Symbolic Techniques for Planning with Extended Goals in Non-Deterministic Domains

Marco Pistore, Renato Bettin, and Paolo Traverso

ITC-IRST
Via Sommarive 18, 38050 Povo, Trento, Italy
{pistore,bettin,traverso}@irst.itc.it

Abstract. Several real world applications require planners that deal with non-deterministic domains and with temporally extended goals. Recent research is addressing this planning problem. However, the ability of dealing in practice with large state spaces is still an open problem. In this paper we describe a planning algorithm for extended goals that makes use of BDD-based symbolic model checking techniques. We implement the algorithm in the MBP planner, evaluate its applicability experimentally, and compare it with existing tools and algorithms. The results show that, in spite of the difficulty of the problem, MBP deals in practice with domains of large size and with goals of a certain complexity.

1 Introduction

Research in classical planning has focused on the problem of providing algorithms that can deal with large state spaces. However, in some application domains (like robotics, control, and space applications) classical planners are difficult to apply, due to restrictive assumptions on the planning problem they are designed (and highly customized) for. Restrictive assumptions that result from practical experiences are, among others, the hypotheses about the determinism of the planning domain and the fact that goals are sets of final desired states (reachability goals). Several recent works address either the problem of planning for reachability goals in *non-deterministic domains* (see for instance [7, 15, 3, 10]), or the problem of planning for *temporally extended goals* that define conditions on the whole execution paths (see for instance [8, 1]). Very few works in planning relax both the restrictions on deterministic domains and on reachability goals, see, e.g., [11, 14]. These works show that planning for temporally extended goals in non-deterministic domains is theoretically feasible in a rather general framework. However, they leave open the problem of dealing in practice with the large state spaces that are a characteristic of most real world applications. Indeed, the combination of the two aspects of non-determinism and temporally extended goals makes the problem of plan generation significantly more difficult than considering one of the two aspects separately. From the one side, planning for temporally extended goals requires general synthesis algorithms that cannot be customized and optimized to the special case of reachability goals. From the other side, compared to planning for extended goals in deterministic domains, the planner has to take into account the fact that temporal properties must be checked on all the execution paths that result from the non-deterministic outcomes of actions. These two factors make *practical* planning for extended goals in *large* non-deterministic domains an open and challenging problem.

In this paper we address this problem. The starting point is the work presented in [14], which provides a theoretical framework for planning in non-deterministic domains. In [14], goals are formulas in the CTL temporal logic [9]. CTL provides the ability to express goals that take into account the fact that a plan may non-deterministically

result in many possible different executions and that some requirements can be enforced on all the possible executions, while others may be enforced only on some executions. In order to show that planning for CTL goals is feasible theoretically, [14] presents a planning algorithm that searches through an explicit representation of the state-space. This however limits its applicability to trivial examples. In this paper we describe in detail a novel planning algorithm based on symbolic model checking techniques, and evaluate it experimentally. The algorithm is a major departure from that presented in [14], both theoretically and practically. Theoretically, while [14] is an explicit-state depth-first forward search, the algorithm presented here is formulated directly by using symbolic model checking techniques. Starting from the goal formula, it builds an automaton that is then used to control the symbolic search on sets of states. From the practical point of view, the algorithm opens up the possibility to scale-up to large state spaces. We implement the algorithm in the MBP planner [2], and provide an extensive experimental evaluation. The experiments show that planning in such a general setting can still be done in practice in domains of significant dimensions, e.g., domains with more than 10^8 states, and with goals of a certain complexity. We compare MBP with SIMPLAN [11], a planner based on explicit-state search, and show the significant benefits of planning based on symbolic model checking w.r.t. explicit-state techniques. We also compare the algorithm for extended goals with the MBP algorithms presented in [6, 7], that have been customized to deal with reachability goals. The general algorithm for extended goals introduces a rather low overhead, in spite of the fact that it deals with a much more general problem.

The paper is structured as follows. Section 2 presents the basic definitions on planning for extended goals in non-deterministic domains. The core of the paper are Sections 3 and 4. The former presents the planning algorithm, while the latter describes its implementation in the MBP planner and shows the experimental evaluation. Finally, Section 5 draws some conclusions and discusses related work.

2 Non-Deterministic Domains, Extended Goals and Plans

In this section we recall briefly the basic definitions for planning with extended goals in non-deterministic domains. See [14] for examples and further explanations.

Definition 1. *A* (*non-deterministic*) *planning domain* $\mathcal{D}$ *is a tuple* $(\mathcal{B}, \mathcal{Q}, \mathcal{A}, \rightarrow)$, *where* $\mathcal{B}$ *is the finite set of* (*basic*) *propositions,* $\mathcal{Q} \subseteq 2^{\mathcal{B}}$ *is the set of states,* $\mathcal{A}$ *is the finite set of actions, and* $\rightarrow \subseteq \mathcal{Q} \times \mathcal{A} \times \mathcal{Q}$ *is the transition relation. We write* $q \xrightarrow{a} q'$ *for* $(q, a, q') \in \rightarrow$.

