SAT-Based planning in complex domains: Concurrency, Constraints and Nondeterminism

Claudio Castellini Enrico Giunchiglia
Armando Tacchella
DIST — Università di Genova
Viale Causa 13, 16145 Genova, Italy

Abstract

Planning as satisfiability is a very efficient technique for classical planning, i.e., for planning domains in which both the effects of actions and the initial state are completely specified. In this paper we present $\mathcal{C}$-SAT, a SAT-based procedure capable of dealing with planning domains having incomplete information about the initial state, and whose underlying transition system is specified using the highly expressive action language $\mathcal{C}$. Thus, $\mathcal{C}$-SAT allows for planning in domains involving (*i*) actions which can be executed concurrently; (*ii*) (ramification and qualification) constraints affecting the effects of actions; and (*iii*) nondeterminism in the initial state and in the effects of actions. We first prove the correctness and the completeness of $\mathcal{C}$-SAT, discuss some optimizations, and then we present $\mathcal{C}$-PLAN, a system based on $\mathcal{C}$-SAT. $\mathcal{C}$-PLAN works on any $\mathcal{C}$ planning problem, but some optimizations have not been fully implemented yet. Nevertheless, the experimental analysis shows that SAT-based approaches to planning with incomplete information are viable, at least in the case of problems with a high degree of parallelism.

1 Introduction

Propositional reasoning is a fundamental problem in many areas of Computer Science. Many researchers have put, and still put, years of effort in the design and implementation of new and more powerful SAT solvers. Moreover, the source code of many of these solvers is freely available and can be used as the core engine of systems able to deal with more complex tasks. For example, in [Giunchiglia *et al.*, 2000] a state-of-the-art SAT solver is used as the basis for the development of 8 efficient decision procedures for classical modal logics. In [Cadoli *et al.*, 1998], a SAT solver is at the basis of a decider able to deal with Quantified Boolean Formulas (QBFs). In [Kautz and Selman, 1998], a SAT solver is used as the search engine for a very efficient planning system able to deal with STRIPS action descriptions and a completely specified initial state.

In [Biere *et al.*, 1999, Copty *et al.*, 2001], SAT solvers are effectively applied to verify complex industrial designs.

In this paper we present $\mathcal{C}$-SAT, a SAT-based procedure for checking the existence of plans of length n in $\mathcal{C}$ [Giunchiglia and Lifschitz, 1998] domains with incomplete information about the initial state. Notice that for any finite action description in any Boolean action language [Lifschitz, 1997], there is an equivalent one —i.e., having the same transition diagram— specified in $\mathcal{C}$. Thus, $\mathcal{C}$-SAT is fully general, and allows to consider domains involving, e.g.,

- actions which can be executed concurrently,
- (ramification and/or qualification) constraints affecting the effects of actions [Lin and Reiter, 1994], and
- nondeterminism in the initial state and/or in the effects of actions.

Because of nondeterminism, $\mathcal{C}$-SAT employs a generate and test approach. The generate and test steps correspond to satisfiability and validity checks respectively, and both are performed using a procedure built on top of a SAT solver. In the case of deterministic planning domains, at most one plan is generated and then proved valid. Notice the relationship to situation-calculus type planning in first order logic (FOL) [Green, 1969], and to planning via Quantified Boolean Formulas (QBFs) [Rintanen, 1999a]. In both FOL and QBF reasoning, it is possible to combine plan generation and verification in a single validity check. In FOL, the plan is retrieved by extracting variable bindings. In QBF reasoning, the plan corresponds to the assignment to the variables corresponding to actions.

We first prove the correctness and the completeness of $\mathcal{C}$-SAT, discuss some optimizations, and then we present $\mathcal{C}$-PLAN, a planning system incorporating $\mathcal{C}$-SAT as the core engine. In order to have optimality of the returned plan, $\mathcal{C}$-PLAN repeatedly calls $\mathcal{C}$-SAT, checking for the existence of plans of increasing length. Our goal in developing $\mathcal{C}$-SAT and $\mathcal{C}$-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, e.g., concurrency and/or constraints and/or nondeterminism. The experimental analysis shows that this is indeed the case, at least in the case of problems with a high degree of parallelism.

The paper is structured as follows. In Section 2 we briefly review the action language $\mathcal{C}$ and show a simple example that will be used throughout the whole paper. In Section 3 we state the formal results laying the grounds to the proposed procedure, which is presented and proved correct and complete in Section 4. Some optimizations are discussed in Section 5. Then, in Section 6 we show the structure of a system incorporating the proposed ideas and present some experimental analysis. We end the paper with the conclusions in Section 7.

2 Action language $\mathcal{C}$

2.1 Syntax and Semantics

We start with a set of atoms partitioned into the set of *fluent symbols* and the set of *action symbols*. A *formula* is a propositional combination of atoms. An *action* is an interpretation of the action symbols. Intuitively, to execute an action α means to execute concurrently the "elementary actions" represented by the action symbols satisfied by α.

An *action description* is a set of

- *static laws*, of the form:

$$\textbf{caused } F \textbf{ if } G, \tag{1}$$

- and *dynamic laws*, of the form:

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H, \tag{2}$$

where F, G, H are formulas such that F and G do not contain action symbols. Both in (1) and in (2), F is called the *head* of the law.

Consider an action description D. A *state* is an interpretation of the fluent symbols that satisfies $G \supset F$ for every static law (1) in D. A *transition* is a triple $\langle\sigma, \alpha, \sigma'\rangle$ where σ, σ' are states and α is an action; intuitively σ is the initial state of the transition and σ' is its resulting state. A formula F is *caused* in a transition $\langle\sigma, \alpha, \sigma'\rangle$ if it is

- the head of a static law (1) in D such that σ' satisfies G, or
- the head of a dynamic law (2) in D such that σ' satisfies G and $\sigma \cup \alpha$ satisfies H.

A transition $\langle\sigma, \alpha, \sigma'\rangle$ is *causally explained* in D if its resulting state σ' is the only interpretation of the fluent symbols that satisfies all formulas caused in this transition.

The *transition diagram* represented by an action description D is the directed graph which has the states of D as vertices, and which includes an edge from σ to σ' labeled α for every transition $\langle\sigma, \alpha, \sigma'\rangle$ that is causally explained in D.

2.2 An example

In rules (2), we do not write "**if** G" when G is a tautology.

Consider the following elaboration of the "safe from baby" example [Myers and Smith, 1988, Giunchiglia and Lifschitz, 1995]. There are a box and a baby crawling on the floor. The box is not dangerous for the baby if it contains dolls or if it is on the table. Otherwise, it is dangerous (e.g., because it contains hammers). We introduce the three fluent symbols *Safe*, *OnTable*, *Dolls*, and the static rules

$$\begin{array}{c}\textbf{caused } \mathit{Safe} \textbf{ if } \mathit{OnTable} \lor \mathit{Dolls}, \\ \textbf{caused } \neg\mathit{Safe} \textbf{ if } \neg\mathit{OnTable} \land \neg\mathit{Dolls}.\end{array} \tag{3}$$

The box can be moved by the mother or the father of the baby. The action symbols are *MPutOnTable*, *FPutOnTable*, *MPutOnFloor*, *FPutOnFloor*. The direct effects of actions are defined by the following dynamic rules:

$$\begin{array}{l} \textbf{caused } \mathit{OnTable} \textbf{ after } \mathit{MPutOnTable} \vee \mathit{FPutOnTable}, \\ \textbf{caused } \neg \mathit{OnTable} \textbf{ after } \mathit{MPutOnFloor} \vee \mathit{FPutOnFloor}. \end{array} \tag{4}$$

However, if the box does not contain dolls (e.g., if it is full of hammers), it can be moved on the table only by the mother and the father concurrently:

$$\textbf{caused } \mathit{False} \textbf{ after } \neg \mathit{Dolls} \wedge \neg(\mathit{MPutOnTable} \equiv \mathit{FPutOnTable}), \tag{5}$$

where *False* represents falsehood. Technically, if P is an arbitrarily chosen fluent symbol, the above rule is an abbreviation for the two rules obtained from (5) substituting first P and then $\neg P$ for *False*. All the fluents but *Safe* are inertial.[1] This last fact is expressed by a pair of dynamic rules of the form

$$\begin{array}{c} \textbf{caused } P \textbf{ if } P \textbf{ after } P, \\ \textbf{caused } \neg P \textbf{ if } \neg P \textbf{ after } \neg P, \end{array} \tag{6}$$

for each fluent P different from *Safe* [McCain and Turner, 1997].

The transition diagram of the action description consisting of (3)–(6) is depicted in Figure 1. Both in the Figure and in the rest of the paper, an action α [resp. a state σ] is represented by the set of action symbols [resp. fluents] satisfied by α [resp. σ].

Consider Figure 1. As it can be observed, the transition system consists of two separated subsystems. In the first (upper in the Figure), the baby is safe no matter the location of the box. This is the case when the box is full of dolls. In the other (lower in the Figure), the baby is safe only when the box is on the table. The example shows only a few of the many expressive capabilities that $\mathcal{C}$ has. For example, the rules in (3) play the role of ramification constraints [Lin and Reiter, 1994]: Moving the box on the table has the indirect effect of making the baby safe. (5) is a generalization of the traditional action preconditions from the STRIPS literature: Here an action is a set of elementary actions, and *Dolls* is a precondition for the execution of $\{\mathit{MPutOnTable}\}$ and $\{\mathit{FPutOnTable}\}$. The semantics of $\mathcal{C}$ takes into account the fact that several elementary actions can be executed concurrently. Besides this, $\mathcal{C}$ allows also, e.g., for expressing qualification constraints, fluents that change by themselves, actions with nondeterministic effects. See [Giunchiglia and Lifschitz, 1998] for more details and examples.

[1] Intuitively, saying that a fluent is inertial corresponds to saying that by default it keeps its truth value after the execution of an action. In our example, if we say that also *Safe* is inertial, the transition diagram associated to the description does not change. However, in general, not all fluents are inertial, and adding the inertiality default for defined fluents (i.e., for fluents whose truth value is determined by the truth values of the others, like *Safe*) may lead to unwanted conclusions. See, for example, Lifschitz' two switches example [Lifschitz, 1990], and Section "Noninertial Fluents" in [Giunchiglia and Lifschitz, 1998] for more details.

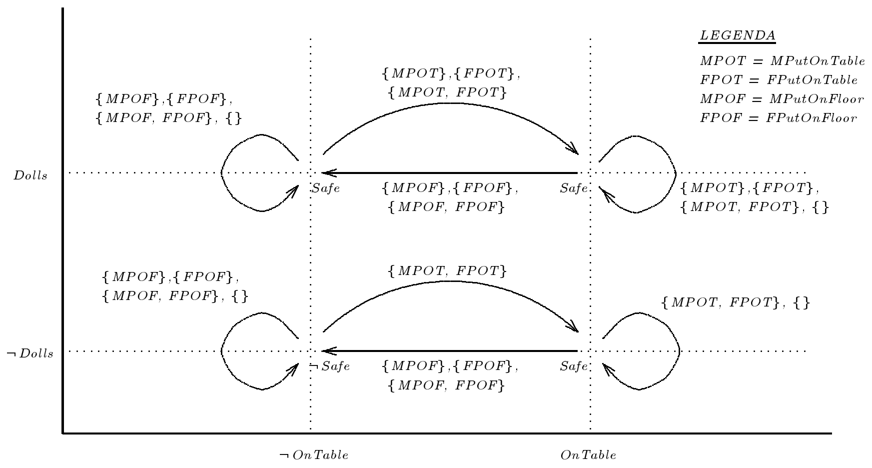

Figure 1: The transition diagram for (3)–(6).

2.3 Computing causally explained transitions

An action description is *finite* if its signature is finite and it consists of finitely many laws. For any finite action description D, it is possible to compute a propositional formula tr_i^D, called the *transition relation* of D, whose satisfying assignments correspond to the causally explained transitions of D [Giunchiglia and Lifschitz, 1998]. To make the computation of tr_i^D easier to present, we restrict to finite action descriptions in which the head of the rules is a literal ([Giunchiglia and Lifschitz, 1998] describes how to compute tr_i^D in the general case). In this case, we say that the action description is *definite*.

