ITC irst

CENTRO PER LA RICERCA
SCIENTIFICA E TECNOLOGICA

38050 Povo (Trento), Italy
Tel.: +39 0461 314312
Fax: +39 0461 302040
e−mail: prdoc@itc.it − url: http://www.itc.it

CONFORMANT PLANNING VIA MODEL CHECKING

Cimatti A., Roveri M.

August 1999

Technical Report # 9908−02

Conformant Planning via Model Checking

Alessandro Cimatti[1] and Marco Roveri[1,2]

[1] ITC-IRST, Via Sommarive 18, 38055 Povo, Trento, Italy,
[2] DSI, University of Milano, Via Comelico 39, 20135 Milano, Italy
{cimatti,roveri}@irst.itc.it

Abstract. Conformant planning is the problem of finding a sequence of actions that is guaranteed to achieve the goal for any possible initial state and nondeterministic behavior of the planning domain. In this paper we present a new approach to conformant planning. We propose an algorithm that returns the set of all conformant plans of minimal length if the problem admits a solution, otherwise it returns with failure. Our work is based on the planning via model checking paradigm, and relies on symbolic techniques such as Binary Decision Diagrams to compactly represent and efficiently analyze the planning domain. The algorithm, called CMBP, has been implemented in the MBP planner. CMBP is strictly more expressive than the state of the art conformant planner CGP. Furthermore, an experimental evaluation suggests that CMBP is able to deal with uncertainties more efficiently than CGP.

1 Introduction

The planning via model checking [5, 8, 7, 9] paradigm is based on the interpretation of a planning domain as a finite state automaton [5]. A high level action language, AR [10], is used to describe complex, nondeterministic domains with multiple initial states, and actions with conditional and uncertain effects. Symbolic representation and exploration techniques on the style of symbolic model checking [3, 15], based on the use of Binary Decision Diagrams (BDDs) [2], allow for efficient planning in nondeterministic domains. The planning algorithm presented in [8] allows to find *strong* plans, i.e. conditional (contingent) plans which are guaranteed to achieve the goal for any initial state and any possible nondeterministic evolution of the domain. The algorithms defined in [7] and in [9] also allow for the generation of iterative trial-and-error strategies.

The work in [8, 7, 9] rely on the hypothesis of complete run-time observability. That is, the status of the world after the execution of a (possibly nondeterministic) action is assumed to be completely observable. The derived plans can be (heavily) conditioned to run-time observations. However, in many real world situations, sensorial information may be costly or unavailable, and techniques are needed to deal with incomplete run-time observability. In this work we extend the planning via model checking paradigm by proposing a new algorithm for conformant planning, i.e. the problem of finding a plan achieving the goal for any possible contingency in total absence of run-time information. Since no information is available at run time, the plan can not be conditioned to run-time observation, and thus it must be a sequence of actions, i.e. a classical plan. Differently from the classical planning problem, however, here a sequence of actions can result in (many) different executions, depending on the initial state and on the different uncertain outcomes of actions. This makes conformant planning much harder than classical planning.

The conformant planning algorithm is applicable to complex planning domains, with conditional actions, uncertainty in the initial state and in the outcomes of actions. The algorithm is complete, i.e. it returns with failure if and only if the problem admits no conformant solution. If a solution exists, it returns *all* conformant plans of minimal length. The algorithm has been implemented in MBP (Model Based Planner) [5, 8, 7], a planner developed on top of the NuSMV [4] model checker, and an experimental analysis has been carried out. The experimental results show that the algorithm can solve rather complex problems, and compares nicely with the state of the art conformant planner CGP [19]. In particular, it is able to express and solve problems with uncertain effects of actions, which can not be expressed in CGP. Furthermore, differently from CGP, our algorithm is not directly related to the *number* of initial states and uncertainties in action effects, and can plan rather efficiently in highly nondeterministic domains.

This paper is structured as follows. In section 2 we present some necessary background. In section 3 we describe the algorithm, and in section 4 we present the experimental results. In section 5 we draw the conclusions and discuss some future research.

2 Background

A planning domain is a 4-tuple $\mathcal{D} = (\mathcal{F}, \mathcal{S}, \mathcal{A}, \mathcal{R})$, where $\mathcal{F}$ is the (finite) set of fluents (atomic propositions), $\mathcal{S} \subseteq 2^{\mathcal{F}}$ is the set of states, $\mathcal{A}$ is the (finite) set of actions, and $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition relation. Intuitively, a state is identified with the set of propositions holding in it. $\mathcal{R}(s, \alpha, s')$ holds iff when executing the action α in the state s the state s' is a possible outcome. An action α is not applicable in s iff there is no state s' such that $\mathcal{R}(s, \alpha, s')$ holds. An action α has an uncertain outcome in s if there are two distinct states s' and s'' such that $\mathcal{R}(s, \alpha, s')$ and $\mathcal{R}(s, \alpha, s'')$. In the following we assume a planning domain $\mathcal{D}$ is given. We say that an action α is applicable in the set of states S if it is applicable to every state of S. The result of executing an action α in the set of states S (also called the image of S under α), written $\mathit{Exec}[\alpha](S)$, is the set of all possible outcomes of the execution of α in any state of S, i.e.