The transition relation describes how an action leads from one state to possibly many different states. We require that relation $\rightarrow$ is total, i.e., for every $q \in \mathcal{Q}$ there is some $a \in \mathcal{A}$ and $q' \in \mathcal{Q}$ such that $q \xrightarrow{a} q'$. We denote with $\mathrm{Act}(q) \triangleq \{a : \exists q'.\ q \xrightarrow{a} q'\}$ the set of the actions that can be performed in state q, and with $\mathrm{Exec}(q, a) \triangleq \{q' : q \xrightarrow{a} q'\}$ the set of the states that can be reached from q performing action $a \in \mathrm{Act}(q)$.

Definition 2. *Let* $\mathcal{B}$ *be the set of basic propositions of a domain* $\mathcal{D}$ *and let* $b \in \mathcal{B}$. *The syntax of an* (*extended*) *goal* g *for* $\mathcal{D}$ *is the following:*
$$g ::= \top \mid \perp \mid b \mid \neg b \mid g \wedge g \mid g \vee g \mid \mathrm{AX}\,g \mid \mathrm{EX}\,g \mid$$
$$\mathrm{A}(g\,\mathrm{U}\,g) \mid \mathrm{E}(g\,\mathrm{U}\,g) \mid \mathrm{A}(g\,\mathrm{W}\,g) \mid \mathrm{E}(g\,\mathrm{W}\,g).$$
We define the following abbreviations: $\mathrm{AF}\,g = \mathrm{A}(\top\,\mathrm{U}\,g)$, $\mathrm{EF}\,g = \mathrm{E}(\top\,\mathrm{U}\,g)$, $\mathrm{AG} = \mathrm{A}(g\,\mathrm{W}\,\perp)$, *and* $\mathrm{EF} = \mathrm{E}(g\,\mathrm{W}\,\perp)$.

Extended goals are expressed with CTL formulas. CTL allows us to express goals that take into account non-determinism. For instance, it is possible to express the different forms of reachability goals considered in [6, 7]: the goal $\mathrm{EF}\, b$ requires plans to have a chance of reaching a set of final desired states where b holds ("weak" planning), $\mathrm{AF}\, b$ requires plans that are guaranteed to achieve the goal ("strong" planning), and $\mathrm{A}(\mathrm{EF}\, b\, \mathrm{W}\, b)$ requires plans that try to achieve the goal with iterative trial-and-error strategies ("strong-cyclic" planning). We can express also different kinds of maintainability goals, e.g., $\mathrm{AG}\, g$ ("maintain g"), $\mathrm{AG}\, \neg g$ ("avoid g"), $\mathrm{EG}\, g$ ("try to maintain g"), $\mathrm{EG}\, \neg g$ ("try to avoid g"), and $\mathrm{AG}\, \mathrm{EF}\, g$ ("always maintain a possibility to reach g"). Moreover, reachability and maintainability requirements can be combined, like in the cases of $\mathrm{AF}\, \mathrm{AG}\, g$ ("reach a set of states where g can be maintained"). See [14] for a larger set of examples of extended goals that can be expressed in CTL.

In order to satisfy extended goals, we need to consider plans that are strictly more expressive than plans that simply map states of the world to actions to be executed, like universal plans [16], memory-less policies [3], and state-action tables [6, 7]. In the case of temporally extended goals, actions to be executed may also depend on the "internal state" of the executor, which can take into account, e.g., previous execution steps. More precisely, a plan can be defined in terms of an *action function* that, given a state and an *execution context* encoding the internal state of the executor, specifies the action to be executed, and in terms of a *context function* that, depending on the action outcome, specifies the next execution context.

Definition 3. *A* plan *for a domain $\mathcal{D}$ is a tuple $\pi = (C, c_0, act, ctxt)$, where C is a set of (execution) contexts, $c_0 \in C$ is the initial context, $act : \mathcal{Q} \times C \rightharpoonup \mathcal{A}$ is the action function, and $ctxt : \mathcal{Q} \times C \times \mathcal{Q} \rightharpoonup C$ is the context function.*

If we are in state q and in execution context c, then $act(q, c)$ returns the action to be executed by the plan, while $ctxt(q, c, q')$ associates to each reached state q' the new execution context. Functions *act* and *ctxt* may be partial, since some state-context pairs are never reached in the execution of the plan.

The execution of a plan results in a change in the current state and in the current context. It can therefore be described in terms of transitions between state-context pairs, like $(q, c) \xrightarrow{a} (q', c')$. Due to the non-determinism of the domain, a plan may lead to an infinite number of different executions. In [14] a finite presentation of all possible executions is obtained by defining an *execution structure*. It is a Kripke structure [9] whose set of states is the set state-context pairs, and whose transitions are the transitions of the plan. According to [14], a plan satisfies a goal g if CTL formula g holds on the execution structure corresponding to the plan.

