Planning as Satisfiability in Nondeterministic Domains

Paolo Ferraris and **Enrico Giunchiglia**

DIST — Università di Genova
Viale Causa 13, 16145 Genova, Italy
{otto,enrico}@dist.unige.it

Abstract

We focus on planning as satisfiability in simple nondeterministic domains. By "simple" we mean specified in a simple extension to the STRIPS formalism allowing for specifying actions with nondeterministic effects. This allows us to simplify and extend the theory presented in (Giunchiglia 2000). The result is a planning system which, in simple nondeterministic domains, is competitive with other state-of-the-art planners.

Introduction

Planning as satisfiability (Kautz & Selman 1992) has emerged as one of the most effective approaches to classical planning. Complex problems with 100 actions can be solved in minutes by the latest SAT-based planning system Blackbox (Kautz & Selman 1998). However, the applicability of the SAT-based approach to planning has been confined to classical problems, i.e., to planning domains in which both the effects of actions and the initial state are completely specified. Recently (Giunchiglia 2000), the second author has presented a SAT-based procedure for planning in $\mathcal{C}$ action descriptions (Giunchiglia & Lifschitz 1998) with incompletely specified initial state. The action language $\mathcal{C}$ —among other expressive capabilities— allows to specify domains with concurrent actions, constraints, and nondeterminism.

In this paper we focus on planning as satisfiability in simple nondeterministic domains. By "simple" we mean specified in a simple extension to the STRIPS formalism allowing for specifying actions with nondeterministic effects. This allows us to simplify and extend the encodings and the procedures presented in (Giunchiglia 2000).

About the encodings, we show that various encodings are possible, differing for the action representation ("regular" or "simple-split") and/or planning strategy ("sequential" or "parallel"), see, e.g., (Kautz & Selman 1996; Kautz, McAllester, & Selman 1997; Ernst, Millstein, & Weld 1997) for the corresponding notions in the classical case. These encodings can be automatically generated in polynomial time starting from the initial action description.

About the procedures, we adopt some simple heuristics like the "split on action first" idea from (Giunchiglia, Massarotto, & Sebastiani 1998). We also write down some simple clauses ruling out useless branches from the search space.

The result is a planning system which, in simple nondeterministic domains, is competitive with other state-of-the-art planners.

STRIPS + Nondeterminism

Traditionally, the description of an action in STRIPS consists of the specification of its preconditions and effects as sets of fluent literals. For an action with nondeterministic effects, we specify its possible effects as a set whose elements are sets of fluent literals. Thus, an action A is described using the following notation

$$\begin{array}{lll} A & \text{pre}: & P; \\ & \text{d-eff}: & E; \\ & \text{i-eff}: & N_1, \ldots, N_n; \end{array} \qquad (1)$$

$(n \geq 1)$ in which $\{P\}, \{E\}, N_1, \ldots, N_n$ are finite sets of fluent literals. Intuitively, P represents the preconditions for the action, E lists its determinate effects, while each N_i represent one of the possible indeterminate outcomes for the action. When $\{P\} = \{\}$, or $\{E\} = \{\}$, or $n = 1$ and $N_1 = \{\}$ (corresponding to an action with no preconditions, or no determinate effects or no indeterminate effects) we omit the corresponding field from the specification of A. In the following, we write *pre*(A), *d-eff*(A), and *i-eff*(A) to denote respectively the sets $\{P\}$, $\{E\}$, and $\{N_1, \ldots, N_n\}$ respectively. We further assume that for any action A, the sets in *i-eff*(A) are consistent with *d-eff*(A), i.e., that for any set N in *i-eff*(A), it is not the case that the set $N \cup$ *d-eff*(A) is inconsistent.

A finite set of actions' specifications is an *action description*. The semantics of an action description can be described in terms of a labeled transition system:

- A *state* is a maximally consistent set of fluent literals, and
- there is a *transition from a state σ to a state σ' with label A* iff A is an action,
  1. *pre*$(A) \subseteq \sigma$,
  2. $\sigma' = (\sigma \cap \sigma') \cup \Gamma$, where Γ is the union of *d-eff*(A) with *some* of the sets in *i-eff*(A).

The first condition states when A is executable. The second condition says that a fluent keeps its value unless affected either by *d-eff*(A) or *i-eff*(A).