$$\mathit{Exec}[\alpha](S) \doteq \{s' \mid \mathcal{R}(s, \alpha, s') \ \mathit{with}\ s \in S\}$$

If s is a state, we write $\mathit{Exec}[\alpha](s)$ instead of $\mathit{Exec}[\alpha](\{s\})$. The *weak* preimage of a set of states S under the action α, written $\mathit{WPreImage}[\alpha](S)$, is the set of all states where the execution of α can lead to S. In symbols,

$$\mathit{WPreImage}[\alpha](S) \doteq \{s \mid \mathcal{R}(s, \alpha, s') \ \mathit{with}\ s' \in S\}$$

We call this set weak preimage to stress the fact that, for every state in it, reaching S when executing α is possible but not necessary. The *strong* preimage of a set S under the action α, written $\mathit{SPreImage}[\alpha](S)$, is the set of all states where α is applicable and every possible execution is in S. I.e.,

$$\mathit{SPreImage}[\alpha](S) \doteq \{s \mid \emptyset \neq \mathit{Exec}[\alpha](s) \subseteq S\}$$

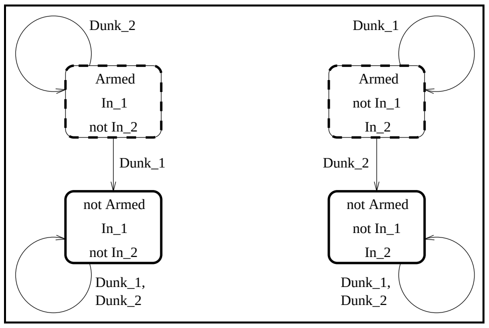

Fig. 1. The automaton for the BT domain

In this paper we consider plans to be sequences of actions. We use ϵ for the 0-length plan, α to denote an action, π and ρ to denote plans, and $\pi;\rho$ for plan concatenation. The applicability set of a plan is the set of states from which we can execute any prefix of the plan without ending up in a state where the rest of the plan is not applicable. The execution of a plan in a set of states is the set of "final" states of the possible execution traces from any of the initial states.

Definition 1 (Applicability set of a Plan). *Let π be a plan. The applicability set of π, written $Appl[\pi]$, is a subset of $\mathcal{S}$ defined as follows:*

1. $Appl[\epsilon] = \mathcal{S}$;
2. $Appl[\alpha] = \{s \mid Exec[\alpha](s) \neq \emptyset\}$;
3. $Appl[\alpha;\rho] = \{s \mid s \in Appl[\alpha],\ \text{and}\ Exec[\alpha](s) \subseteq Appl[\rho]\}$;

Definition 2 (Plan Execution). *Let S be a finite set of states. Let π be a plan for $\mathcal{D}$. The execution of π in S, written $Exec[\pi](S)$, is defined as:*

1. $Exec[\epsilon](S) = S$;
2. $Exec[\alpha](S) = \{s' \mid s \in S,\ \text{and}\ \mathcal{R}(s,\alpha,s')\}$;
3. $Exec[\alpha;\pi](S) = Exec[\pi](Exec[\alpha](S))$;

The classical example used to illustrate conformant planning is the bomb in the toilet (BT) problem. Figure 1 depicts the corresponding automaton. There are two packages, and one of them contains an armed bomb. It is possible to dunk either package in the toilet (actions $Dunk_1$ and $Dunk_2$). Dunking the package containing the bomb has the effect of disarming the bomb, while dunking the other package has no effect. Initially the bomb is armed, but there is uncertainty in the initial configuration since it is not known where the bomb is (dashed line states). We want to find a conformant solution to the problem of disarming the bomb, i.e. a sequence of actions that will disarm the bomb for all initial states. In this case, there are two possible conformant plans of length 2, namely dunking both packages in either order.

A planning probelm is a triple $(\mathcal{D}, Init, Goal)$, where $\mathcal{D}$ is the planning domain, and $Init$ and $Goal$ are nonempty sets of states of $\mathcal{D}$. In the following, when clear from the context, we omit the domain from a planning problem. A formal characterization of conformant planning can be given as follows.

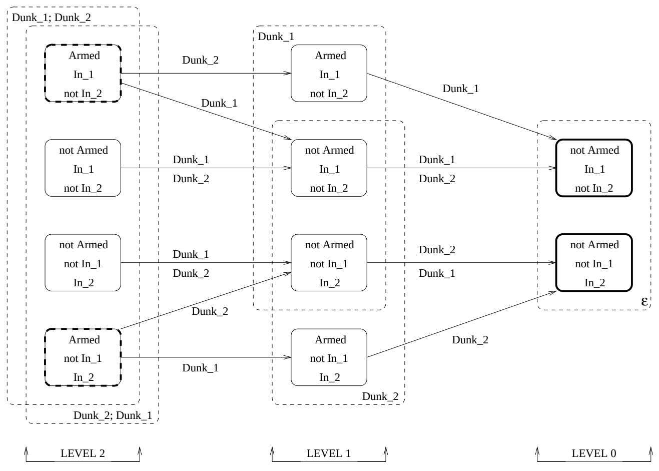

Fig. 2. Solving the BT problem

Definition 3 (Conformant Plan). *The plan π is a conformant plan for (a conformant solution to) the planning problem $(\mathcal{D}, \mathit{Init}, \mathit{Goal})$ iff $\mathit{Init} \subseteq \mathit{Appl}[\pi]$, and $\mathit{Exec}[\pi](\mathit{Init}) \subseteq \mathit{Goal}$.*

In words, a plan π is a conformant solution to a planning problem $(\mathit{Init}, \mathit{Goal})$ if two conditions are satisfied. First, π must be applicable in *Init*, i.e. after executing any prefix of π in any of the initial states, the remaining plan is always applicable. Second, all the states resulting from the execution of π in *Init* must be goal states.

