klase

Planning as SAT- and QBF-Based reasoning

Participating Units: University of Genova, University of Roma

Abstract. Planning is a traditional research area in AI. The last decade has witnessed a large body of work in the development of languages and algorithms for planning, up to the point where complex, realistic problems can be modeled and solved in seconds. At the same time, thanks to the generality of the problem formulation, several applicative problems can be recasted into planning. This is also true in the SOC area, where planning can be used to automatically compose distributed services to provide new services satisfying domain and user specified requirements. In this deliverable, we focus on planning algorithmis based on propositional satisfiability (SAT) and Quantified Boolean Formulas (QBFs).

Document Identifier	Deliverable D4.5
Project	MIUR-FIRB project RBNE0195K5 "Knowledge Level Automated Software Engineering"
Version	v1.0
Date	May 24, 2006
State	Final
Distribution	Public

Acknowledgements.

This document is part of a research project funded by the FIRB 2001 Programme of the “Ministero dell’Istruzione, dell’Università e della Ricerca” as project number RBNE0195K5.

The partners in this project are: Istituto Trentino di Cultura (Coordinator), Università degli Studi di Trento, Università degli Studi di Genova, Università degli Studi di Roma “La Sapienza”, DeltaDator S.p.A..

Executive Summary

Planning is a traditional research area in AI. The last decade has witnessed a large body of work in the development of languages and algorithms for planning, up to the point where complex, realistic problems can be modeled and solved in seconds. At the same time, thanks to the generality of the problem formulation, several applicative problems can be recasted into planning. This is also true in the SOC area, where planning can be used to automatically compose distributed services to provide new services satisfying domain and user specified requirements.

In this deliverable we recap the research performed by the partners of the KLASE project to design advanced planning procedures based on propositional satisfiability (SAT) and Quantified Boolean Formulas (QBFs). In particular, the simple and well known planning as satisfiability framework is extended along three directions. In the first (Chapter 1) we show how conformant planning problems with domains expressed in $\mathcal{C}$ or $\mathcal{C}+$ (see deliverable 4.2) can be solved with SAT-based and/or QBF techniques. In the second (Chapter 2), we consider the issue of extended goals expressed as LTL formulas in the classical and the more general conformant planning cases. Finally, we consider the classical planning problem in the presence of "soft-goals" (Chapter 3), i.e., goals that do not have to be necessarily satisfied but whose satisfaction is desiderable.

Contents

Chapter 1

SAT- and QBF-Based Planning with language $\mathcal{C}$ and $\mathcal{C}+$

In this chapter we present $\mathcal{C}$-SAT and $\mathcal{C}$-QSAT, a SAT and QBF-based respectively procedure for checking the existence of plans of length n in $\mathcal{C}+$ [GL98, GLL$^+$04] domains with incomplete information about the initial state. Notice that for any finite action description in any Boolean action language [Lif97], there is an equivalent one —i.e., having the same transition diagram— specified in $\mathcal{C}+$. Thus, $\mathcal{C}$-SAT and $\mathcal{C}$-QSAT are fully general, and allow to consider domains involving, e.g.,

- actions which can be executed concurrently,
- (ramification and/or qualification) constraints affecting the effects of actions [LR94], 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). $\mathcal{C}$-QSAT uses QBF solvers as reasoning engines and thus combines plan generation and verification in a single validity check, along the lines of what has been done in [Rin99a].

We first prove the correctness and the completeness of $\mathcal{C}$-SAT and $\mathcal{C}$-QSAT, and then we present $\mathcal{C}$-PLAN, a planning system incorporating $\mathcal{C}$-SAT and $\mathcal{C}$-QSAT as core engines. In order to have optimality of the returned plan, $\mathcal{C}$-PLAN repeatedly calls $\mathcal{C}$-SAT and/or $\mathcal{C}$-QSAT, checking for the existence of plans of increasing length. Our goal in developing $\mathcal{C}$-PLAN has been to see whether the good performances obtained by SAT-based planners in the classical case would extend to more complex problems involving, 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 chapter is structured as follows. In Section 1.1 we briefly review the Boolean fragment of the action language $\mathcal{C}$+ and show a simple example that will be used throughout the whole chapter. In Section 1.2 we state the formal results laying the grounds to the proposed procedures, which are presented and proved correct and complete in Section 1.3 and in Section 1.4. Then, in Section 1.5 we show the structure of a system incorporating the proposed ideas and present some experimental analysis. We end the chapter with the conclusions in Section 1.7.

1.1 Action language $\mathcal{C}$+

Here, for simplicity, we restrict to the simple case in which all variables are Boolean, i.e., to the case in which the domain *Dom*(c) of each constant symbol c has two symbols, i.e., $|$*Dom*$(c)| = 2$. See [GLL$^+$04] or deliverable d4.2 for the syntax and semantics of the general case. However, it should be reminded that, assuming that *Dom*(c) is finite for each symbol c, it is always possible to replace each non Boolean symbol with a collection of Boolean symbols and get an equivalent action description (see Section "Reducing Multi-Valued Formulas to Classical Formulas") in d4.2.

1.1.1 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:

$$\textbf{caused } F \textbf{ if } G, \tag{1.1}$$

- and *dynamic laws*, of the form:

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H, \tag{1.2}$$

where F, G, H are formulas such that F and G do not contain action symbols. Both in (1.1) and in (1.2), 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 (1.1) 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 (1.1) in D such that σ' satisfies G, or

- the head of a dynamic law (1.2) 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.

1.1.2 An example

In rules (1.2), we do not write "**if** G" when G is a tautology.

Consider the following elaboration of the "safe from baby" example [MS88, GL95]. 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

$$\begin{array}{c} \textbf{caused } \mathit{Safe} \textbf{ if } \mathit{OnTable} \vee \mathit{Dolls}, \\ \textbf{caused } \neg \mathit{Safe} \textbf{ if } \neg \mathit{OnTable} \wedge \neg \mathit{Dolls}. \end{array} \tag{1.3}$$

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:

$$\begin{array}{c} \textbf{caused } \mathit{OnTable} \textbf{ after } \mathit{MPutOnTable} \vee \mathit{FPutOnTable}, \\ \textbf{caused } \neg \mathit{OnTable} \textbf{ after } \mathit{MPutOnFloor} \vee \mathit{FPutOnFloor}. \end{array} \tag{1.4}$$

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:

$$\textbf{caused } \mathit{False} \textbf{ after } \neg \mathit{Dolls} \wedge \neg (\mathit{MPutOnTable} \equiv \mathit{FPutOnTable}), \tag{1.5}$$

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 (1.5) 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

$$\begin{array}{c} \textbf{caused } P \textbf{ if } P \textbf{ after } P, \\ \textbf{caused } \neg P \textbf{ if } \neg P \textbf{ after } \neg P, \end{array} \tag{1.6}$$

[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 [Lif90], and Section "Noninertial Fluents" in [GL98] for more details.

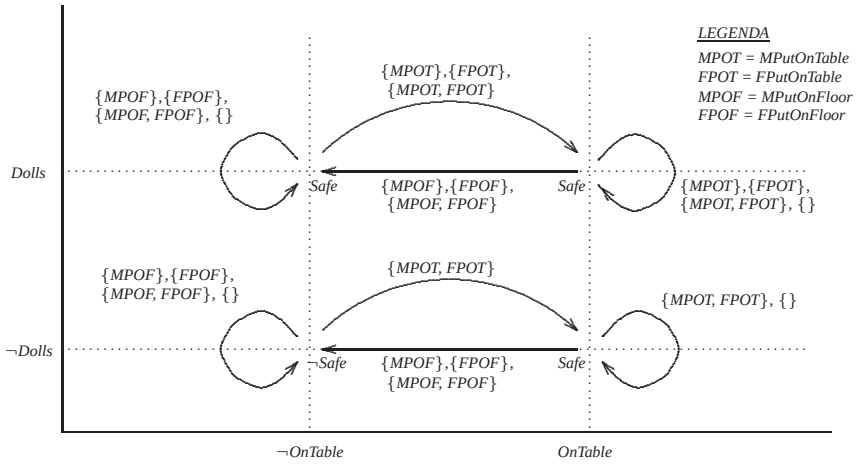

Figure 1.1: The transition diagram for (1.3)–(1.6).

for each fluent P different from *Safe* [MT97].

The transition diagram of the action description consisting of (1.3)–(1.6) is depicted in Figure 1.1. Both in the Figure and in the rest of the chapter, an action α [resp. a state σ] is represented by the set of action symbols [resp. fluents] satisfied by α [resp. σ].

Consider Figure 1.1. 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 (1.3) play the role of ramification constraints [LR94]: Moving the box on the table has the indirect effect of making the baby safe. (1.5) 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 $\{$*MPutOnTable*$\}$ and $\{$*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 [GL98, GLL$^+$04] for more details and examples.

1.1.3 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 [GL98, GNT04a]. 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 ([GLL$^+$04] 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 (1.1) with $\langle F, G\rangle$, and the dynamic law (1.2) 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 (1.3)–(1.6), tr_i^D is equivalent to the conjunction of the formulas:

$$\begin{gathered}
\mathit{Dolls}_{i+1} \equiv \mathit{Dolls}_i, \\
\mathit{Safe}_i \equiv \mathit{OnTable}_i \vee \mathit{Dolls}_i, \\
\mathit{Safe}_{i+1} \equiv \mathit{OnTable}_{i+1} \vee \mathit{Dolls}_{i+1}, \\
\mathit{OnTable}_{i+1} \equiv \mathit{MPutOnTable}_i \vee \mathit{FPutOnTable}_i \vee \\
\mathit{OnTable}_i \wedge \neg \mathit{MPutOnFloor}_i \wedge \neg \mathit{FPutOnFloor}_i, \\
(\mathit{MPutOnTable}_i \vee \mathit{FPutOnTable}_i) \supset \neg(\mathit{MPutOnFloor}_i \vee \mathit{FPutOnFloor}_i), \\
\neg \mathit{Dolls}_i \supset (\mathit{MPutOnTable}_i \equiv \mathit{FPutOnTable}_i).
\end{gathered} \tag{1.7}$$

In the rest of the chapter, tr_i^D is the formula defined as above if D is definite, and the formula defined analogously to $ct_1(D)$ in [GL98], 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 ([GL98]) *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.*

1.2 Possible and Valid plans

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

$$\sigma^0, \alpha^1, \sigma^1, \ldots, \alpha^n, \sigma^n \tag{1.8}$$

($n \geq 0$) 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 (1.8).

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 (1.8) 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.

1.2.1 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 (1.8) 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

$$\langle \neg \mathit{OnTable}, (1.3)\text{–}(1.6), \mathit{Safe} \rangle. \tag{1.9}$$

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

The following Theorem is similar to Proposition 2 in [MT98], and is an easy consequence of Proposition 2 in this chapter.

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 [KS92]. 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 (1.9), 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.

1.2.2 Valid Plans

As pointed out in [MT98], 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 (1.8) such that σ^0 is an initial state, σ^n is a goal state. According to this definition, the plan $\{$*MPutOnTable*$\}$ would be valid for the planning problem (1.9). Indeed, $\{$*MPutOnTable*$\}$ is not valid since this action is not "executable" in the initial states satisfying $\neg$*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

$$\sigma^0, \alpha^1, \sigma^1, \ldots, \alpha^k, \sigma^k \tag{1.10}$$

with $k < n$, α^{k+1} is executable in σ^k. For example, if D consists of (1.3)–(1.6), the plan $\{$*MPutOnFloor*$\}; \{$*FPutOnFloor*$\}$ is always executable in any state, while the plan $\{$*MPutOnFloor*$\}; \{$*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 (1.8) for D. For example, in the case of (1.3)–(1.6), the state $\{\}$ (i.e., the state which does not satisfy any fluent symbol) is a possible result of executing $\{$*MPutOnFloor*$\}; \{$*FPutOnFloor*$\}$ in any state satisfying $\neg$*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 (1.9), the plan $\{$*MPutOnTable*$\}$ is not valid, while $\{$*MPutOnTable,FPutOnTable*$\}$ is valid. Notice that valid plans correspond to "conformant" plans in [SW98]. 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*

$$I_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{tr}_i^D \models G_n. \tag{1.11}$$

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 (1.8) 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 $I_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{tr}_i^D$. By Proposition 2, there exists a corresponding history (1.8). 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 (1.11) holds. Since $\vec{\alpha}$ is not valid there exists a history (1.8) 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 $I_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{tr}_i^D$. However, since (1.11) holds, μ satisfies G_n, which contradicts the fact that σ^n does not satisfy G. □

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

$$\exists p^1 \ldots \exists p^n \mathit{tr}_i^D[P_{i+1}^1/p^1, \ldots, P_{i+1}^n/p^n] \tag{1.12}$$

where $P^1, \ldots, P^n$ are all the fluent symbols in D, and $\mathit{tr}_i^D[P_{i+1}^1/p^1, \ldots, P_{i+1}^n/p^n]$ 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

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

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

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

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

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

whence the thesis. □

Notice that the propositional variables and the bounding quantifiers can always be eliminated in (1.12), 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 (1.12) by simple syntactic manipulations, see [FG00]. For example, if tr_i^D is the conjunction of the formulas (1.7), then Poss_i^D is equivalent to the conjunction of the following formulas:

$$\begin{gathered} \mathit{Safe}_i \equiv \mathit{OnTable}_i \vee \mathit{Dolls}_i, \\ (\mathit{MPutOnTable}_i \vee \mathit{FPutOnTable}_i) \supset \neg(\mathit{MPutOnFloor}_i \vee \mathit{FPutOnFloor}_i), \\ \neg\mathit{Dolls}_i \supset (\mathit{MPutOnTable}_i \equiv \mathit{FPutOnTable}_i). \end{gathered} \tag{1.13}$$

From (1.13) follows that in the action description (1.3)–(1.6) the actions $\{\mathit{MPutOnTable}\}$ and $\{\mathit{FPutOnTable}\}$ are not executable in states satisfying $\neg\mathit{Dolls}$.