Action descriptions describe what are the effects of executing actions. Consider for example the following elaboration of the "bomb in the toilet" problem from (Mcdermott 1987). There is a finite set P of packages and a finite set T of toilets. In this section, we assume that there are two packages P_1 and P_2 and one toilet T_1. One of the packages is armed because it contains a bomb. Dunking a package in a toilet disarm the package, but may clog the toilet. Dunking a package in a toilet is possible only if the toilet is not clogged and the package has been not previously dunked. Flushing clears a toilet. This scenario is described by the following action description, in which p and t are metavariables ranging over the sets P and T respectively:

$$\begin{array}{lll} \mathit{Dunk}(p,t) & \text{pre}: & \neg\mathit{Clogged}(t), \neg\mathit{Dunked}(p); \\ & \text{d-eff}: & \neg\mathit{Armed}(p), \mathit{Dunked}(p); \\ & \text{i-eff}: & \{\mathit{Clogged}(t)\}, \{\}; \end{array} \quad (2)$$

$$\mathit{Flush}(t) \quad \text{d-eff}: \quad \neg\mathit{Clogged}(t);$$

According to the indeterminate effects of dunking, the toilet will become clogged, or nothing will happen.

In order to specify a planning problem, we need to specify an action description D, but also which are the initial and goal states of D: the task is to determine a "valid plan", i.e., a finite sequence of actions which is "always executable" and such that any of its "possible outcomes" is a goal state.

Formally, a *planning problem* is a triple $\langle I, D, G\rangle$, in which D is an action description while I and G are two fluent formulas. A state σ of D is *initial* [resp. *goal*] if σ satisfies I [resp. G].[1] Thus, for the "bomb in the toilet", the planning problem in which initially one of the packages is armed and the toilet is clear; and with goal to disarm all packages, can be specified by the triple:

$$\langle (\mathit{Armed}(P_1) \equiv \neg\mathit{Armed}(P_2)) \wedge \neg\mathit{Clogged}(T_1), (2), \neg\mathit{Armed}(P_1) \wedge \neg\mathit{Armed}(P_2)\rangle. \quad (3)$$

A *plan* is a finite sequence of actions.

Consider a planning problem $\langle I, D, G\rangle$.

A *history* is a path in the transition diagram of D, that is, a finite sequence

$$\sigma^0, A^1, \sigma^1, \ldots, A^n, \sigma^n \quad (4)$$

($n \geq 0$) such that $\sigma^0, \sigma^1, \ldots, \sigma^n$ are states; $A^1, \ldots, A^n$ are actions; and, for each $i \in \{i, \ldots, n\}$, D has a transition from σ^{i-1} to σ^i with label A^i.

A plan $A^1; \ldots; A^n$ ($n \geq 0$) is *possible* for $\langle I, D, G\rangle$ if there exists a history (4) such that σ^0 is an initial state, and σ^n is a goal state. Notice that —unless the action description is deterministic and there is only one initial state— a possible plan is not ensured to achieve the goal. For example, considering the planning problem (3), the plans $\mathit{Dunk}(P_1, T_1)$ and $\mathit{Dunk}(P_1, T_1); \mathit{Dunk}(P_2, T_1)$ are both possible. However, intuitively none is valid: the first does not achieve the goal if the bomb is in P_2, while the second is not executable if dunking P_1 clogs the toilet.

An action A *is executable in a state* σ if D has a transition from σ to a state σ' with label A. Let σ^0 be a state. A plan $A^1; \ldots; A^n$ is *always executable in* σ^0 if for any history

$$\sigma^0, A^1, \sigma^1, \ldots, A^k, \sigma^k$$

with $k < n$, A^{k+1} is executable in σ^k. A state σ^n is a *possible result of executing a plan* $A^1; \ldots; A^n$ *in* σ^0 if there exists a history (4) for D.

A plan $A^1; \ldots; A^n$ is *valid* if for any initial state σ^0,

- $A^1; \ldots; A^n$ is always executable in σ^0, and
- any possible result of executing $A^1; \ldots; A^n$ in σ^0 is a goal state.

Indeed, the plan $\mathit{Dunk}(P_1, T_1); \mathit{Flush}(T_1); \mathit{Dunk}(P_2, T_1)$ is valid for the planning problem (3). Notice that if we assume that flushing is possible only when the toilet is clogged, no valid plan exists for (3).