3 The Conformant Planning Algorithm

The conformant planning algorithm uses as data structures states-plan (SP) tables, of the form $SPT = \{(S_1.\pi_1)\dots(S_n.\pi_n)\}$ where, for $i = 1,\dots,n$, S_i is a set of states, π_i is a sequence of actions, and $\pi_i \neq \pi_j$ for all $j \neq i$. We call $(S_i.\pi_i)$ a states-plan pair, and S_i the set of states indexed by π_i. When no ambiguity arises, we write $SPT(\pi_i)$ for S_i. The intuition is that π_i is a conformant solution for any planning problem (S, Goal), with $S \subseteq S_i$. Thus we call S_i *conformance set* of π_i in SPT.

The algorithm proceeds backwards, from the goal to the initial states. It performs a breadth first search, building at each step conformant plans of increasing length. The status of the search (a level) is represented by a SP table, containing plans of the same length. The SP tables are stored in an array, *SPTarr*, *SPTarr*$[i]$ being the SP table corresponding to the i-th level of search.

Figure 2 describes how the algorithm solves the BT problem. The goal states are depicted with a thick solid line. A SP pair is depicted as states encircled by a dashed line, annotated by the indexing plan. The SP table at level 0, *SPTarr*$[0]$, is $\{(\mathit{Goal}.\epsilon)\}$, i.e. the set of goal states indexed by the 0-length plan ϵ. (Notice that ϵ is a conformant solution to every problem with goal set *Goal* and initial states contained in *Goal*.) The SP table at level 1, *SPTarr*$[1]$, contains two SP pairs with (overlapping) sets of states indexed by the length 1 plans $Dunk_1$ and

```
function CONFORMANTPLAN(Init, Goal)
begin
    $i = 0$;
    $SPTarr[0] := \{ (Goal. \epsilon) \}$;
    $Plans = $ GETPLANS$(Init, SPTarr[0])$;
    while $((SPTarr[i] \neq \emptyset) \wedge (Plans = \emptyset))$ do
        $i := i + 1$;
        $SPTarr[i] :=$ CONFORMANTPREIMAGE$(SPTarr[i-1])$;
        $SPTarr[i] :=$ CONFORMANTPRUNE$(SPTarr, i)$;
        $Plans :=$ GETPLANS$(Init, SPTarr[i])$;
    done
    if $(SPTarr[i] = \emptyset)$ then
        return Fail;
        else return Plans;
end
```

Fig. 3. The conformant planning algorithm.

$Dunk_2$. The set indexed by $Dunk_1$, $SPTarr[1](Dunk_1)$, contains all states where $Dunk_1$ is applicable and all possible resulting states are in *Goal*. Notice that for *all* the states in this set, $Dunk_1$ leads to the goal. Thus, $Dunk_1$ is a conformant plan for every subset of $SPTarr[1](Dunk_1)$. However, neither of the SP pairs of length 1 corresponds to a conformant solution to our problem, because neither of the corresponding conformance sets contains all the initial states. Notice also that, under the hypothesis of complete observability, after one step the Strong Planning procedure presented in [8] would return a conditional plan specifying to execute only one action, i.e. dunk exactly the package containing the bomb. At level 2, $SPTarr[2]$ contains two SP pairs corresponding to the plans $Dunk_1; Dunk_2$ and $Dunk_2; Dunk_1$. Both the corresponding sets of states contain all the initial states (thick dashed line). This means that for all initial states either plan is applicable and will result in a goal state. Thus, we have found two conformant plans for the BT problem. These plans are conformant for any choice of the initial states. $SPTarr[2]$ does not contain all possible plans of length 2. $Dunk_1; Dunk_1$ and $Dunk_2; Dunk_2$ are not present. The reason is that, for each i, $SPTarr[2](Dunk_i; Dunk_i)$ would not differ from $SPTarr[1](Dunk_i)$. In other words, $Dunk_i; Dunk_i$ is subsumed by $Dunk_i$, and can be pruned. In general, if the expansion of a further level only results in no plans or plans which are subsumed by shorter plans, then the algorithm terminates concluding that the problem admits no conformant plan.

3.1 Set-theoretic view

The conformant planning algorithm CONFORMANTPLAN(*Init*,*Goal*), presented in Figure 3, takes in input a planning problem in form of the set of states *Init* and *Goal*. It returns *Fail* if and only if the problem admits no conformant solution. If a conformant plan exists, CONFORMANTPLAN(*Init*, *Goal*) returns the set of all the conformant plans of minimal length.

