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; Simmons 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 $\mathcal{AR}$ (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),

which repeatedly sense the world, select an appropriate action, execute it, and iterate until the goal is reached.

- We prove that the procedure always terminates with a strong solution, i.e. a plan which guarantees goal achievement regardless 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.

This paper is structured as follows. We review the planning language $\mathcal{AR}$, and its semantics given in terms of finite state automata. Then we define the notion of "strong planning solution" and the extended language for plans. We define the planning procedure which generates universal plans, and we prove its formal properties. Finally we describe the implementation in MBP and present some preliminary experimental results.

Non-deterministic Domains as Finite Automata

An $\mathcal{AR}$ language is characterized by a finite nonempty set of *actions*, and a finite nonempty set of *fluents*. Each fluent F is associated to a finite nonempty set Rng_F of *values* (the *range* of F) [1]. An *atomic formula* has the form $(F = V)$, where F is a fluent, and $V \in Rng_F$. In case Rng_F is $\{\top, \bot\}$ (where $\top$ and $\bot$ stand for truth and falsity, respectively), the fluent is said to be propositional or boolean, and the atomic formula $F = \top$ is abbreviated with F. A *formula* is a propositional combination of atomic formulas (we use the usual connectives for negation $\neg$, conjunction $\wedge$ and disjunction $\vee$). Fluents can be either *inertial* or *non-inertial* depending on whether they obey or not to the law of inertia. $\mathcal{AR}$ allows for propositions of the following form (where A is an action, F is an inertial fluent, and P and Q are formulas).

$$A \textbf{ causes } P \textbf{ if } Q \qquad (1)$$

$$A \textbf{ possibly changes } F \textbf{ if } Q \qquad (2)$$

$$\textbf{always } P \qquad (3)$$

$$\textbf{initially } P \qquad (4)$$

$$\textbf{goal } G \qquad (5)$$

[1] Our approach is restricted to the finite case. This hypothesis, though quite strong, is acceptable for many practical cases.

Intuitively, (1) states that P holds after we execute A in a state in which Q holds; (2) states that F may non-deterministically change its value if we execute A in a state in which Q holds; (3) and (4) state that P holds in any state and in the initial states, respectively; (5) defines the goal states as the ones where G holds. The proposition A **has preconditions** P, stating that A can be executed only in states in which P holds, is an abbreviation for A **causes** $\bot$ **if** $\neg P$. A *(non-deterministic) domain description* is a set of propositions of the form $(1-4)$. A *(non-deterministic) planning problem description* is a set of propositions of the form $(1-5)$.

We explain the intuitive meaning of $\mathcal{AR}$ through the simple example in figure 1. The planning problem description describes a situation in which a pack can be moved from the railway, truck or airplane station of a city airport (e.g. London Heatrow) to different city airports (e.g. London Gatwick, and Luton). An informal pictorial representation of the domain is given in figure 2. The possible actions are *drive-train*, *wait-at-light*, *drive-truck*, *make-fuel*, *fly* and *air-truck-transit*. *pos* is an inertial fluent the range of which is {*train-station*, *truck-station*, *air-station*, *Victoria-station*, *city-center*, *Gatwick*, *Luton*}; *fuel* is an inertial propositional fluent, *light* is a non-inertial fluent the range of which is {*green*, *red*}, *fog* is a non inertial propositional fluent. We abbreviate $pos = v_1 \vee \ldots \vee pos = v_n$ with $pos = \{v_1, \ldots, v_n\}$. This example shows how $\mathcal{AR}$ allows for describing non-deterministic actions (*drive-truck* and *drive-train*) and non-deterministic environmental changes (non-inertial fluents *light* and *fog*). It is also possible to express conditional effects, i.e. the fact that an action may have different effects depending on the state in which it is executed (e.g., see actions *wait-at-light* and *fly*). Finally, we can express partial specifications of the initial situation, corresponding to a set of possible initial states. All of this makes the language far more expressive than STRIPS-like (Fikes & Nilsson 1971) or ADL-like (Penberthy & Weld 1992) languages. In the following $\mathcal{D}$ is a given planning problem description, $\mathcal{F}$ is the set of its fluents, and $\mathcal{A}$ the set of its actions.