Encoding possible plans

As in the classical case, various types of encodings are possible. Here we restrict to the generalization of the encodings:

- Explanatory-Regular-Parallel (ERP),
- Explanatory-Regular-Sequential (ERS), and
- Explanatory-simple-Split-Sequential (ESS),

i.e., the ones that experimentally proved to lead to the best computational results in the classical case (see, e.g., (Ernst, Millstein, & Weld 1997; Giunchiglia, Massarotto, & Sebastiani 1998)).

Consider an action description D. In the ERP encodings, for each time point i, we have the following formulas (when we consider the action description (2), p, p' and t, t' are metavariables ranging over P and T respectively):

- an action A executed at time i implies that its preconditions and determinate effects are true at time i and $i+1$ respectively:

$$A_i \supset \Big(\bigwedge_{L \in \mathit{pre}(A)} L_i \wedge \bigwedge_{L \in \mathit{d\text{-}eff}(A)} L_{i+1}\Big).$$

  In the case of (2), we have:

$$\begin{array}{l} \mathit{Dunk}_i(p,t) \supset \neg\mathit{Clogged}_i(t) \wedge \neg\mathit{Dunked}_i(p) \\ \qquad\qquad \wedge\neg\mathit{Armed}_{i+1}(p) \wedge \mathit{Dunked}_{i+1}(p), \\ \mathit{Flush}_i(t) \supset \neg\mathit{Clogged}_{i+1}(t). \end{array}$$

- An action A executed at time i implies that some of its indeterminate effects are true at time $i+1$:

$$A_i \supset \bigvee_{N \in \mathit{i\text{-}eff}(A)} \bigwedge_{L \in N} L_{i+1}.$$

  In the case of (2), the corresponding formulas are tautologies.

- If a fluent literal L becomes true at time $i+1$, then an action A affecting it either through its determinate or indeterminate effects must have been executed at time i:

[1] We say that a state σ satisfies a fluent formula F if $\sigma \cup \{F\}$ is consistent.

– If $L \in$ *d-eff*(A) then it is enough to impose that A has been executed at time i,
– If $L \in N \in$ *i-eff*(A) then we have to impose that A has been executed at time i but also that all the literals in N are true at time $i+1$.

In a formula:

$$L_{i+1} \wedge \neg L_i \supset \bigvee_{A:L \in \textit{d-eff}(A)} A_i \vee \bigvee_A \bigvee_{N:N \in \textit{i-eff}(A), L \in N} (A_i \wedge \bigwedge_{L':L' \in N} L'_{i+1}).$$

In the case of (2), we get formulas equivalent to:

$$\begin{gathered}\neg \textit{Armed}_{i+1}(p) \wedge \textit{Armed}_i(p) \supset \bigvee_{t \in T} \textit{Dunk}_i(p,t),\\ \neg \textit{Armed}_i(p) \supset \neg \textit{Armed}_{i+1}(p),\\ \neg \textit{Clogged}_{i+1}(t) \wedge \textit{Clogged}_i(t) \supset \textit{Flush}_i(t),\\ \textit{Clogged}_{i+1}(t) \wedge \neg \textit{Clogged}_i(t) \supset \bigvee_{p \in P} \textit{Dunk}_i(p,t),\\ \textit{Dunked}_{i+1}(p) \wedge \neg \textit{Dunked}_i(p) \supset \bigvee_{t \in T} \textit{Dunk}_i(p,t),\\ \textit{Dunked}_i(p) \supset \textit{Dunked}_{i+1}(p).\end{gathered}$$

- Two distinct actions A and B cannot be executed at time i if they are mutex:

$$\neg(A_i \wedge B_i). \tag{5}$$

For the ERP encodings, A and B are *mutex* if the effects of A contradicts with the possible effects or the preconditions of B; or the other way around. More formally, if there exists two sets $N \in$ *i-eff*(A) and $N' \in$ *i-eff*(B) such that *d-eff*$(A) \cup N \cup$ *pre*$(B) \cup N'$ or *pre*$(A) \cup N \cup$ *d-eff*$(B) \cup N'$ are contradictory. In the case of (2), this imposes

$$\begin{gathered}\neg(\textit{Dunk}_i(p,t) \wedge \textit{Flush}_i(t)),\\ \neg(\textit{Dunk}_i(p,t) \wedge \textit{Dunk}_i(p',t)) \qquad p \neq p',\end{gathered}$$

and, in the case there is one more than one toilet, also

$$\neg(\textit{Dunk}_i(p,t) \wedge \textit{Dunk}_i(p,t')) \qquad t \neq t'.$$

In the ERS and ESS encodings, any two distinct actions A and B are considered to be mutex. As a matter of fact, the ERS encodings are as the ERP, except that we have (5) for any two distinct actions A and B. The ESS encodings, are obtained from the ERS encodings by substituting each action

$$\textit{ActionPred}_i(\textit{Arg}_1, \ldots, \textit{Arg}_n)$$

with the conjunction

$$(\textit{ActionPred}_i\textit{-Arg}_1 \wedge \ldots \wedge \textit{ActionPred}_i\textit{-Arg}_n)$$