3   The Symbolic Planning Algorithm

A planning problem requires to build a plan that satisfies the goal and is compatible with the given domain (i.e., it is executable). The algorithm we propose works by building an automaton, called the *control automaton* that is used to guide the search of a plan. The states of the control automaton are the contexts of the plan that is being built, and the transitions represent the possible evolutions of the contexts when actions are executed. Control automata are strictly related to the tree automata proposed in [12] as the basic structure for performing CTL synthesis. The outline of the symbolic planning algorithm is the following:

```
function symbolic-plan(g_0) : Plan
    aut := build-aut(g_0)
    assoc := build-assoc(aut)
    plan := extract-plan(aut, assoc)
    return plan
```

It works in three main steps. In the first step, *build-aut* constructs the control automaton for the given goal. In the second step, *build-assoc* exploits the control automaton to guide the symbolic exploration of the domain, and associates a set of states in the planning domain to each state in the control automaton. Intuitively, these are the states for which a plan exists from the given context. In the third step, *extract-plan* constructs a plan by exploiting the information on the states associated to the contexts.

Control automata. Control automata are the key element for the definition of the planning algorithm.

Definition 4. *A control automaton is a tuple $A = (C, c_0, T, R)$, where:*
- *C is the set of control states (or contexts), and $c_0 \in C$ is the initial control state.*
- *$T : C \rightarrow \mathcal{P}(Prop(\mathcal{B}) \times \mathcal{P}(C) \times C)$ is the transition function, where $Prop(\mathcal{B})$ is any propositional formula constructed from basic propositions $b \in \mathcal{B}$.*
- *$R = \{B_1, \dots, B_n\}$, with $B_i \subseteq C$, is the set of the* red blocks *of the automaton.*

The transitions in $T(c)$ describe the different valid evolutions from context c. For each $(P, Es, A) \in T(c)$, component P constrains the states where the transitions is applicable, while components Es and A describe the contexts that must hold in the next states, according to the transition. More precisely, each context $E \in Es$ must hold for "some" of the next states, while A defines the context that must hold for "all the other" next states. The distinction between contexts Es and context A is necessary in the case of non-deterministic domain, since it permits to distinguish between behaviors that the plan should enforce on all next states, or only on some of them.

Component R defines conditions on the valid infinite executions of a plan. These coincide with the so called "acceptance conditions" of automata theory [13]. Acceptance conditions are necessary to distinguish the control states of a plan where the execution can persist forever from the control states that should be left eventually in order to allow for a progress in the fulfillment of the goal. More precisely, a given red block $B \in R$ is used to represent all the control states in which the execution is trying to reach or achieve a given condition. If an execution of a plan persists inside B, then the condition is never reached, the execution is not accepted and the plan is not valid. If a control state does not appear in any red block, then it corresponds to a situation where only safety or maintainability goals have to be fulfilled, so no progress is required.

Construction of the control automata. We now define how the control automaton is constructed from the goal.

Definition 5. *The control automaton $A = build\text{-}aut(g_0)$ is built according to rules:*
- *$c_0 = [g_0] \in C$.*

- *If $[g_1, \dots, g_n] \in C$ then, for each $(P, EX, AX) \in progr(g_1 \wedge \cdots \wedge g_n)$ and for each partition $\{EX_1, \dots, EX_n\}$ of EX:*

    $(P , \{order\text{-}goals(AX \cup EX_i) : i = 1..n\} , order\text{-}goals(AX)) \in T(c)$.

    Moreover, order-goals$(AX \cup EX_i) \in C$ for $i = 1..n$ and order-goals$(AX) \in C$.

- *For each strong until subgoal g of g_0, let $B_g = \{c \in C : \textbf{head}(c) = g\}$; if $B_g \neq \emptyset$, then $B_g \in R$.*

Each state of the control automaton corresponds to an ordered list of subgoals, representing those subgoals that the executor should currently achieve. The order of the subgoals represents their priorities. Indeed, there are situations in which it is necessary to distinguish control states that correspond to the same subgoals according to their priorities. Consider for instance the case of goal $g_{AG} = AG(AF\,p \land AF\,q)$, that requires to keep achieving both condition p and condition q. Two different contexts generated in the construction of the control automaton for this goal are $[AF\,p, AF\,q, g_{AG}]$ and $[AF\,q, AF\,p, g_{AG}]$. They have the same subgoals, but the first one gives priority to goal $AF\,p$, while the second one gives priority to goal $AF\,q$. By switching between the two contexts, the plan guarantees that both the conditions p and q are achieved infinitely often. In general, the first goal **head**(c) in a context c is the goal with the highest priority, and it is the goal that the planning algorithm is trying to achieve first.

In order to be able to define the priority of the subgoals we need to distinguish three categories of formulas: the *strong until* goals ($A(\_U\_)$ and $E(\_U\_)$), the *weak until* goals ($A(\_W\_)$ and $E(\_W\_)$), and the *transient* goals ($AX\_$, $EX\_$, $\_\lor\_$, and $\_\land\_$). A transient goal is "resolved" in one step, independently from their priority: prefixes AX and EX, for instance, express conditions only on the next execution step. Weak until goals are allowed to hold forever, without being resolved. Therefore, we assign a low priority to transient and weak until goals. Strong until goals must be instead eventually resolved for the plan to be valid. We assign a high priority to these goals, and, among the strong until goals, we give priority to the goals that are active since more steps. Namely, the longer a strong until goal stays active and unresolved, the "more urgent" the goal becomes. In Definition 5 the ordering of subgoals sg is performed by function *order-goals*(sg, c). The input context c represents the list of the subgoals in the old context; it is necessary to determine the priority among the strong until goals that are already active in the old context.