Consider a definite action description D. For any number i and any formula H in the signature of D, H_i is the expression obtained from H by substituting each atom B with B_i. Intuitively, the subscript i represents time. If P is a fluent symbol, the atom P_i expresses that P holds at time i. If A is an action symbol, the atom A_i expresses that A is among the elementary actions executed at time i. In the following, we abbreviate the static law (1) with $\langle F, G\rangle$, and the dynamic law (2) with $\langle F, G, H\rangle$. tr_i^D is the conjunction of

- for each fluent literal F, the formula

$$F_{i+1} \equiv \bigvee_{G:\langle F,G\rangle\in D} G_{i+1} \vee \bigvee_{G,H:\langle F,G,H\rangle\in D} G_{i+1} \wedge H_i,$$

- for each static causal law $\langle F, G\rangle$ in D, the formula

$$G_i \supset F_i.$$

For example, if D consists of (3)–(6), tr_i^D is equivalent to the conjunction of the formulas:

$$
\begin{array}{c}
Dolls_{i+1} \equiv Dolls_i, \\
Safe_i \equiv OnTable_i \vee Dolls_i, \\
Safe_{i+1} \equiv OnTable_{i+1} \vee Dolls_{i+1}, \\
OnTable_{i+1} \equiv MPutOnTable_i \vee FPutOnTable_i \vee \\
OnTable_i \wedge \neg MPutOnFloor_i \wedge \neg FPutOnFloor_i, \\
(MPutOnTable_i \vee FPutOnTable_i) \supset \neg(MPutOnFloor_i \vee FPutOnFloor_i), \\
\neg Dolls_i \supset (MPutOnTable_i \equiv FPutOnTable_i).
\end{array} \tag{7}
$$

In the rest of the paper, tr_i^D is the formula defined as above if D is definite, and the formula defined analogously to $ct_1(D)$ in [Giunchiglia and Lifschitz, 1998], otherwise. Furthermore, we will identify any interpretation μ of D with the conjunction of the literals satisfied by μ. Thus, given also the previous notational convention, if μ is an assignment and i is a natural number, μ_i has to be understood as $\bigwedge_{L:L \text{ is a literal}, \mu \models L} L_i$.

Proposition 1 ([Giunchiglia and Lifschitz, 1998]) *Let D be a finite action description. Let $\langle \sigma, \alpha, \sigma' \rangle$ be a transition of D. $\langle \sigma, \alpha, \sigma' \rangle$ is causally explained in D iff $\sigma_i \wedge \alpha_i \wedge \sigma'_{i+1}$ entails tr_i^D.*

3 Possible plans and Valid plans

A *history* for an action description D is a path in the corresponding transition diagram, that is, a finite sequence

$$\sigma^0, \alpha^1, \sigma^1, \ldots, \alpha^n, \sigma^n \tag{8}$$

$(n \geq 0)$ such that $\sigma^0, \sigma^1, \ldots, \sigma^n$ are states, $\alpha^1, \ldots, \alpha^n$ are actions, and

$$\langle \sigma^{i-1}, \alpha^i, \sigma^i \rangle \qquad (1 \leq i \leq n)$$

are causally explained transitions of D. n is the *length* of the history (8).

Let D be a finite action description. Consider D. As an easy consequence of Proposition 1, we get the following proposition that will be used later.

Proposition 2 *Let D be a finite action description. If $\sigma^0, \sigma^1, \ldots, \sigma^n$ are states, $\alpha^1, \ldots, \alpha^n$ are actions $(n \geq 0)$, then (8) is a history iff $\wedge_{i=0}^{n-1}(\sigma_i^i \wedge \alpha_i^{i+1}) \wedge \sigma_n$ entails $\wedge_{i=0}^{n-1} tr_i^D$.*

Thus, an assignment satisfying $\wedge_{i=0}^{n-1} tr_i^D$ corresponds to a history and vice versa. In the above Proposition, superscripts to states/actions denote the particular state/action, and subscripts denote a time step. For example α_i^{i+1} denotes the $(i+1)$-th action at the i-th time step.

A *planning problem* for D is characterized by two formulas I and G in the fluent signature, i.e., it is a triple $\langle I, D, G \rangle$. A state σ is *initial* [resp. *goal*] if σ satisfies I [resp. G]. A planning problem $\langle I, D, G \rangle$ is *deterministic* if

- there is only one initial state, and
- for any state σ and action α, there is at most one causally explained transition $\langle \sigma, \alpha, \sigma' \rangle$ in D.

A *plan* is a finite sequence $\alpha^1; \ldots; \alpha^n$ $(n \geq 0)$ of actions.

3.1 Possible plans

Consider a planning problem $\pi = \langle I, D, G \rangle$. A plan $\alpha^1; \ldots; \alpha^n$ is *possible* for π if there exists a history (8) for D such that σ^0 is an initial state, and σ^n is a goal state. For example, the plan consisting of the empty sequence of actions is possible for the planning problem

$$\langle \neg \mathit{OnTable}, (3)\text{–}(6), \mathit{Safe} \rangle. \tag{9}$$

In fact, there exists an initial state which is also a goal state.

The following Theorem is similar to Proposition 2 in [McCain and Turner, 1998], and is an easy consequence of Proposition 2 in this paper.

Theorem 1 *Let $\pi = \langle I, D, G \rangle$ be a planning problem. A plan $\alpha^1; \ldots; \alpha^n$ is possible for π iff the formula*

$$\wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge I_0 \wedge \wedge_{i=0}^{n-1} tr_i^D \wedge G_n$$

is satisfiable.

The execution of a possible plan is not ensured to lead to a goal state. Indeed, if D is deterministic and there is only one initial state, then executing a possible plan leads to a goal state. This is the idea underlying planning as satisfiability in [Kautz and Selman, 1992]. However, this is not always the case in our setting, where actions can be nondeterministic and there can be multiple initial states. For example, considering the planning problem (9), executing the possible plan consisting of the empty sequence of actions is not ensured to lead to a goal state: In fact, there is an initial state which is not a goal state.

3.2 Valid Plans

As pointed out in [McCain and Turner, 1998], in order to be sure that a plan $\alpha^1; \ldots; \alpha^n$ is good (they say "valid"), it is not enough to check that for any history (8) such that σ^0 is an initial state, σ^n is a goal state. According to this definition, the plan $\{\mathit{MPutOnTable}\}$ would be valid for the planning problem (9). Indeed, $\{\mathit{MPutOnTable}\}$ is not valid since this action is not "executable" in the initial states satisfying $\neg \mathit{Dolls}$. Intuitively, we have to check that the plan is also "always executable" in any initial state, i.e., executable for any initial state and any possible outcome of the actions in the plan. To make the notion of valid plan precise, we need the following definitions.

Consider a finite action description D and a plan $\vec{\alpha} = \alpha^1; \ldots; \alpha^n$.

An action α is *executable* in a state σ if for some state σ', $\langle \sigma, \alpha, \sigma' \rangle$ is a causally explained transition of D. Let σ^0 be a state. The plan $\vec{\alpha}$ is *always executable in* σ^0 if for any history

$$\sigma^0, \alpha^1, \sigma^1, \ldots, \alpha^k, \sigma^k \tag{10}$$

with $k < n$, α^{k+1} is executable in σ^k. For example, if D consists of (3)–(6), the plan $\{\mathit{MPutOnFloor}\}; \{\mathit{FPutOnFloor}\}$ is always executable in any state, while the plan $\{\mathit{MPutOnFloor}\}; \{\mathit{FPutOnTable}\}$ is always executable only in the states satisfying Dolls.

Assume that $\vec{\alpha}$ is a plan which is always executable in a state σ^0. A state σ^n is a *possible result of executing* $\vec{\alpha}$ *in* σ^0 if there exists a history (8) for D. For example, in the case of (3)–(6), the state $\{\}$ (i.e., the state which does not satisfy any fluent symbol) is a possible result of executing $\{\mathit{MPutOnFloor}\}; \{\mathit{FPutOnFloor}\}$ in any state satisfying $\neg \mathit{Dolls}$.

Let $\pi = \langle I, D, G \rangle$ be a planning problem. A plan $\vec{\alpha} = \alpha^1; \ldots; \alpha^n$ is *valid* for π if for any initial state σ^0,

- $\vec{\alpha}$ is always executable in σ^0, and
- any possible result of executing $\vec{\alpha}$ in σ^0 is a goal state.

Considering the planning problem (9), the plan $\{\mathit{MPutOnTable}\}$ is not valid, while $\{\mathit{MPutOnTable}, \mathit{FPutOnTable}\}$ is valid. Notice that valid plans correspond to "conformant" plans in [Smith and Weld, 1998]. In the following, we use the terms "conformant" and "valid" without distinction.

Proposition 3 *Let $\pi = \langle I, D, G \rangle$ be a planning problem. Let $\vec{\alpha} = \alpha^1; \ldots; \alpha^n$ be a plan which is always executable in any initial state. $\vec{\alpha}$ is valid for π iff*

$$I_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} tr_i^D \models G_n. \tag{11}$$

Proof of Proposition 3: $\vec{\alpha}$ is always executable in any initial state by hypothesis. Thus, $\vec{\alpha}$ is valid iff (by definition) for any history (8) such that σ^0 is an initial state (i.e., σ^0 satisfies I), σ^n is a goal state (i.e., σ^n satisfies G).

We consider the two directions separately.

$\Rightarrow$) $\vec{\alpha}$ is valid. Let μ be an interpretation that satisfies $I_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} tr_i^D$. By Proposition 2, there exists a corresponding history (8). Indeed, since $\vec{\alpha}$ is valid, σ^n satisfies G, and thus σ_n^n entails G_n.

$\Leftarrow$) Assume $\vec{\alpha}$ is not valid and that (11) holds. Since $\vec{\alpha}$ is not valid there exists a history (8) in which

- σ^0 is an initial state, i.e., σ^0 satisfies I, and
- σ^n is not a goal state, i.e., σ^n does not satisfy G.

By Proposition 2, to this history there corresponds an assignment μ satisfying $I_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} tr_i^D$. However, since (11) holds, μ satisfies G_n, which contradicts the fact that σ^n does not satisfy G. $\square$

Thus, if a plan $\vec{\alpha}$ is always executable in any initial state, Proposition 3 establishes a necessary and sufficient condition for determining whether $\vec{\alpha}$ is valid. Our next step is to define, on the basis of tr_i^D, the transition relation trt_i^D of a new automaton in which every action is always executable, thus enabling us to use Proposition 3: Intuitively, assuming that an action α is not executable in a state σ, we add transitions leading from σ with label α to "bad" states. A state σ is bad if goal states are not reachable from σ. In order to do this, we first need to know when an action is executable in a state. Let $Poss_i^D$ be the formula

$$\exists p^1 \ldots \exists p^n \, tr_i^D[P_{i+1}^1/p^1, \ldots, P_{i+1}^n/p^n] \tag{12}$$

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.

Proposition 4 *Let α be an action and let σ be a state of a finite action description D. α is executable in σ iff $\sigma_i \wedge \alpha_i$ entails $Poss_i^D$.*

Proof of Proposition 4: Let $Trans^D$ be the set of causally explained transitions of D. By Proposition 1, tr_i^D is logically equivalent to

$$\bigvee_{\sigma,\alpha,\sigma' : \langle \sigma,\alpha,\sigma' \rangle \in Trans^D} (\sigma_i \wedge \alpha_i \wedge \sigma'_{i+1})$$

Thus, $tr_i^D[P_{i+1}^1/p^1, \ldots, P_{i+1}^n/p^n]$ is

$$\bigvee_{\sigma,\alpha,\sigma' : \langle \sigma,\alpha,\sigma' \rangle \in Trans^D} (\sigma_i \wedge \alpha_i \wedge \sigma'_{i+1}[P_{i+1}^1/p^1, \ldots, P_{i+1}^n/p^n]),$$

and then the existential closure of the above formula is equivalent to

$$\bigvee_{\sigma,\alpha : \exists \sigma' \langle \sigma,\alpha,\sigma' \rangle \in Trans^D} (\sigma_i \wedge \alpha_i)$$

whence the thesis. □

Notice that the propositional variables and the bounding quantifiers can always be eliminated in (12), although the formula can become much longer in the process. However, if for each action we have its preconditions explicitly listed (as, e.g., in STRIPS), it is possible to compute the propositional formula equivalent to (12) by simple syntactic manipulations, see [Ferraris and Giunchiglia, 2000]. For example, if tr_i^D is the conjunction of the formulas (7), then $Poss_i^D$ is equivalent to the conjunction of the following formulas:

$$\begin{gathered} Safe_i \equiv OnTable_i \vee Dolls_i, \\ (MPutOnTable_i \vee FPutOnTable_i) \supset \neg(MPutOnFloor_i \vee FPutOnFloor_i), \\ \neg Dolls_i \supset (MPutOnTable_i \equiv FPutOnTable_i). \end{gathered} \tag{13}$$

From (13) follows that in the action description (3)–(6) the actions $\{MPutOnTable\}$ and $\{FPutOnTable\}$ are not executable in states satisfying $\neg Dolls$.