(see, e.g., (Kautz, McAllester, & Selman 1997; Ernst, Millstein, & Weld 1997)). Thus, in our example, we substitute $\textit{Dunk}_i(p,t)$ with $\textit{Dunk\_pkg}_i(p) \wedge \textit{Dunk\_into}_i(t)$. Factorization techniques analogous to those described in (Ernst, Millstein, & Weld 1997) for the classical case, allow to simplify/eliminate some of the clauses. For example, we have the clauses corresponding to, e.g.,

$$\textit{Dunk\_pkg}_i(p) \supset \neg \textit{Dunked}_i(p)$$

instead of those corresponding to

$$\textit{Dunk\_pkg}_i(p) \wedge \textit{Dunk\_into}_i(t) \supset \neg \textit{Dunked}_i(p).$$

It is not difficult to see that, if actions do not have indeterminate effects, each of the above encodings boils down to the corresponding encoding proposed in the classical setting.

Let tr_i^D be the conjunction of the above formulas for the particular encoding chosen. Consider a planning problem $\langle I, D, G \rangle$. For each number $n \geq 0$, each assignment[2] μ satisfying

$$I_0 \wedge \bigwedge_{i=0}^{n-1} tr_i^D \wedge G_n \tag{6}$$

corresponds to a possible plan:

- In the case of the ERS encodings, we have that for each $i \in \{0, \ldots, n-1\}$ at most one action variable A_i is assigned to true by μ: this guarantees a totally ordered plan. Analogously for the ESS encodings.
- In the case of the ERP encodings, for each $i \in \{0, \ldots, n-1\}$ more than one action variable A_i may be assigned to true by μ. This results in a partially ordered set of actions. However, any total order consistent with the partial order corresponds to a possible plan. We can choose to lexicographically order any initially unordered pair of actions.

Computing valid plans

Consider a planning problem $\langle I, D, G \rangle$. Let μ be an assignment satisfying (6). Let *act*(μ) be the conjunction of the literals in μ corresponding to actions. Let *plan*(μ) be the possible plan corresponding to μ.

In order to determine whether *plan*(μ) is valid, the fundamental observation is that if each action is executable in any state, *plan*(μ) is valid iff

$$\textit{act}(\mu) \wedge I_0 \wedge \bigwedge_{i=0}^{n-1} tr_i^D \models G_n.$$

On the basis of this observation, our next step is to define a new propositional formula trt_i^D which intuitively represents the transition relation of an automaton Σ

- whose states are doubled (wrt the automaton corresponding to D) by introducing a new fluent symbol Z, and
- whose transitions are labeled with the actions of D and are such that there is a transition from a state σ to a state σ' with label A if and only if
  - σ and σ' satisfy $\neg Z$ and D has a transition from σ_D to σ'_D with label A, or
  - σ satisfies $\neg Z$, σ' satisfies Z and A is not executable in σ_D, or
  - σ and σ' satisfy Z,

  where σ_D is the restriction of σ to the fluent signature of D (and similarly for σ'_D).

Thus in Σ any action is "executable" in any state; and (4) is a history of D iff

$$\sigma^0 \cup \{\neg Z\}, A^1, \sigma^1 \cup \{\neg Z\}, \ldots, A^n, \sigma^n \cup \{\neg Z\}$$

[2] An *assignment* is a set of literals. An assignment μ *satisfies* a formula P if (i) μ is a maximally consistent set of the literals in P, and (ii) $\mu \cup \{P\}$ is consistent.

is a path of Σ.

Let $Poss_i^D$ be the conjunction of the formulas

$$(A_i \supset \bigwedge_{P \in pre(A)} P_i) \tag{7}$$

over all actions A of D. Intuitively, the formula $Poss_i^D$ says which action is executable in which state at time i. As in (Giunchiglia 2000), we define trt_i^D to be the formula

$$(tr_i^D \wedge \neg Z_i \wedge \neg Z_{i+1}) \vee ((Z_i \vee \neg Poss_i^D) \wedge Z_{i+1}),$$

where Z is a newly introduced fluent symbol. It is not difficult to prove that the above itemized properties hold.