$\mathcal{AR}$ domain descriptions are given semantics in terms of automata (Giunchiglia, Kartha, & Lifschitz 1997). Intuitively, a domain description is associated with an automaton. The states of the automaton represent the admissible valuations to fluents, a *valuation* being a function that associates to each fluent F an element of Rng_F. The transitions from state to state in the automaton represent execution of actions. We overview now the translation from $\mathcal{D}$ into formulas which directly codify the corresponding finite state automaton and which allow us to generate plans by exploiting symbolic model checking techniques (this construction was first defined and proved correct in (Cimatti *et al.* 1997)). For each fluent F

$$\textit{drive-train}\ \textbf{has preconditions}\ pos = \textit{train-station} \lor (pos = \textit{Victoria-station} \land light = green) \tag{6}$$
$$\textit{drive-train}\ \textbf{causes}\ pos = \textit{Victoria-station}\ \textbf{if}\ pos = \textit{train-station} \tag{7}$$
$$\textit{drive-train}\ \textbf{causes}\ pos = Gatwick\ \textbf{if}\ pos = \textit{Victoria-station} \land light = green \tag{8}$$
$$\textit{wait-at-light}\ \textbf{causes}\ light = green\ \textbf{if}\ light = red \tag{9}$$
$$\textit{wait-at-light}\ \textbf{causes}\ light = red\ \textbf{if}\ light = green \tag{10}$$
$$\textit{drive-truck}\ \textbf{has preconditions}\ pos = \{\textit{truck-station}, \textit{city-center}\} \land fuel \tag{11}$$
$$\textit{drive-truck}\ \textbf{causes}\ pos = \textit{city-center}\ \textbf{if}\ pos = \textit{truck-station} \tag{12}$$
$$\textit{drive-truck}\ \textbf{causes}\ pos = Gatwick\ \textbf{if}\ pos = \textit{city-center} \tag{13}$$
$$\textit{drive-truck}\ \textbf{possibly changes}\ fuel\ \textbf{if}\ fuel \tag{14}$$
$$\textit{make-fuel}\ \textbf{has preconditions}\ \neg fuel \tag{15}$$
$$\textit{make-fuel}\ \textbf{causes}\ fuel \tag{16}$$
$$\textit{fly}\ \textbf{has preconditions}\ pos = \textit{air-station} \tag{17}$$
$$\textit{fly}\ \textbf{causes}\ pos = Gatwick\ \textbf{if}\ \neg fog \tag{18}$$
$$\textit{fly}\ \textbf{causes}\ pos = Luton\ \textbf{if}\ fog \tag{19}$$
$$\textit{air-truck-transit}\ \textbf{has preconditions}\ pos = \textit{air-station} \tag{20}$$
$$\textit{air-truck-transit}\ \textbf{causes}\ pos = \textit{truck-station} \tag{21}$$
$$\textbf{initially}\ (pos = \{\textit{train-station}, \textit{air-station}\} \lor (pos = \textit{truck-station} \land fuel)) \tag{22}$$
$$\textbf{goal}\ pos = Gatwick \tag{23}$$

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

in $\mathcal{F}$, we introduce a variable F ranging over 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 $pos = Gatwick$ represents the set of all possible valuations such that the fluent *pos* has value *Gatwick*, and any other fluent F has any value in Rng_F. Propositional connectives (conjunction, disjunction, negation) represent set-theoretic operations over sets of valuations (union, intersection, complement). For instance, the formula $(pos = Gatwick) \lor fog$ represents the union of the sets of valuations where *pos* has value *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.

$$State \doteq \bigwedge_{(\textbf{always}\ P) \in \mathcal{D}} P; \qquad Init \doteq State \land \bigwedge_{(\textbf{initially}\ P) \in \mathcal{D}} P;$$
$$Goal \doteq State \land \bigwedge_{(\textbf{goal}\ G) \in \mathcal{D}} G.$$

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 $\mathcal{A}$ of the form (A **causes** P **if** Q), which are active in the current state (i.e. Q holds). This is expressed formally by the following definition.