One of the key steps in the construction of the transition function of a control automaton is the function *progr*. It associates to each goal g the conditions that g defines on the current state and on the next states to be reached, according to the CTL semantics. This is obtained by unrolling the weak and strong until operators, as shown by the following rules:

- *progr*$(A(g\,U\,g')) = (progr(g) \land AX\,A(g\,U\,g')) \lor progr(g')$ and
  progr$(E(g\,U\,g')) = (progr(g) \land EX\,E(g\,U\,g')) \lor progr(g')$;
- *progr*$(A(g\,W\,g')) = (progr(g) \land AX\,A(g\,W\,g')) \lor progr(g')$ and
  progr$(E(g\,W\,g')) = (progr(g) \land EX\,E(g\,W\,g')) \lor progr(g')$.

Function *progr* commutes with the other operators: e.g., *progr*$(g \land g') = progr(g) \land progr(g')$. For instance, *progr*$(AG\,p) = p \land AX\,AG\,p$, *progr*$(EF\,q) = q \lor EX\,EF\,p$, and *progr*$(AG\,p \land EF\,q) = (p \land q \land AX\,AG\,p) \lor (p \land AX\,AG\,p \land EX\,EF\,q)$. We can assume that formula *progr*(g) is written in disjunctive normal form, as in the examples above. Each disjunct corresponds to an alternative evolution of the domain, i.e., to alternative plans we can search for. Each disjunct consists of the conjunction of three kinds of formulas, the propositional ones b and $\neg b$, those of the form $EX\,f$, and those of the form $AX\,h$. In the algorithm, we make this structure explicit and represent *progr*(g) as a set of triples

$$progr(g) = \{(P_i, EX_i, AX_i) \mid i \in I\}.$$

where $p \in P_i$ are the propositional formulas of the i-th disjunct of *progr*(g), and $f \in EX_i$ ($h \in AX_i$) if $EX\,f$ ($AX\,h$) belongs to the i-th disjunct.

In the case the component EX of a disjunct (P, EX, AX) contains more than one subgoal, the generation of the control automaton has to take into account that there are different ways to distribute the subgoals in EX to the set of next states. For instance, if set EX contains two subgoals, then we can require that both the subgoals hold in the same next state, or that they hold in two distinct next states. In the general case, any partition $EX_1, \ldots, EX_n$ of the subgoals in EX corresponds to a possible way to associate the goals to the next states. Namely, for each $i = 1..n$, there must be some next state where subgoals $AX \cup EX_i$ hold. In all the other states, subgoals in AX must hold.

A plan for a given goal is not valid if it allows for executions where a strong until goals becomes eventually active and is then never resolved. In order to represent these undesired behaviors, the construction of the automaton generates a red block B_g for each set of contexts that share the same "higher-priority" strong until goal g.

Associating states to contexts. Once the control automaton for a goal g_0 is built, the planning algorithm proceeds by associating to each context in the automaton a set of states in the planning domain. The association is built by function *build-assoc*:

```
1  function build-assoc(aut) : Assoc
2      foreach c ∈ aut.C do assoc[c] := Q
3      green-block := {c ∈ C : ∀B ∈ aut.R . c ∉ B}
4      blocks := aut.R ∪ {green-block}
5      while (∃B ∈ blocks . need-refinement(B)) do
6          if B ∈ aut.R then foreach c ∈ B do assoc[c] := ∅
7          while (∃c ∈ B . need-update(c)) do
8              assoc[c] := update-ctxt(aut, assoc, c)
9      return assoc
```

The algorithm starts with an optimistic association, that assigns all the states $\mathcal{Q}$ in the domain to each context (line 2). The association is then iteratively refined. At every iteration of the loop (lines 5-8), a block of contexts is chosen, and the corresponding associations are updated. Those states are removed from the association, from which the algorithm discovers that the goals in the context are not satisfiable. The algorithm terminates when a fixpoint is reached, that is, whenever no further refinement of the association is possible: in this case, function *need-refinement(B)* at line 5 evaluates to false for each $B \in blocks$ and the guard of the **while** fails. The chosen block of contexts may be either one of the red blocks, or the block of states that are not in any red block (this is the "green" block of the automaton). In the case of the green block, the refinement step must guarantee only that all the states associated to the contexts are "safe": that is, they never lead to contexts where the goal cannot be achieved anymore. This refinement (lines 7-8) is obtained by choosing a context in the green block and by "refreshing" the corresponding set of states (function *update-ctxt*). Once the fixpoint is reached and all the refresh steps on the states in B do not change the association (i.e., no context in B needs updates), the loop at lines 7-8 is left, and another block is chosen. In the case of a red block, not only does the refinement guarantee that the states in the association are "safe", but also that the contexts in the red block are eventually left. Indeed, as we have seen, executions that persist forever in the control states of a red block are not valid. To this purpose, the sets of states associated to the red-block are initially emptied (line 6). Then, iteratively, one of the control states in the red-block is chosen, and its association is updated (lines 7-8). In this way, a least fixpoint is computed for the states associated to the red block.