From the above, we have that $plan(\mu)$ is valid iff

$$act(\mu) \wedge I_0 \wedge \neg Z_0 \wedge \bigwedge_{i=0}^{n-1} trt_i^D \models G_n \wedge \neg Z_n. \tag{8}$$

Thus, we can

- *generate* possible plans by finding assignments μ satisfying (6), and
- *test* whether the $plan(\mu)$ is valid by checking whether (8) holds.

This is the idea behind the procedure represented in Figure 1, stated correct and complete in the case of planning problems whose action description is written in $\mathcal{C}$ (Giunchiglia & Lifschitz 1998). Notice that in (Giunchiglia 2000), $Poss_i^D$ is defined as the formula

$$\exists p^1 \ldots \exists p^n tr_i^D[P_{i+1}^1/p^1, \ldots, P_{i+1}^n/p^n] \tag{9}$$

where $P^1, \ldots, P^n$ are all the fluent symbols in D, and $tr_i^D[P_{i+1}^1/p^1, \ldots, P_{i+1}^n/p^n]$ denotes the formula obtained from tr_i^D by substituting each fluent P_{i+1}^k with a distinct propositional variable p^k.

(7) and (9) are not equivalent: (9) entails (7) but not viceversa. However (for the ERP and ERS encodings, and similarly for the ESS), if we add to (7) a conjunct $\neg(A_i \wedge B_i)$ for each pair of actions A and B

- which are mutex, or
- in the case of ERP encodings, such that $d\text{-}eff(A) \cup d\text{-}eff(B)$ is contradictory,

the two formulas become equivalent. These additional conjuncts are not needed for (8) to hold. In fact, in (8), μ is an assignment satisfying (6): as a consequence, $act(\mu)$ entails such additional conjuncts. Notice that, from a computational point of view, computing a propositional formula equivalent to (9) may require an exponential space.

Consider Figure 1. First notice the loop in PLAN_TEST: this takes into account that the assignments generated by $\text{PLAN\_GEN}_{\text{DP}}$ may be partial. Then, notice that the procedure $\text{PLAN\_GEN}_{\text{DP}}$ is essentially the Davis Putnam (DP) procedure (Davis & Putnam 1960; Davis, Longemann, & Loveland 1962) modulo the call to PLAN_TEST whenever it finds an assignment satisfying the input formula. Because of this, we can use any state-of-the-art SAT solvers *almost* as a blackbox. Indeed, we need to modify the source code to perform the above mentioned call to PLAN_TEST, and —if implemented— we need to rule out also the pure-literal rule (see, e.g., (Giunchiglia *et al.* 1998) for more details). Such modifications have been carried out on various state-of-the-art SAT solvers, like Bohm's solver (see (Giunchiglia *et al.* 1998)), Zhang's SATO (see (Giunchiglia & Tacchella 2000)), and Bayardo' and Schrag's REL_SAT (see (Wolfman & Weld 1999)).

$P := I_0 \wedge \bigwedge_{i=0}^{n-1} tr_i^D \wedge G_n$;
$V := I_0 \wedge \neg Z_0 \wedge \bigwedge_{i=0}^{n-1} trt_i^D \wedge \neg(G_n \wedge \neg Z_n)$;
function PLAN() **return** $\text{PLAN\_GEN}_{\text{DP}}(cnf(P), \{\})$.

function $\text{PLAN\_GEN}_{\text{DP}}(\varphi, \mu)$
  if $\varphi = \{\}$ **then return** PLAN_TEST(μ);
  if $\{\} \in \varphi$ **then return** *False*;
  if { a unit clause $\{L\}$ occurs in φ } **then**
    return $\text{PLAN\_GEN}_{\text{DP}}(assign(L, \varphi), \mu \cup \{L\})$;
  $A :=$ { an atom occurring in φ };
  return $\text{PLAN\_GEN}_{\text{DP}}(assign(A, \varphi), \mu \cup \{A\})$ **or**
    $\text{PLAN\_GEN}_{\text{DP}}(assign(\neg A, \varphi), \mu \cup \{\neg A\})$.

function PLAN_TEST(μ)
  foreach {assignment μ' s.t. $\mu \subseteq \mu'$}
    if not SAT($act(\mu') \wedge V$) **then exit with** $plan(\mu')$;
  return *False*.