$$Res^0 \doteq State \land State[F_1/F_1', \ldots, F_n/F_n'] \land \bigwedge_{(A\ \textbf{causes}\ P\ \textbf{if}\ Q) \in \mathcal{D}} ((\mathrm{Act}{=}A \land Q) \supset P[F_1/F_1', \ldots, F_n/F_n']) \tag{24}$$

We denote with $[F_1/F_1', \ldots, F_n/F_n']$ the simultaneous substitution of each F_i with F_i'. Intuitively, the conjunct $State[F_1/F_1', \ldots, F_n/F_n']$ imposes that the valuations to next fluent variables must be states, while $P[F_1/F_1', \ldots, F_n/F_n']$ imposes that the effect of the action must hold in the next state whenever Q holds in the current state. Res^0 identifies all constraints on what *must change*, but says nothing about what *does not change*. The transition relation of the automaton *Res* is obtained by minimizing what does not change,

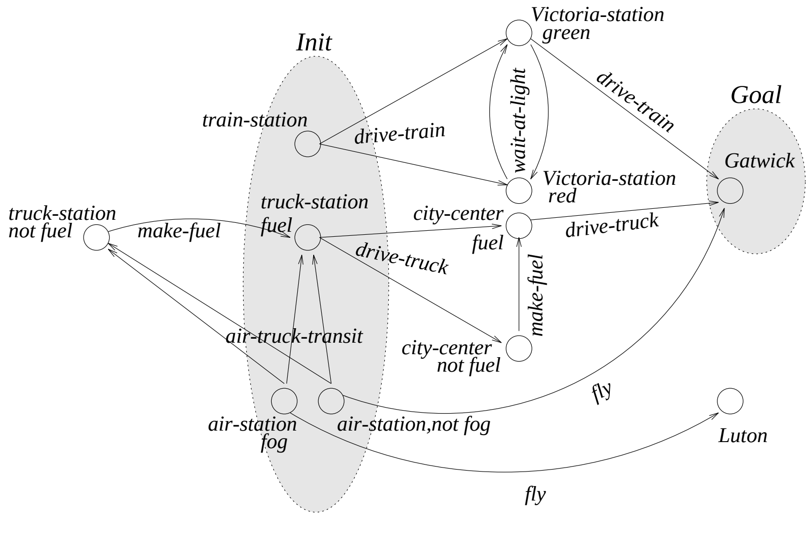

Figure 2: An informal description of the domain of figure 1.

eliminating all those valuations where variations of fluents are not necessary. In the following we assume that $F_1, \ldots, F_m$ [$F_{m+1}, \ldots, F_n$, resp.] are the inertial [not inertial, resp.] fluents of $\mathcal{A}$, listed according to a fixed enumeration. The formula *Res* is defined as follows.

$$\begin{aligned} Res \doteq\ & Res^0 & \wedge \\ & \neg\exists v_1 \ldots v_n.(Res^0[F_1'/v_1, \ldots, F_n'/v_n] & \wedge \\ & \textstyle\bigwedge_{1 \le i \le m}(F_i = v_i \vee F_i' = v_i) & \wedge \\ & \textstyle\bigvee_{1 \le i \le m}(F_i' \neq v_i)) & \wedge \\ & \bigwedge_{(A \textbf{ possibly changes } F_h \textbf{ if } Q) \in \mathcal{D}} ((Act = A \wedge Q) \supset F_h' = v_h) \end{aligned} \quad (25)$$

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 $F_1, \ldots, F_n$ 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 $((Act = A \wedge Q) \supset F_h' = v_h)$, 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 Φ be the set of propositions constructed from fluents in $\mathcal{F}$. The set of (Extended) Plans $\mathcal{P}$ for $\mathcal{D}$ is the least set such that*

1. *if $\alpha \in \mathcal{A}$, then $\alpha \in \mathcal{P}$;*

2. *if* $\alpha, \beta \in \mathcal{P}$, *then* $\alpha; \beta \in \mathcal{P}$;
3. *if* $\alpha, \beta \in \mathcal{P}$, *then* $\alpha \cup \beta \in \mathcal{P}$;
4. *if* $\alpha, \beta \in \mathcal{P}$ *and* $p \in \Phi$, *then* **if** p **then** α **else** $\beta \in \mathcal{P}$;
5. *if* $\alpha \in \mathcal{P}$ *and* $p \in \Phi$, *then* **if** p **then** $\alpha \in \mathcal{P}$;
6. *if* $\alpha \in \mathcal{P}$ *and* $p \in \Phi$, *then* **while** p **do** $\alpha \in \mathcal{P}$.