We can now define the new transition relation trt_i^D as the formula

$$(tr_i^D \wedge \neg Z_i \wedge \neg Z_{i+1}) \vee (\neg Poss_i^D \wedge Z_{i+1}) \vee (Z_i \wedge Z_{i+1}), \tag{14}$$

where Z is a newly introduced fluent symbol. Given that $tr_i^D \supset Poss_i^D$ holds, (14) is logically equivalent to

$$(\neg Z_i \vee Z_{i+1}) \wedge (tr_i^D \vee Z_{i+1}) \wedge (\neg Poss_i^D \vee Z_i \vee \neg Z_{i+1}). \tag{15}$$

Intuitively, if s^f is the fluent signature of D, trt_i^D determines (in the sense of Proposition 1) the transition relation of an automaton

- whose set of states corresponds to the set of assignments of the signature $s^f \cup \{Z\}$, 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 α if and only if
  - σ and σ' satisfy $\neg Z$ and $\langle \sigma_D, \alpha, \sigma'_D \rangle$ is a causally explained transition of D, or
  - σ satisfies $\neg Z$, σ' satisfies Z and α is not executable in σ_D, or
  - σ and σ' satisfy Z,

  where σ_D is the restriction of σ to s^f (and similarly for σ'_D).

The above intuition is made precise by the following Proposition.

Proposition 5 *Let D be a finite action description. Let σ, σ' be two states of D, and let α be an action. The following three facts hold:*

1. *$\langle \sigma, \alpha, \sigma' \rangle$ is a causally explained interpretation of D iff $\sigma_i \wedge \neg Z_i \wedge \alpha_i \wedge \sigma'_{i+1} \wedge \neg Z_{i+1}$ entails (14).*
2. *α is not executable in σ iff $\sigma_i \wedge \neg Z_i \wedge \alpha_i \wedge Z_{i+1}$ entails (14).*
3. *(14) entails $Z_i \supset Z_{i+1}$.*

Proof of Proposition 5: The first two items follows by Propositions 1, 4. The last item is an easy consequence of the fact that (14) is logically equivalent to (15). □

In the following Theorem, $State_0^D$ is the formula

$$\bigwedge_{F,G:\langle F,G \rangle \in D} G_0 \supset F_0,$$

representing the set of "possible initial states". If D is (3)–(6), then $State_0^D$ is equivalent to

$$Safe_0 \equiv OnTable_0 \vee Dolls_0.$$

Theorem 2 *Let D be a finite action description. A plan $\alpha^1;\ldots;\alpha^n$ is valid for a planning problem $\langle I, D, G\rangle$ iff*

$$I_0 \wedge State_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1}\alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} trt_i^D \models G_n \wedge \neg Z_n. \qquad (16)$$

Proof of Theorem 2: We consider the two directions separately. Let $\vec{\alpha} = \alpha^1;\ldots;\alpha^n$.

$\Rightarrow$) $\vec{\alpha}$ is valid by hypothesis. Let μ be an interpretation that satisfies $I_0 \wedge State_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1}\alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} trt_i^D$. Since $\vec{\alpha}$ is valid, for any history (10) with $k < n$ α^{k+1} is executable in σ^k. Thus (Proposition 4), μ satisfies $Poss_k^D$, for each $k < n$. Since μ satisfies $\wedge_{i=0}^{n-1} trt_i^D \wedge \neg Z_0$, and trt_i^D is equivalent to (15), it follows that μ satisfies $\neg Z_1, \ldots, \neg Z_n$ and $\wedge_{i=0}^{n-1} tr_i^D$. If μ satisfies $\wedge_{i=0}^{n-1} tr_i^D$, by Proposition 2, there exists a corresponding history (8). Thus, since $\vec{\alpha}$ is valid, σ^n satisfies G and μ satisfies G_n. The thesis follows from the fact that μ can be arbitrarily chosen.

$\Leftarrow$) Assume $\vec{\alpha}$ is not valid and that (16) holds. Since $\vec{\alpha}$ is not valid then there exists some initial state σ^0 such that

1. $\vec{\alpha}$ is not always executable in σ^0, or
2. a possible result of executing $\vec{\alpha}$ in σ^0 is not a goal state.

We consider only the first case: The proof in the second case can be done along the lines of the proof of Proposition 3 (notice that —once we have proved the first case— we can assume that $\vec{\alpha}$ is always executable in σ_0). If $\vec{\alpha}$ is not always executable in σ^0, then there exists a history (10) with $k < n$ such that α^{k+1} is not executable in σ^k. Let μ be the assignment to the variables in (16) defined by

$$\mu(F_i) = \begin{cases} \sigma^i(F) & \text{if } F \text{ is a fluent symbol of } D \text{ and } i \leq k, \\ \sigma^k(F) & \text{if } F \text{ is a fluent symbol of } D \text{ and } k < i \leq n, \\ \alpha^{i+1}(F) & \text{if } F \text{ is an action symbol of } D \text{ and } i \leq k, \\ \alpha^{k+1}(F) & \text{if } F \text{ is an action symbol of } D \text{ and } k < i < n, \\ \mathit{False} & \text{if } F_i = Z_i \text{ and } i \leq k, \\ \mathit{True} & \text{if } F_i = Z_i \text{ and } k < i \leq n. \end{cases}$$

By construction, μ satisfies $I_0 \wedge State_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1}\alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} trt_i^D$. However, by construction μ also satisfies Z_n, which contradicts the hypotheses. ☐

Considering the planning problem (9), Theorem 2 can be used, e.g., to establish that the plans $\{\mathit{MPutOnTable}\}$,$\{\mathit{MPutOnTable},\mathit{FPutOnTable}\}$ are respectively not valid and valid. To check it, consider the formula

$$I_0 \wedge State_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1}\alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} trt_i^D. \qquad (17)$$

In both cases $n = 1$. But,

- When $\alpha^1 = \{MPutOnTable\}$, (17) is equivalent to the conjunction of the formulas

$$\begin{gathered}\neg OnTable_0,\\ Safe_0 \equiv Dolls_0,\\ \neg Z_0,\\ MPutOnTable \wedge \neg FPutOnTable \wedge \neg MPutOnFloor \wedge \neg FPutOnFloor,\end{gathered}$$

  and the formulas

$$\begin{gathered}Safe_1 \vee Z_1,\\ Dolls_0 \vee Z_1,\\ Dolls_1 \vee Z_1,\\ OnTable_1 \vee Z_1,\\ \neg Dolls_0 \vee \neg Z_1.\end{gathered}$$

  This conjunction does not entail $Safe_1 \wedge \neg Z_1$. Indeed, $\{MPutOnTable\}$ is a valid plan for (9) if *Dolls* initially holds.

- When $\alpha^1 = \{MPutOnTable, FPutOnTable\}$, (17) is equivalent to the conjunction of

$$\begin{gathered}\neg OnTable_0,\\ Safe_0 \equiv Dolls_0,\\ \neg Z_0,\\ MPutOnTable \wedge FPutOnTable \wedge \neg MPutOnFloor \wedge \neg FPutOnFloor,\end{gathered}$$

  and the formulas

$$\begin{gathered}Safe_1,\\ OnTable_1,\\ Dolls_1 \equiv Dolls_0,\\ \neg Z_1.\end{gathered}$$

  This conjunction obviously entails $Safe_1 \wedge \neg Z_1$.

4 Computing valid plans in $\mathcal{C}$

Consider a planning problem $\pi = \langle I, D, G\rangle$. Thanks to Theorem 2, we may divide the problem of finding a valid plan for π into two parts:

1. *generate* a (possible) plan, and
2. *test* whether the generated plan is also valid.

The testing phase can be performed using any state-of-the-art complete SAT solver. According to Theorem 2, a plan $\alpha^1;\ldots;\alpha^n$ is valid if and only if

$$I_0 \wedge State_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1}\alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} trt_i^D \wedge \neg(G_n \wedge \neg Z_n) \qquad (18)$$

is not satisfiable. For the generation phase, different strategies can be used:

$P := I_0 \wedge \wedge_{i=0}^{n-1} tr_i^D \wedge G_n$;
$V := I_0 \wedge State_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} trt_i^D \wedge \neg(G_n \wedge \neg Z_n)$;

function $\mathcal{C}$-SAT()
  return $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$($cnf(P)$, {}).

function $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$(φ, μ)
  if $\varphi = \{\}$ **then return** $\mathcal{C}$-SAT_TEST(μ); /* base */
  if $\{\} \in \varphi$ **then return** *False*; /* backtrack */
  if { a unit clause $\{L\}$ occurs in φ } /* unit */
    then return $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$($assign(L, \varphi)$,$\mu \cup \{L\}$);
  L := { a literal occurring in φ };
  return $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$($assign(L, \varphi)$,$\mu \cup \{L\}$) **or** /* split */
      $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$($assign(\overline{L}, \varphi)$,$\mu \cup \{\overline{L}\}$).

function $\mathcal{C}$-SAT_TEST(μ)
  α := { the set of literals in μ corresponding to action literals};
  foreach {plan $\alpha^1; \ldots; \alpha^n$ s.t. each element in α is a conjunct in $\wedge_{i=0}^{n-1} \alpha_i^{i+1}$}
    if not SAT($\wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge V$) **then exit with** $\alpha^1; \ldots; \alpha^n$;
  return *False*.

Figure 2: $\mathcal{C}$-SAT, $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$ and $\mathcal{C}$-SAT_TEST.

1. generation of arbitrary plans of length n,
2. generation of a subset of the possible plans of length n,
3. generation of the whole set of (possible) plans of length n.

By checking the validity of each generated plan, we obtain a correct but possibly incomplete procedure in the first two cases; and a correct and complete procedure in the last case. Given a planning problem π and a natural number n, we say that a procedure is

- *correct* (for π, n) if any returned plan $\alpha_1; \ldots; \alpha_n$ is valid for π, and
- *complete* (for π, n) if it returns *False* when there is no valid plan $\alpha_1; \ldots; \alpha_n$ for π.

By Theorem 1, possible plans of length n can be generated by finding assignments satisfying the formula

$$I_0 \wedge \wedge_{i=0}^{n-1} tr_i^D \wedge G_n. \tag{19}$$

This can be accomplished using incomplete SAT solvers like GSAT [Selman *et al.*, 1992], or complete SAT solvers like SATO [Zhang, 1997] or SIM [Giunchiglia *et al.*, 2001]. For the generation of the whole set of possible plans, we may use a complete SAT solver, and

- at step 0, ask for an assignment satisfying (19), and
- at step $i+1$, ask for an assignment satisfying (19) and the negation of the plans generated in the previous steps,

till no more satisfying assignments are found. This method for generating all possible plans has the advantage that the SAT decider is used as a blackbox. The obvious disadvantage is that the size of the input formula checked by the SAT solver may become exponentially bigger than the original one. A better solution is to invoke the test *inside* the SAT procedure whenever a possible plan is found. In the case of the Davis-Logemann-Loveland (DLL) procedure [Davis *et al.*, 1962], we get the procedure $\mathcal{C}$-SAT represented in Figure 2. In the Figure,

- *cnf*(P) is a set of clauses corresponding to P. The transformation from a formula into a set of clauses can be performed using the conversions based on "renaming" (see, e.g., [Tseitin, 1970, Plaisted and Greenbaum, 1986]).
- $\overline{L}$ is the literal complementary to L.
- For any literal L and set of clauses φ, $assign(L, \varphi)$ is the set of clauses obtained from φ by
  - deleting the clauses in which L occurs as a disjunct, and
  - eliminating $\overline{L}$ from the others.