Figure 1: PLAN and $\text{PLAN\_GEN}_{\text{DP}}$

Heuristics

The procedure PLAN is essentially a "generate and test" procedure. As such, care must be taken in order to avoid the generation of useless cases. To this extent, we have devised the heuristics described in the following paragraphs (here we consider only regular encodings: analogous considerations hold for ESS encodings).[3]

Split on actions first: in $\text{PLAN\_GEN}_{\text{DP}}$, when choosing the atom A, we first give preference to variables corresponding to actions. Splitting on action variables has proven to be a very effective heuristics in the classical case (see (Giunchiglia, Massarotto, & Sebastiani 1998)). In our setting, this imposes that —in $\text{PLAN\_GEN}_{\text{DP}}$ search tree— branching nodes corresponding to fluent variables occur only at the bottom. Given an assignment corresponding to a possible plan which fails the validity test, it is easy to modify $\text{PLAN\_GEN}_{\text{DP}}$ in order to incorporate a simple backjumping schema to the latest assigned action variable. The combined result is that a possible plan will be generated and then tested for validity at most once.

For any two actions A and B such that some of B's preconditions contradict some of the indirect effects of A, we add to (6) the clause

$$\neg A_i \vee \neg B_{i+1},$$

[3] We use the term "heuristics" since they are meant to improve the performances "on the average". It is indeed the case that on some problem, better performances can be obtained by turning some heuristic off. However, in our experience, in most cases this does not happen.

for each $i < n$. In fact, any possible plan corresponding to an assignment which violates the above clause, will fail the validity test. In the case of (2), this allows to add the clauses

$$\neg \mathit{Dunk}_i(p,t) \vee \neg \mathit{Dunk}_{i+1}(p',t),$$

for any two packages p and p' and $i < n$.

For each action A such that $\mathit{pre}(A) \cup \mathit{d\text{-}eff}(A)$ is consistent, we can add to (6)

$$\neg A_i \vee \neg A_{i+1},$$

for each $i < n$. In fact, for any such action A, if $A_1, \ldots, A, A, \ldots, A_n$ is a valid plan, then also $A_1, \ldots, A, \ldots, A_n$ is valid. In the case of (2), this implies that

$$\neg \mathit{Flush}_i(t) \vee \neg \mathit{Flush}_{i+1}(t)$$

can be safely added, for any toilet t and $i < n$.

For each time step $i < n-1$, we can assume that if we do not execute any action at i, then we will not execute any action at $i+1$. For (2), if *Act* is the whole set of actions in the domain, this would imply adding the clauses corresponding to

$$\bigwedge_{A \in \mathit{Act}} \neg A_i \supset \bigwedge_{A \in \mathit{Act}} \neg A_{i+1}.$$

For the bomb-in-the-toilet problem, we will have $|P| \times |T| + |T|$ clauses, each with $|P| \times |T| + |T| + 1$ disjuncts. In order not to have so many long clauses, we introduce the action *NoOp*, and write the clauses

$$\bigvee_{p \in P, t \in T} \mathit{Dunk}_i(p,t) \vee \bigvee_{t \in T} \mathit{Flush}_i(t) \vee \mathit{NoOp}_i,$$

$$\mathit{NoOp}_i \supset \mathit{NoOp}_{i+1},$$

and the $|P| \times |T| + |T|$ binary clauses corresponding to

$$\mathit{NoOp}_i \supset \bigwedge_{p \in P, t \in T} \neg \mathit{Dunk}_i(p,t) \wedge \bigwedge_{t \in T} \neg \mathit{Flush}_i(t).$$

Finally, if we have already checked the non existence of valid plans of length k, when looking for valid plans of length $n > k$, we can restrict our search for possible plans which falsify the goal at step k. In the case of (2), if the goal is to disarm all the packages, we add the clause:

$$\bigvee_{p \in P} \mathit{Armed}_k(p).$$

Even though not necessary (because entailed) we also add the unit clause $\neg \mathit{NoOp}_k$.

Experimental analysis

We have developed $\mathcal{C}$-PLAN, a system implementing the above procedures on top of *SAT (Giunchiglia & Tacchella 2000). For a high-level description of $\mathcal{C}$-PLAN, see (Giunchiglia 2000). Here, it suffices to say that $\mathcal{C}$-PLAN accepts $\mathcal{C}$ action descriptions written in *CCALC*[4], and that in our tests we have looked for plans of increasing length starting from $n = 1$.