The algorithm proceeds as follows, by filling the array of SP tables *SPTarr*. First it checks if there are plans of length 0, i.e. if ϵ is a solution. The function

GETPLANS, given a SP table and a set of (initial) states, computes the set of all possible conformant plans contained in the SP table.

$$\textsc{GetPlans}(Init, SPT) \doteq \{\pi \mid \textit{there exists } (S.\pi) \in SPT \textit{ and } Init \subseteq S\} \quad (1)$$

If no conformant plan of length i exists ($(Plans = \emptyset)$ in line 4), then we enter the loop, and build conformant plans of increasing length (lines 5 to 8). The iteration terminates (line 4) when either a plan is found ($Plans \neq \emptyset$), or the space of conformant plans has been completely explored ($SPTarr[i] = \emptyset$).

At each iteration, the function CONFORMANTPREIMAGE is called to build a new SP table, containing conformant plans of length i, extending the conformant plans of length $i-1$ contained in $SPTarr[i-1]$.

$$\begin{aligned}&\textsc{ConformantPreimage}(SPT) \doteq \qquad (2)\\ &\quad \{(S\ .\ \alpha;\pi) \mid \textit{there exists } (S'.\pi) \in SPT, \textit{ and } S = SPreImage[\alpha](S') \neq \emptyset\}\end{aligned}$$

The resulting SP table is then stored in the i-th position of $SPTarr$. The function CONFORMANTPRUNE is responsible to remove from the newly generated SP table the plans which are either subsumed by other plans of the same length, or by plans present in the SP tables built at previous steps. It takes in input the array of SP tables $SPTarr$, and an index of the current step.

$$\begin{aligned}&\textsc{ConformantPrune}(SPTarr, i) \doteq\\ &\{(S'.\pi') \in SPTarr[i] \mid\\ &\qquad \textit{there is no } (S.\pi) \in SPTarr[i] \textit{ such that } \pi \neq \pi' \textit{ and } S' \subsetneq S, \qquad (3)\\ &\qquad \textit{and for all } j < i, \textit{ there is no } (S.\pi) \in SPTarr[j].(S' \subset S)\}\end{aligned}$$

The termination of the algorithm follows from the calls to CONFORMANTPRUNE, which guarantee that the set of explored conformance sets is monotonically increasing, and thus a fix point is eventually reached when a plan does not exist (given the finiteness of the domain). The optimality of the algorithm follows from the breadth-first style of the search.

3.2 Symbolic representation

From a conceptual point of view the algorithm of Figure 3 is rather simple. The problem is how to implement it efficiently. The basic idea, mutuated from symbolic model checking [15,3], is to represent the sets to be computed (e.g. sets of states, SP tables) symbolically, by means of propositional and quantified boolean formulae (QBF). These formulae, in turn, are represented and efficiently manipulated as BDDs. In the rest of this section we reinterpret the algorithm in terms of manipulation of propositional formulae. The issues related to BDDs are discussed in the next section.

We have a vector $\mathbf{x}$ of (distinct) boolean variables, called *state* variables, used to encode sets of states. For instance, for the BT problem, the variables in $\mathbf{x}$ could be $Armed$, In_1 and In_2. A state corresponds to a complete assignment to the variables in $\mathbf{x}$. The assignment $\{(Armed.\top)(In_1.\top)(In_2.\bot)\}$ (we write $\top$

and $\perp$ for the true and false truth values) corresponds to the state where the bomb is in package 1, and armed. We use formulae as representatives of the set of their models. Thus, a propositional formula in the variables in $\mathbf{x}$, written $\phi(\mathbf{x})$, represents the set of the states corresponding to the assignments which make ϕ true. For instance, the formula $\neg Armed$ represents the set of goal states, i.e. the states where the bomb is not armed. The formula $Armed \wedge (In_1 \leftrightarrow \neg In2)$ represents the set of initial states.

Another vector of *action* variables, $\boldsymbol{\alpha}$, is used to represent actions. For the BT problem, with a sequential encoding (i.e. assuming that only one action can be executed at each time), we can use one boolean variable Act, where the assignment $\{(Act.\top)\}$ represents the action $Dunk_1$, and the assignment $\{(Act.\perp)\}$ represents the action $Dunk_2$. A formula $\psi(\mathbf{x},\boldsymbol{\alpha})$ represents a relation between states and actions (e.g., a universal plan, or an applicability condition). The formula $Armed \wedge Act$ specifies a relation holding between action $Dunk_1$, and every state where $Armed$ holds.

Transitions are 3-tuples containing a state (the initial state of the transition), an action (the action being executed), and a state (the resulting state of the transition). To represent the final state of transitions we use an additional vector of (next) state variables $\mathbf{x'}$. The transition relation of the automaton corresponding to the planning domain is thus represented by a formula $\mathcal{R}(\mathbf{x},\boldsymbol{\alpha},\mathbf{x'})$, each satisfying assignment of which represents a particular transition.

In order to represent SP tables, we need a way to represent plans. A plan of length i is represented as an assignment to the vectors of plan variables, $\boldsymbol{\alpha_1},\ldots,\boldsymbol{\alpha_i}$, where each vector of variables $\boldsymbol{\alpha_n}$ ranges over actions, and represents the n-th action of a plan. For the BT problem, the assignment $\{(Act_1.\top)(Act_2.\perp)\}$ represents the plan $Dunk_1; Dunk_2$. The formula $\neg Act_1$ represents the set of the two plans of length 2 $Dunk_2; Dunk_1$ and $Dunk_2; Dunk_2$, since it imposes no constraint on the second action. In the following we assume that the variables in $\mathbf{x}, \mathbf{x'}, \boldsymbol{\alpha}, \boldsymbol{\alpha_1},\ldots,\boldsymbol{\alpha_i}$ are all distinct. An SP table containing plans of length i is represented by a formula in the state variables $\mathbf{x}$ and plan variables $\boldsymbol{\alpha}, \boldsymbol{\alpha_1},\ldots,\boldsymbol{\alpha_i}$.