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

$$(\mathit{tr}_i^D \wedge \neg Z_i \wedge \neg Z_{i+1}) \vee (\neg \mathit{Poss}_i^D \wedge Z_{i+1}) \vee (Z_i \wedge Z_{i+1}), \tag{1.14}$$

where Z is a newly introduced fluent symbol. Given that $\mathit{tr}_i^D \supset \mathit{Poss}_i^D$ holds, (1.14) is logically equivalent to

$$(\neg Z_i \vee Z_{i+1}) \wedge (\mathit{tr}_i^D \vee Z_{i+1}) \wedge (\neg \mathit{Poss}_i^D \vee Z_i \vee \neg Z_{i+1}). \tag{1.15}$$

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 (1.14).*
2. *α is not executable in σ iff $\sigma_i \wedge \neg Z_i \wedge \alpha_i \wedge Z_{i+1}$ entails (1.14).*
3. *(1.14) 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 (1.14) is logically equivalent to (1.15). □

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 (1.3)–(1.6), then State_0^D is equivalent to

$$\mathit{Safe}_0 \equiv \mathit{OnTable}_0 \vee \mathit{Dolls}_0.$$

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*

$$I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D \models G_n \wedge \neg Z_n. \tag{1.16}$$

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 $I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D$. Since $\vec{\alpha}$ is valid, for any history (1.10) 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 $\wedge_{i=0}^{n-1} \mathit{trt}_i^D \wedge \neg Z_0$, and trt_i^D is equivalent to (1.15), it follows that μ satisfies $\neg Z_1, \ldots, \neg Z_n$ and $\wedge_{i=0}^{n-1} \mathit{tr}_i^D$. If μ satisfies $\wedge_{i=0}^{n-1} \mathit{tr}_i^D$, by Proposition 2, there exists a corresponding history (1.8). 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 (1.16) 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 (1.10) with $k < n$ such that α^{k+1} is not executable in σ^k. Let μ be the assignment to the variables in (1.16) 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 $I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D$. However, by construction μ also satisfies Z_n, which contradicts the hypotheses. □

Considering the planning problem (1.9), Theorem 2 can be used, e.g., to establish that the plans $\{$*MPutOnTable*$\}$,$\{$*MPutOnTable,FPutOnTable*$\}$ are respectively not valid and valid. To check it, consider the formula

$$I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D. \tag{1.17}$$

In both cases $n = 1$. But,

- When $\alpha^1 = \{$*MPutOnTable*$\}$, (1.17) is equivalent to the conjunction of the formulas

$$\begin{gathered} \neg \mathit{OnTable}_0, \\ \mathit{Safe}_0 \equiv \mathit{Dolls}_0, \\ \neg Z_0, \\ \mathit{MPutOnTable}_0 \wedge \neg \mathit{FPutOnTable}_0 \wedge \neg \mathit{MPutOnFloor}_0 \wedge \neg \mathit{FPutOnFloor}_0, \end{gathered}$$

and the formulas

$$\begin{gathered} \mathit{Safe}_1 \vee Z_1, \\ \mathit{Dolls}_0 \vee Z_1, \\ \mathit{Dolls}_1 \vee Z_1, \\ \mathit{OnTable}_1 \vee Z_1, \\ \neg \mathit{Dolls}_0 \vee \neg Z_1. \end{gathered}$$

This conjunction does not entail $\mathit{Safe}_1 \wedge \neg Z_1$. Indeed, $\{$*MPutOnTable*$\}$ is a valid plan for (1.9) if *Dolls* initially holds.

- When $\alpha^1 = \{$*MPutOnTable,FPutOnTable*$\}$, (1.17) is equivalent to the conjunction of

$$\begin{gathered}\neg \mathit{OnTable}_0,\\ \mathit{Safe}_0 \equiv \mathit{Dolls}_0,\\ \neg Z_0,\\ \mathit{MPutOnTable}_0 \wedge \mathit{FPutOnTable}_0 \wedge \neg \mathit{MPutOnFloor}_0 \wedge \neg \mathit{FPutOnFloor}_0,\end{gathered}$$

and the formulas

$$\begin{gathered}\mathit{Safe}_1,\\ \mathit{OnTable}_1,\\ \mathit{Dolls}_1 \equiv \mathit{Dolls}_0,\\ \neg Z_1.\end{gathered}$$

This conjunction obviously entails $\mathit{Safe}_1 \wedge \neg Z_1$.

1.3 $\mathcal{C}$-SAT: a SAT-based decision procedure for planning 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

$$I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D \wedge \neg(G_n \wedge \neg Z_n) \tag{1.18}$$

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

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

$P := I_0 \wedge \wedge_{i=0}^{n-1} \mathit{tr}_i^D \wedge G_n$;
$V := I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D \wedge \neg(G_n \wedge \neg Z_n)$;

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

function $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$(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$_{\mathrm{DLL}}$(*assign*(L, φ),$\mu \cup \{L\}$);
  L := { a literal occurring in φ };
  return $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$(*assign*(L, φ),$\mu \cup \{L\}$) **or** /* split */
      $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$(*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 1.2: $\mathcal{C}$-SAT, $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$ and $\mathcal{C}$-SAT_TEST.

- *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

$$I_0 \wedge \wedge_{i=0}^{n-1} \mathit{tr}_i^D \wedge G_n. \tag{1.19}$$

This can be accomplished using incomplete SAT solvers like GSAT [SLM92], or complete SAT solvers like SATO [Zha97] or SIM [GMTZ01]. 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 (1.19), and
- at step $i + 1$, ask for an assignment satisfying (1.19) 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 [DLL62], we get the procedure $\mathcal{C}$-SAT represented in Figure 1.2. 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., [Tse70, PG86]).

- $\overline{L}$ is the literal complementary to L.

- For any literal L and set of clauses φ, *assign*(L, φ) 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.

Theorem 3 *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 Theorem 3: $\mathcal{C}$-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. □

1.3.1 Optimizations

Consider a planning problem $\pi = \langle I, D, G\rangle$ and a natural number n. The procedure $\mathcal{C}$-SAT in Figure 1.2 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., [Dec90, Pro93, BS97]). 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 1.2 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.

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 (1.9), 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 (1.9), 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 (1.9) 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

$$N(\pi') \geq N(\pi''). \tag{1.20}$$

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 (1.9), 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 (1.20) 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$_{\mathrm{DLL}}$ procedure in Figure 1.2 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 (1.20) holds for each deterministic version π'' of π.

In (1.9), 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.

Incorporating Backjumping and Learning

Backjumping and learning are two familiar concepts in constraint satisfaction, and can produce significant speed-ups (see, e.g., [Dec90, Pro93, BS97]). Furthermore, the incorporation of analogous techniques is reported to lead to analogous improvements in plan-graph based planning (see [Kam00]).

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 [Pro93, BS97, GMTZ01]).

Here we extend the procedures described in [BS97, GMTZ01], 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 (1.10) 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.

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

$$I_0 \wedge \wedge_{i=0}^{n-1} \mathit{tr}_i^D \wedge (\vee_{i=0}^{n} G_i) \tag{1.21}$$

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

$$I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D \models \vee_{i=0}^{n} (G_i \wedge \neg Z_i). \tag{1.22}$$

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

$$\begin{array}{c}((tr_i^D \wedge \neg \mathit{NoOp}_i) \vee ((\wedge_{P\in s^f} P_{i+1} \equiv P_i) \wedge \mathit{NoOp}_i))\\ \wedge(\mathit{NoOp}_i \supset \wedge_{A\in s^a} \neg A_i),\end{array} \tag{1.23}$$

   where s^f and s^a are, respectively, the fluent and action signatures of D, and *NoOp* is a newly introduced action symbol. Intuitively, (1.23) 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

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

   for each state σ of D. Thus, on the basis of $tr_i^{D'}$, the formulas $\mathit{Poss}_i^{D'}$ and $trt_i^{D'}$ are defined as usual, while in the definition of the formulas P/V in Figure 1.2 we have to replace $tr_i^{D'}$, $trt_i^{D'}$ for tr_i^D, trt_i^D respectively. Finally, in Figure 1.2, 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 $\alpha^i(\mathit{NoOp}) = \mathit{True}$. 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

$$\wedge_{i=0}^{n-2}(\mathit{NoOp}_i \supset \mathit{NoOp}_{i+1}) \tag{1.24}$$

   to the definition of P in Figure 1.2. 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 (1.24), also $\mathit{NoOp}_1, \ldots, \mathit{NoOp}_{n-1}$: 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; $\neg\mathit{NoOp}_0$ is set to true and NoOp_1 is also set to true because of a splitting step. Again, because of (1.24), also $\mathit{NoOp}_2, \ldots, \mathit{NoOp}_{n-1}$ are set to true: These possible plans correspond to plans of π having length 1, and the computation proceeds along the same lines.

1.4 $\mathcal{C}$-QSAT: a QBF-based decision procedure for planning in $\mathcal{C}$+

Theorem 2 provides us also a direct encoding of a planning problem as a QBF satisfaction problem, as sanctioned by the following theorem.

Theorem 4 *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 $\wedge_{i=0}^{n-1}\alpha_i^{i+1}$ entails*

$$\forall s_0^f \ldots s_n^f Z_0 \ldots Z_n (I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D) \supset (G_n \wedge \neg Z_n). \tag{1.25}$$

where s^f is the fluent signature of D.

Proof of Theorem 4: The proof is an easy consequence of the fact that a formula α is a tautology if and only if its universal closure (obtained by universally quantifying all the symbols in α) is true. □

Theorem 4 says that there is a one-to-one correspondence between the valid plans of a planning problem $\langle I, D, G\rangle$ and the assignment satisfying (1.25). Thus, considering the planning problem (1.9), Theorem 4 can be used, e.g., to establish that the plans {*MPutOnTable*},{*MPutOnTable,FPutOnTable*} are respectively not valid and valid. To check it, consider the formula

$$I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D. \tag{1.26}$$

In both cases $n = 1$, but,

- When $\alpha^1 = \{\mathit{MPutOnTable}\}$, if we substitute $\mathit{MPutOnTable}_0$ with true and the other action variables with false, (1.26) is equivalent to the conjunction of the formulas

  $$\neg \mathit{OnTable}_0,$$
  $$\mathit{Safe}_0 \equiv \mathit{Dolls}_0,$$
  $$\neg Z_0,$$

  and the formulas

  $$\mathit{Safe}_1 \vee Z_1,$$
  $$\mathit{Dolls}_0 \vee Z_1,$$
  $$\mathit{Dolls}_1 \vee Z_1,$$
  $$\mathit{OnTable}_1 \vee Z_1,$$
  $$\neg \mathit{Dolls}_0 \vee \neg Z_1.$$

  This conjunction does not entail $\mathit{Safe}_1 \wedge \neg Z_1$. Indeed, {*MPutOnTable*} is a valid plan for (1.9) if *Dolls* initially holds.

$Q := \forall s_0^f \ldots s_n^f Z_0 \ldots Z_n (I_0 \wedge \mathit{State}_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D) \supset (G_n \wedge \neg Z_n)$;

function $\mathcal{C}$-QSAT() **return** QBF-SOLVER(*cnf*(Q)).

Figure 1.3: $\mathcal{C}$-QSAT.

- When $\alpha^1 = \{$*MPutOnTable*,*FPutOnTable*$\}$, if we substitute $\mathit{MPutOnTable}_0$ and $\mathit{FPutOnTable}_0$ with true and the other action variables with false, (1.26) is equivalent to the conjunction of
$$\neg \mathit{OnTable}_0,$$
$$\mathit{Safe}_0 \equiv \mathit{Dolls}_0,$$
$$\neg Z_0,$$
and the formulas
$$\mathit{Safe}_1,$$
$$\mathit{OnTable}_1,$$
$$\mathit{Dolls}_1 \equiv \mathit{Dolls}_0,$$
$$\neg Z_1.$$
This conjunction obviously entails $\mathit{Safe}_1 \wedge \neg Z_1$.