Notice the "for each" iteration in the $\mathcal{C}$-SAT_TEST procedure. Indeed, α may correspond to a partial assignment to the action signature. In order not to miss a possible plan we need to iterate over all the possible total assignments which extend α.

Main Theorem *Let $\pi = \langle I, D, G\rangle$ be a planning problem. Let n be a natural number. $\mathcal{C}$-SAT is correct and complete for π, n.*
Proof of Main Theorem: $\mathcal{C}$-SAT is correct: This is a consequence of Theorem 2. $\mathcal{C}$-SAT is complete: All possible plans are tested for validity. Indeed, if a plan is not possible, it is also not valid. Thus, if there is a valid plan, one will be returned. If there is no valid plan, *False* will be returned. □

5 Optimizations

Consider a planning problem $\pi = \langle I, D, G\rangle$ and a natural number n. The procedure $\mathcal{C}$-SAT in Figure 2 only checks the existence of valid plans of length n. Indeed, even assuming that a plan is returned, we are not guaranteed about its optimality (we say that a plan of length n is *optimal* if it is valid and there is no valid plan of length $< n$). This is a direct consequence of the expressive power of $\mathcal{C}$, in which the nonexistence of a valid plan of length n does not ensure the nonexistence of a valid plan of length $m < n$. Thus, if we want to have a procedure returning optimal plans, we have to consider $n = 0, 1, 2, 3, 4, \ldots$, and for each value of n, call $\mathcal{C}$-SAT, exiting as soon as a valid plan is found. However, the resulting procedure has three weaknesses, listed below:

1. For each n, there may be a huge number of possible plan to be generated and tested. As we will see in the first subsection, it is possible to avoid generating and testing all possible plans for π, at the same time maintaining the correctness and completeness of $\mathcal{C}$-SAT.

2. Considering $\mathcal{C}$-SAT, we see that there is no interaction between the generation (done by $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$) and the testing (done by $\mathcal{C}$-SAT_TEST) phases. $\mathcal{C}$-SAT_TEST, if given a not valid plan, returns *False*, causing $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$ to backtrack to the latest choice point. However, it is well known that backtracking procedures may explore huge portions of the search space without finding a solution because of some wrong choices at the beginning of the search tree. The standard solution in SAT is to incorporate backjumping and learning schemas (see, e.g., [Dechter, 1990, Prosser, 1993, Bayardo, Jr. and Schrag, 1997]). In the second subsection, we show that it is possible to incorporate backjumping and learning also in $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$, thus overcoming the above mentioned problems.

3. There is no re-use of the computation done at different n-s. This is due to the fact that we have a separate run of $\mathcal{C}$-SAT for each value of n. However, we can modify the theory presented in Section 3 and $\mathcal{C}$-SAT in order to return an optimal plan of length $m \leq n$ if there is one, and *False* otherwise. This is the topic of the third and final subsection.

5.1 Eliminating possible plans

At a fixed length n, one drawback of $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$ is that it generates all the possible plans of length n, and there can be exponentially many. However, it is possible to significantly reduce the number of possible plans generated without loosing completeness. The basic idea is to consider only plans which are possible in a "deterministic version" of the original planning problem. In a deterministic version, all the sources of nondeterminism (in the initial state, in the outcome of the actions) are eliminated. The reason why this does not hinder completeness, is an easy consequence of the following proposition.

Proposition 6 *Let $\pi = \langle I, D, G\rangle$, $\pi' = \langle I', D', G'\rangle$ be two planning problems in the same fluent and action signatures, and such that*

1. *every initial state of D' is also an initial state of D, (i.e., $I' \supset I$ is valid),*

2. *for every action α, the set of states of D in which α is executable is a subset of the set of states of D' in which α is executable, (i.e., $Poss^D \supset Poss^{D'}$ is valid),*

3. *every causally explained transition of D' is also a causally explained transition of D, (i.e., $tr^{D'} \supset tr^D$ is valid),*

4. every goal state of π is also a goal state of π' (i.e., $G \supset G'$ is valid).

If a plan is not possible for π' then it is not valid for π.

Proof of Proposition 6: Assume π and π' satisfy the hypotheses of the Proposition. As a direct consequence of the definition of valid plan, we have that any valid plan for π is also a valid plan for π'. The thesis follows from the fact that, for any planning problem (and thus also for π') a valid plan is also a possible plan. □

Consider a planning problem $\pi = \langle I, D, G \rangle$ with possibly many initial states and a nondeterministic action description D.

According to the above Proposition, in $\mathcal{C}$-PLAN we can:

- generate possible plans by considering any planning problem π' satisfying the conditions in the Proposition 6, and
- test whether each of the generated possible plans is indeed valid.

The result is still a correct and complete planning procedure for π, n. Indeed, in choosing π', we want to minimize the set of possible plans generated and then tested. Hence, we want π' to be a "deterministic version of π".

A planning problem $\pi' = \langle I', D', G' \rangle$ is a *deterministic version of* π if the conditions in Proposition 6 are satisfied, and

1. I' is satisfied by a single state,
2. for each action α, the set of states in which α is executable in D is equal to the set of states of D' in which α is executable,
3. for any action α and state σ, there is at most one state σ' such that $\langle \sigma, \alpha, \sigma' \rangle$ is a causally explained transition of D',
4. G is equal to G'.

Of course (unless the planning problem is already deterministic) there are many deterministic versions: Going back to our planning problem (9), we can either choose that the box is full of hammers, or of dolls, (i.e., we can assume that either *Dolls* or $\neg$*Dolls* initially holds). In more complex scenarios there can be exponentially many deterministic versions, and the obvious question is whether there is one which is "best" according to some criterion. If we consider the planning problem (9), we see that assuming that initially *Dolls* holds, would lead to the generation and the test of the possible plan of length 0; while with the assumption $\neg$*Dolls* the first possible plan generated and tested has length 1. Indeed, any valid plan for (9) has length greater or equal to 1. This is not by chance. In fact, let S be the set of deterministic versions of π. For each planning problem π' in S, let $N(\pi')$ be the length of the shortest plan for π'. Then, as an easy consequence of Proposition 6, the length of any valid plan for π is greater or equal to

$$max_{\pi' \in S} N(\pi').$$

On the basis of this fact, we say that $\pi' \in S$ is *better than* $\pi'' \in S$ if

$$N(\pi') \geq N(\pi''). \tag{20}$$

In other words, we prefer the deterministic versions which start to have solutions (each corresponding to a possible plan for the original planning problem) for the biggest possible value of n. In our planning problem (9), this would lead us to choose the deterministic version in which $\neg$*Dolls* holds.

Determining the set of deterministic versions of π is not an easy task in general. Even assuming that the computation of the elements in S is easy, determining for each pair π', π'' of elements in S whether (20) holds, seems impractical. In the following, for simplicity we assume to have nondeterminism only in the initial state. (Analogous considerations hold for actions with nondeterministic effects.) Under this assumption, we modify our $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$ procedure in Figure 2 in order to do the following:

- once an assignment μ satisfying $cnf(P)$ is found, we determine the assignment $\mu' \subseteq \mu$ to the fluent variables at time 0,
- if the possible plan corresponding to μ is not valid, and the planning problem is not already deterministic, then we disallow future assignments extending μ', by adding to I the clause consisting of the complement of the literals satisfied by μ'.

In this way, we progressively eliminate some initial states for which there is a deterministic version having a possible plan of length n. At the end, i.e., when we get to a deterministic planning problem π', π' is a deterministic version of π, and (20) holds for each deterministic version π'' of π.

In (9), the above procedure would do the following:

- At $n = 0$, an assignment satisfying $Dolls_0$ and P will be generated. The corresponding possible plan consisting of the empty sequence of actions will be tested for validity, and rejected. As a consequence, $\neg$*Dolls* will be added to I.
- At $n = 1$, there is only one possible plan for the new planning problem, and this plan is also valid.

5.2 Incorporating Backjumping and Learning

Backjumping and learning are two familiar concepts in constraint satisfaction, and can produce significant speed-ups (see, e.g., [Dechter, 1990, Prosser, 1993, Bayardo, Jr. and Schrag, 1997]). Furthermore, the incorporation of analogous techniques is reported to lead to analogous improvements in plan-graph based planning (see [Kambhampati, 2000]).

We do not enter into the details about how to implement backjumping and learning in SAT, and assume that the reader is familiar with the topic (See [Prosser, 1993, Bayardo, Jr. and Schrag, 1997, Giunchiglia *et al.*, 2001]).

Here we extend the procedures described in [Bayardo, Jr. and Schrag, 1997, Giunchiglia *et al.*, 2001], by adding the rejection of assignments corresponding to possible but not valid plans. Indeed, what we could do —assuming μ is an assignment corresponding to a possible but not valid plan $\vec{\alpha} = \alpha^1; \ldots; \alpha^n$— is to return *False* and set $\vee_{i=0}^{n-1} \neg\alpha_i^{i+1}$ as the initial working reason. However, are there any better choices? According to the definition of valid plan, $\vec{\alpha}$ may be not valid for two reasons:

1. there is a history (10) with $k < n$, σ^0 an initial state, and α^{k+1} is not executable in σ^k, or

2. $\vec{\alpha}$ is always executable in any initial state, but one of the possible outcomes of executing $\vec{\alpha}$ in an initial state is not a goal state.

In both cases, $\mathcal{C}$-SAT_TEST determines an assignment μ' satisfying $\wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge V$, and thus returns *False*. Also notice that in the first case, μ' satisfies $\neg Z_0, \ldots, \neg Z_k, Z_{k+1}, \ldots, Z_n$ with $k < n$. Then we can set $\vee_{i=0}^{k} \neg\alpha_i^{i+1}$ as the initial working reason for rejecting μ: Any assignment satisfying $\wedge_{i=0}^{k} \alpha_i^{i+1}$ does not correspond to a valid plan. Of course, setting $\vee_{i=0}^{k} \neg\alpha_i^{i+1}$ as working reason for rejecting μ is better than setting $\vee_{i=0}^{n-1} \neg\alpha_i^{i+1}$: Since $k < n$ each disjunct in $\vee_{i=0}^{k} \neg\alpha_i^{i+1}$ is also in $\vee_{i=0}^{n-1} \neg\alpha_i^{i+1}$, and, if $k < n-1$, the vice versa is not true.

5.3 Learning from previous attempts for smaller n-s

A third source of inefficiency in our system is that the facts "learned" for smaller n-s are not used for the current value of n. For example, we may discover over and over again that a certain action is not executable after a sequence of other actions. In order to overcome this particular problem, the obvious solution is to add the clauses learned at previous steps (and corresponding to the "initial working reasons" described in previous subsection) also to the current step. Of course, there can be exponentially many such clauses, and this approach does not seem feasible in practice. A much better solution is to avoid searching for valid plans of increasing length. Instead, we may generate possible plans of length $k \leq n$ by satisfying

$$I_0 \wedge \wedge_{i=0}^{n-1} tr_i^D \wedge (\vee_{i=0}^{n} G_i) \tag{21}$$

and then test if for some $k \leq n$ $\alpha^1; \ldots; \alpha^k$ is valid by checking whether

$$I_0 \wedge State_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} trt_i^D \models \vee_{i=0}^{n} (G_i \wedge \neg Z_i). \tag{22}$$

In this way,

- we do not have a distinct run of $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$ for each value of $k \leq n$, and thus clauses learned for $k \leq n$ are naturally maintained and re-used, but

- we have lost optimality: If a plan is returned, it is not guaranteed to be the shortest one.

In order to regain optimality, we proceed as follows. Consider a planning problem $\pi = \langle I, D, G\rangle$, and let tr_i^D be defined as usual.