Using a symbolic representation, we exploit the fact that if a variable v does not occur in ϕ, then it is irrelevant for the truth value of ϕ: any satisfying assignment of ϕ where the truth value of v is reversed is still a satisfying assignment. In general, the cardinality of the set represented by a given formula has a multiplying factor of two to the power of the number of variables which do not occur in the formula. This explains why a symbolic representation can have a dramatic improvement over an explicit-state (enumerative) representation.

In the following we describe in terms of propositional and QBF transformations some of the operations of the algorithm. The complete description can be found in [6]. We indicate with $\phi[\mathbf{v'}/\mathbf{v}]$ the parallel substitution (also called "shifting") in the formula ϕ of the variables in vector $\mathbf{v}$ with the (corresponding) variables in $\mathbf{v'}$. The computation of ConformantPreimage(SPT), can be described as follows (where SPT is the SP table in input, representing plans of length $i-1$):

$$\textsc{ConformantPreimage}(\mathit{SPT}) \doteq \qquad (4)$$
$$(\forall \mathbf{x}'.(\mathcal{R}(\mathbf{x},\boldsymbol{\alpha},\mathbf{x}') \rightarrow \mathit{SPT}(\mathbf{x},\boldsymbol{\alpha}_{\mathbf{i-1}},\ldots,\boldsymbol{\alpha}_{\mathbf{1}})[\mathbf{x}'/\mathbf{x}]) \quad \wedge \quad \exists \mathbf{x}'.\mathcal{R}(\mathbf{x},\boldsymbol{\alpha},\mathbf{x}'))[\boldsymbol{\alpha}_{\mathbf{i}}/\boldsymbol{\alpha}]$$

The free variables of the resulting formula are the current state variables $\mathbf{x}$ and the plan variables $\boldsymbol{\alpha}_{\mathbf{i}},\ldots,\boldsymbol{\alpha}_{\mathbf{1}}$. The action variables $\boldsymbol{\alpha}$ in $\mathcal{R}$ are renamed to plan variables $\boldsymbol{\alpha}_{\mathbf{i}}$. The next state variables in $\mathcal{R}$ and in *SPTarr* (resulting from the shifting of $\mathbf{x}$ to $\mathbf{x}'$) are universally quantified away. Each set of assignments satisfying (4) and agreeing on the values assigned to plan variables represents a relation between a set of states and a plan of length i, i.e. a SP pair.

GetPlans extracts the assignments to plan variables such that the corresponding set contains the initial states. In symbols,

$$\textsc{GetPlans}(\mathit{Init},\mathit{SPT}) \doteq \forall \mathbf{x}.(\mathit{Init}(\mathbf{x}) \rightarrow \mathit{SPT}(\mathbf{x},\boldsymbol{\alpha}_{\mathbf{i}},\ldots,\boldsymbol{\alpha}_{\mathbf{1}})) \qquad (5)$$

4 Experimental results

In this section we discuss some implementational issues, and present some results of the experimental evaluation (all the details are given in [6]). The conformant planning algorithm was implemented in MBP. MBP is based on the NuSMV model checker, is written in C, and uses the CUDD [20] state-of-the-art BDD package. MBP takes in input planning domains described in AR [10], generates the corresponding symbolic representation, and can apply different planning algorithms to the specified planning problems. In the following we call CMBP the conformant planning algorithm implemented in MBP.

The conformant planners which are most significant for comparison with CMBP are CGP [19] and QbfPlan [17]. CGP extends the ideas of GraphPlan [1] to deal with uncertainty. Basically, a planning graph is built of every possible sequence of possible worlds, and constraints among planning graphs are propagated to ensure conformance. We consider CGP the state of the art in conformant planning. CGP was shown to outperform several other planners such as Buridan [16] and UDTPOP [14] (see [19] for a detailed comparison).

QbfPlan is (our name for) the planning system by Rintanen. QbfPlan generalizes the idea of SAT-based planning [12, 13, 11] to nondeterministic domains, by encoding problems in QBF. Given a bound on the length of the plan, first a QBF encoding of the problem is generated, and then a QBF solver [18] is called. If no solution is found, a new encoding for a longer plan must be generated and solved. QbfPlan is interesting for comparison, since it relies on a symbolic representation based on QBF (although it differs from CMBP in many other ways).

Both CGP and QbfPlan are incomplete, i.e. can not conclude that a planning problem has no conformant solutions. CMBP, on the other hand, thanks to the pruning step, is complete, i.e. it can discover whether no solution exists. In the experimental evaluation, for a fair comparison, CMBP was run by disabling the pruning primitives.