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

By Theorem 4, the valid plans of π are the assignment satisfying (1.25), and thus it is sufficient to invoke a QBF solver on (1.25) as a black box. Unfortunately, most of the available QBF solvers take in input formulas whose matrix is in Conjunctive Normal Form (CNF), and thus it is necessary to preliminary convert the matrix of (1.25) in CNF. This is what the procedure $\mathcal{C}$-QSAT in Figure 1.3 does. In the Figure,

- *cnf*(Q) is the QBF whose matrix is converted to a set of clauses. This transformation can be performed using the conversions based on "renaming" (see, e.g., [Tse70, PG86]).
- QBF-SOLVER is an arbitrary QBF solver. In the case the QBF solver assumes the input formula to be closed (i.e., in which all the symbols are quantified), then it is assumed that *cnf*$(())Q)$ returns the existential closure of Q.

Theorem 5 *Let $\pi = \langle I, D, G \rangle$ be a planning problem. Let n be a natural number. $\mathcal{C}$-QSAT is correct and complete for π, n.*

Proof of Theorem 5: Correctness and completeness of $\mathcal{C}$-QSAT are easy consequences of Theorem 4. □

Notice that some of the currently available QBF solvers implement backjumping and learning, corresponding to the analogous optimizations introduced for $\mathcal{C}$-SAT and that in $\mathcal{C}$-QSAT are inherited for free. Also, assuming we have a QBF solver as back-end, it is possible to return "contingent plans", i.e., plans in which the action to be executed depend on the value of some observable fluents. Well-known encodings exists, see, e.g., [Rin99a, Tur02].

1.5 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. $\mathcal{C}$-PLAN uses *CCALC* system by Norman McCain as front-end and then can use either the procedure $\mathcal{C}$-SAT for determining the existence of a plan of a given length, or a QBF solver, as described in Sections 1.3 and 1.4. In the following, we call $\mathcal{C}$-SATPLAN and $\mathcal{C}$-QSATPLAN the system $\mathcal{C}$-PLAN when using $\mathcal{C}$-SAT and $\mathcal{C}$-QSAT as back engine respectively. Before discussing the implementation and presenting some experiments with $\mathcal{C}$-SATPLAN and $\mathcal{C}$-QSATPLAN we briefly present the *CCALC* system.

1.5.1 *CCALC*

The original version of the *CCALC* was written by Norman McCain as part of his dissertation. Now the system is being maintained by Texas Action Group at Austin.[2] It can be downloaded from its home page

`http://www.cs.utexas.edu/users/tag/ccalc/ .`

To illustrate the syntax of the input language of *CCALC*, we show in Figures 1.6 and 1.7 how to rewrite in that language the $\mathcal{C}$+ description of the Monkey and Bananas domain (see deliverable d4.2) given in Figures 1.4 and 1.5.

CCALC is written in Prolog, and it follows the Prolog tradition of capitalizing variables. This is why, in the language of *CCALC*, $Loc(Monkey) = l$ turns into `loc(monkey)=L`.

The ranges of variables, described at the beginning of Figure 1.4, are given names in the sort declarations at the beginning of Figure 1.6. The extent of each sort is defined in object declarations.

The constant declaration

```
loc(thing)              :: inertialFluent(location)
```

has the same meaning as the declaration

[2] `http://www.cs.utexas.edu/users/tag .`

Notation: x ranges over $\{\mathit{Monkey}, \mathit{Bananas}, \mathit{Box}\}$; l ranges over $\{L_1, L_2, L_3\}$.

Simple fluent constants:	Domains:
$\mathit{Loc}(x)$	$\{L_1, L_2, L_3\}$
$\mathit{HasBananas}, \mathit{OnBox}$	Boolean

Action constants:	Domains:
$\mathit{Walk}(l)$, $\mathit{PushBox}(l)$, *ClimbOn*, *ClimbOff*, *GraspBananas*	Boolean

Causal laws:

caused $\mathit{Loc}(\mathit{Bananas}) = l$ **if** $\mathit{HasBananas} \wedge \mathit{Loc}(\mathit{Monkey}) = l$
caused $\mathit{Loc}(\mathit{Monkey}) = l$ **if** $\mathit{OnBox} \wedge \mathit{Loc}(\mathit{Box}) = l$

$\mathit{Walk}(l)$ **causes** $\mathit{Loc}(\mathit{Monkey}) = l$
nonexecutable $\mathit{Walk}(l)$ **if** $\mathit{Loc}(\mathit{Monkey}) = l$
nonexecutable $\mathit{Walk}(l)$ **if** OnBox

$\mathit{PushBox}(l)$ **causes** $\mathit{Loc}(\mathit{Box}) = l$
$\mathit{PushBox}(l)$ **causes** $\mathit{Loc}(\mathit{Monkey}) = l$
nonexecutable $\mathit{PushBox}(l)$ **if** $\mathit{Loc}(\mathit{Monkey}) = l$
nonexecutable $\mathit{PushBox}(l)$ **if** OnBox
nonexecutable $\mathit{PushBox}(l)$ **if** $\mathit{Loc}(\mathit{Monkey}) \neq \mathit{Loc}(\mathit{Box})$

Figure 1.4: Action description *MB*, Part 1.

```
loc(thing)              :: simpleFluent(location)
```

accompanied by the fluent dynamic law

```
inertial loc(X)
```

where `X` is a variable for `things`. That is to say, this declaration says that any expression consisting of the symbol `loc` followed by a `thing` in parentheses is a simple fluent constant, that the domain of each of these constants is the set of `locations`, and that these constants are postulated to be inertial. The symbol `inertialFluent` is used more often than `simpleFluent` in *CCALC* action descriptions, because the inertia assumption is required for almost every simple fluent constant. When the symbol `inertialFluent` in the declaration of a constant is not followed by a sort, the constant is assumed to be Boolean. This convention is used in the declarations of `onBox` and `hasBananas`. Similarly, the action constants declared in Figure 1.6 are understood to be Boolean, since no other domain is specified.

ClimbOn **causes** *OnBox*
nonexecutable *ClimbOn* **if** *OnBox*
nonexecutable *ClimbOn* **if** $Loc(Monkey) \neq Loc(Box)$

ClimbOff **causes** $\neg$*OnBox*
nonexecutable *ClimbOff* **if** $\neg$*OnBox*

GraspBananas **causes** *HasBananas*
nonexecutable *GraspBananas* **if** *HasBananas*
nonexecutable *GraspBananas* **if** $\neg$*OnBox*
nonexecutable *GraspBananas* **if** $Loc(Monkey) \neq Loc(Bananas)$

nonexecutable $Walk(l) \wedge PushBox(l)$
nonexecutable $Walk(l) \wedge$ *ClimbOn*
nonexecutable $PushBox(l) \wedge$ *ClimbOn*
nonexecutable *ClimbOff* $\wedge$ *GraspBananas*

exogenous c	for every action constant c
inertial c	for every simple fluent constant c

Figure 1.5: Action description *MB*, Part 2.

The declaration

```
walk(location)              :: exogenousAction
```

has the same meaning as the declaration

```
walk(location)              :: action
```

accompanied by the action dynamic law

```
exogenous walk(L)
```

where `L` is a variable for `locations`. That is to say, this declaration says that any expression consisting of the symbol `walk` followed by a `location` in parentheses is a Boolean-valued action constant, and that these constants are postulated to be exogenous. The symbol `exogenousAction` is used more often than `action` in *CCALC* action descriptions, because most actions are exogenous.

The rest of the *CCALC* encoding of the Monkey and Bananas domain consists of causal laws. It is almost identical to the list of causal laws in Figures 1.4, 1.5.

```
:- sorts
  thing;
  location.

:- objects
  monkey,bananas,box     :: thing;
  l1,l2,l3               :: location.

:- variables
  L                      :: location.

:- constants
  loc(thing)             :: inertialFluent(location);
  hasBananas,onBox       :: inertialFluent;

  walk(location),
  pushBox(location),
  climbOn,
  climbOff,
  graspBananas           :: exogenousAction.

caused loc(bananas)=L if hasBananas & loc(monkey)=L.
caused loc(monkey)=L if onBox & loc(box)=L.

walk(L) causes loc(monkey)=L.
nonexecutable walk(L) if loc(monkey)=L.
nonexecutable walk(L) if onBox.

pushBox(L) causes loc(box)=L.
pushBox(L) causes loc(monkey)=L.
nonexecutable pushBox(L) if loc(monkey)=L.
nonexecutable pushBox(L) if onBox.
nonexecutable pushBox(L) if loc(monkey)\=loc(box).
```

Figure 1.6: Monkey and Bananas in the language of *CCALC*, Part 1.

```
climbOn causes onBox.
nonexecutable climbOn if onBox.
nonexecutable climbOn if loc(monkey)\=loc(box).

climbOff causes -onBox.
nonexecutable climbOff if -onBox.

graspBananas causes hasBananas.
nonexecutable graspBananas if hasBananas.
nonexecutable graspBananas if -onBox.
nonexecutable graspBananas if loc(monkey)\=loc(bananas).

nonexecutable walk(L) & pushBox(L).
nonexecutable walk(L) & climbOn.
nonexecutable pushBox(L) & climbOn.
nonexecutable climbOff & graspBananas.
```

Figure 1.7: Monkey and Bananas in the language of *CCALC*, Part 2.

Our version of *CCALC* computes the set of clauses corresponding to the transition relation tr_0^D of any (not necessarily definite) action description D.

1.5.2 $\mathcal{C}$-SATPLAN

Figure 1.8 shows the overall architecture of $\mathcal{C}$-SATPLAN. 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 1.2.

Consider Figure 1.8.

As we have already seen, *CCALC* computes the set of clauses corresponding to the transition relation tr_0^D of any (not necessarily definite) action description D.

CTCALC determines, on the basis of tr_0^D, the set of clauses corresponding to trt_0^D. *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

caused *False* **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 [FG00]. We also assume that, given a planning problem $\langle I, D, G\rangle$, each assignment satisfying I is a state

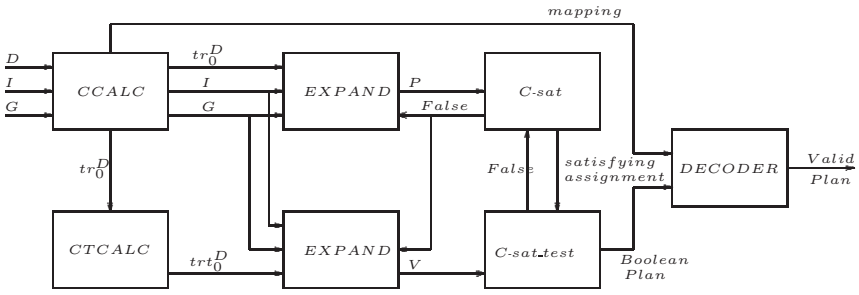

Figure 1.8: Architecture of $\mathcal{C}$-SATPLAN

of D.[3]

$\mathcal{C}$-SAT and $\mathcal{C}$-SAT_TEST implement the algorithms in Figure 1.2 and the optimizations in Sections 1.3.1, 1.3.1. In the experiments here reported, $\mathcal{C}$-SAT and $\mathcal{C}$-SAT_TEST are implemented on top of SIM [GMTZ01]. SIM is an efficient library for SAT developed by our group, and features many splitting heuristics and size/relevance learning [BS97].[4] In all the following experiments, both $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$ and $\mathcal{C}$-SAT_TEST use the unit-based heuristics described in [LA97] and relevance learning of size 4 [BS97].

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 $\mathcal{C}$-SATPLAN we consider an elaboration of the traditional "bomb in the toilet" problem from [McD87]. 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 [BG00]: 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:`

[3]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 (1.1) in D.

[4]In the previous version, used for the experimental analysis described in [FG00], these modules were based on *SAT [GT00], and *SAT was based on the SATO SAT solver [Zha97].

	GPT	CMBP		QBFPLAN			$\mathcal{C}$-SATPLAN				
$\|P\|$-$\|T\|$	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 1.1: Bomb in the toilet: Classic version

`//www.ldc.usb.ve/~hector/`.

- Cimatti and Roveri's CMBP [CR99, CR00]: In CMBP, the planning domain is represented as a finite state automaton, and Binary Decision Diagrams (BDDs) [Bry92] are used to represent and search the automaton. CMBP allows for planning domains specified in the $\mathcal{AR}$ language [GKL97], 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 [Rin99a]: 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. 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 $\mathcal{C}$-SATPLAN. In our experiments, we used the latest version of the solver QSAT [Rin99b, Rin01], 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 $\mathcal{C}$-SATPLAN 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 1.1. 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 $\mathcal{C}$-SATPLAN,
  - 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"),
  - 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 1.1, $\mathcal{C}$-SATPLAN 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 1.2 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 [CR00]. As we can see from Table 1.2, when there is only one toilet $\mathcal{C}$-SATPLAN'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			$\mathcal{C}$-SATPLAN				
$\|P\|$-$\|T\|$	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 1.2: Bomb in the toilet: Multiple toilets, clogging, one bomb.