1. Instead of considering $\langle I, D, G\rangle$, we consider the planning problem $\langle I, D', G\rangle$, where D' is characterized by $tr_i^{D'}$ defined as

$$\begin{array}{c}((tr_i^D \wedge \neg NoOp_i) \vee ((\wedge_{P\in s^f} P_{i+1} \equiv P_i) \wedge NoOp_i)) \\ \wedge(NoOp_i \supset \wedge_{A\in s^a} \neg A_i),\end{array} \tag{23}$$

   where s^f and s^a are, respectively, the fluent and action signatures of D, and $NoOp$ is a newly introduced action symbol. Intuitively, (23) defines (in the sense of Proposition 1) the transition relation of an automaton obtained from the transition system associated to D by adding a transition

$$\langle \sigma, \{NoOp\}, \sigma\rangle$$

   for each state σ of D. Thus, on the basis of $tr_i^{D'}$, the formulas $Poss_i^{D'}$ and $trt_i^{D'}$ are defined as usual, while in the definition of the formulas P/V in Figure 2 we have to replace $tr_i^{D'}$, $trt_i^{D'}$ for tr_i^D, trt_i^D respectively. Finally, in Figure 2, assuming $\alpha^1; \ldots; \alpha^n$ is a plan of D' such that

$$\wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge V$$

   is not satisfiable, then we have to exit with the sequence of actions of D obtained from $\alpha^1; \ldots; \alpha^n$ by removing each α^i such that $\alpha^i(NoOp) = True$. The result is a correct and complete procedure for π, k, with $k \leq n$.

2. To obtain optimality of the returned plan, we have to do some more work. In fact, we have to guarantee that the sequence of possible plans generated and tested corresponds to plans of π of increasing length. In order to do this, we add the clauses

$$\wedge_{i=0}^{n-2}(NoOp_i \supset NoOp_{i+1}) \tag{24}$$

   to the definition of P in Figure 2. Then, we start the generation of the shortest possible plans by forcing $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$ to split first on the literal $NoOp_i$ not yet assigned and with the smallest index i. In this way, we start looking for possible plans satisfying $NoOp_0$, and thus because of (24), also $NoOp_1, \ldots, NoOp_{n-1}$: These possible plans correspond to plans of π having length 0. If π has no possible plan of length 0, backtrack to $NoOp_0$ happens; $\neg NoOp_0$ is set to true and $NoOp_1$ is also set to true because of a splitting step. Again, because of (24), also $NoOp_2, \ldots, NoOp_{n-1}$ are set to true: These possible plans correspond to plans of π having length 1, and the computation proceeds along the same lines.

6 Implementation and Experimental Analysis

We have implemented $\mathcal{C}$-PLAN, a system incorporating the ideas herein described. $\mathcal{C}$-PLAN is still a prototype and is undergoing further developments.

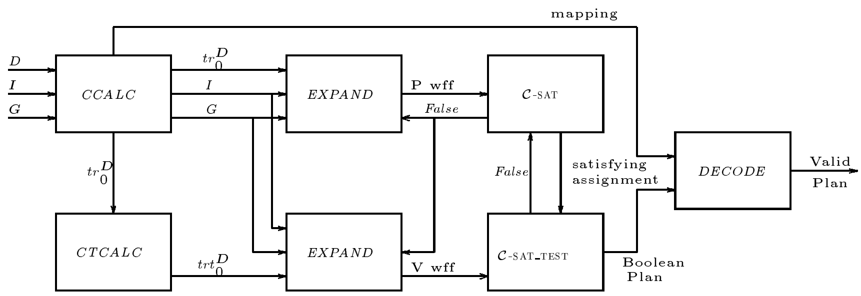

Figure 3: Architecture of $\mathcal{C}$-PLAN

The current version implements the procedures in Figure 2 with the optimizations described in Section 5.1 and in Section 5.2. It also assumes that, given a planning problem $\langle I, D, G\rangle$, each assignment satisfying I is a state of D.[2] Figure 3 shows the overall architecture of $\mathcal{C}$-PLAN. In the Figure, blocks stand for modules of the system, and arrows show the data flow. The input/output behavior of each module is specified by the corresponding labels on the arrows. For the meaning of the labels see also Figure 2.

Consider Figure 3.

CCALC computes the set of clauses corresponding to the transition relation tr_0^D of any (not necessarily definite) action description D. The *CCALC* module has been kindly provided by Norman McCain and is part of the *CCALC* system, see `http://www.cs.utexas.edu/users/tag/cc`.

CTCALC determines, on the basis of tr_0^D, the set of clauses corresponding to trt_0^D. In our current version, *CTCALC* assumes that the set of preconditions of each action is explicit. More precisely, if the actions satisfying α are not executable in the states satisfying H, a dynamic causal law

$$\textbf{caused } \mathit{False} \textbf{ after } H \wedge \alpha$$

has to belong to D. With this assumption, the computation of $Poss_i^D$ and thus of trt_i^D starting from tr_i^D can be done with simple syntactic manipulations, see [Ferraris and Giunchiglia, 2000]. Finally, both *CCALC* and *CTCALC* implement clause form transformations based on renaming (see, e.g., [Plaisted and Greenbaum, 1986]).

$\mathcal{C}$-SAT and $\mathcal{C}$-SAT_TEST implement the algorithms in Figure 2 and the optimizations in Sections 5.1, 5.2. Currently, $\mathcal{C}$-SAT and $\mathcal{C}$-SAT_TEST are implemented on top of SIM [Giunchiglia *et al.*, 2001]. SIM is an efficient library

[2] This is not a restriction. Given any planning problem $\langle I, D, G\rangle$, we can instead consider the planning problem $\langle I', D, G\rangle$ where I' is obtained from I by adding a conjunct $G \supset F$ for each static law (1) in D.

for SAT developed by our group, and features many splitting heuristics and size/relevance learning [Bayardo, Jr. and Schrag, 1997].[3] In all the following experiments, both $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$ and $\mathcal{C}$-SAT_TEST use the unit-based heuristics described in [Li and Anbulagan, 1997] and relevance learning of size 4 [Bayardo, Jr. and Schrag, 1997].

EXPAND is a module generating P/V formulas with a bigger value of n at each step. The lowest and highest values for n to try can be fixed by input parameters.

To evaluate the effectiveness of $\mathcal{C}$-PLAN we consider an elaboration of the traditional "bomb in the toilet" problem from [McDermott, 1987]. There is a finite set P of packages and a finite set T of toilets. One of the packages is armed because it contains a bomb. Dunking a package in a toilet disarms the package and is possible only if the package has not been previously dunked. We first consider planning problems with $|P| = 2, 4, 6, 8, 10, 15, 20$, and $|T| = 1$. We compare our system with

- Bonet and Geffner's GPT [2000]: In GPT, planning as heuristic search is performed in the space of belief states, where a belief state is a set of states or, more in general, a probability distribution over states. Both conformant and contingent planning are possible: In the conformant case, the plan returned is guaranteed to be optimal. One limitation of GPT is that the complexity of some operations performed during the preprocessing and the search scale with the dimension of the state space. As the authors say, if the state space is sufficiently large, GPT does not even get off the ground. GPT has been downloaded from Hector Geffner's web page `http://www.ldc.usb.ve/~hector/`.

- Cimatti and Roveri's CMBP [1999, 2000]: In CMBP, the planning domain is represented as a finite state automaton, and Binary Decision Diagrams (BDDs) [Bryant, 1992] are used to represent and search the automaton. CMBP allows for planning domains specified in the $\mathcal{AR}$ language [Giunchiglia *et al.*, 1997], and it can be very efficient. One limitation of CMBP is that the size of the BDD representation of a formula (standing for a belief state or for the transition relation of the automaton) critically depends on the way variables are ordered in the BDD. Furthermore, in some cases, the size of the BDD may become exponential no matter what ordering is used. CMBP has been downloaded from `http://www.cs.washington.edu/research/jair/contents/v13.html`. CMBP has been run with the option `-ptt` as suggested by the authors during a personal communication.

- Rintanen's QBFPLAN [1999a]: In QBFPLAN, the existence of a conformant plan of length n corresponds, via an encoding, to the satisfiability of a Quantified Boolean Formula whose size polynomially increases with n.

[3] In the previous version, used for the experimental analysis described in [Ferraris and Giunchiglia, 2000], these modules were based on *SAT [Giunchiglia and Tacchella, 2000], and *SAT was based on the SATO SAT solver [Zhang, 1997].

	GPT	CMBP		QBFPLAN			$\mathcal{C}$-PLAN				
$\|P\|$-$\|T\|$	Total	#s	Total	#s	Last	Tot-S	#s	#pp	Last	Tot-S	Total
2-1	0.03	2	0.00	1	0.00	0.00	1	1	0.00	0.00	0.00
4-1	0.03	4	0.01	1	0.00	0.00	1	1	0.00	0.00	0.00
6-1	0.04	6	0.02	1	0.00	0.00	1	1	0.00	0.00	0.00
8-1	0.15	8	0.08	1	0.00	0.00	1	1	0.00	0.00	0.00
10-1	0.27	10	0.61	1	0.00	0.00	1	1	0.00	0.00	0.00
15-1	17.05	15	42.47	1	0.01	0.01	1	1	0.00	0.00	0.00
20-1	MEM	–	MEM	1	0.03	0.03	1	1	0.00	0.00	0.00

Table 1: Bomb in the toilet: Classic version

QBFPLAN uses parallel encoding, i.e., it allows to execute multiple, non-conflicting, elementary actions in a single time step. For optimal (parallel) planning, conformant plans of length $1, 2\ldots$ are searched till one is found, as for $\mathcal{C}$-PLAN. In our experiments, we used the latest version of the solver QSAT [Rintanen, 1999b, Rintanen, 2001], i.e., QSAT v1.0 of February, 27th 2002. Both QBFPLAN and the latest version of QSAT have been kindly provided by the author.

These are among the most recent conformant planners. The encoding of each problem is the same for the four planners, modulo the different representation language. Both $\mathcal{C}$-PLAN and QBFPLAN use a parallel encoding, while CMBP and GPT are sequential planners: They execute one action per time step.

The results for these four systems are shown in Table 1. In the table, we show:

- For GPT, the total time the system takes to solve the problem.
- For CMBP, the number of steps (column "#s") (i.e., the number of elementary actions) and the total time needed to solve the problem (column "Total").
- For QBFPLAN,
  - the number of steps (i.e., the number of parallel actions) (column "#s"),
  - the search time taken by the system at the last step (column "Last"),
  - the total search time (column "Tot-S"), i.e., the sum over all steps of the time taken by QSAT, thus excluding the times necessary to build the QBF formula at each step.
- For $\mathcal{C}$-PLAN,
  - the number of steps (i.e., the number of parallel actions) (column "#s"),
  - the number of possible plans generated before finding a valid one (column "#pp"),
  - the search time taken by the system at the last step (column "Last"),

	GPT	CMBP		QBFPLAN			$\mathcal{C}$-PLAN				
$\|P\|$-$\|T\|$	Total	#s	Total	#s	Last	Tot-S	#s	#pp	Last	Tot-S	Total
2-1	0.10	3	0.00	3	0.00	0.00	3	6	0.00	0.00	0.01
2-5	0.04	2	0.01	1	0.00	0.00	1	1	0.00	0.00	0.00
2-10	0.05	2	0.03	1	0.01	0.01	1	1	0.00	0.00	0.00
4-1	0.04	7	0.00	7	0.01	0.09	7	540	0.12	0.15	0.65
4-5	0.23	4	0.79	1	0.00	0.00	1	1	0.00	0.00	0.00
4-10	2.23	4	11.30	1	0.01	0.01	1	1	0.00	0.00	0.01
6-1	0.09	11	0.04	11	0.06	6.02	11	52561	15.39	49.39	221.55
6-5	3.29	7	16.80	3	0.05	6.73	3	98346	56.92	57.34	419.53
6-10	74.15	–	MEM	1	0.03	0.03	1	1	0.00	0.00	0.01
8-1	0.41	15	0.20	15	0.19	721.66	–	–	–	–	TIME
8-5	32.07	11	112.48	3	1.51	26.91	–	–	–	–	TIME
8-10	MEM	–	MEM	1	0.04	0.04	1	1	0.00	0.00	0.01
10-1	2.67	19	1.55	–	–	TIME	–	–	–	–	TIME
10-5	MEM	15	974.45	–	–	TIME	–	–	–	–	TIME
10-10	MEM	–	MEM	1	0.08	0.08	1	1	0.00	0.00	0.04

Table 2: Bomb in the toilet: Multiple toilets, clogging, one bomb.