	CMBP				CGP	
	|P|	#P.	|BDD|	Time	|L|	Time
BT(2)	2	2	3	0.000	1	0.000
BT(4)	4	24	37	0.000	1	0.000
BT(6)	6	720	287	0.020	1	0.010
BT(8)	8	40320	1337	0.150	1	0.020
BT(10)	10	3628800	7919	1.330	1	0.020

	CMBP				CGP	
	|P|	#P.	|BDD|	Time	|L|	Time
BTC(2)	3	2	11	0.010	3	0.000
BTC(3)	5	6	28	0.010	5	0.010
BTC(4)	7	24	102	0.010	7	0.030
BTC(5)	9	120	225	0.050	9	0.130
BTC(6)	11	720	483	0.160	11	0.860
BTC(7)	13	5040	1005	0.520	13	2.980
BTC(8)	15	40320	2773	1.850	15	13.690
BTC(9)	17	362880	5876	6.020	17	41.010
BTC(10)	19	3628800	12336	16.020	19	157.590

QbfPlan			
BTC(6)		BTC(10)	
|P|	Time	|P|	Time
1	0.00	1	0.02
2	0.01	2	0.03
3	0.26	3	0.78
4	0.63	4	2.30
5	1.53	5	4.87
6	2.82	6	8.90
7	6.80	7	22.61
8	14.06	8	52.72
9	35.59	9	156.12
10	93.34	10	410.86
11	(+) 2.48	11	1280.88
		13	3924.96
		14	—
		...	...
		18	—
		19	(+) 16.84

Table 1. Results for the BT and BTC problems.

CMBP is strictly more expressive than CGP, which can handle uncertainty only in the initial state (although [19] describes how the approach can be extended to actions with uncertain effects). The comparison with CGP was carried out only on the cases with uncertainty on the initial condition. QbfPlan is able to handle actions with uncertain effects. This is done by introducing auxiliary (choice) variables, the assignments to which correspond to the different possible outcomes of actions. These variables need to be quantified universally to ensure conformance of the solution. However, the encoding generator of QbfPlan has ML code as its input format. The comparison with QbfPlan is limited to the (few) problems for which the encodings already existed.

For CMBP and CGP, all the examples were run by setting a limit to the depth of the search. Since MBP uses a serial encoding, the limit corresponds to the maximum length of the plan. In CGP, the limit is on the number of levels in the planning graph. The chosen limit was enough to find a solution for the tested problems in both systems. Differently from e.g. BlackBox [11], QbfPlan does not have a heuristic to guess the "right" length of the plan. Given a limit in the length of the plan, it generates all the encodings up to the specified length, and repeatedly calls the QBF decider on encodings of increasing length until a plan is found. We specified as limit the length of the shortest solution. BDD based computations are known to be sensitive to a number of factors, such as the ordering of variables. For all the examples reported here, CMBP used a fixed ordering strategy: action variables were positioned at the top, then plan variables, and state variables. Variables of a given kind were interleaved with the corresponding auxiliary variables (e.g. $\mathbf{x}$ with $\mathbf{x}'$, $\boldsymbol{\alpha}_\mathbf{i}$ with $\boldsymbol{\beta}_\mathbf{i}$). Dynamic variable reordering was disabled. The tests were performed on an Intel 300MhZ Pentium-II, 512MB RAM, running Linux. CGP is implemented in LISP, and was compiled and run under Allegro CL 4.3 [Linux/X86;R1]. CPU time was limited to 7200 sec (two hours) for each test. In the following tables, unless otherwise specified, we write — for a test that was not completed within the above time limit.

The evaluation was performed by running the systems on a number of pa-

	Low Unc.							Mid Unc.				High Unc.			
BMTC		CMBP				CGP			CMBP	CGP			CMBP	CGP	
(p,t)	IS	|P|	#P.	|BDD|	Time	|L|	Time	IS	Time	|L|	Time	IS	Time	|L|	Time
(2,2)	2	2	4	15	0.000	1	0.000	4	0.000	2	0.010	8	0.000	2	0.030
(3,2)	3	4	48	70	0.010	3	0.020	6	0.010	3	0.040	12	0.020	4	13.560
(4,2)	4	6	768	268	0.040	3	0.030	8	0.060	4	0.460	16	0.090	4	145.830
(5,2)	5	8	15360	662	0.180	5	1.390	10	0.260	5	13,180	20	0.340	4	—
(6,2)	6	10	368640	1499	0.640	5	3.490	12	0.830	5	—	24	1.150		
(7,2)	7	12	1.03e7	3250	2.100	7	508.510	14	2.780			28	3.390		
(8,2)	8	14	3.30e8	8357	7.960	7	918.960	16	10.380			32	12.330		
(9,2)	9	16	1.18e10	17944	22.820	7	—	18	30.370			36	35.510		
(10,2)	10	18	4.75e11	37968	72.730			20	87.370			40	121.740		
(2,4)	2	2	24	31	0.000	1	0.000	8	0.010	1	0.020	32	0.010	2	1.610
(3,4)	3	3	144	122	0.030	1	0.010	12	0.050	2	0.290	48	0.150	2	8.690
(4,4)	4	4	576	426	0.100	1	0.010	16	0.320	2	0.730	64	0.840	2	32.190
(5,4)	5	6	57600	1985	0.680	3	0.500	20	1.610	2	—	80	3.420	3	—
(6,4)	6	8	5806080	5905	3.350	3	1.160	24	6.900			96	12.650		
(7,4)	7	10	6.58e08	14939	14.210	3	2.410	28	23.090			112	40.410		
(8,4)	8	12	8.44e10	40237	77.420	3	8.540	32	232.150			128	932.820		
(9,4)	9		—	—	—	4	—	36	—			144	—		
(10,4)	10											160			
(2,6)	2	2	60	56	0.010	1	0.010	16	0.010	1	0.200	128	0.090	2	337.604
(3,6)	3	3	720	423	0.090	1	0.010	24	0.080	1	0.830	192	1.040	2	1459.110
(4,6)	4	4	8640	1879	0.510	1	0.040	32	1.190	2	30.630	256	6.460	2	5643.450
(5,6)	5	5	86400	6137	3.080	1	0.060	40	12.260	2	30.140	320	40.770	2	—
(6,6)	6	6	518400	14265	17.490	1	0.100	48	118.600	2	57.300	384	1819.520		
(7,6)	7	8	2.03e08	67489	5939.520	3	211.720	56	—	2	—	448	—		
(8,6)	8		—	—	—	3	1015.160	64				512			
(9,6)	9					3	3051.990	72				576			
(10,6)	10					2	—	80				640			