Extended plans are a subset of program terms in Dynamic Logic (DL) (Harel 1984) modeling constructs for SDL and non-deterministic choice.

Definition 0.2 (Plan Execution) *Let Φ be the set of propositions constructed from fluents in $\mathcal{F}$ and let $p \in \Phi$. Let S be a finite set of states. Let α be an extended plan for $\mathcal{D}$. The execution of α in S, written $Exec[\alpha](S)$, is defined as:*

1. $Exec[\alpha](S) = \{s' | s \in S, Res(s, \alpha, s')\}$, if $\alpha \in \mathcal{A}$;
2. $Exec[\alpha; \beta](S) = Exec[\beta](Exec[\alpha](S))$;
3. $Exec[\alpha \cup \beta](S) = Exec[\alpha](S) \cup Exec[\beta](S)$;
4. $Exec[\textbf{if } p \textbf{ then } \alpha \textbf{ else } \beta](S) = Exec[\alpha](S \cap p) \cup Exec[\beta](S \cap \neg p)$;
5. $Exec[\textbf{if } p \textbf{ then } \alpha](S) = Exec[\alpha](S \cap p) \cup \neg p$;
6. $Exec[\textbf{while } p \textbf{ do } \alpha](S) = \mathbf{lfp}_Z[S \cup Exec[\alpha](Z \cap p)] \cap \neg p$.

The execution of a basic action α in the set of states S is the set of states which can be reached by applying α to any of the states in S [2]. The execution of extended plans is similar to the execution of programs in DL. The basic difference is that in our case the execution of an iteration is guaranteed to have a least fix point by the finiteness of the domain. We are interested in plans where actions are guaranteed to be applicable regardless of non-determinism. The following definition captures this notion.

Definition 0.3 (Applicability set of a Plan) *Let Φ be the set of propositions constructed from fluents in $\mathcal{F}$ and let $p \in \Phi$. Let α be an extended plan for $\mathcal{D}$. The applicability set of α, written $Appl[\alpha]$, is defined as follows:*

1. $Appl[\alpha] = \{s | Exec[\alpha](s) \neq \emptyset\}$ if $\alpha \in \mathcal{A}$;
2. $Appl[\alpha; \beta] = \{s | s \in Appl[\alpha], \text{ and } Exec[\alpha](s) \subseteq Appl[\beta]\}$;
3. $Appl[\alpha \cup \beta] = Appl[\alpha] \cap Appl[\beta]$;
4. $Appl[\textbf{if } p \textbf{ then } \alpha \textbf{ else } \beta] = (Appl[\alpha] \cap p) \cup (Appl[\beta] \cap \neg p)$;
5. $Appl[\textbf{if } p \textbf{ then } \alpha] = (Appl[\alpha] \cap p) \cup \neg p$;
6. $Appl[\textbf{while } p \textbf{ do } \alpha] = \mathbf{lfp}_Z[\neg p \cup [p \cap Appl[\alpha] \cap \mathbf{AX}_\alpha(Z)]]$, *where* $\mathbf{AX}_\alpha(S) = \{s | Exec[\alpha](s) \subseteq S\}$.

Definition 0.4 (Strong solution) *A plan α is a strong solution to a non-deterministic planning problem with initial states Init and goal Goal, if $Init \subseteq Appl[\alpha]$ and for all states $s \in Exec[\alpha](Init)$, s satisfies Goal.*

[2] For the sake of simplicity, this notion is presented here as a set-theoretical operation. The computation of the corresponding formula, though standard in model checking (basically it is the *image computation* of a set for a given transition relation), is rather complicated and involves the instantiation of the variable Act with α in Res, conjunction with the formula representing S, existential quantification of current variables (F_i), and substitution of F_i' with F_i (backward shifting).

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. $\subseteq, \setminus$).