- the total search time (column "Tot-S"), i.e., the sum over all steps of the time taken by $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$ and $\mathcal{C}$-SAT_TEST,
- the total time taken by the system to solve the problem, excluding the off-line time taken by *CCALC* and *CTCALC* (column "Total"). This timing does not coincide with "Tot-S" because it includes also the time required by *EXPAND*, and by other internal procedures.

Times are in seconds, and all the tests have been run on a Pentium III, 850MHz, 512MBRAM running Linux SUSE 7.0. For practical reasons, we stopped the execution of a system if its running time exceeded 1200s of CPU time or if it required more than the 512MB of available RAM. In the table, the first case is indicated with "TIME" and the second with "MEM".

As it can be seen from Table 1, $\mathcal{C}$-PLAN and QBFPLAN take full advantage of their ability to execute multiple elementary actions concurrently. Indeed, they solve the problem in only one step, by dunking all the packages. Furthermore, the time taken by these system is not or barely measurable. CMBP and GPT have comparable performances, with CMBP being better of a factor of 2-3. However, the most interesting data about these systems is that when $|P| = 20$ they both run out of memory. As we have already said, both GPT and CMBP can require huge amounts of memory.

We also consider the elaboration of the "bomb in the toilet" in which dunking a package clogs the toilet. There is the additional action of flushing the toilet, which is possible only if the toilet is clogged. The results are shown in Table 2 for $|P| = 2, 4, 6, 8, 10$ and $|T| = 1, 5, 10$. With one toilet, these problems are the "sequential version" of the previous. With multiple toilets they are similar to the "BMTC" problems in [Cimatti and Roveri, 2000]. As we can see from Table 2, when there is only one toilet $\mathcal{C}$-PLAN's "Total" time slows down rapidly compared to the other solvers. Indeed, $|T| = 1$ represents the purely sequential case in which the only valid plan consists in repeatedly dunking a package and

	GPT	CMBP		QBFPLAN			$\mathcal{C}$-PLAN				
$\lvert P\rvert$-$\lvert T\rvert$	Total	#s	Total	#s	Last	Tot-S	#s	#pp	Last	Tot-S	Total
2-1	0.03	3	0.00	3	0.00	0.00	3	3	0.00	0.00	0.00
2-5	0.04	2	0.00	1	0.00	0.00	1	1	0.00	0.00	0.00
2-10	0.24	2	0.02	1	0.01	0.01	1	1	0.00	0.00	0.02
4-1	0.17	7	0.01	7	0.06	0.18	7	15	0.01	0.02	0.02
4-5	0.06	4	0.54	1	0.01	0.01	1	1	0.01	0.00	0.01
4-10	0.38	4	7.13	1	0.02	0.02	1	1	0.02	0.00	0.02
6-1	0.08	11	0.03	11	0.90	47.94	11	117	0.25	1.39	2.01
6-5	0.33	7	10.71	3	0.71	124.14	3	48	0.62	0.66	1.36
6-10	7.14	–	MEM	1	0.36	0.36	1	1	0.00	0.00	0.00
8-1	0.06	15	0.17	–	–	TIME	15	1195	12.23	147.25	184.29
8-5	2.02	11	90.57	–	–	TIME	3	2681	14.84	15.60	317.13
8-10	MEM	–	MEM	1	11.73	11.73	1	1	0.00	0.00	12.68
10-1	0.21	19	1.02	–	–	TIME	–	–	–	–	TIME
10-5	12.51	15	591.33	–	–	TIME	–	–	–	–	TIME
10-10	MEM	–	MEM	1	889.90	889.90	1	1	0.00	0.00	0.06

Table 3: Bomb in the toilet: Multiple toilets, clogging, possibly multiple bombs.

flushing the toilet till all the packages have been dunked. By analysing these numbers and profiling the code, we discovered that

- most of the search time is spent by $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$: On all the experiments we tried, each call of $\mathcal{C}$-SAT_TEST takes a hardly measurable time. Thus, the potential exponential cost of verifying a plan does not arise in practice, at least on these experiments (but also on the other experiments we tried). As a matter of facts, each time a plan is verified, the corresponding set of unit clauses is added to the V formula and (if the plan is not valid) the empty set of clauses is generated after very few splits.

- in some cases, the search time is negligible wrt the total time spent by the system which takes into account also the time, e.g., to expand the formula at each step. This is evident when $|P| = 6$ and $|T| = 1, 5$.

In any case, $\mathcal{C}$-PLAN's performances are not too bad compared to the ones of the other solvers: $\mathcal{C}$-PLAN, CMBP, GPT, QBFPLAN do not solve 4, 3, 3, 2 problems respectively. As expected, $\mathcal{C}$-PLAN and QBFPLAN run out of time, while the other planners run out of memory.

Finally, we consider the same problem as before, except that we do not know how many packages are armed. These problems, wrt the ones previously considered, present a higher degree of uncertainty. We consider the same values of $|P|$ and $|T|$ and report the same data as before. The results are shown in Table 3.

Contrarily to what could be expected, $\mathcal{C}$-PLAN's performances are much better on the problems in Table 3 than on those in Table 2. This is most evident if we compare the number of plans generated and tested by $\mathcal{C}$-PLAN before finding a solution. For example, if we consider the four packages and one toilet problem,

- with one bomb, as in Table 2, $\mathcal{C}$-PLAN generates 540 possible plans and takes 0.65s to solve the problem (0.15s of search time),

- with possibly multiple bombs, as in Table 3, $\mathcal{C}$-PLAN generates 15 possible plans and takes 0.02s to solve the problem (0.02s is also the search time).

To understand why, consider the case in which there is only one toilet and two packages P_1 and P_2. For $n = 0$,

- If we know that there is one bomb, then there are no possible plans.
- If we know nothing about the initial state, then there is the possible plan consisting of the empty sequence of actions (corresponding to assuming that neither P_1 nor P_2 is armed). In this case, because of the determinization, $\mathcal{C}$-PLAN adds a clause to the initial state saying that at least one package is armed.

For $n = 1$, $\mathcal{C}$-PLAN tries 2 possible plans in both scenarios. Assuming that the plan in which P_1 is dunked is generated first,

- If we know that there is one bomb, the plan is rejected, and —because of the determinization— a clause is added to the initial state allowing $\mathcal{C}$-PLAN to conclude that the bomb is in P_2. Then, for $n = 2$ and $n = 3$, any plan in which P_2 is dunked is possible.
- If we know nothing about the initial state, the plan is rejected, and — because of the determinization— a clause saying that the bomb is not in P_1 or is in P_2 is added to the initial state. Then, $\mathcal{C}$-PLAN generates the other plan in which only P_2 is dunked. Also this plan is rejected and a clause saying that a bomb is in P_1 or not in P_2 is added to the initial state. Thus, there is now only one initial state satisfying all the constraints, namely the one in which both P_1 and P_2 are armed. This allows $\mathcal{C}$-PLAN to conclude that there are no possible plans for $n = 2$, and to immediately generate a valid plan at $n = 3$.

In any case, the optimizations described in Section 5.1 and Section 5.2 do help a lot. Indeed, if we consider the four packages and one toilet problem, and disable the optimizations,

- if we have one bomb, as in Table 2, $\mathcal{C}$-PLAN generates 2145 possible plans and takes 0.54s to solve the problem (0.24s in the last step),
- if we have possibly multiple bombs, as in Table 3, $\mathcal{C}$-PLAN generates 3743 possible plans and takes 0.93s to solve the problem (0.72s in the last step).

However, it turns out that for these domains, backjumping and learning do not help much: By disabling them, we get roughly the same times.

Also CMBP and GPT perform better on the problems in Table 3 than on the problems in Table 2: Overall, $\mathcal{C}$-PLAN, CMBP and GPT do not solve respectively 2, 3 and 2 of the problems in Table 3. The situation is different for QBFPLAN: Now it is not able to solve 4 problems. The motivation lies in the particular pruning heuristics used by QSAT. In particular, QSAT performs a partial elimination of the universal quantifiers, which is most effective when the formula contains few universal quantifiers, as in the case of the QBFs resulting from the problems in Table 2.[4] Overall, we get roughly the same picture that we had before: $\mathcal{C}$-PLAN and QBFPLAN take full advantage of their ability to concurrently execute actions, and thus behave well on problems with multiple toilets. In some cases, both CMBP and GPT exhaust all the available memory.

On the basis of these comparative tests and given that $\mathcal{C}$-PLAN is still at an early stage of development, we can conclude that $\mathcal{C}$-PLAN (and QBFPLAN) is competitive with both CMBP and GPT on problems with a high degree of parallelism. This is not surprising, since analogous results have been obtained in the classical setting. In particular, in [Haslum and Geffner, 2000a], BLACKBOX [Kautz and Selman, 1998], GRAPHPLAN [Blum and Furst, 1995], STAN [Long and Fox, 1999] and HSPr [Haslum and Geffner, 2000a], are comparatively tested on some logistics and rockets problems: The conclusion is that on these problems SAT approaches appear to do best.

The bomb in the toilet problems are a classic for testing planners with incomplete information. However, they do not lend themselves to be good benchmarks for SAT-based planners. Indeed, the classical bomb in the toilet is evidently not a good benchmark for SAT-based planners like $\mathcal{C}$-PLAN ($\mathcal{C}$-PLAN takes 0.00s to solve all the problems we considered). Furthermore, there are few instances of these problems. In order to have more instances, we have to consider multiple toilets, possibly multiple bombs, the possibility that one toilet becomes clogged because of a dunking, etc. etc.. Of course, by adding parameters to the original problem, we get more and more instances. However, with these additional parameters, it is no longer clear what is (are) the parameter(s) ruling the expected difficulty of the problem. Ideally, what we would like, is the ability to generate as many problems as we want by having a direct control over the problems characteristics, such as size and expected hardness (see, e.g., [Achlioptas *et al.*, 2000]). More precisely, what we would like is a test set meeting the following five requirements:

1. Each problem should have a "small bound": In other words, we have to be able to determine a solution or the absence of a solution testing the system for small n (compared to the size of the problem). Indeed, given that the size of the P/V formulas is polynomial in n, but n can be exponential in the number of fluents of the input action description, it does not make sense to apply our approach for big values of n.

[4] The encodings produced with QBFPLAN have the following form

$$\exists a_1 \ldots a_n \forall d_1 \ldots d_m \exists v_1 \ldots v_f (\\ (\neg d_1 \wedge \ldots \wedge \neg d_m \supset s_1) \wedge (\neg d_1 \wedge \ldots \wedge d_m \supset s_2) \wedge \ldots \wedge (d_1 \wedge \ldots \wedge d_m \supset s_{2^m}) \wedge \Phi)$$

where $a_1, \ldots, a_n$ are the variables corresponding to actions; $s_1, \ldots, s_{2^m}$ are conjunctions of literals, corresponding to all the possible initial states; $d_1, \ldots, d_m$ are "dummy" variables; $v_1, \ldots, v_f$ are the variables corresponding to fluents; see [Rintanen, 1999a] for more details. In the experiments in Table 2 there are $|P|$ possible initial states, and thus $log_2|P|$ dummy variables. In the experiments in Table 3 there are $2^{|P|}$ possible initial states, and thus $|P|$ dummy variables.

2. Each problem should be "SAT challenging": Finding a possible plan should be not an easy task. This is necessary in order to stress the SAT-capabilities of the system.

3. Each problem has to have a "predictable difficulty" on average: We would like to have a parameter d whose value rules the expected difficulty of the problem. By increasing d, we should get more difficult problems.

4. For each value of d, it should be possible to "randomly generate" as many problems as we want: This is necessary in order to get statistically meaningful results of the planners' performances.

5. (If possible) the problems should be "meaningful": It would be nice if each instance corresponded to a real-world problem.