Table 2. Results for the BMTC problems

rameterized problem domains. The first class of problems we tackled is based on the classical bomb in the toilet problem, BT(p), where p is the parametric number of packages. The results for the BT problems are shown in Table 1 (upper left). The columns relative to CMBP are the length of the plan (|P|), the number of plans (#P.), the size of the BDD representing the set of conformant solutions (|BDD|), and the run time needed for searching the automaton (expressed in seconds). The columns relative to CGP are the number of levels in the planning graphs, and the computation time needed for the search. For the BT problem CGP is almost insensitive to the problem size, and outperforms CMBP. One reason for this is that CGP inherits from GraphPlan the ability to deal with parallel actions efficiently, and the BT problem is intrinsically parallel (the depth of the planning graph is always one, i.e. all packages can be dunked in parallel).

We call BTC(p) the extension where dunking a package (always) clogs the toilet, and flushing can remove the clogging. The results for this problems are shown in Table 1. Since the BTC does not allow for parallel actions, the impact of the depth of the plan length becomes significant, and CMBP outperforms CGP. The performance of QbfPlan is reported in the rightmost table, only for the 6 and 10 package problems. Notice that each line reports the time needed to decide whether there is a plan of length i. QbfPlan is outperformed both by CGP and by CMBP. QbfPlan does not exploit the computations performed to analyze previous levels, and thus needs to restart from scratch problems of increasing length. In the rest of the comparison we do not consider QbfPlan.

	CMBP				CGP	
	|P|	#P.	|BDD|	Time	|L|	Time
RING(2)	5	2	10	0.010	3	0.070
RING(3)	8	2	23	0.030	4	—
RING(4)	11	2	35	0.060		
RING(5)	14	2	47	0.320		
RING(6)	17	2	59	1.460		
RING(7)	20	2	71	7.190		
RING(8)	23	2	83	35.380		
RING(9)	26	2	95	167.690		

CGP on RING(5)				
IS	|L|	Time	|L|	Time
1	5	0.010	9	0.020
2	5	0.060	9	0.140
4	5	0.420	9	1.950
8	5	6.150	9	359.680
16	5	—	9	—

Table 3. Results for the RING problems.

The next class of problems, called BMTC(p,t), is the generalization of the BTC problem to the case of multiple toilets. The results are reported in Table 2. (IS is the number of initial states.) In the first class of tests ("Low Uncertainty" columns), the only uncertainty is the position of the bomb, while toilets are known to be not clogged. The basic feature of the problem is that it becomes more parallelizable when the number of toilets increases. CGP is able to fully exploit this feature, while CMBP suffers because of its serial encoding. With many toilets CGP outperforms CMBP. However, the behavior of CGP degrades as soon as more than 5 levels in the planning graph need to be explored. Consider the results for the BMTC(6,2) and BMCT(7,2) problems. Notice also that CMBP finds all the 10321920 conformant solutions to BMTC(7,2) in 2.100 seconds.

The "Mid" and "High" columns show the results in presence of more uncertainty in the initial state. In the second [third, respectively] class of tests, the status of every other [every, resp.] toilet can be either clogged or non clogged. This increases the number of possible initial states. The results show that CMBP is much less sensitive to the number of initial states, CGP is almost unable to solve what were trivial problems.

We considered another class of problems, where we have a ring of rooms, each of them with a window, which can be either open, closed or locked. The robot can move (either clockwise or counterclockwise), close the window of the room where it is, and lock it if closed. The goal is to have all windows locked. In the problem RING(r), where r is the number of rooms, the position of windows obeys the law of inertia, i.e. it remains unchanged unless changed by an action of the robot. The uncertainty in the initial states can be both in the position of the robot, and in the status of the windows. The maximum number of initial states is $r * 3^r$, corresponding to full uncertainty on the position of the robot and on the status of each window. The results, in the case of maximum uncertainty, are reported in on the left in Table 3. On the right, we plot (for the RING(5) problem) the dependency of CGP on the number of initial states combined with the number of levels to be explored (different goals were provided which require the exploration of different levels).

Finally, we considered problems with full uncertainty in action effects, which can not be expressed in CGP. In the BTUC(p), clogging is an uncertain outcome of dunking a package. In the URING(r), at each time instant, each window can open or close nondeterministically if it is not locked. The results are reported in Table 4. The run times are lower than in the inertial cases, this is due to the fact that there is no need to represent the effects of the law of inertia.

