2 MBP finds the set of optimal strong solutions in 0.03 seconds. (All tests were performed on a 200MHz Pentium MMX with 96MB RAM.) We have also experimented with a non-deterministic moving target search problem. The "hunter-prey" (Koenig & Simmons 1995). In the simple case presented in (Koenig & Simmons 1995) (6 vertex nodes with one initial state) MBP finds the strong optimal solution in 0.05 seconds. We have experimented with larger configurations: the strong solution is found in 0.30 seconds with 50 nodes, 0.85 seconds with 100 nodes, 1.532 seconds with 500 nodes and 54.66 seconds with 1000 nodes (these times include the encoding of the automaton into OBDDs). We have extended the problem from a given initial state to the set of any possible initial state. MBP takes 0.07 seconds with 6 nodes, 0.40 seconds with 50 nodes, 1.08 seconds with 100 nodes, 18.77 seconds with 500 nodes and 70.00 seconds with 1000 nodes. These tests, far from being a definite experimental analysis, appear very promising. An extensive test of the strong planning routine on a set of realistic domains is the object of future work.

Conclusion and related work

In this paper we have presented an algorithm for solving planning problems in non-deterministic domains. The algorithm generates automatically universal plans which are guaranteed to achieve a goal regardless of non-determinism. It always terminates, even in the case no solution exists, it is correct and complete, and finds optimal plans, in the sense that the worst-case length of the plan (in terms of number of actions) is minimal with respect to other possible solution plans. Universal plans are encoded compactly through OBDDs as state-action tables which can be directly executed by the planner. The planning and execution algorithm have been implemented in MBP, a planner based on model checking, which has been extended to execute "extended plans", i.e. plans containing conditionals and iterations. We have tested the planning algorithm on an example proposed for real-time search in non-deterministic domains (Koenig & Simmons 1995) and the performance are very promising.

In the short term, we plan to add to the planning algorithm the ability of generating optimized state-action tables. For instance, it would be possible (and relatively easy) to extend the planning procedure to take into account the preconditions and the determinism of actions in order to limit the number of tests which have to be applied at run time. In this paper we do not deal with the problem of generating plans which need to contain iterations. In our current implementation, unbounded loops are always avoided since they might non-deterministically loop forever, and therefore
they do not guarantee a strong solution. In general, iterations may be of great importance, for instance to codify a trial-and-error strategy ("repeat pick up block until succeed"), or to accomplish cyclic missions. On the one side, it is possible to extend the framework to generate iterative plans which strongly achieve the goal under the assumption that non-determinism is "fair", i.e. a certain condition will not be ignored forever (Cimatti, Roveri, & Traverso 1998a). On the other, the notion of goal must be extended in order to express more complex (e.g. cyclic) behaviors. A further main future objective is the investigation of the possibility of interleaving flexibly the planning procedure with its execution.

As far as we know, this is the first automatic planning procedure which generates universal plans which are encoded as OBDDs and are guaranteed to achieve the goal in a non-deterministic domain. While expressive frameworks to deal with non-determinism have been previously devised (see for instance (Stephan & Biundo 1993; Steel 1994)), the automatic generation of plans in these frameworks is still an open problem. Koening and Simmons (Koenig & Simmons 1995) describe an extension of real-time search (called min-max LRTA*) to deal with non-deterministic domains. As any real-time search algorithm, min-max LRTA* does not plan ahead (it selects actions at run time depending on a heuristic function) and, in some domains (Koenig & Simmons 1995), it is not guaranteed to find a solution. Some of the work in conditional planning (see for instance (Warren 1976; Peot & Smith 1992)) allows for a limited form of non-determinism, through the use of an extension of STRIPS operators, called conditional actions, i.e. actions with different, mutually exclusive sets of outcomes. The domain descriptions we deal with are far more expressive and, in spite of this expressiveness, the planning algorithm is guaranteed to terminate and to find a strong solution. Moreover, an intrinsic problem in conditional planning is the size and complexity of the plans. The use of a extended plans with iterations and that of OBDDs to encode state-action tables reduces significantly this problem.

References