In order to meet this last requirement, we started with a classical robot navigation problem. We are given an $N \times N$ grid, and M robots (with $M < N$) can move in it. They start from one border of the grid and their goal is to reach the opposite side. In what follows, we assume that they start from the left border. Their duty is made not trivial because each location in the grid may, or may be not, occupied by an object. In order to have a small bound and have SAT-challenging problems, we assume that the locations of the objects obey the "pigeonhole" principle: There is at least one object per column, and no more than one per row. Pigeonhole formulas are well-known in the SAT-literature and they are a standard benchmark for SAT-solvers. Furthermore, given that in each column there is at most one object, if there exists a valid plan, then there is one whose length is $\leq 2(N-1)$. Finally, as in [Achlioptas *et al.*, 2000], in order to have a parameter ruling the difficulty of the problem, we assume that the location of d of the N objects is unknown: When $d = N$, we only know that the objects obey the pigeonhole principle, and therefore have a high-degree of uncertainty in the location of the objects. When $d = 1$, we exactly know their location, and the problem boils down to a classical planning problem. Thus, $d = 1$ represents the basic case in which it should be very easy to find the valid solutions: The location of all the objects is known, and we are facing a classical planning problem.

For each value of d, we generate 100 different instances by randomly placing N objects in the grid according to the pigeonhole principle, and then by removing d of the N objects. We consider the case in which we have $N = 5, 7, 9$; $d = 1, 2, 3$; and $M = 1, 2$ robots. The first robot starts from the bottom-left corner and, when $M = 2$, the second starts from the top-left corner. For each setting of the values for N and d we report the data for the samples in which the "Total" time is

- the 1%-percentile, i.e., the minimum, (row "min"), or
- the 25%-percentile (row "25%"), or
- the 50%-percentile, i.e., the median, (row "med"), or

N	d		$M=1$						$M=2$					
			Plan	#s	#pp	Last	Tot-S	Total	Plan	#s	#pp	Last	Tot-S	Total
5	1	min	Y	5	1	0.01	0.01	0.25	Y	5	1	0.01	0.01	0.68
		25%	Y	5	1	0.01	0.01	0.31	Y	6	1	0.01	0.02	1.11
		med	Y	6	1	0.00	0.00	0.47	N	8	0	0.01	0.03	1.32
		75%	N	8	0	0.00	0.00	0.65	N	8	0	0.02	0.08	1.41
		max	Y	8	1	0.01	0.02	0.95	Y	8	1	0.02	0.07	2.10
	2	min	Y	5	1	0.01	0.01	0.27	Y	5	1	0.00	0.00	0.69
		25%	Y	5	1	0.01	0.01	0.36	N	8	0	0.02	0.07	1.36
		med	Y	6	3	0.00	0.00	0.73	N	8	1	0.02	0.14	1.62
		75%	Y	8	1	0.00	0.01	0.89	N	8	6	0.02	1.05	3.35
		max	Y	7	51	0.53	0.65	10.17	-	-	-	-	-	TIME
	3	min	Y	5	1	0.00	0.00	0.44	Y	5	2	0.01	0.01	1.00
		25%	N	8	4	0.01	0.20	1.07	N	8	5	0.03	0.53	3.40
		med	N	8	6	0.45	0.69	1.82	N	8	13	0.03	2.84	7.11
		75%	Y	7	19	2.27	2.49	5.01	N	8	102	0.03	29.62	49.51
		max	N	8	142	27.15	34.94	45.12	-	-	-	-	-	TIME
7	1	min	Y	7	1	0.03	0.03	1.99	Y	7	1	0.04	0.04	6.08
		25%	Y	7	1	0.02	0.02	2.12	Y	8	1	0.06	0.10	8.47
		med	Y	8	1	0.03	0.06	3.02	Y	8	1	0.06	0.10	8.72
		75%	Y	8	1	0.04	0.06	3.15	Y	9	1	0.05	0.14	11.41
		max	Y	10	1	0.04	0.14	5.48	Y	10	1	0.06	0.20	14.70
	2	min	Y	7	1	0.03	0.03	2.08	Y	7	1	0.03	0.03	6.28
		25%	Y	7	1	0.02	0.02	3.09	Y	9	1	0.07	0.16	11.74
		med	Y	8	1	0.03	0.05	4.45	Y	8	3	0.05	0.08	14.29
		75%	N	12	1	0.05	0.37	5.04	Y	9	10	5.28	5.38	28.29
		max	Y	11	8	3.19	3.97	16.51	-	-	-	-	-	TIME
	3	min	Y	7	1	0.03	0.03	3.75	N	12	0	0.10	0.52	9.23
		25%	Y	8	1	0.03	0.06	6.03	Y	9	12	4.78	4.90	41.34
		med	Y	9	2	0.03	0.09	11.00	Y	9	109	339.89	340.00	439.07
		75%	Y	9	12	12.42	15.56	26.60	-	-	-	-	-	TIME
		max	-	-	-	-	-	TIME	-	-	-	-	-	TIME
9	1	min	Y	9	1	0.08	0.08	11.46	Y	9	1	0.13	0.13	32.72
		25%	Y	9	1	0.09	0.09	11.76	Y	9	1	0.13	0.13	33.89
		med	Y	10	1	0.10	0.18	14.66	Y	10	1	0.20	0.33	42.57
		75%	Y	10	1	0.10	0.19	15.60	Y	10	1	0.15	0.28	43.61
		max	Y	11	1	0.11	0.31	20.17	Y	11	1	0.16	0.44	54.18
	2	min	Y	9	1	0.08	0.08	11.73	Y	9	1	0.12	0.12	33.38
		25%	Y	9	1	0.08	0.08	17.69	Y	9	2	0.12	0.12	57.19
		med	Y	10	1	0.11	0.19	22.90	Y	10	1	0.14	0.26	67.37
		75%	Y	10	3	2.01	2.10	27.91	Y	11	12	26.12	26.37	130.15
		max	-	-	-	-	-	TIME	-	-	-	-	-	TIME
	3	min	N	16	0	0.18	1.21	19.69	N	16	0	0.36	2.44	40.62
		25%	Y	9	1	0.09	0.09	24.00	Y	10	2	0.20	0.40	90.95
		med	Y	10	1	0.10	0.18	31.31	Y	10	7	0.22	0.36	200.54
		75%	Y	11	10	34.00	37.74	84.65	-	-	-	-	-	TIME
		max	-	-	-	-	-	TIME	-	-	-	-	-	TIME

Table 4: $\mathcal{C}$-PLAN's performances on robot navigation problems

- the 75%-percentile (row "75%"), or
- the 100%-percentile, i.e., the maximum, (row "max"),

of the 100 timings we obtained. We remind that the *Q%-percentile* of a set S of values is the value V such that Q% of the values in S are smaller or equal to V. The set of statistics consisting of the {1%, 25%, 50%, 75%, 100%}-percentiles is known as the "five-number summary" and is most useful for comparing distributions [Moore and McCabe, 1993]. For these samples, we show the same data as before, and also (column "Plan") whether the problem has a solution (value "Y") or not (value "N"). The results are shown in Table 4. Notice that we only test $\mathcal{C}$-PLAN, the main reason being that it is not clear to us how to naturally represent these problems in the languages of the other planners. As it can be seen from the Table 4, $\mathcal{C}$-PLAN's performances get worse and worse as M

increases: When $M = 1$, $\mathcal{C}$-PLAN is not able to solve problems having $d = 3$ and $N = 7, 9$. When $M = 2$, $\mathcal{C}$-PLAN is not able to solve problems having $d = 2, 3$ and $N = 5, 7, 9$. This confirms our expectations.

Considering the data in the Table, we see that the sometimes the search time spent by the system is very small compared to its total running time (see, e.g., the data for $N = 9$). In these cases, most of the time is taken by other operations internal to the system, like the expansion. About the expansion, it is worth remarking that in applications the action description formalizing the scenario is rarely changed: Most of the times, it is the initial state and/or the goal that change from time to time. This opens up the possibility to perform the expansion of both P and V in two steps:

- by computing off-line the formulas $\wedge_{i=0}^{n-1} tr_i^D$ and $State_0^D \wedge \neg Z_0 \wedge_{i=0}^{n-1} trt_i^D$, for the given action description D, and for each plausible n,
- by adding on-line the conjuncts corresponding to the specific initial and goal states (represented by the formulas I_0, G_n for P, and I_0, $\neg(G_n \wedge \neg Z_n)$ for V).

Of course, in this way the on-line time necessary to compute the P and V formulas at each step becomes negligible.

7 Conclusions

We have presented a SAT-based procedure capable of dealing with planning domains having incomplete information about the initial state, and whose underlying transition system is specified in $\mathcal{C}$. $\mathcal{C}$ allows for, e.g., concurrency, constraints, and nondeterminism. We proved the correctness and completeness of the procedure, discussed some optimizations, and then we presented $\mathcal{C}$-PLAN, a system based on our procedure. The experimental analysis shows that SAT-based approaches to planning with incomplete information can be competitive with CMBP [Cimatti and Roveri, 1999, Cimatti and Roveri, 2000], GPT [Bonet and Geffner, 2000], and QBFPLAN [Rintanen, 1999a] at least in the case of problems with a high degree of parallelism. We also propose a benchmark set for evaluating SAT-based planners dealing with uncertainty.

In the last few years, there has been a growing interest in developing planners dealing with uncertainty. If we restrict our attention to conformant planners, some of the most popular are CMBP [Cimatti and Roveri, 1999, Cimatti and Roveri, 2000], GPT [Bonet and Geffner, 2000], CGP [Smith and Weld, 1998], QBFPLAN [Rintanen, 1999a] and WSPDF [Finzi *et al.*, 2000].

CMBP is based on the representation of the planning domain as a finite state automaton, and uses BDDs [Bryant, 1992] to compactly represent and efficiently search the automaton. As we already said, the algorithm is based on breadth first, backward search, and is able to return all conformant plans of minimal length. Furthermore, it is able to determine the absence of a solution. Because of the breadth first search, CMBP can be very effective. However, it is well known that the size of BDDs critically depends on an either statically or dynamically fixed variable ordering, and that there are some problems which cause an exponential blow up of the size of BDDs for any variable ordering. As we have seen, on some problems, CMBP clogs all the available memory of our computer. Finally, CMBP's input language is based on the action language $\mathcal{AR}$ [Giunchiglia *et al.*, 1997], and thus misses some of the $\mathcal{C}$ expressive capabilities, like concurrency and qualification constraints. More recently, the authors proposed a new approach in which heuristic search is combined with the symbolic representation of the automaton, and proposed a new planner (called HSCP) which outperforms CMBP by orders of magnitude with a much lower memory consumption [Bertoli *et al.*, 2001]. However, HSCP is not guaranteed to return optimal plans.

In GPT the conformant planning problem is seen as a deterministic search problem in the space of belief states. In GPT, search is based on the A* algorithm [Nilsson, 1980], and each belief state is stored separately. As a consequence GPT performances strictly rely on the goodness of the function estimating the distance of a belief state, to the belief state representing the goal. Indeed, GPT is able to conclude that a planning problem has no solution by exhaustively exploring the space of belief states. Finally, GPT input language is based on PDDL, extended to deal with probabilities and uncertainty, and thus it has some features (i.e., probabilities) that $\mathcal{C}$ misses, and it misses some of $\mathcal{C}$ expressive capabilities. It is worth remarking that the problem of effectively extending heuristic search based approaches in order to deal with concurrency is still open. As we have seen, GPT can clog all the available memory of the computer.

CGP (standing for "Conformant GraphPlan") extends the classical plan-graph approach [Blum and Furst, 1995] to deal with uncertainty in the initial state (in the corresponding paper, the authors say how to extend the approach to the case of actions with nondeterministic effects). The basic idea behind CGP is initialize a separate plan-graph for each possible determinization of the given planning problem. Thus, CGP performs poorly if run on problems with an high degree of uncertainty. According to the experimental results presented in [Cimatti and Roveri, 2000, Haslum and Geffner, 2000b], CGP is outperformed by CMBP and GPT. Finally, CGP's input language is less expressive than $\mathcal{C}$.