	CMBP			
	|P|	#P.	|BDD|	Time
BTUC(2)	3	2	11	0.000
BTUC(3)	5	6	28	0.000
BTUC(4)	7	24	102	0.020
BTUC(5)	9	120	225	0.050
BTUC(6)	11	720	483	0.170
BTUC(7)	13	5040	1005	0.530
BTUC(8)	15	40320	2773	1.830
BTUC(9)	17	362880	5876	6.020
BTUC(10)	19	3628800	12336	17.730

	CMBP			
	|P|	#P.	|BDD|	Time
URING(2)	5	2	10	0.000
URING(3)	8	2	23	0.010
URING(4)	11	2	35	0.030
URING(5)	14	2	47	0.080
URING(6)	17	2	59	0.200
URING(7)	20	2	71	0.530
URING(8)	23	2	83	1.370
URING(9)	26	2	95	4.600
URING(10)	29	2	107	14.320

Table 4. Results for the BTUC and URING problems.

5 Conclusions and Future Work

In this paper we presented a new algorithm for conformant planning. The algorithm is applicable to complex planning domains, with conditional actions, uncertainty in the initial state and in the outcomes of actions, and nondeterministic changes in the environment. The algorithm returns the set of all conformant plans of minimal length, if a solution to the planning problem exists. Otherwise, it terminates with failure. This work relies on and extends the planning via symbolic model checking paradigm presented in [5, 8, 7, 9]. The algorithm has been designed to be implemented efficiently taking full advantage of the symbolic representation based on BDD. The experimental results show that the algorithm is able to solve rather complex problems, and compares nicely with the state of the art conformant planner CGP, and with QBFPLAN. First, CMBP is complete, i.e. it is able to decide whether a conformant plan exists. Second, CMBP is strictly more expressive than CGP, as it allows for uncertainty in the action effects. Furthermore, CGP suffers from the enumerative nature of its algorithm, and its qualitative behavior seem to depend heavily on the *number* of possible situations to be considered. The experimental evaluation suggests that CMBP is able to deal with uncertainties more efficiently than CGP.

A first direction of future activity is the investigation of parallel encodings. CMBP inherits from MBP a serial encoding, and is thus outperformed by CGP in problems with a high degree of parallelizability (e.g. when multiple toilets are available). Furthermore, optimization techniques typical of symbolic model checking, such as partitioning techniques [3], could be used to reduce the computational cost of relational products and pruning. We have also developed another algorithm for conformant planning, based on a forward (rather than backward) traversal of the state space. Another direction of future research includes its experimental evaluation, and its integration with the backward algorithm presented in this paper. Finally, conformant planning via model checking will be extended to deal with the general case of planning under partial observability.

References

1. Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence 1–2*, 90:279–298, 1997.
2. R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

3. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2):142–170, June 1992.
4. A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 495–499, Trento, Italy, July 1999. Springer.
5. A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via Model Checking: A Decision Procedure for $\mathcal{AR}$. In S. Steel and R. Alami, editors, *Proceeding of the Fourth European Conference on Planning*, number 1348 in LNAI, pages 130–142, Toulouse, France, September 1997. Springer-Verlag.
6. A. Cimatti and M. Roveri. Conformant Planning via Model Checking. Technical Report 9908-02, ITC-IRST, Trento, Italy, August 1999.
7. A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proceeding of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin, 1998. AAAI-Press.
8. A. Cimatti, M. Roveri, and P. Traverso. Strong Planning in Non-Deterministic Domains via Model Checking. In *Proceeding of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, Carnegie Mellon University, Pittsburgh, USA, June 1998. AAAI-Press.
9. M. Daniele, P. Traverso, and M. Y. Vardi. Strong Cyclic Planning Revisited. In Susanne Biundo, editor, *Proceeding of the Fifth European Conference on Planning*. Durham, UK, September 1999. Springer-Verlag.
10. E. Giunchiglia, G. N. Kartha, and V. Lifschitz. Representing action: Indeterminacy and ramifications. *Artificial Intelligence*, 95(2):409–438, 1997.
11. H. Kautz and B. Selman. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. In *Working notes of the Workshop on Planning as Combinatorial Search*, Pittsburgh, PA, USA, June 1998.
12. Henry A. Kautz, David McAllester, and Bart Selman. Encoding Plans in Propositional Logic. In *Proc. KR-96*, 1996.
13. Henry A. Kautz and Bart Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proc. AAAI-96*, 1996.
14. Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, September 1995.
15. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publ., 1993.
16. M. Peot. *Decision-Theoretic Planning*. PhD thesis, Dept. Engineering-Economic Systems — Stanford University, 1998.
17. J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intellegence Research*, 1999. Accepted for publication.
18. J. Rintanen. Improvements to the Evaluation of Quantified Boolean Formulae. In *16th Iinternational Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, August 1999. To appear.
19. David E. Smith and Daniel S. Weld. Conformant graphplan. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 889–896, Menlo Park, July 26–30 1998. AAAI Press.
20. F. Somenzi. CUDD: CU Decision Diagram package — release 2.1.2. Department of Electrical and Computer Engineering — University of Colorado at Boulder, April 1997.