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$_{\mathrm{DLL}}$: 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, $\mathcal{C}$-SATPLAN's performances are not too bad compared to the ones of the other solvers: $\mathcal{C}$-SATPLAN, CMBP, GPT, QBFPLAN do not solve 4, 3, 3, 2 problems respectively. As expected, $\mathcal{C}$-SATPLAN 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 1.3.

Contrarily to what could be expected, $\mathcal{C}$-SATPLAN's performances are much better on the problems in Table 1.3 than on those in Table 1.2. This is most evident if we compare

	GPT	CMBP		QBFPLAN			$\mathcal{C}$-SATPLAN				
$\|P\|$-$\|T\|$	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 1.3: Bomb in the toilet: Multiple toilets, clogging, possibly multiple bombs.

the number of plans generated and tested by $\mathcal{C}$-SATPLAN before finding a solution. For example, if we consider the four packages and one toilet problem,

- with one bomb, as in Table 1.2, $\mathcal{C}$-SATPLAN generates 540 possible plans and takes 0.65s to solve the problem (0.15s of search time),
- with possibly multiple bombs, as in Table 1.3, $\mathcal{C}$-SATPLAN 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, $\mathcal{C}$-SATPLAN adds a clause to the initial state saying that at least one package is armed.

For $n = 1$, $\mathcal{C}$-SATPLAN 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 $\mathcal{C}$-SATPLAN 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, $\mathcal{C}$-SATPLAN 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 $\mathcal{C}$-SATPLAN 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 1.3.1 and Section 1.3.1 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 1.2, $\mathcal{C}$-SATPLAN 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 1.3, $\mathcal{C}$-SATPLAN 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 1.3 than on the problems in Table 1.2: Overall, $\mathcal{C}$-SATPLAN, CMBP and GPT do not solve respectively 2, 3 and 2 of the problems in Table 1.3. 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 1.2.[5] Overall, we get roughly the same picture that we had before: $\mathcal{C}$-SATPLAN 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 $\mathcal{C}$-SATPLAN is still at an early stage of development, we can conclude that $\mathcal{C}$-SATPLAN (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,

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

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

where $a_1, \dots, a_n$ are the variables corresponding to actions; $s_1, \dots, s_{2^m}$ are conjunctions of literals, corresponding to all the possible initial states; $d_1, \dots, d_m$ are "dummy" variables; $v_1, \dots, v_f$ are the variables corresponding to fluents; see [Rin99a] for more details. In the experiments in Table 1.2 there are $|P|$ possible initial states, and thus $log_2|P|$ dummy variables. In the experiments in Table 1.3 there are $2^{|P|}$ possible initial states, and thus $|P|$ dummy variables.

in [HG00a], BLACKBOX [KS98], GRAPHPLAN [BF95], STAN [LF99] and HSPr [HG00a], 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 $\mathcal{C}$-SATPLAN ($\mathcal{C}$-SATPLAN 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., [AGKS00]). 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.

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

N	d		$M=1$ Plan	#s	#pp	Last	Tot-S	Total	$M=2$ 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 1.4: $\mathcal{C}$-SATPLAN's performances on robot navigation problems

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 [AGKS00], 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
- 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 [MM93]. 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 1.4. Notice that we only test $\mathcal{C}$-SATPLAN, 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 1.4, $\mathcal{C}$-SATPLAN's performances get worse and worse as M increases: When $M = 1$, $\mathcal{C}$-SATPLAN is not able to solve problems having $d = 3$ and $N = 7, 9$. When $M = 2$, $\mathcal{C}$-SATPLAN 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 $\wedge_{i=0}^{n-1} \mathit{tr}_i^D$ and $\mathit{State}_0^D \wedge \neg Z_0 \wedge_{i=0}^{n-1} \mathit{trt}_i^D$, 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.

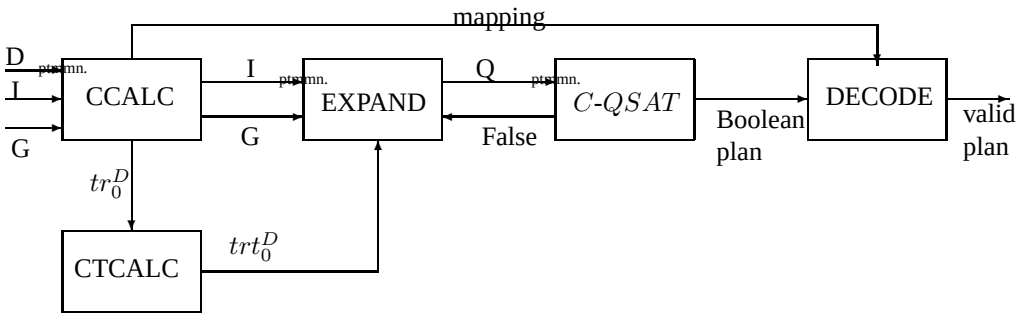

Figure 1.9: Architecture of $\mathcal{C}$-QSATPLAN

1.5.3 $\mathcal{C}$-QSATPLAN

As for $\mathcal{C}$-SATPLAN, $\mathcal{C}$-QSATPLAN assumes that, given a planning problem $\langle I, D, G\rangle$, each assignment satisfying I is a state of D. We recall that this is not a restriction since any planning problem $\langle I, D, G\rangle$, can be equivalently replaced by $\langle I', D, G\rangle$ where I' is obtained from I by adding a conjunct $G \supset F$ for each static law (1.1) in D.

Figure 1.9 shows the overall architecture of $\mathcal{C}$-QSATPLAN. 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 1.3.

Consider Figure 1.9.

As we have already seen, *CCALC* computes the set of clauses corresponding to the transition relation tr_0^D of any (not necessarily definite) action description D.

CTCALC determines, on the basis of tr_0^D, the set of clauses corresponding to trt_0^D. *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

caused *False* **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 [FG00]. Finally, both *CCALC* and *CTCALC* implement clause form transformations based on renaming (see, e.g., [PG86]).

The performances of $\mathcal{C}$-QSATPLAN heavily depends on the efficiency of the underlying QBF solver used by $\mathcal{C}$-QSAT. Tables 1.5 and 1.6 show the performances of the QBF solvers in the 2006 QBF evaluation on bomb in the toilet and robot problems respectively.

As it can be seen from the tables, the various versions of QUBE are among the most effective. This is even more evident if we consider all the planning problem in the 2006 QBF evaluation, whose results are shown in Table 1.7.

Solver	Total time	Mean time	Std time	Median time	IQ range time	Solved/Total
2clsQ	92.56	30.85	41.67	2.74	2.69	3/3
GRL	257.19	85.73	118.93	3.27	3.26	3/3
openQbf	10506.92	3502.31	2550.35	4505.59	4505.18	2/3
preQuantor	91.18	30.39	41.12	2.61	2.56	3/3
Qbfl	12594.11	4198.04	2544.41	5996.97	5397.27	0/3
Quaffle	6021.14	2007.05	2823.57	12.07	3.18	2/3
QUANTOR	34.72	11.57	14.33	2.9	2.85	3/3
QUANTOR_hc	34.63	11.54	14.32	2.86	2.82	3/3
qube3.0	12.04	4.01	3.33	3.81	3.77	3/3
qube4.0	29.15	9.72	9.63	6.21	6.14	3/3
qube5.0	16.84	5.61	3.91	7.94	7.84	3/3
semprop	76.57	25.52	31.59	6.22	5.94	3/3
sKizzo-0.9-abs	59.36	19.79	27.58	0.53	0.49	3/3
sKizzo-0.9-grn	12.16	4.05	5.37	0.47	0.42	3/3
sKizzo-0.9-std	62.64	20.88	29.06	0.61	0.56	3/3
SQBF	91.43	30.48	41.18	2.66	2.59	3/3
sSolve	47.74	15.91	16.31	8.72	8.19	3/3
ssolve+ut	10.77	3.59	3.54	1.63	1.05	3/3
ssolve-ut	6008.34	2002.78	2826.67	7.45	6.86	2/3
WalkQSAT	29.94	9.98	11.98	3.1	3.09	3/3
yQuaffle	628.53	209.51	275.93	24.22	19.48	2/3

Table 1.5: Performances of various 2006 QBF solvers on bomb in the toilet problems

Solver	Total time	Mean time	Std time	Median time	IQ range time	Solved/Total
2clsQ	94323.38	2358.08	1179.83	2424.08	1240.39	27/40
GRL	2544.15	63.6	27.77	64.42	24.49	40/40
openQbf	12058.6	301.47	76.88	314.04	131.8	40/40
preQuantor	2541.61	63.54	21.46	72.92	32.22	0/40
Qbfl	240027.92	6000.7	0.67	6000.96	0.09	0/40
Quaffle	22600.48	565.01	5.03	563.08	7.27	0/40
QUANTOR	2379.28	59.48	20.77	49.97	40.18	0/40
QUANTOR_hc	2430.95	60.77	21.51	50.16	42.13	0/40
qube3.0	188.58	4.71	2.14	4.64	2.4	40/40
qube4.0	605.1	15.13	7.49	16.5	10.41	40/40
qube5.0	610.99	15.27	7.5	16.7	10.32	40/40
semprop	18785.28	469.63	191.41	600.99	231.39	15/40
sKizzo-0.9-abs	76.07	1.9	0.6	1.78	0.76	40/40
sKizzo-0.9-grn	23945.01	598.63	1.16	598.86	0.56	0/40
sKizzo-0.9-std	15925.43	398.14	1186.77	5.07	182.89	35/40
SQBF	3588.74	89.72	20.75	97.97	35.62	40/40
sSolve	1061.03	26.53	11.77	29.16	17.05	40/40
ssolve+ut	1679.62	41.99	16.25	47.12	19.86	40/40
ssolve-ut	1059.46	26.49	11.76	29.11	17.34	40/40
WalkQSAT	3029.62	75.74	56.55	70.64	20.62	0/40
yQuaffle	130.04	3.25	1.09	3.33	1.37	40/40

Table 1.6: Performances of various 2006 QBF solvers on the robot problems

Solver	Total time	Mean time	Std time	Median time	IQ range time	Solved/Total
2clsQ	2540604.55	3650.29	2745.64	6000.42	5858.86	289/696
GRL	302298.46	434.34	258.32	600.84	516.04	200/696
openQbf	3260737.28	4684.97	2388.27	6000.87	0.51	166/696
preQuantor	60100.79	86.35	400.14	59.66	80.68	115/696
Qbfl	3284614.27	4719.27	2370.31	5998.64	6.02	111/696
Quaffle	831290.96	1194.38	1840.27	559.58	24.27	90/696
QUANTOR	48672.4	69.93	60.07	66.4	42.8	131/696
QUANTOR_hc	48835.71	70.17	59.95	66.96	43.85	131/696
qube3.0	1358381.23	1951.7	2690.69	48.91	5995.8	485/696
qube4.0	1374702.01	1975.15	2684.23	72.87	5996.43	483/696
qube5.0	1384905.45	1989.81	2676.45	71.76	5996.66	483/696
semprop	188425	270.73	291.17	26.56	600.87	402/696
sKizzo-0.9-abs	2643638.2	3798.33	2648.87	6000.08	5884.12	187/696
sKizzo-0.9-grn	2929341.74	4208.82	2673.14	6000.65	5402	117/696
sKizzo-0.9-std	2961065.68	4254.4	2629.59	6000.61	5403.24	149/696
SQBF	2636312.76	3787.81	2544.3	5025.71	5723.13	192/696
sSolve	3049634.92	4381.66	2602.54	6000.25	5428.45	200/696
ssolve+ut	2842360.97	4083.85	2706.07	6000.25	5422.69	199/696
ssolve-ut	2426189.85	3485.9	2815.8	6000.24	5400.05	185/696
WalkQSAT	318711.85	457.92	244.14	600.97	285.21	87/696
yQuaffle	609436.27	875.63	1545.92	558.44	550.74	197/696

Table 1.7: Performances of various 2006 QBF solvers on the planning problems in the 2006 QBF evaluation

QUBE is the solver that solved the highest number of instances. Further, it is the solver which has the lowest mean and median times. As a consequence, QUBE is indeed the solver of choice to be used with $\mathcal{C}$-QSATPLAN. Notice that Table 1.7 involves various forms of planning problems, from conditional (like the bomb in the toilet and robot problems) up to contigent problems where the action to be executed depends on the value of some "observable" fluents.

1.6 Satisfiability Planning and Answer Set Programming

Computationally, the operation of *CCALC* as a planner is a form of satisfiability planning [KS92]. Since causal theories are closely related to logic programs, it can be also viewed as a form of answer set programming [MT99], [Nie99] applied to plan generation [Lif02]. In the case of deterministic domains, the operation of $\mathcal{C}$-SATPLAN is particularly close to the operation of the answer set solver CMODELS [GML04]. This does not come as a surprise, given the close correspondence existing between answer set and SAT solvers [GM05]. The possibility to use answer set solvers as planning systems [Lif02], and the close correspondence between answer set and SAT solvers are the main motivation at the basis of the works [GML04] and [GM05] in the context of this project. Here we do not report about these works and refer to the corresponding papers for the details.