In QBFPLAN, planning in nondeterministic domains is reduced to reasoning about Quantified Boolean Formulas (QBFs). QBFPLAN is not restricted to conformant planning, and can perform conditional planning under partial observability. Among the cited systems, QBFPLAN is the one that most closely resembles $\mathcal{C}$-PLAN: For each planning problem and plan length n, a corresponding QBF formula is generated and then a QBF solver is invoked. Also the search performed by the QBF solver reflects the search performed by $\mathcal{C}$-PLAN: First a sequence of actions is generated, and then its validity is checked. Indeed, for any planning problem specified using any action language —as long as it is possible to compute a propositional formula corresponding to the transition relation— it is relatively easy to specify QBFs whose solutions correspond to the existence of a solution at a given length. However, in $\mathcal{C}$-PLAN we have a decoupling between the plan generation and the plan validation phases. Such decoupling allows to incorporate different procedures for the generation phase.

For instance, if we have nondeterminism only in the initial state, then we can use a solver incorporating the optimization introduced in Section 5.1. According to such optimization, one possible initial state ruled out at a certain time step, no longer comes to play in the subsequent time steps. This is not possible when the solving phase is just a call to a solver used as a black box.

WSPDF is a simple (i.e., consisting of few lines of code) planner based on regression and written in GOLOG. WSPDF's good performances rely on domain dependent control knowledge that is added to prune bad search paths. With the addition of control knowledge, WSPDF can be very effective. However, because of this additional information, WSPDF plays in a different league than all the above mentioned planners, including $\mathcal{C}$-PLAN.

Our work can be seen as a follow up of [McCain and Turner, 1998]. In [McCain and Turner, 1998], the language of causal theories is considered, and the notions of possible and valid plans are introduced. The action language $\mathcal{C}$ is based on [McCain and Turner, 1997], and is less expressive than the language of causal theories used in [McCain and Turner, 1998]. However, the focus in [McCain and Turner, 1998] is on stating some conditions under which a possible plan is also valid. No procedure for computing valid plans in the general case (e.g., with multiple initial states or actions with nondeterministic effects) is given.

In [Ferraris and Giunchiglia, 2000], it is showed that the general theory here presented can be specialized to deal with "simple" nondeterministic domains. Intuitively, a domain is "simple", if

- there are no static laws;
- concurrency is not allowed; and
- each elementary action A is characterized by $m + 2$ ($m \geq 1$) finite sets of fluents $P, E, N_1, \ldots, N_m$: P and E list respectively A's preconditions and effects as in STRIPS, while each N_i represents one of the possible outcomes of A.

For "simple" nondeterministic domains, "regular parallel" or "simple-split sequential" encodings in the style of [Kautz and Selman, 1996, Kautz *et al.*, 1997, Ernst *et al.*, 1997] are possible. According to the experimental analysis done in [Ferraris and Giunchiglia, 2000], the regular parallel encodings are those leading to the best performances, as in the classical case.

The encodings and optimizations presented in [Ferraris and Giunchiglia, 2000] are possible because of the restrictions on the syntax of the possible action descriptions. As we said in the introduction, the procedure $\mathcal{C}$-SAT here described is fully general because it allows to consider any finite $\mathcal{C}$ action description, and for any finite action description in any Boolean action language, there is an equivalent $\mathcal{C}$ action description. Given the generality of the procedure, and the results of our experimental analysis, we believe that SAT-based approaches to planning with incomplete information can be very effective at least on problems with a high degree of parallelism. This belief is also confirmed by the results in [Haslum and Geffner, 2000a] for the classical case, and by the very positive results that SAT-based approaches are having in formal verification. In this field, following the proposal of [Biere *et al.*, 1999], a verification problem is converted into a SAT problem by unrolling the transition relation n-times (as proposed by Kautz and Selman in planning [Kautz and Selman, 1992]) and then by adding the initial state and the negation of the safety property to be verified, as additional conjuncts. A SAT solver is then applied to the resulting formula to check whether it is satisfiable (in which case the property is violated) or not. This approach has the same main weakness of $\mathcal{C}$-PLAN, namely it is not applicable for big n-s. However, for relatively small n-s (corresponding in planning to problems with a high degree of parallelism), this approach has showed to outperform all the others, see [Biere *et al.*, 1999, Shtrichman, 2000, Copty *et al.*, 2001].

Finally, a very different SAT-based procedure for conformant planning has been very recently proposed in [Kurien *et al.*, 2002]. The idea behind such procedure is to start with a solution which works in some deterministic version of the initial planning problem, and then to continue extending/modifying it till it works in all the possible deterministic versions. Some optimizations/heuristics are presented in order to improve performances.

Acknowledgements

We thank Vladimir Lifschitz, and Hudson Turner for comments and discussions on an earlier version of this paper. Thanks to Alessandro Cimatti and Hector Geffner for useful discussions about model checking and heuristic search respectively. Jussi Rintanen kindly provided QBFPLAN and the latest version of QSAT. A special thank to Paolo Ferraris for many fruitful discussions on the topic of this paper. Paolo has also participated to the design of the architecture of $\mathcal{C}$-PLAN, and he developed the first version of $\mathcal{C}$-PLAN. Norman McCain has made possible to integrate *CCALC* in $\mathcal{C}$-PLAN. This work is supported by ASI and CNR.

References

[Achlioptas *et al.*, 2000] Dimitris Achlioptas, Carla Gomes, Henry Kautz, and Bart Selman. Generating satisfiable problem instances. In *Proc. AAAI*, 2000.

[Bayardo, Jr. and Schrag, 1997] Roberto J. Bayardo, Jr. and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 203–208, Menlo Park, July 27–31 1997. AAAI Press.

[Bertoli *et al.*, 2001] Piergiorgio Bertoli, Alessandro Cimatti, and Marco Roveri. Heuristic search + symbolic model checking = efficient conformant planning.

In *Proc. of the International Joint Conference on Artificial Intelligence (IJ-CAI'2001)*, 2001.

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, 1999.

[Blum and Furst, 1995] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proc. of IJCAI-95*, pages 1636–1642, 1995.

[Bonet and Geffner, 2000] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. AIPS*, 2000.

[Bryant, 1992] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[Cadoli *et al.*, 1998] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proc. AAAI*, 1998.

[Cimatti and Roveri, 1999] Alessandro Cimatti and Marco Roveri. Conformant planning via model checking. In *Lecture Notes in Computer Science, Proc. of the 5th European Conference on Planning (ECP-99).* Springer, September 1999.

[Cimatti and Roveri, 2000] Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[Copty *et al.*, 2001] Fady Copty, Limor Fix, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Vardi. Benefits of bounded model checking at an industrial setting. In *Proc. 13th International Computer Aided Verification Conference (CAV)*, 2001.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[Dechter, 1990] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, January 1990.

[Ernst *et al.*, 1997] Michael Ernst, Todd Millstein, and Daniel Weld. Automatic SAT-compilation of planning problems. In *Proc. IJCAI-97*, 1997.

[Ferraris and Giunchiglia, 2000] Paolo Ferraris and Enrico Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. AAAI-2000*, 2000.

[Finzi *et al.*, 2000] Alberto Finzi, Fiora Pirri, and Ray Reiter. Open world planning in the situation calculus. In *Proc. AAAI*, 2000.

[Gent *et al.*, 2000] Ian Gent, Hans Van Maaren, and Toby Walsh, editors. *SAT2000. Highlights of Satisfiability Research in the Year 2000.* IOS Press, 2000.

[Giunchiglia and Lifschitz, 1995] Enrico Giunchiglia and Vladimir Lifschitz. Dependent fluents. In *Proc. IJCAI-95*, pages 1964–1969, 1995.

[Giunchiglia and Lifschitz, 1998] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630, 1998.

[Giunchiglia and Tacchella, 2000] Enrico Giunchiglia and Armando Tacchella. *SAT: a system for the development of modal decision procedures. In *Proc. of the 17th International Conference on Automated Deduction (CADE'2000)*, pages 291–296, 2000.

[Giunchiglia *et al.*, 1997] Enrico Giunchiglia, G. Neelakantan Kartha, and Vladimir Lifschitz. Representing action: indeterminacy and ramifications. *Artificial Intelligence*, 95:409–443, 1997.

[Giunchiglia *et al.*, 2000] E. Giunchiglia, F. Giunchiglia, and A. Tacchella. SAT-Based Decision Procedures for Classical Modal Logics. *Journal of Automated Reasoning*, 2000. To appear. Reprinted in [Gent *et al.*, 2000].

[Giunchiglia *et al.*, 2001] Enrico Giunchiglia, Marco Maratea, Armando Tacchella, and Davide Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001), LNAI 2083*, 2001.

[Green, 1969] Cordell Green. Application of theorem proving to problem solving. In *Proc. IJCAI-69*, pages 219–240, 1969.

[Haslum and Geffner, 2000a] Patrick Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *Proc. AIPS*, pages 140–149, 2000.

[Haslum and Geffner, 2000b] Patrick Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *Proc. AIPS*, pages 140–149, 2000.

[Kambhampati, 2000] Subbarao Kambhampati. Planning Graph as (Dynamic) CSP: Exploiting CBL, DDB and other CSP search techniques in Graphplan. *JAIR*, 12:1–34, 2000.

[Kautz and Selman, 1992] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI-92*, pages 359–363, 1992.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic and stochastic search. In *Proc. AAAI-96*, pages 1194–1201, 1996.

[Kautz and Selman, 1998] Henry Kautz and Bart 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, held in conjunction with AIPS-98*, 1998.

[Kautz *et al.*, 1997] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proc. KR-96*, pages 374–384, 1997.

[Kurien *et al.*, 2002] James A. Kurien, Pandurang P. Nayak, and David E. Smith. Fragment-based conformant planning. In *Proc. 6th Int.nl Conf. o Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 2002.

[Li and Anbulagan, 1997] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 366–371, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers.

[Lifschitz, 1990] Vladimir Lifschitz. Frames in the space of situations. *Artificial Intelligence*, 46:365–376, 1990.

[Lifschitz, 1997] Vladimir Lifschitz. On the logic of causal explanation. *Artificial Intelligence*, 96:451–465, 1997.

[Lin and Reiter, 1994] Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation*, 4:655–678, 1994.

[Long and Fox, 1999] Derek Long and Maria Fox. The efficient implementation of the plan-graph. *JAIR*, 10:85–115, 1999.

[McCain and Turner, 1997] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proc. AAAI-97*, pages 460–465, 1997.

[McCain and Turner, 1998] Norman McCain and Hudson Turner. Fast satisfiability planning with causal theories. In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 1998.

[McDermott, 1987] Drew McDermott. A critique of pure reason. *Computational Intelligence*, 3:151–160, 1987.

[Moore and McCabe, 1993] D. S. Moore and G. P. McCabe. *Introduction to the Practice of Statistics*. W. H. Freeman and Co., 1993.

[Myers and Smith, 1988] Karen Myers and David Smith. The persistence of derived information. In *Proc. AAAI-88*, pages 496–500, 1988.

[Nilsson, 1980] Nils Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA., 1980.

[Plaisted and Greenbaum, 1986] D.A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[Prosser, 1993] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[Rintanen, 1999a] Jussi Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[Rintanen, 1999b] Jussi Rintanen. Improvements to the evaluation of quantified boolean formulae. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*, pages 1192–1197, S.F., July 31–August 6 1999. Morgan Kaufmann Publishers.

[Rintanen, 2001] Jussi Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formulae. In *Proc. LPAR*, volume 2250 of *LNCS*, pages 362–376, 2001.

[Selman *et al.*, 1992] B. Selman, H. Levesque., and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proc. of the 10th National Conference on Artificial Intelligence*, pages 440–446, 1992.

[Shtrichman, 2000] O. Shtrichman. Tuning SAT checkers for bounded model-checking. In *Proc. 12th International Computer Aided Verification Conference (CAV)*, 2000.

[Siekmann and Wrightson, 1983] Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag, 1983.

[Smith and Weld, 1998] David Smith and Daniel Weld. Conformant graphplan. In *Proc. AAAI-98*, pages 889–896, 1998.

[Tseitin, 1970] G. Tseitin. On the complexity of proofs in propositional logics. *Seminars in Mathematics*, 8, 1970. Reprinted in [Siekmann and Wrightson, 1983].

[Zhang, 1997] H. Zhang. SATO: An efficient propositional prover. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 272–275, Berlin, July13–17 1997. Springer.