The core step of *build-assoc* is function *update-ctxt*($aut, assoc, c$). It takes as input the automaton, $aut = (C, c_0, T, R)$, the current association of states *assoc* and a context $c \in C$, and returns the new set of states to be associated to c.

$$update\text{-}ctxt(aut, assoc, c) \triangleq \{q \in \mathcal{Q} :$$
$$\exists a \in \mathcal{A}, \exists (P, Es, A) \in T(c)$$
$$q \in states\text{-}of(P) \wedge$$
$$(q, a) \in strong\text{-}pre\text{-}image(assoc[A]) \wedge$$
$$(q, a) \in multi\text{-}weak\text{-}pre\text{-}image(\{assoc[E] : E \in Es\})\}.$$

For a state to be associated to the context, the next states corresponding to the execution of some action $a \in \mathcal{A}$ should satisfy the transition conditions of the automaton. Let us consider an action a and an element $(P, Es, A) \in T(c)$. Formula P describes conditions on the current states. Only those states that satisfy property P are valid (condition $q \in states\text{-}of(P)$). A is the context that should be reached for "all the other" next states, i.e., for all the next states not associated with any context in Es. Since all the contexts in Es contain a superset of the goals in context A, we check, without loss of generality, that *all* the next states are valid for context A. In order to satisfy this constraint, function *strong-pre-image* is exploited on the set *assoc*$[A]$ of states that are associated to context A. Function *strong-pre-image*(Q) returns the state-action pairs that guarantee to reach states in Q:

$$strong\text{-}pre\text{-}image(Q) \triangleq \{(q, a) : a \in \mathrm{Act}(q) \wedge \mathrm{Exec}(q, a) \subseteq Q\}.$$

Set Es contains the contexts that must be reached for some next states. To satisfy this constraint, function *multi-weak-pre-image* is called on the set $\{assoc[E] : E \in Es\}$ whose elements are the sets of states that are associated to the contexts in Es. Function *multi-weak-pre-image* returns the state-action pairs that guarantee to cover all the sets of states received in input:

$$multi\text{-}weak\text{-}pre\text{-}image(Qs) \triangleq$$
$$\{(q, a) : a \in \mathrm{Act}(q) \wedge \exists i : Qs \mapsto \mathrm{Exec}(q, a) . \forall Q \in Qs . i(Q) \in Q\}.$$

This function can be seen as a generalization of function *weak-pre-image*(Q), that computes the state-action pairs that may lead to a state in Q: *weak-pre-image*(Q) $= \{(q, a) : \mathrm{Exec}(q, a) \cap Q \neq \emptyset\}$. Indeed, in function *multi-weak-pre-image*(Qs) an injective map is required to exist from the Qs to the next states obtained by the execution of the state-action pair. This map guarantees that there is at least one next state in each set of states is Qs. We remark that function *update-ctxt* is the critical step of the algorithm, in terms of performance. Indeed, this is the step where the domain is explored to compute pre-images of sets of states. BDD-based symbolic techniques [4] are exploited in this step to obtain a compact representation of the sets of states associated to the contexts, and to allow for an efficient exploration of the domain.

Extracting the plan Once the association *assoc* from contexts to sets of states is built for automaton *aut*, a plan can be easily obtained. The set of contexts for the plan coincides with the set of contexts of the control automaton *aut*. The information necessary to define functions *act* and *ctxt* is implicitly computed during the execution of the function *build-assoc*. Indeed, functions *update-ctxt* and *multi-weak-pre-image* determine, respectively, the action $act(q, c)$ to be performed from a given state q in a given context c, and the next execution context $ctxt(q, c, q')$ for any possible next state q'. A plan can thus be obtained from a given assignment by executing one more step of the refinement function and by collecting these information.

4 Experimental Evaluation

We have implemented the planning algorithm inside the MBP planner. MBP [2] is built on top of a state-of-the-art symbolic model checker, NuSMV [5]. Further information on MBP can be found at `http://sra.itc.it/tools/mbp/`.

The experimental evaluation is designed to test the scalability of the approach, both in terms of the size of the domain and in terms of the complexity of the goal. In the experiments we also draw a comparison with planning algorithms for extended goals based on an explicit-state exploration of the domain, and in particular with SimPlan. SimPlan [11] implements different approaches to planning in non-deterministic domains. We focus on the logic-based planning component, where extended goals can be expressed in (an extension of) Linear Temporal Logic (LTL). LTL formulas can be used in SimPlan to describe user-defined, domain- and goal-dependent control strategies, that can provide aggressive pruning of the search space. In the experiments we test the performance of SimPlan with and without strategies. Another important comparison term are the algorithms provided by MBP for the specific case of reachability goals [6, 7]. Some of the experiments are designed to evaluate the overhead of the general algorithm for extended goals w.r.t. the optimized algorithms.

We consider the "robot delivery" planning domain, first described in [11]. This domain describes a building, composed by 8 rooms connected by 7 doors. A robot can move from room to room, picking up and putting down objects. Some rooms in the domain may be designed as "producer" and as "consumer" rooms: an object of a certain type can disappear if positioned in the corresponding consumer room, and can then reappear in one of the producer rooms. Furthermore, in order to add non-determinism to the system, some of the doors may be designed to close without intervention of the robot: they are called "kid-doors" in [11].