1.7 Conclusions and Related Work

We have presented SAT- and QBF-based procedures 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 procedures, discussed some optimizations, and then we presented $\mathcal{C}$-PLAN, a system based on our procedures. The experimental analysis shows that SAT-based approaches to planning with incomplete information (i.e., $\mathcal{C}$-SATPLAN) can be competitive with CMBP [CR99, CR00], GPT [BG00], and QBFPLAN [Rin99a] 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 case of $\mathcal{C}$-QSATPLAN, we have shown that by using QBF solvers as black boxes, $\mathcal{C}$-QSATPLAN inherits for free the advance in the development of the QBF field, as witnessed by the yearly QBF solvers comparative evaluations [BST04, NPT06].

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 [CR99, CR00], GPT [BG00], CGP [SW98], QBFPLAN [Rin99a] and WSPDF [FPR00].

CMBP is based on the representation of the planning domain as a finite state automaton, and uses BDDs [Bry92] 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 $\mathcal{AR}$ [GKL97], 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 [BCR01]. 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 [Nil80], 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 [BF95] 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 [CR00, HG00b], 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), as in $\mathcal{C}$-QSATPLAN. 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 $\mathcal{C}$-SATPLAN and $\mathcal{C}$-QSATPLAN: 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 $\mathcal{C}$-SATPLAN: First a sequence of actions is generated, and then its validity is checked. Indeed, for any planning problem spec-

ified 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 $\mathcal{C}$-SATPLAN 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 1.3.1. 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. However, QBFPLAN and $\mathcal{C}$-QSATPLAN have the advantage that QBF solvers are used as black-boxes and thus they can be very easily integrated. As a consequence, both QBFPLAN and $\mathcal{C}$-QSATPLAN can take advantage of the latest advances in the development of QBF solvers, each year incorporating the most effective one.

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 $\mathcal{C}$-SATPLAN.

Our work can be seen as a follow up of [MT98]. In [MT98], 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 [MT97], and is less expressive than the language of causal theories used in [MT98]. However, the focus in [MT98] 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 [FG00], 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 [KS96, KMS96, EMW97] are possible. According to the experimental analysis done in [FG00], the regular parallel encodings are those leading to the best performances, as in the classical case.

The encodings and optimizations presented in [FG00] 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 [HG00a] 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 [BCCZ99], a verification problem is converted into a SAT problem by unrolling the transition relation n-times (as proposed by Kautz and Selman in planning [KS92]) 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 $\mathcal{C}$-SATPLAN, 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 [BCCZ99, Sht00, CFG$^+$01].

Finally, a very different SAT-based procedure for conformant planning has been very recently proposed in [KNS02]. 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.

Chapter 2

SAT- and QBF-Based Planning with Extended Goals

2.1 Introduction

We address the problem of "safe planning". By "safe planning", we mean the task of generating/validating plans that not only achieve the goal, but verify also a set of other user defined properties, e.g., safety properties. More in details, we show how it is possible to define procedures

- for the validation of user defined plans with respect to given (safety) properties, and
- for the generation of plans satisfying the desired properties.

We focus on the planning as satisfiability approach introduced in [KS92] in the classical setting, and extended in [GML04] for dealing with nondeterministic domains. Thus, both the validation and the generation of plans are reduced to the problem of checking the satisfiability of certain formulas. Differently from [KS92, GML04] which reduce to propositional satisfiability (SAT), our target logical formalism will be that of Quantified Boolean Formulas (QBFs). This is because we use the highly expressive *Quantified Linear Temporal Logic* [Fin82] to express the properties the system has to fulfill. The availability of effective QBF solvers [BST04] makes the approach attractive.

The chapter is structured as follows. In Section 2.2 we briefly introduce the basic planning terminology and definitions used in the rest of the chapter. Section 2.3 is devoted to the introduction of the temporal logic we use to formally define the properties that safe plans have to meet. In Section 2.4 we show how it is possible to check whether a valid plan (either user defined, or automatically generated by a planner) is also safe. The automatic generation of safe plans is the object of Section 2.5. In Section 2.6, we draw some conclusions, and discuss some related work.

2.2 Possible and Valid Plans

We start with a set of atoms partitioned into a set of *fluent symbols*, a set of *action symbols*, and a set of *quantified symbols*. Intuitively, a fluent represents a property of the world that changes over time, because of the execution of actions. A *propositional formula* is a propositional combination of atoms, and a *Quantified Boolean Formula (QBF)* is a propositional formula in which all the quantified symbols are either existentially or universally quantified. An *action* is an interpretation of the action symbols. A *state* is an interpretation of the fluent signature. A *transition* is a triple $\langle s, a, s' \rangle$ where s, s' are states and a is an action: Intuitively s is the initial state of the transition, and s' is its resulting state. Intuitively, to execute an action a means to execute concurrently the "elementary actions" represented by the action symbols satisfied by a.

An *action description* D is a finite set of expressions describing how actions change the state of the world, i.e., the set of possible transitions. We do not make any assumption about the language used to specify D, except the following:

1. the effects of actions depend on the state of the world in which they are executed.

This restriction does not allow for "non-markovian" action descriptions like the one definable in $\mathcal{ARD}$ [GL95]. Under the assumption that D is Markovian, it is possible to associate a transition diagram with each action description D. The *transition diagram* represented by D is the directed graph which has the states of D as vertices, and which includes an edge from s to s' labeled a for every transition $\langle s, a, s' \rangle$ that is possible according to D. Notice that D may be deterministic or non-deterministic.

Besides the above assumption, we also assume that

2. it is possible to compute a QBF, called tr_i^D, whose satisfying assignments correspond to the possible transitions caused by the execution of an action in D.

More in detail, we assume that in tr_i^D there is a propositional variable A_i for each action symbol A, and two propositional variables F_i and F_{i+1} for each fluent symbol F in D: Intuitively, F_i represents the value of F in the initial state of the transition, and F_{i+1} represents the value of F in the resulting state, after having performed the action.

As in the standard planning literature, a planning problem for D is characterized by an initial state and a set of goal states. In our setting, the initial and the goal states are represented as two QBFs in the fluent signature. Thus, for us, a *planning problem* is a triple $\langle I, D, G \rangle$, where I and G are QBFs encoding the initial and goal state(s) respectively, and D is an action description. A *plan* (of *length* $n \geq 0$) is a finite sequence $a^1; \ldots; a^n$ of actions.

Consider a planning problem $\pi = \langle I, D, G \rangle$. Intuitively, to ensure that a plan $a^1; \ldots; a^n$ is valid, we have to check

- that the plan is "always executable" in any initial state, i.e., executable for any initial state and any possible outcome of the actions in the plan, and

- that any "possible result" of executing the plan in any initial state is a goal state.

In order to make the above definition precise we have to give the following definitions.

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

$$s^0, a^1, s^1, \ldots, a^n, s^n \tag{2.1}$$

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

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

are transitions which are possible according to D. n is the *length* of the history (2.1).

A plan $\vec{a} = a^1; \ldots; a^n$ is *possible* for π if there exists a history (2.1) for D such that

- s^0 is an initial state, and

- s^n is a goal state.

An action a is *executable* in a state s if for some state s', $\langle s, a, s' \rangle$ is a possible transition according to D. Let s^0 be a state. The plan $a^1; \ldots; a^n$ is *always executable in* s^0 if for any history

$$s^0, a^1, s^1, \ldots, a^k, s^k$$

with $k < n$, a^{k+1} is executable in s^k. Assume that $\vec{a}$ is a plan which is always executable in a state s^0. A state s^n is a *possible result of executing* $\vec{a}$ *in* s^0 if there exists a history (2.1) for D.

A plan $\vec{a} = a^1; \ldots; a^n$ is *valid* for π if for any initial state s^0,

- $\vec{a}$ is always executable in s^0, and

- any possible result of executing $\vec{a}$ in s^0 is a goal state.

Indeed, if D is deterministic, any possible plan is also valid. However, this is not the case if D is nondeterministic. A possible plan *may* lead to the goal; while valid plans are ensured to reach a goal state despite the potential multiple initial states and nonderminism in the action description. In the planning literature, valid plans are also called "conformant" [SW98].

2.3 Safe Plans

Consider a planning problem $\pi = \langle I, D, G \rangle$. In the previous section we have formally defined the notion of valid plan as a sequence of actions which is guaranteed to reach a goal state. Depending on the characteristics of π (e.g., depending on the language used to describe D) different planning procedures can be used. However, as we have already argued in the introduction, we do not only want that a plan be "valid", but also "safe" with respect to a set of user-defined properties. For example, we do not want a plan which can go through a state which is potentially dangerous for humans.

In the formal verification literature, these (safety) properties are nicely formalized by means of temporal logics. Here we consider *Quantified (Propositional) Linear Temporal Logic (QLTL)* [Fin82], which extends the more popular (Propositional) Lineral Temporal Logic [Pnu77] in that it allows for quantification over variables. Here variables are assumed to be *rigid*, i.e., that their value does not change over time. The definition of a QLTL formula (used for the specification of the properties) is the following:

- A fluent or a quantified symbol is a *QLTL formula*, and
- The negation of a fluent or of a quantified symbol is a *QLTL formula*, and
- If α and β are QLTL formulas, also

$$\begin{array}{cccc} (\alpha \wedge \beta), & (\alpha \vee \beta), & \mathcal{X}_s\alpha, & \mathcal{X}_w\alpha, \\ \mathcal{F}\alpha, & \mathcal{G}\alpha, & (\alpha \mathcal{U} \beta), & (\alpha \mathcal{R} \beta), \end{array}$$

  are QLTL formulas, and

- If α is a QLTL formula, and x is quantified symbol, $\exists x\alpha$ and $\forall x\alpha$ are QLTL formulas. We also say that the occurences of x in α and to the right of the quantifier are *bounded*.

As before, we consider QLTL formulas in which all the quantified symbols are bounded.

Notice that we have assumed QLTL formulas to be in negation normal form, i.e., that negations occur only in front of fluent symbols. This is not a limitation given that —according to the semantics defined below— (assuming we relax for a moment the assumption to be in negation normal form)

$$\begin{array}{c} \neg\mathcal{X}_s\alpha \equiv \mathcal{X}_w\neg\alpha, \\ \neg\mathcal{F}\alpha \equiv \mathcal{G}\neg\alpha, \\ \neg(\alpha \mathcal{U} \beta) \equiv (\neg\alpha \mathcal{R} \neg\beta). \end{array} \tag{2.2}$$

Some of the above equivalences hold in standard QLTL. However, the meaning of the temporal connectives $\mathcal{X}_w, \mathcal{F}, \mathcal{G}, \mathcal{R}$ have to be adjusted in order to accommodate the fact that we are dealing with a finite time line (see [Eme90a], pag. 1006). Intuitively, if n represent our horizon,

- $\mathcal{X}_s\alpha$ reads "there exists a successor moment and α holds there",
- $\mathcal{X}_w\alpha$ reads "if there exists a successor moment then α holds there",
- $\mathcal{F}\alpha$ reads "for some subsequent time $\leq n$ α holds",
- $\mathcal{G}\alpha$ reads "for all subsequent times $\leq n$ α holds",
- $\alpha\mathcal{U}\beta$ reads "for some subsequent time $\leq n$ β holds, and α holds until then",
- $\alpha\mathcal{R}\beta$ reads "if for some subsequent time $\leq n$ β does not hold, then α holds in a state before then".

According to the above intuitive meanings, the following equivalences hold:

$$\mathcal{F}\alpha \equiv \top\mathcal{U}\alpha, \qquad \mathcal{G}\alpha \equiv \bot\mathcal{R}\alpha, \tag{2.3}$$

where $\top, \bot$ are the symbols for truth and falsity respectively.