[4]For *CCALC*, see `http://www.cs.utexas.edu/users/tag/cc`.

	CGP		ERP		ERS		ESS	
$\|P\|$-$\|T\|$	#s	CPU	#s	CPU	#s	CPU	#s	CPU
2-1	3	0.01	3	0.00	3	0.00	3	0.00
3-1	5	0.02	5	0.01	5	0.00	5	0.01
4-1	7	0.07	7	0.01	7	0.01	7	0.02
5-1	9	0.27	9	0.03	9	0.04	9	0.04
6-1	11	1.36	11	0.06	11	0.06	11	0.08
2-2	1	0.01	1	0.00	2	0.01	2	0.00
3-2	3	0.01	3	0.01	4	0.02	4	0.01
4-2	3	0.04	3	0.03	6	0.07	6	0.03
5-2	5	0.52	5	0.06	8	0.16	8	0.13
6-2	5	1.26	5	0.10	10	0.32	10	3.20
2-3	1	0.00	1	0.00	2	0.01	2	0.00
3-3	1	0.01	1	0.01	3	0.05	3	0.04
4-3	3	0.07	3	0.04	5	0.20	5	0.06
5-3	3	0.16	3	0.07	7	0.50	7	0.08
6-3	3	0.35	3	0.17	9	1.11	9	0.14

Table 1: Deterministic actions, multiple initial states

To test $\mathcal{C}$-PLAN effectiveness, we have compared it with CGP (Smith & Weld 1998).CGP has been compiled and run using Allegro Common Lisp Trial Edition 5.0.1. $\mathcal{C}$-PLAN is written in C and has been compiled with `gcc -O2`. The testing machine has been an Intel PC `PII350MHz` with `256MbRAM`, running `SUSE Linux 6.2`. Since CGP only deals with nondeterminism in the initial state, we have modified (2) in such a way that dunking a package always clogs the toilet. Initially only one package is armed, the toilets are not clogged, and the goal is to disarm all the packages. As in (Smith & Weld 1998), we (i) have considered problems having from 2 to 6 packages, and from 1 to 3 toilets, and (ii) have assumed that flushing is possible only if the toilet is clogged. For $\mathcal{C}$-PLAN, we have tested its performances using the three encodings ERP, ERS and ESS described previously.

Table 1 shows the results. For CGP, we report the number of levels and the CPU time needed (excluding the time taken by the garbage collector). For each of three encodings used to test $\mathcal{C}$-PLAN, we report the number of steps needed, and the CPU time needed by $\mathcal{C}$-PLAN for the last step. CPU times are in seconds, and repeating the runs produces very similar results.

Consider Table 1. As it can be observed, $\mathcal{C}$-PLAN is competitive with CGP, which experimentally proved to outperform previous planners for nondeterministic domains (like BURIDAN (Kushmerick, Hanks, & Weld 1995) and UDTPOP (Peot 1998)) by orders of magnitude (see (Smith & Weld 1998) for details). Comparing the performances of $\mathcal{C}$-PLAN on the different encodings, we see that ERP leads to the fewest steps and best performances.

In order to evaluate the performances of $\mathcal{C}$-PLAN when using different encodings on nondeterministic domains, we have tested it using the action description (2), in which dunking a package may clog the toilet. The initial and goal states are as before. The results are in Table 2.

Consider Table 2. ERP encodings lead to the best per-

	ERP		ERS		ESS	
$\|P\|$-$\|T\|$	#s	last	#s	last	#s	last
2-1	3	0.00	3	0.00	3	0.01
3-1	5	0.01	5	0.01	5	0.01
4-1	7	0.02	7	0.02	7	0.02
5-1	9	0.05	9	0.05	9	0.04
6-1	11	0.09	11	0.08	11	0.08
2-2	1	0.01	2	0.01	2	0.01
3-2	3	0.02	4	0.04	4	0.05
4-2	3	0.07	6	0.12	6	0.75
5-2	5	0.12	8	4.14	8	112.96
6-2	5	6.76	10	538.41	10	5780.81
2-3	1	0.00	2	0.02	2	0.01
3-3	1	0.01	3	0.07	3	0.04
4-3	3	0.07	5	0.55	5	0.72
5-3	3	0.27	7	80.45	8	4.88
6-3	3	0.88	9	7590.51	9	5.85