The experiments have been performed on a Pentium III 700 MHz with 4 Gb RAM of memory running Linux. The time limit was set to 1 hour (3600 seconds). All the experiments have been run on 5 random instances. In the tables, we report the average time required to complete. In the case of MBP, the reported times include also the pre-processing time necessary in MBP to build the symbolic representation of the planning domain. In the case only some of the instances terminate in the time limit, we report the average on the instances that terminate and the number t of terminated instances as $[t/5]$. If all the instances of an experiment do not terminate, the corresponding cell is left empty.

The first two experiments coincide with the experiments proposed in [11]. **Experiment 1** consists in moving objects into given rooms and then maintain them there. No producer and consumer rooms are present in this experiment. We fix the number of objects present in the domain to 5. The number n of objects to be moved ranges from 1 to 5, while the number k of kid-doors ranges from 0 to 7. The CTL goal is the following:

$$\text{AF AG } (in(obj_1, room_1) \land \cdots \land in(obj_n, room_n)).$$

Experiment 2 consists in reactively delivering produced objects to the corresponding consumer room. The number p of producer and consumer rooms ranges 1 to 4, while the number k of kid-doors ranges form 0 to 7. The CTL goal is the following:

$$\text{AG } \Big(\bigwedge_{i=1..p} in(obj_i, prod_i) \to \text{AF } (in(obj_i, cons_i)) \Big)$$

The results of these two experiments are reported in Tables 1 and 2 for MBP, for SimPlan with control strategies (SimPlan with CS), and for SimPlan without control formulas (SimPlan w/o CS). MBP and SimPlan exhibit complementary behaviors

	MBP					SIMPLAN with CS					SIMPLAN w/o CS		
	n = 1	n = 2	n = 3	n = 4	n = 5	n = 1	n = 2	n = 3	n = 4	n = 5	n = 1	n = 2	n = 3
k = 0	0.7	3.4	22.1	143.9	1094.6	0.5	0.7	1.2	1.6	1.7	311.9	1145.8 [1/5]	-
k = 1	0.7	4.5	33.7	195.3	1219.6	1.0	1.9	6.3	4.8	9.1	106.5 [2/5]	0.5 [1/5]	-
k = 2	0.8	5.0	38.9	275.1	1648.2	8.4	11.4	11.5	116.7	128.6 [4/5]	-	-	-
k = 3	0.8	6.4	41.2	276.9	2163.2 [2/5]	16.0	40.1 [4/5]	378.9 [3/5]	727.0 [3/5]	-	-	-	-
k = 4	1.0	5.6	45.7	336.9	2185.3 [3/5]	22.2	1478.5 [3/5]	275.7 [2/5]	-	-	-	-	-
k = 5	1.2	7.8	43.4	350.2	1866.1 [3/5]	680.5	352.8 [2/5]	420.1 [1/5]	-	-	-	-	-
k = 6	1.2	8.8	52.1	426.2	2505.1 [3/5]	1143.2 [4/5]	-	-	-	-	-	-	-
k = 7	1.4	9.4	42.7	303.3	2886.1 [2/5]	-	-	-	-	-	-	-	-

Table 1. Results of Experiment 1.

	MBP				SIMPLAN with CS				SIMPLAN w/o CS			
	p = 1	p = 2	p = 3	p = 4	p = 1	p = 2	p = 3	p = 4	p = 1	p = 2	p = 3	p = 4
k = 0	0.0	6.4	124.6	2053.6	0.3	2.4	28.7	303.6	33.2 [4/5]	721.1 [1/5]	-	-
k = 1	0.0	6.1	137.6	2426.9	0.8	15.1	360.2	309.8 [3/5]	9.2 [2/5]	17.8 [1/5]	-	-
k = 2	0.0	6.3	112.5	2684.1	15.0	63.0 [4/5]	918.0 [1/5]	-	-	-	-	-
k = 3	0.0	5.6	123.1	2063.1	245.7	3289.8 [1/5]	-	-	-	-	-	-
k = 4	0.1	12.8	130.6	2325.5	12.2 [3/5]	-	-	-	-	-	-	-
k = 5	0.1	11.3	140.6	2944.9	1386.9 [1/5]	-	-	-	-	-	-	-
k = 6	0.1	7.9	130.3	2940.2 [3/5]	1104.6 [1/5]	-	-	-	-	-	-	-
k = 7	0.1	12.4	140.8	2703.2	1.8 [1/5]	-	-	-	-	-	-	-

Table 2. Results of Experiment 2.

in these tests. The performance of MBP is left rather unaffected when kid-doors are added, but the time required to find a plan grows exponentially in the number n of objects to be moved and in the number p of producer rooms. SIMPLAN with control strategies scales linearly with respect to the number n of objects, and behaves better than MBP also when the number p of producer rooms grows. SIMPLAN, however, suffers remarkably when kid-doors are added to the domain.