Let h be a history (2.1), and α a QLTL formula. We say that that h *satisfies* α *(*$h \models^n \alpha$*)* iff $h \models_0^n \alpha$, where, for any $i \leq n$, the definition of $h \models_i^n \alpha$ is given inductively on the structure of the QLTL formulas:

1. $h \models_i^n \top$ and $h \not\models_i^n \bot$,
2. $h \models_i^n \alpha$ if $s^i \models \alpha$ and α is a fluent literal,
3. $h \models_i^n \neg\alpha$ if it is not the case that $h \models_i^n \alpha$,
4. $h \models_i^n (\alpha \wedge \beta)$ if $h \models_i^n \alpha$ and $h \models_i^n \beta$,
5. $h \models_i^n (\alpha \vee \beta)$ if $h \models_i^n \alpha$ or $h \models_i^n \beta$,
6. $h \models_i^n \mathcal{X}_s\alpha$ if $i < n$ and $h \models_{i+1}^n \alpha$,
7. $h \models_i^n \mathcal{X}_w\alpha$ if $i = n$ or $h \models_{i+1}^n \alpha$,
8. $h \models_i^n \mathcal{F}\alpha$ if $\exists j$: $i \leq j \leq n$, $h \models_j^n \alpha$,
9. $h \models_i^n \mathcal{G}\alpha$ if $\forall j$: $i \leq j \leq n$, $h \models_j^n \alpha$,
10. $h \models_i^n \alpha\mathcal{U}\beta$ if $\exists j$: $i \leq j \leq n$, $h \models_j^n \beta$ and $\forall k$: $i \leq k < j$, $h \models_k^n \alpha$,
11. $h \models_i^n \alpha\mathcal{R}\beta$ if $\forall j$: $i \leq j \leq n$, $h \models_j^n \beta$ or $\exists k$: $i \leq k < j$, $h \models_k^n \alpha$,
12. $h \models_i^n \forall x\alpha$ if $h \models_i^n \alpha(\top/x)$ and $h \models_i^n \alpha(\bot/x)$,
13. $h \models_i^n \exists x\alpha$ if $h \models_i^n \alpha(\top/x)$ or $h \models_i^n \alpha(\bot/x)$,

where $\alpha(\top/x)$ is the formula obtained from α by replacing all the occurences of x with $\top$. Similarly for $\alpha(\bot/x)$. Given the above semantics, it is easy to see that both (2.2) and (2.3) hold.

Temporal logic allows us to specify different interesting requirements on plans. The user can specify a safety property with the QLTL formula $\mathcal{G}\alpha$, intuitively standing for "the property α should be maintained", or $\mathcal{G}\neg\beta$, meaning for instance that "a dangerous or undesired situation β should be avoided". The QLTL formula $\mathcal{GF}\alpha$ can be used to specify that "always a state where α holds should be eventually reached", while $\mathcal{FG}\beta$ states that "the system should get to the point where property β can be maintained". As further examples, QLTL formulas can be composed in order to specify a sequential ordering in which properties should be verified: $\mathcal{F}(\alpha \wedge \mathcal{F}\beta)$ states that "α should be eventually true before β", while $\mathcal{F}(\alpha \wedge \mathcal{FG}\beta)$ can be used to specify that "α should be reached first, and then β should be reached and maintained". Finally, quantifiers allow for more compact (and thus arguably simpler) representations [CDLS96]. For example, $\mathcal{F}\alpha \wedge \mathcal{F}\neg\alpha$ can be equivalently but more compactly represented as $\forall x\mathcal{F}(x \equiv \alpha)$.

2.4 Safe Planning: Plan Validation

We now consider the problem of plan validation with respect to a finite set of properties. Let α be the conjunction of the properties, let $\pi = \langle I, D, G\rangle$ a planning problem, and let $\vec{a} = a^1; \ldots; a^n$ be a valid plan.

We say that $\vec{a}$ is *safe* with respect to α if each history (2.1) satisfies α. The problem of validating $\vec{a}$ with respect to α can be re-cast as a satisfiability problem. The basic idea is to check whether the execution of the actions in $\vec{a}$ "entails" the property α, reformulated as a QBF. In the following, for any number i and formula H, H_i is the expression obtained from H by substituting each atom B with B_i. Intuitively, the subscript i represents time:

- If F is a fluent symbol, the atom F_i expresses that F 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.

The reformulation $[\alpha]^n$ of α as a QBF is defined as $[\alpha]^n_0$, and $[\alpha]^n_i$ is:

1. $[\alpha]^n_i$ is α_i, if α is a fluent literal,
2. $[\alpha \wedge \beta]^n_i$ is $[\alpha]^n_i \wedge [\beta]^n_i$,
3. $[\alpha \vee \beta]^n_i$ is $[\alpha]^n_i \vee [\beta]^n_i$,
4. $[\mathcal{X}_s\alpha]^n_i$ is $\bot$ if $i = n$, and is $[\alpha]^n_{i+1}$ otherwise,
5. $[\mathcal{X}_w\alpha]^n_i$ is $\top$ if $i = n$, and is $[\alpha]^n_{i+1}$ otherwise,

6. $[\mathcal{F}\alpha]_i^n$ is $\vee_{j=i}^n [\alpha]_j^n$,
7. $[\mathcal{G}\alpha]_i^n$ is $\wedge_{j=i}^n [\alpha]_j^n$,
8. $[\alpha \mathcal{U} \beta]_i^n$ is $\vee_{j=i}^n ([\beta]_j^n \wedge_{k=i}^{j-1} [\alpha]_k^n)$,
9. $[\alpha \mathcal{R} \beta]_i^n$ is $\wedge_{j=i}^n ([\beta]_j^n \vee_{k=i}^{j-1} [\alpha]_k^n)$,
10. $[\forall x \alpha]_i^n$ is $\forall x [\alpha]_i^n$,
11. $[\exists x \alpha]_i^n$ is $\exists x [\alpha]_i^n$.

For any QLTL formula, the correspondence between its semantics and its translation as a QBF is immediate. Given the above mapping, and under the assumption that $\vec{a}$ is valid for π, the check that $\vec{a}$ is safe with respect to α amounts to check that

$$I_0 \wedge \wedge_{i=0}^{n-1} a_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{tr}_i^D \models [\alpha]^n. \tag{2.4}$$

For example, if α is a safety property $\mathcal{G}\beta$, (2.4) holds if for any history (2.1) in which s^0 is an initial state, s^i satisfies β for any $i : 0 \leq i \leq n$.

2.5 Safe Planning: Plan Generation

Consider a finite set of properties, Let α be their conjunction, and let $\pi = \langle I, D, G \rangle$ a planning problem. Safe planning is the task of finding a valid plan which is also safe with respect to α. As before, we do not make any assumption about D: it can be deterministic or nondeterministic, it can allow for the concurrent execution of actions or not. However, for simplicity we assume that I is satisfied by at most one state.

The simplest approach for the generation of safe plans, is to generate (possible) plans, and test whether each generated plan is valid and safe. In this way, we obtain

1. a correct but possibly incomplete safe procedure in general, and
2. a correct and complete safe procedure if we generate and test all the possible plans.

Given a planning problem π and a natural number n, we say that a procedure for safe planning is

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

In the planning as satisfiability framework, all the possible plans can be generated by enumerating the assignments satisfying

$$I_0 \wedge \wedge_{i=0}^{n-1} \mathit{tr}_i^D \wedge G_n. \tag{2.5}$$

Here the situation is the same, except that (2.5) may be a QBF. The check whether a plan $\vec{a} = a^1, \ldots, a^n$ is safe —assuming we already know is valid— can be done as described in Section 2.4. Indeed, if I is satisfied by only one state (as we assumed at the beginning of the Section) and D is deterministic, any possible plan is also valid, and thus we are done: we can easily devise a procedure that generates valid plans till one which is also safe is found.

However, as described in [Giu00], the check whether $\vec{a}$ is valid is complicated in the case that D is non deterministic. In fact, the check that a possible plan is also valid amounts to perform an entailment check similar to the one in (2.4). More in detail, a plan $a^1; \ldots; a^n$ is valid iff

$$I_0 \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} a_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D \models G_n \wedge \neg Z_n, \tag{2.6}$$

where trt_i^D is defined as in [Giu00] on the basis of tr_i^D, and Z is a newly introduced fluent symbol. Thus, given what we said so far, in order to have a correct and complete procedure for safe planning we should:

- generate (all) the possible plans by satisfying (2.5),
- test if each generated plan $a^1; \ldots; a^n$ is valid by checking whether (2.6) holds, and
- if $a^1; \ldots; a^n$ is valid, check whether it is also safe with respect to a property α, by checking whether (2.4) holds.

This is not necessary. Indeed, the last two checks can be combined into one: A (possible) plan $a^1; \ldots; a^n$ is both valid for π and safe with respect to α iff

$$I_0 \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} a_i^{i+1} \wedge \wedge_{i=0}^{n-1} \mathit{trt}_i^D \models G_n \wedge \neg Z_n \wedge [\alpha]^n.$$

Furthermore, as an optimization, if in the generation phase we consider only the plans corresponding to the assignments satisfying

$$I_0 \wedge \wedge_{i=0}^{n-1} \mathit{tr}_i^D \wedge G_n \wedge [\alpha]^n \tag{2.7}$$

we still get a correct and complete procedure for safe planning: The assignment satisfying (2.7) and not (2.5) for sure do not correspond to safe plans.

2.6 Conclusions and Related Work

We have extended the planning as satisfiability approach to the case of "safe planning", i.e. to the generation and validation of plans that have to guarantee both the reachability of a goal state and the fact that user defined temporal properties hold. We allow domains to be non-deterministic, as several applications require. In non-deterministic domains, a plan may result in several different possible executions. Safe planning provides the ability of generating plans and validating them against all the possibly different executions.

In the literature, there are various works in the "planning as model checking" literature, first introduced in [CGGT97] (see also [GT99] for an introduction), and then extended to generate valid plans for reachability goals in non-deterministic domains, see, e.g., [CRT98, BCRT01]. The approach is based on the idea that plans can be generated and validated by searching through sets of states compactly represented by BDDs. Differently from the work that we present here, in these works (see, e.g., [PT01, PBT01]) safe plans can be specified with formulas in the CTL temporal logic [Eme90b]. CTL allows the user to distinguish between requirements that should hold on all the possible non-deterministic plan executions, and others that may hold only on some executions. The plans that are generated and validated can include conditionals and iterations. The work has been implemented in the MBP planner [BCP+01].

Chapter 3

SAT-Based Planning with Preferences

3.1 Introduction

Planning as Satisfiability [KS92] is one of the most well-known and effective technique for classical planning: SATPLAN [KS99] is a planner based on propositional satisfiability (SAT) and has been the winning system in the deterministic track for optimal planners in the 4th International Planning Competition (IPC-4) [HE05] and a co-winner in the recent IPC (IPC-5) which has just terminated. Further, we have seen in the previous chapters, how the technique can be extended to deal with nondeterministic domains and/or extended goals. Given a planning problem π the basic idea of planning as satisfiability is to convert the problem of determining the existence of a plan for π with a fixed makespan n into a SAT formula φ such that there is a one-to-one correspondence between the plans of π with makespan n and the interpretations satisfying φ. Of course, for SATPLAN effectiveness, it is crucial the availability of very effective SAT solvers, like MINISAT [ES03]. The same holds more in general for any planning system based on SAT and/or QBF solvers, like QUBE. MINISAT is based on the Davis-Logemann-Loveland procedure (DLL) like most of the state-of-the-art SAT checkers, and won the last SAT competition in 2005. QUBE [GNT04b] is the QF solver developed by the unit of Genova, won the probabilistic class at the last QBF evaluation, and is one of the most effective solvers for dealing with problems coming from planning and/or formal verification.

We now show how it is possible to easily extend SATPLAN and, more in general, any planner based on SAT and/or QBF, in order to handle both qualitative and quantitative preferences. In the resulting system, qualitative preferences are handled via a simple modification in the heuristic of the DLL solver (MINISAT in our case), while quantitative preferences are reduced to qualitative ones. We first consider the problem of planning with "soft goals", i.e., a goal whose achievement is desired but is not necessary. We modified SATPLAN in order to handle preferences, obtaining a system that we call SATPLAN(P). Then we show that with our approach SATPLAN(P) is as effective as SATPLAN in solving the same problems but without soft goals. We show that such good performances of

SATPLAN(P) are due to the relative low number of (soft) goals usually present in planning problems. To validate this claim we consider the issue of determining "minimal" plans: in this case, any action variable corresponds to a preference and it is thus common to have problems with several thousands of preferences. We show that the performances of SATPLAN(P) are not affected even in this settings for many problems. However, SATPLAN(P) can be much slower than SATPLAN when considering problems with a very high ratio of number of preferences (i.e., action variables) to the total number of variables. Of course, in many cases, SATPLAN(P) returns plans with better quality than the standard SATPLAN both in the case of planning with soft goals and in the case of determining minimal plans. Our analysis is conducted considering both qualitative and quantitative preferences, different reductions from quantitative to qualitative ones, and most of the planning domains considered in the first 4 IPCs and that SATPLAN can handle.

The chapter is structured as follows. We first introduce some basic notation and terminology (Section 3.2), then we introduce preferences (Section 3.3) and show how it is possible to incorporate them in planning as satisfiability (Section 3.4). The implementation and experimental analysis is given in Section 3.5. The last section contains the related work and some final remarks.

3.2 Basic preliminaries

Let $\mathcal{F}$ and $\mathcal{A}$ be the set of *fluents* and *actions* respectively. A *state* is an interpretation of the fluent signature. A *complex action* is an interpretation of the action signature. Intuitively, a complex action α models the concurrent execution of the actions satisfied by α.

A *planning problem* is a triple $\langle I, tr, G\rangle$ where

- I is a Boolean formula over $\mathcal{F}$ and represents the set of *initial states*;
- tr is a Boolean formula over $\mathcal{F} \cup \mathcal{A} \cup \mathcal{F}'$ where $\mathcal{F}' = \{f' : f \in \mathcal{F}\}$ is a copy of the fluent signature and represents the *transition relation* of the automaton describing how (complex) actions affect states (we assume $\mathcal{F} \cap \mathcal{F}' = \emptyset$);
- G is a Boolean formula over $\mathcal{F}$ and represents the set of *goal states*.