The basic planning step is the StrongPreImage function, defined as

$$\begin{aligned} &\text{StrongPreImage}(S) = \\ &\quad \{<s, \alpha> : s \in State, \alpha \in \mathcal{A}, \emptyset \neq Exec[\alpha](s) \subseteq S\} \end{aligned} \tag{26}$$

StrongPreImage(S) returns the set of pairs state-action $<s, \alpha>$ such that action α is applicable in state s ($\emptyset \neq Exec[\alpha](s)$) and the (possibly non-deterministic) execution of α in s necessarily leads inside S ($Exec[\alpha](s) \subseteq S$). Intuitively, 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 StrongPreImage(*Gatwick*) is

$$\begin{array}{lll} \{ & <(\text{pos=Victoria-station} \wedge \text{light=green}), & \text{drive-train}>, \\ & <(\text{pos=city-center} \wedge \text{fuel}), & \text{drive-truck}>, \\ & <(\text{pos=air-station} \wedge \neg\text{fog}), & \text{fly}> \quad \} \end{array}$$

The planning algorithm 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 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. 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). 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 *Fail* is returned).

The state-action table returned by StrongPlan is a compact encoding of an iterative conditional plan.

while $(pos = \{\textit{Victoria-station}, \textit{city-center}, \textit{air-station}, \textit{train-station}, \textit{truck-station}\})$ **do**

[1]	**if**	$(pos = \textit{Victoria-station} \wedge light = green)$	**then**	*drive-train*	$\cup$
[1]	**if**	$(pos = \textit{city-center} \wedge fuel)$	**then**	*drive-truck*	$\cup$
[1]	**if**	$(pos = \textit{air-station} \wedge \neg fog)$	**then**	*fly*	$\cup$
[2]	**if**	$(pos = \textit{Victoria-station} \wedge light = red)$	**then**	*wait-at-light*	$\cup$
[2]	**if**	$(pos = \textit{city-center} \wedge \neg fuel)$	**then**	*make-fuel*	$\cup$
[3]	**if**	$(pos = \textit{train-station})$	**then**	*drive-train*	$\cup$
[3]	**if**	$(pos = \textit{truck-station} \wedge fuel)$	**then**	*drive-truck*	$\cup$
[4]	**if**	$(pos = \textit{truck-station} \wedge \neg fuel)$	**then**	*make-fuel*	$\cup$
[5]	**if**	$(pos = \textit{air-station} \wedge fog)$	**then**	*air-truck-transit*	

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

procedure StrongPlan(G)
  $OldAcc := \emptyset$;
  $Acc := G$;
  $SA := \emptyset$;
  while $(Acc \neq OldAcc)$
    if $Init \subseteq Acc$
      then return SA;
    $PreImage :=$ StrongPreImage(Acc);
    $SA := SA \cup$ PruneStates($PreImage, Acc$);
    $OldAcc := Acc$;
    $Acc := Acc \cup$ ProjectActions($PreImage$);
  return $Fail$;

Figure 3: The Strong Planning Algorithm.

The idea is that conditionals are needed to take into account the different forms of non-determinism, while iteration is used to cycle until the goal has been reached. In figure 4 we show the plan corresponding to the state-action table SA returned by StrongPlan for the example in figure 2. Each if-then branch in the plan directly corresponds to one state-action pair in SA. The while test is the disjunction of the predicates describing each state in SA (in the following called *Domain* of the SA).

Let us consider now how StrongPlan builds the state-action table generating the plan of figure 4. On the left, in square brackets, is the StrongPlan iteration at which each state-action pair is added to SA. Notice how, at iteration 3, the pair

<(pos=Victoria-station∧light=green), wait-at-light>

is pruned away (by PruneStates) since the corresponding states have been previously accumulated. This is an example of loop checking performed by StrongPlan which prunes away plans which generate loops. Similarly, <(pos=air-station∧ ¬fog),air-truck-transit> is pruned away, since the resulting plan would be longer than the plan where *fly* is applied in $pos = \textit{air-station} \wedge \neg fog$.