Some remarks are in order on the different behaviors of the two systems in the first experiment in the case the number n of objects to be moved grows. The number of steps that the robot must perform grows linearly in the number of objects, while the number of interesting states of the planning domain grows exponentially. MBP searches for a plan for all the states in the domain. This explains its exponential grow. The search control strategies in SIMPLAN, instead, prune most of the search space, at least in the case no kid-doors are present. The plan is therefore built in linear time w.r.t. its length. When the search control strategies are disallowed in SIMPLAN, a larger portion of the state space should be explored, and the performance becomes much worse. Indeed, plans are found in the time limit only for very small values of parameters k, n, and p.

Experiment 3 is designed to compare the performance of the general extended-goals planning algorithm of MBP with the optimized algorithms provided by MBP for reachability goals. In this case, the robot is required to reach a state where a goal condition is satisfied. The goal conditions have the following form:

$$p = in(obj_1, room_1) \wedge \cdots \wedge in(obj_n, room_n).$$

	k = 0					k = 7				
	n = 1	n = 2	n = 3	n = 4	n = 5	n = 1	n = 2	n = 3	n = 4	n = 5
AF p	0.7	3.8	26.9	160.2	1316.3	0.3	0.3	0.4	0.4	0.4
Strong	0.4	2.3	16.1	139.5	766.8	0.3	0.3	0.3	0.4	0.4
SIMPLAN with CS	0.4	0.7	1.2	1.6	2.1	-	-	-	-	-
SIMPLAN w/o CS	-	-	-	-	-	-	-	-	-	-
A(EF p W p)	1.3	9.9	46.1	601.6	2547.3	2.3	12.9	75.0	589.8	-
SC-Global	0.6	3.6	26.0	266.6	1253.3	1.1	5.0	39.9	238.6	1880.6
SC-Local	0.2	1.3	9.3	111.1	615.1	2.1	15.3	152.8	1525.8	-

Table 3. Results of Experiment 3.

We consider three optimized algorithms for reachability goals. The first algorithm tries to build *Strong* plans, i.e., plans that reach condition p despite non-determinism. It corresponds to temporally extended goal AF p. The other algorithms try to build *Strong-Cyclic* plans by exploiting two different approaches, the *Global* and the *Local* approach described in [6]. We recall from Section 2 that a strong-cyclic plan defines a trial-and-error strategy, corresponding to the temporally extended goal A(EF p W p). Strong-cyclic plans cannot be expressed in SIMPLAN: indeed, SIMPLAN is not able to express those goals that require a combination of universal and existential path quantifiers.

Strong-cyclic plans are interesting in the cases where strong plans do not exist due to the non-determinism in the domain. In order to allow for such situations, in this experiment we use a variant of the robot delivery domain, where the robot may fail to open a kid-door. (In the original domain, the robot always succeeds in opening a door; kid can close it again only from the next round.) If a kid-door is on the route of the robot, no strong plan exists for that problem: the robot can only try to open the door until it succeeds.

The results of this experiment are shown in Table 3. We report only the case of 0 kid-doors (where strong plans always exist), and the case of 7 kid-doors (where strong plans never exist). In all the cases, the number n of objects to be moved ranges from 1 to 5. The upper part of the table covers the "strong" reachability planning problem and compares the optimized MBP algorithm (Strong), the general algorithm on goal AF p, and, for completeness, SIMPLAN. In the case $k = 7$, the times in Table 3 are those required by MBP to report that no strong plan exists. The lower part of the table considers the "strong-cyclic" reachability planning problem and compares the two strong-cyclic algorithms of MBP (SC-Global and SC-Local) and the general MBP algorithm on goal A(EF p W p). The experiment shows that the generic algorithm for temporally extended goals compares quite well with respect to the optimized algorithms provided by MBP. Indeed, the generic algorithm requires about twice the time needed by the optimized algorithm in the case of the strong plans and about $2.5\times$ the time of the optimized "global" algorithms for the strong-cyclic planning. The "local" algorithm behaves better than the generic algorithm (and than the "global" one) in the case a strong plan exists, i.e., $k = 0$; it behaves worse in the case no plan exists, i.e., $k = 7$. This difference in the performances of the MBP algorithms derives from the overhead introduced in the generic algorithm by the need of managing generic goals, and from the optimizations present in the specific algorithms. For instance, the Strong and SC-Local algorithms stop when a plan is found for all the initial states, while the generic algorithm stops only when a fixpoint is reached.

Experiment 4 tests the scalability of the algorithm w.r.t. the complexity of the goal. In particular, we consider the case of sequential reachability goals in the modified domain of Experiment 3. Given a sequence $p_1, \ldots, p_t$ of conditions to be reached, with

$$p_i = in(obj_{i,1}, room_{i,1}) \wedge \cdots \wedge in(obj_{i,n}, room_{i,n}),$$

	k = 0						k = 7					
	t = 1	t = 2	t = 3	t = 4	t = 5	t = 6	t = 1	t = 2	t = 3	t = 4	t = 5	t = 6
AF _	35.1	97.5	158.7	196.1	197.9	290.4	0.3	0.5	0.4	0.6	0.6	0.8
SIMPLAN with CS	1.2	2.9	4.2	6.8	7.8	11.5	-	-	-	-	-	-
SIMPLAN w/o CS	-	-	-	-	-	-	-	-	-	-	-	-
A(EF _ W _)	128.1	195.1	307.7	358.0	583.0	632.4	151.3	279.8	342.8	544.0	642.9	799.4