The above definition of planning problem differs from the traditional ones in which the description of actions' effects on a state is described in an high-level action language like STRIPS or PDDL. We preferred this formulation because the techniques we are going to describe are largely independent of the action language used, at least from a theoretical point of view. The only assumption that we make is that the description is deterministic: there is only one state satisfying I and the execution of a (complex) action α in a state s can lead to at most one state s'. More formally, for each state s and complex action α there is at most one interpretation extending $s \cup \alpha$ and satisfying tr.

Consider a planning problem $\pi = \langle I, tr, G \rangle$.

In the following, for any integer i

- if F is a formula in the fluent signature, F_i is obtained from F by substituting each $f \in \mathcal{F}$ with f_i,
- tr_i is the formula obtained from tr by substituting each symbol $p \in \mathcal{F} \cup \mathcal{A}$ with p_{i-1} and each $f \in \mathcal{F}'$ with f_i.

If n is an integer, the *planning problem* π *with makespan* n is the Boolean formula π_n defined as

$$I_0 \wedge \wedge_{i=1}^{n} tr_i \wedge G_n \qquad (n \geq 0) \tag{3.1}$$

and a *plan for* π_n is an interpretation satisfying (3.1).

For example, considering the planning problem of going to work from home. Assuming that we can use the car or the bus or the bike, this scenario can be easily formalized using a single fluent variable *AtWork* and three action variables *Car*, *Bus* and *Bike* with the obvious meaning. The problem with makespan 1 can be expressed by the conjunction of the formulas:

$$\begin{array}{c} \neg \mathit{AtWork}_0, \\ \mathit{AtWork}_1 \equiv \neg \mathit{AtWork}_0 \equiv (\mathit{Car}_0 \vee \mathit{Bus}_0 \vee \mathit{Bike}_0), \\ \mathit{AtWork}_1, \end{array} \tag{3.2}$$

in which the first formula corresponds to the initial state, the second to the transition relation, and the third to the goal state. (3.2) has 7 plans (i.e., satisfying interpretations), each corresponding to a non-empty subset of $\{\mathit{Car}_0, \mathit{Bus}_0, \mathit{Bike}_0\}$. For instance, in the plan corresponding to $\{\mathit{Car}_0, \mathit{Bus}_0\}$ both the car and the bike are used to get to work. If we want to avoid any two actions in $\{\mathit{Car}_0, \mathit{Bus}_0, \mathit{Bike}_0\}$ to occur in parallel, the following mutex axioms are to be added as part of the formulas encoding the transition relation

$$\neg(\mathit{Car}_0 \wedge \mathit{Bus}_0), \neg(\mathit{Car}_0 \wedge \mathit{Bike}_0), \neg(\mathit{Bus}_0 \wedge \mathit{Bike}_0).$$

3.3 Preferences and Optimal Plans

Let π_n be a planning problem π with makespan n.

In addition to the goals, it may be desiderable to have plans satisfying other conditions. For example, considering the problem (3.2), in addition to being at work at time 1, we may want to avoid taking the bus (at time 0). Formally this preference is expressed by the formula $\neg \mathit{Bus}_0$, and it amounts to prefer the plans satisfying $\neg \mathit{Bus}_0$ to those satisfying Bus_0. In general, there can be more than one preference, and it may not be possible to satisfy all of them. For example, in (3.2) it is not possible to satisfy the three possible preferences $\neg \mathit{Bike}_0$, $\neg \mathit{Bus}_0$ and $\neg \mathit{Car}_0$.

A "qualitative" solution to the problem of conflicting preferences is to define a partial order on them. A *qualitative preference (for π_n)* is a pair $\langle P, \prec \rangle$ where P is a set of formulas (the preferences) whose atoms are in π_n, and $\prec$ is a partial order on P. The partial order can be empty, meaning that all the preferences are equally important. The partial order can be extended to plans for π_n. Consider a qualitative preference $\langle P, \prec \rangle$. Let π_1 and π_2 be two plans for π_n. π_1 *is preferred to* π_2 *(wrt* $\langle P, \prec \rangle$*)*, iff

1. they satisfy different sets of preferences, i.e., $\{p : p \in P, \pi_1 \models p\} \neq \{p : p \in P, \pi_2 \models p\}$, and
2. for each preference p_2 satisfied by π_2 and not by π_1 there is another preference p_1 satisfied by π_1 and not by π_2 with $p_1 \prec p_2$.

The second condition says that if π_1 does not satisfy a preference p_2 which is satisfied by π_2, then π_1 is preferred to π_2 only if there is a good reason for $\pi_1 \not\models p_2$, and this good reason is that $\pi_1 \models p_1$, $p_1 \prec p_2$ and $\pi_2 \not\models p_1$. We write $\pi_1 \prec \pi_2$ to mean that π_1 is preferred to π_2. It is easy to see that $\prec$ defines a partial order on plans for π_n wrt $\langle P, \prec \rangle$. A plan π is *optimal for* π_n *(wrt* $\langle P, \prec \rangle$*)* if it is a minimal element of the partial order on plans for π_n, i.e., if there is no plan π' for π_n with $\pi' \prec \pi$ (wrt $\langle P, \prec \rangle$).

A "quantitative" approach to solve the problem of conflicting preferences is to assign weights to each of them, and then minimize/maximize a given objective function involving the preferences and their weights. In most cases the objective function is the weighted sum of the preferences: this has been the case of each planning problem with preferences in the last IPC. With this assumption, a *quantitative preference (for π_n)* can be defined as a pair $\langle P, c \rangle$ where P is a set of formulas in π_n signature (as before) and c is a function associating an integer to each preference in P. Without loss of generality, we can further assume that $c(p) \geq 0$ for each $p \in P$ and that we are dealing with a maximization problem. Thus, a plan is *optimal (wrt* $\langle P, c \rangle$*)* if it maximizes[1]

$$\sum_{p \in P : \pi \models p} c(p). \tag{3.3}$$

For instance, considering the planning problem (3.2), if we have the qualitative (resp. quantitative) preference

- $\langle \{\neg \mathit{Bike}_0, \neg \mathit{Bus}_0, \neg \mathit{Car}_0\}, \emptyset \rangle$, (resp. $\langle \{\neg \mathit{Bike}_0, \neg \mathit{Bus}_0, \neg \mathit{Car}_0\}, c \rangle$, where c is the constant function 1) then there are three optimal plans, corresponding to $\{\mathit{Bike}_0\}, \{\mathit{Bus}_0\}, \{\mathit{Car}_0\}$.
- $\langle \{\neg \mathit{Bike}_0, \neg \mathit{Bus}_0, \neg \mathit{Car}_0\}, \{\neg \mathit{Bike}_0 \prec \neg \mathit{Car}_0\} \rangle$, (resp. $\langle \{\neg \mathit{Bike}_0, \neg \mathit{Bus}_0, \neg \mathit{Car}_0\}, c \rangle$, where $c(\neg \mathit{Bike}_0) = 2$ while $c(\neg \mathit{Bus}_0) = c(\neg \mathit{Car}_0) = 1$) then there are two optimal plans, corresponding to $\{\mathit{Bus}_0\}, \{\mathit{Car}_0\}$.

[1] Assuming that $c(p) < 0$ for some $p \in P$, we can replace p with $\neg p$ in P and define $c(\neg p) = -c(p)$: the set of optimal plans does not change. Given $\langle P, c \rangle$ and assuming we are interested in minimizing the objective function (3.3), we can consider the quantitative preference $\langle P', c' \rangle$ where $P' = \{\neg p : p \in P\}$ with $c'(\neg p) = c(p)$, and then look for a plan maximizing $\sum_{p \in P' : \pi \models p} c'(p)$.

- $\langle\{\mathit{Bike}_0 \vee \mathit{Bus}_0\}, \emptyset\rangle$, (resp. $\langle\{\mathit{Bike}_0 \vee \mathit{Bus}_0\}, c\rangle$, where c is the constant function 1) then all the plans except for the one corresponding to $\{\mathit{Car}_0\}$ are optimal.

3.4 Planning as Satisfiability with preferences

Consider a planning problem with makespan n π_n, and a qualitative preference $\langle P, \prec\rangle$. In planning as satisfiability, plans for π_n are generated by invoking a SAT solver on π_n. Optimal plans for π_n can be obtained by

1. Encoding the preference P as a formula to be conjoined with π_n; and
2. Modifying DLL in order to search first for optimal plans, i.e., to branch according to the partial order $\prec$.

The resulting procedure is reported in Figure 3.1 in which:

- for each $p \in P$, $v(p)$ is a newly introduced variable;
- $v(P)$ is the set of new variables, i.e., $\{v(p) : p \in P\}$;
- $v(\prec) = \prec'$ is the partial order on $v(P)$ defined by $v(p) \prec' v(p')$ iff $p \prec p'$;
- *cnf*(φ), where φ is a formula, is a set of clauses (i.e., set of sets of literals) such that for any interpretation μ in the signature of *cnf*(φ), $\mu \models$ *cnf*(φ) iff $\mu \models \varphi$: there are well known methods for computing *cnf*(φ) in linear time by introducing additional variables.
- S is an *assignment*, i.e., a consistent set of literals. An assignment S corresponds to the partial interpretation mapping to true the literals $l \in S$.
- l is a literal and $\overline{l}$ is the complement of l;
- φ_l returns the set of clauses obtained from φ by (i) deleting the clauses $C \in \varphi$ with $l \in C$, and (ii) deleting $\overline{l}$ from the other clauses in φ;
- *ChooseLiteral*$(\varphi, S, P', \prec')$ returns an *unassigned* literal l (i.e., such that $\{l, \overline{l}\} \cap S = \emptyset$) in φ such that either all the variables is P' are assigned, or $l \in P'$ and all the other variables $v(p) \in P'$ with $v(p) \prec l$ are assigned.

As it can be see from the figure, OPT-DLL is the standard DLL except for the modification in the heuristic, i.e., *ChooseLiteral* which initially selects literals according to the partial order $\prec'$. If we have two preferences $p_1 = (\neg\mathit{Bike}_0 \wedge \neg\mathit{Bus}_0 \wedge \neg\mathit{Car}_0)$ and $p_2 = (\neg\mathit{Bike}_0 \wedge \neg\mathit{Bus}_0)$ with $p_1 \prec p_2$ and we consider the problem (3.2), OPT-DLL returns the plan corresponding to $\{\mathit{Car}_0\}$ determined while exploring the branch extending $\{\neg v(p_1), v(p_2)\}$. This plan is optimal, as sanctioned by the following theorem.

function QL-PLAN(π_n,P,$\prec$)
1 **return** OPT-DLL(*cnf*($\pi_n \wedge \wedge_{p \in P}(v(p) \equiv p)$),$\emptyset$,$v(P)$,$v(\prec)$)

function OPT-DLL(φ,S,P',$\prec'$)
2 **if** ($\emptyset \in \varphi$) **return** *False*;
3 **if** ($\varphi = \emptyset$) **return** S;
4 **if** ($\{l\} \in \varphi$) **return** OPT-DLL$(\varphi_l, S \cup \{l\}, P', \prec')$;
5 l := *ChooseLiteral*$(\varphi, S, P', \prec')$;
6 V := OPT-DLL$(\varphi_l, S \cup \{l\}, P', \prec')$;
7 **if** ($V \neq$ *False*) **return** V;
8 **return** OPT-DLL$(\varphi_{\overline{l}}, S \cup \{\overline{l}\}, P', \prec')$.

Figure 3.1: The algorithm of QL-PLAN

Theorem 6 *Let π_n be a planning problem with makespan n and let $\langle P, \prec \rangle$ be a qualitative preference for π_n.* QL-PLAN*(π_n,P,$\prec$) returns*

1. *False if π_n has no plans, and*
2. *an optimal plan for π_n wrt $\langle P, \prec \rangle$, otherwise.*

Consider now a quantitative preference $\langle P, c \rangle$. The problem of finding an optimal plan for π_n wrt $\langle P, c \rangle$ can be solved again using OPT-DLL in Figure 3.1 as core engine. The basic idea is to encode the value of the objective function (3.3) as a sequence of bits $b_{n-1}, \ldots, b_0$ and then consider the qualitative preference $\langle \{b_{n-1}, \ldots, b_0\}, \{b_{n-1} \prec b_{n-2}, \ldots, b_1 \prec b_0\} \rangle$. In more details, let *Bool*(P, c) a Boolean formula such that:

1. if $n = \lceil log_2((\sum_{p \in P} c(p)) + 1) \rceil$, *Bool*$(P, c)$ contains n new variables $b_{n-1}, \ldots, b_0$; and
2. for any plan π satisfying π_n, there exists a unique interpretation μ to the variables in $\pi_n \wedge$ *Bool*(P, c) such that
   (a) μ extends π and satisfies $\pi_n \wedge$ *Bool*(P, c);
   (b) $\sum_{p \in P : \pi \models p} c(p) = \sum_{i=0}^{n-1} \mu(b_i) \times 2^i$, where $\mu(b_i)$ is 1 if μ assigns b_i to true, and is 0 otherwise.