Table 2: Nondeterministic actions, multiple initial states

formances, sometimes orders of magnitude better than the others. This is not surprising in that it confirms the results obtained in the classical setting. What is more surprising are the performances of $\mathcal{C}$-PLAN when using the ERS or the ESS encodings. Sometime one encoding dominates the other by orders of magnitude, and sometime it is the other way around. Even more, $\mathcal{C}$-PLAN performances do not seem predictable. Introducing an additional package or toilet may cause an unpredictable change in $\mathcal{C}$-PLAN performances. A similar phenomenon has already been observed in propositional satisfiability. As discussed in (Gomes, Selman, & Kautz 1998), the performances of SAT solvers based on the DP procedure can be highly unpredictable: minor changes in the search procedure or in the propositional formula can drastically alter the solution time. As a matter of fact, $\mathcal{C}$-PLAN is heavily based on the DP procedure. Because of this, it may well be the case that by introducing controlled randomization and rapid restart into $\mathcal{C}$-PLAN, we can obtain the speed-ups described in (Gomes, Selman, & Kautz 1998), and, at the same time, make $\mathcal{C}$-PLAN more predictable.

Conclusions

In this paper we have focused on planning as satisfiability in simple nondeterministic domains. We have simplified and extended the theory presented in (Giunchiglia 2000). For example, we have shown that we do not need the (possibly very expensive) computation of a propositional formula equivalent to (9). We have also shown that different encodings are possible, along the lines of what has been done in the classical case: defining "bitwise", or "overloaded simple-split" encodings (see (Ernst, Millstein, & Weld 1997)) seems relatively easy. We have implemented $\mathcal{C}$-PLAN: The experimental analysis shows that $\mathcal{C}$-PLAN is competitive with CGP.

As for CGP, our objective in developing $\mathcal{C}$-PLAN has been to see whether the good performances obtained by SAT-based planners in the classical case, would extend to more complex problems involving concurrency and/or constraints and/or nondeterminism. The experimental analysis shows that, at least in a "simple nondeterministic" setting, the answer is positive.

Acknowledgments

We thank David Smith for having provided CGP code. We thank Norman McCain for the help on *CCALC*. We thank Armando Tacchella for the help on *SAT. The second author is supported by ASI, CNR and MURST.

References

Davis, M., and Putnam, H. 1960. A computing procedure for quantification theory. *JACM* 7:201–215.

Davis, M.; Longemann, G.; and Loveland, D. 1962. A machine program for theorem proving. *Journal of the ACM* 5(7).

Ernst, M.; Millstein, T.; and Weld, D. Automatic SAT-compilation of planning problems. *Proc. IJCAI-97*.

Giunchiglia, E., and Lifschitz, V. An action language based on causal explanation: Preliminary report. *Proc. AAAI-98*, 623–630.

Giunchiglia, E.; Giunchiglia, F.; Sebastiani, R.; and Tacchella, A. More evaluation of decision procedures for modal logics. *Proc. KR-98*.

Giunchiglia, E.; Massarotto, A.; and Sebastiani, R. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. *Proc. AAAI-98*.

Giunchiglia, E. 2000. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. *Proc. KR-00*.

Giunchiglia, E., and Tacchella, A. *SAT: a system for the development of modal decision procedures. In *Proc. CADE-00*.

Gomes, C. P.; Selman, B.; and Kautz, H. Boosting combinatorial search through randomization. *Proc. AAAI-98*, 431–437.

Kautz, H., and Selman, B. Planning as satisfiability. *Proc. ECAI-92*, 359–363.

Kautz, H., and Selman, B. Pushing the envelope: planning, propositional logic and stochastic search. *Proc. AAAI-96*, 1194–1201.

Kautz, H., and Selman, B. BLACKBOX: A new approach to the application of theorem proving to problem solving. *Working notes of the AIPS-98 Workshop on Planning as Combinatorial Search*.

Kautz, H.; McAllester, D.; and Selman, B. Encoding plans in propositional logic. *Proc. KR-96*, 374–384.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1-2):239–286.

Peot, M. 1998. *Decision-Theoretic Planning*. Ph.D. Dissertation, Stanford University. Dept. of Engineering-Economic Systems.

Smith, D., and Weld, D. Conformant graphplan. *Proc. AAAI-98*, 889–896.

Wolfman, S., and Weld, D. The LPSAT-engine & its application to resource planning. *Proc. IJCAI-99*.