Table 4. Results of Experiment 4.

we consider the "strong" sequential reachability planning problems

$$\mathrm{AF}\,(p_1 \wedge \mathrm{AF}\,(p_2 \wedge \cdots \mathrm{AF}\,(p_t)))$$

and the "strong-cyclic" sequential reachability planning problem

$$\mathrm{A}(\mathrm{EF}\,(p_1 \wedge \cdots \mathrm{A}(\mathrm{EF}\,p_t\,\mathrm{W}\,p_t))\,\mathrm{W}\,(p_1 \wedge \cdots \mathrm{A}(\mathrm{EF}\,p_t\,\mathrm{W}\,p_t))).$$

In Table 4 we present the outcomes of the experiment. In the strong case, we present the results also for SIMPLAN. In the strong-cyclic case a comparison is not possible, as SIMPLAN is not able to represent this kind of goals. The number n of objects is set to 3, while the nesting level t ranges from 1 to 6. The cases of 0 and of 7 kid-doors are considered. The experiment shows that MBP scales about linearly in the number of nested temporal operators, both in the case of strong and in the case of strong-cyclic multiple reachability. In the case of 0 kid-doors and strong reachability, also SIMPLAN with search control strategies scales linearly, and the performance is much better than MBP. Without search strategies, SIMPLAN is not able to complete any of the tests in this experiment.

The experimental evaluation shows that MBP is able to deal with relatively large domains (some of the instances of the considered experiments have more than 10^8 states) and with high non-determinism, and that the performance scales well with respect to the goal complexity. In terms of expressiveness, CTL turns out to be an interesting language for temporally extended goals. With respect to LTL (used by SIMPLAN), CTL can express goals that combine universal and existential path quantifiers: this is the case, for instance, of the strong-cyclic reachability goals. On specific planning problems, (e.g., reachability problems) the overhead w.r.t. optimized state-of-the-art algorithms is acceptable. The comparison with SIMPLAN shows that the algorithms based on symbolic techniques outperform the explicit planning algorithms in the case search control strategies are not allowed. With search control strategies, SIMPLAN performs better than MBP in the case of domains with a low non-determinism. The possibility of enhancing the performance of MBP with search strategies is an interesting topic for future research.

5 Conclusions and Related Work

In this paper, we have described a planning algorithm based on symbolic model checking techniques, which is able to deal with non-deterministic domains and goals as CTL temporal formulas. This work extends the theoretical results presented in [14] by developing a symbolic algorithm that fully exploits the potentiality of BDDs for the efficient exploration of huge state spaces. We implement the algorithm in MBP, and perform a set of experimental evaluations to show that the approach is practical. We test the scalability of the planner depending on the dimension of the domain, on the degree of non-determinism, and on the length of the goal. The experimental evaluation gives positive results, even in the case MBP is compared with hand-tailored domain and goal dependent heuristics (like those of SIMPLAN), or with algorithms optimized to deal with reachability goals.

Besides SIMPLAN, very few attempts have been made to build planners that work in practice in such a general setting like the one we propose. The issue of "temporally extended goals" is certainly not new. However, most of the works in this direction restrict to deterministic domains, see for instance [8, 1]. Most of the planners able to deal with non-deterministic domains, do not deal with temporally extended goals [7, 15, 3].

Planning for temporally extended goals is strongly related to the "synthesis problem" [12]. Indeed, the planner has to synthesize a plan from the given goal specification. We are currently investigating the applicability of the proposed algorithm to synthesis problems.

In this paper we focus on the case of full observability. An extension of the work to the case of planning for extended goals under partial observability is one of the main objectives for future research.

References

1. F. Bacchus and F. Kabanza. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.
2. P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a Model Based Planner. In *Proc. of IJCAI'01 workshop on Planning under Uncertainty and Incomplete Information*, 2001.
3. B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. AIPS 2000*, 2000.
4. 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, 1992.
5. A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: a reimplementation of SMV. In *Proc. STTT'98*, pages 25–31, 1998.
6. A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. Technical report, IRST, Trento, Italy, 2001.
7. A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proc. AAAI'98*, 1998.
8. G. de Giacomo and M.Y. Vardi. Automata-theoretic approach to planning with temporally extended goals. In *Proc. ECP'99*, 1999.
9. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter 16, pages 995–1072. Elsevier, 1990.
10. R. Jensen and M. Veloso. OBBD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intellegence Research*, 13:189–226, 2000.
11. F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67–113, 1997.
12. O. Kupferman and M. Vardi. Synthesis with incomplete informatio. In *Proc. Int. Conf. on Temporal Logic*, 1997.
13. O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Proc. CAV'94*, 1994.
14. M. Pistore and P. Traverso. Planning as Model Checking for Extended Goals in Non-deterministic Domains. In *Proc. IJCAI'01*. AAAI Press, August 2001.
15. J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intellegence Research*, 10:323–352, 1999.
16. M. J. Schoppers. Universal plans for Reactive Robots in Unpredictable Environments. In *Proc. IJCAI'87*, 1987.