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 Π_{SA} corresponding to a given state-action table SA.

$$\Pi_{SA} \doteq \textbf{while } Domain_{SA} \textbf{ do } IfThen_{SA} \tag{27}$$

where $Domain_{SA}$ represents the states in the state-action table SA [3], and $IfThen_{SA}$ is the plan which, given a state s, executes the corresponding action α such that $\langle s, \alpha\rangle \in SA$ (see figure 4). (If SA is empty, then $Domain_{SA} = \perp$ and $IfThen_{SA}$ can be any action.)

$$Domain_{SA} \doteq \bigvee_{\exists \alpha . <\alpha, s> \in SA} s$$

$$IfThen_{SA} \doteq \bigcup_{\langle s, \alpha\rangle \in SA} \textbf{if } s \textbf{ then } \alpha$$

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

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 α in the state $s \in Appl[\alpha]$ are all the sequences of states constructed during any possible execution of α. We define the *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 π is optimal for the planning problem if, for any $s \in Init$, there is no other strong solution with lower worst length.

[3] As explained in section , we collapse the notion of state with that of its symbolic representation as a formula.

Theorem 0.3 *Let $\mathcal{D} = <Res, Init, Goal>$. If there exists a strong solution to $\mathcal{D}$, then* StrongPlan(*Goal*) *returns a state-action table SA such that Π_{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 depends 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 F_i, 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 $Init \not\subseteq Acc$ but $Init \cap Acc \neq \emptyset$, *SA* 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, 15.32 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. Koenig 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

Beetz, M., and McDermott, D. 1994. Improving Robot Plans During Their Execution. In *Proc. of AIPS-94.*

Bryant, R. E. 1992. Symbolic Boolean manipolation with ordered binary-decision diagrams. *ACM Computing Surveys* 24(3):293–318.

Burch, J. R.; Clarke, E. M.; McMillan, K. L.; Dill, D. L.; and Hwang, L. J. 1992. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation* 98(2):142–170.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via Model Checking: A Decision Procedure for $\mathcal{AR}$. In *Proc. of ECP-97.*

Cimatti, A.; Roveri, M.; and Traverso, P. 1998a. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. To appear in *Proc. of AAAI-98.*

Cimatti, A.; Roveri, M.; and Traverso, P. 1998b. Strong Planning in Non-Deterministic Domains via Model Checking. Technical Report 9801-07, IRST, Trento, Italy. Extended version of this paper.

Clarke, E.; Grunberg, O.; and Long, D. 1994. Model checking. In *Proc. of the International Summer School on Deductive Program Design, Marktoberdorf.*

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2(3-4):189–208.

Firby, R. J. 1987. An Investigation into Reactive Planning in Complex Domains. In *Proc. of AAAI-87*, 202–206.

Georgeff, M., and Lansky, A. L. 1986. Procedural knowledge. *Proc. of IEEE* 74(10):1383–1398.

Giunchiglia, E.; Kartha, G. N.; and Lifschitz, V. 1997. Representing action: Indeterminacy and ramifications. *Artificial Intelligence* 95(2):409–438.

Harel, D. 1984. Dynamic Logic. In Gabbay, D., and Guenthner, F., eds., *Handbook of Philosophical Logic*, volume II. D. Reidel Publishing Company. 497–604.

Koenig, S., and Simmons, R. 1995. Real-Time Search in Non-Deterministic Domains. In *Proc. of IJCAI-95*, 1660–1667.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for adl. In *Proc. of KR-92.*

Peot, M., and Smith, D. 1992. Conditional Nonlinear Planning. In *Proc. of AIPS-92*, 189–197.

Schoppers, M. J. 1987. Universal plans for Reactive Robots in Unpredictable Environments. In *Proc. of the 10th International Joint Conference on Artificial Intelligence*, 1039–1046.

Simmons, R. 1990. An Architecture for Coordinating Planning, Sensing and Action. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, 292–297.

Steel, S. 1994. Action under Uncertainty. *J. Logic and Computation, Special Issue on Action and Processes* 4(5):777–795.

Stephan, W., and Biundo, S. 1993. A New Logical Framework for Deductive Planning. In *Proc.IJCAI-93*, 32–38. Morgan Kaufmann.

Warren, D. 1976. Generating Conditional Plans and Programs. In *Proc. of AISB-76.*