If the above conditions are satisfied, we say that *Bool*(P, c) is a *Boolean encoding of* $\langle P, c \rangle$ *with output* $b_{n-1}, \ldots, b_0$. *Bool*(P, c) can be realized in polynomial time in many ways, see, e.g., [War98].

The resulting procedure is represented in Figure 3.2 in which *Bool*(P, c) is a Boolean encoding of $\langle P, c \rangle$ with output $b_{n-1}, \ldots, b_0$, $b(c) = \{b_{n-1}, \ldots, b_0\}$ and $p(c)$ is the partial order $b_{n-1} \prec b_{n-2}, \ldots, b_1 \prec b_0$.

function QT-PLAN(π_n,P,c)
1 **return** OPT-DLL(*cnf*($\pi_n \wedge$ *Bool*(P, c)),$\emptyset$,$b(c)$,$p(c)$)

Figure 3.2: The algorithm of QT-PLAN

If we have two preferences $p_1 = (\neg$*Bike*$_0 \wedge \neg$*Bus*$_0 \wedge \neg$*Car*$_0)$ and $p_2 = (\neg$*Bike*$_0 \wedge \neg$*Bus*$_0)$ with $c(p_1) = 2$ and $c(p_2) = 1$, then two bits b_1 and b_0 are sufficient as output of *Bool*$(\{p_1, p_2\}, c)$. Further, if we consider the problem (3.2), OPT-DLL returns the plan corresponding to $\{$*Car*$_0\}$ determined while exploring the branch extending $\neg b_1, b_0$. This plan is optimal, as sanctioned by the following theorem.

Theorem 7 *Let π_n be a planning problem with makespan n and let $\langle P, c\rangle$ be a quantitative preference for π_n.* QT-PLAN*(π_n,P,c) returns*

1. *False if π_n has no plans, and*
2. *an optimal plan for π_n wrt $\langle P, c\rangle$, otherwise.*

Theorems 6 and 7 are based on results on DLL first stated in [GM06] and in [BBD$^+$04] in the context of constraint programming.

3.5 Implementation and Experimental Analysis

As we already said in the introduction, we use SATPLAN as underlying planning system. We implemented OPT-DLL in MINISAT which is also one of the solvers SATPLAN can use, and that we set as default for SATPLAN.[2]

Considering the benchmarks, our first observation is that a special track with preferences has been set up in the context of the recent IPC-5, and a special focus has been put on "soft goals" which express desired goals that don't have to be necessarily achieved. This required the extension of the PDDL language in order to incorporate preferences as part of the problems' specification [GL06]. Unfortunately, the benchmarks used in the last IPC became available only on June 19th, and SATPLAN does not have the parser for preferences. For these reasons, we considered benchmarks coming from the first 4 IPCs and modify them in order to include soft goals. In particular, given that if $\pi = \langle I, tr, G\rangle$ is a planning problem that SATPLAN can handle, then G is a conjunction of atoms, we modified SATPLAN internals in order to output a SAT formula in which the clauses corresponding to the goal have been substituted with a new set of clauses encoding that the new goal is the disjunction of the atoms in G. Further, if π_n is the planning problem with

[2] SATPLAN's default solver is called SIEGE: we run SATPLAN with SIEGE and MINISAT and we have seen no significant difference in SATPLAN's performances.

makespan n, and S is the set of atoms in G_n, we considered both the qualitative preference $\langle S, \emptyset \rangle$ and the quantitative preference $\langle S, c \rangle$ in which c is the constant function 1. These preferences encode the fact that the formulas in S are now "soft goals": the planner tries to satisfy as many atoms in S as possible and

1. in the case of the qualitative preference, if π is an optimal plan then there is no plan for π_n satisfying a strict superset of $\{p : p \in S, \pi \models p\}$,

2. in the case of the quantitative preference, if π is an optimal plan then there is no plan for π_n satisfying a number of preferences bigger than $|\{p : p \in S, \pi \models p\}|$.

Further, in the quantitative case, we considered Warners' (War98), and also Bailleux and Boufkhad's (BB03) encodings of *Bool*(P, c), denoted with W-encoding and BB-encoding respectively. The size of W-encoding is linear in $|P|$ while BB-encoding is quadratic. However, the latter has better computational properties and it has been consistently reported in the literature to lead to good results [BB03, BR05].

We considered the pipesworld, satellite, airport, promela philosophers and optical, psr, depots, driverLog, zenoTravel, freeCell, logistic, blocks, mprime and mistery domains from the first 4 IPCs. We discarded the problems with only one goal because they would not have "soft goals". Finally we considered SATPLAN, SATPLAN modified in order to handle the qualitative, quantitative with W-encoding and quantitative with BB-encoding preferences and denoted with SATPLAN(s), SATPLAN(w) and SATPLAN(b) respectively. The timeout has been set to 300s, and all the tests have been run on a Linux box equipped with a Pentium IV 2.4GHz processor and 512MB of RAM. The results are shown in Figures 3.3 and 3.4. The left plots shows the performances of the four systems on the problems considered, ordered according to SATPLAN's performances. The right plot shows the number of soft goals each planner does not satisfy, again ordered according to SATPLAN's results.

The first observation from the left plot is that the performances of the three planners with preferences are pretty much the same and can be hardly distinguished from SATPLAN's ones. This is indeed a very good news that could have been a-priori expected, but only to a certain extent: from a planning perspective SATPLAN(w)/(b)/(s) are solving an harder task than SATPLAN, while from a SAT perspective it is well known that imposing some restrictions on the literals to be used for branching can lead to an exponential degradation in the performances. Indeed, DLL is a form of tree resolution and imposing a static variable ordering on the heuristic makes DLL computation an ordered tree resolution: the potential exponential degradation of DLL performances is then a consequence of the fact that ordered and tree resolution cannot p-simulate each other, see, e.g., [BOP03] for more details.

Considering the right plot, as expected SATPLAN does not satisfy many of the soft goals, while SATPLAN(w)/(b)/(s) manage to satisfy all of them in many cases. Interestingly, the number of soft goals not satisfied by SATPLAN(w)/(b)/(s) are in most case

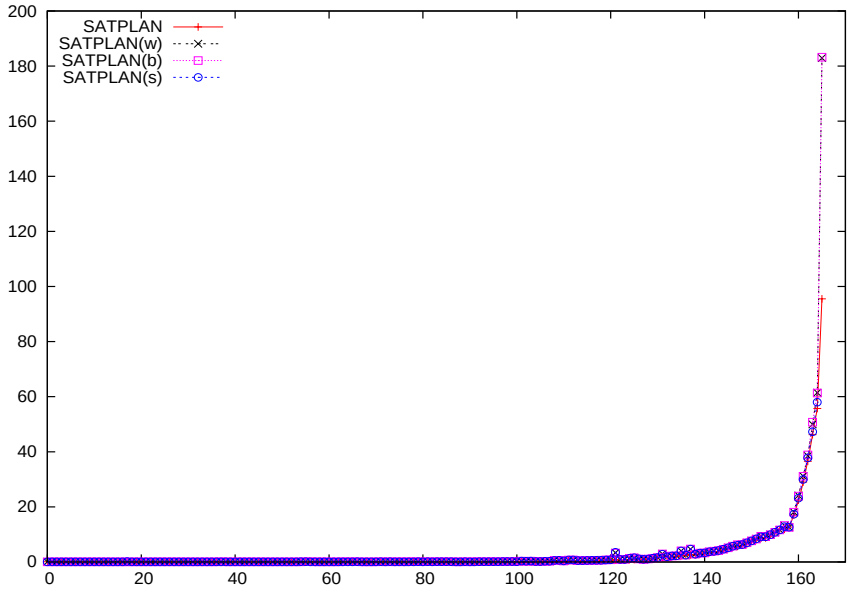

Figure 3.3: Performances of SATPLAN/(w)/(b)/(s): CPU time

equal. This is obvious for SATPLAN(w)/(b) while this is not necessary the case for SATPLAN(s). The right plot provides also an explanation of the good CPU performances of SATPLAN(w)/(b)/(s): in all the problems considered there are at most 30 soft goals. This means that OPT-DLL branching heuristic is forced for at most the initial 30 branches and, while it is known in SAT that the first branches are very important, they are just a few. Further, for the quantitative case, the burden introduced by the Boolean encoding of the objective function is negligible, being an adder of at most 30 bits.

In order to evaluate the effectiveness of SATPLAN(w)/(b)/(s) compared to SATPLAN when the number of preferences is high and the heuristic is highly constrained, we considered the problem of finding a "minimal" plan. More precisely, if π_n is the given planning problem with makespan n, and S is the set of action variables in π_n, we consider

1. the qualitative preference $\langle \{\neg p : p \in S\}, \prec \rangle$ where $\neg p \prec \neg p'$ if p precedes p' according to the lexicographic ordering, and
2. the quantitative preference $\langle \{\neg p : p \in S\}, c \rangle$ in which c is the constant function 1.

These preferences encode the fact that we prefer plans with as few actions as possible. The qualitative preference has also been set in order to further constraint the heuristic up to the point to make it static on the action variables. In this setting, we are expecting a significant degradation in the CPU performances of SATPLAN(w)/(b)/(s) wrt SATPLAN's ones in many cases. The results are in Figures 3.5 and 3.6. In the left plot,

- on the x-axis there are the problems, sorted according to the ratio between the

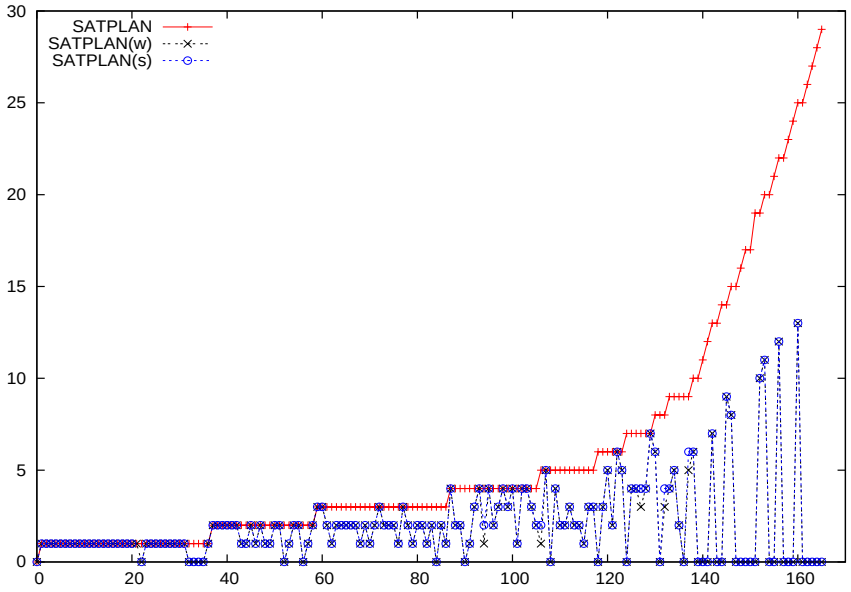

Figure 3.4: Performances of SATPLAN/(w)/(b)/(s): number of unsatisfied soft goals.

number of preferences (i.e., action variables) and the total number of variables in the instance for which a plan is found, and

- on the y-axis there is the ratio between the performances of SATPLAN(w)/(b)/(s) and SATPLAN's ones, in logarithmic scale.

Differently from what we expected, we see that SATPLAN(w)/(s) is as efficient as SATPLAN for a significant initial portion of the plot, though on a few instances SATPLAN(w) does not terminate within the timeout. SATPLAN(s) is also in many cases more efficient than SATPLAN, and this reminds the observation in [GMS98] that preferential splitting on action variables can lead to significant speed-ups. SATPLAN(b) bad performances are easily explained by the fact that the quadratic BB-encoding leads to very big SAT instances: considering for instance the relatively small "optical1" problem, the first satisfiable SAT instance has 2228 and 34462 variables and clauses respectively, while the same numbers when considering the W-encodings and the BB-encodings are 9500 and 56091, 13768 and 1157016 respectively; "optical1" is solved in 1.28s/2.17s/57.14s/1.6s by SATPLAN/(w)/(b)/(s). Considering the problems with ratio < 0.5 on the x-axis, they include all the airport, promela philosophers and optical and some of the psr problems that we considered. It is thus possible to conclude that even when the number of preferences is very high, SATPLAN(w)/(s) can be very effective (or, at least, as effective as SATPLAN) on some domains. Considering the "quality" of the plan returned, measured as its sequential length, the results are in the right plot. The *sequential length* of a plan π is the number of action variables assigned to true. In the plot, for sake of readability, we do not show the results for the airport, promela philosophers and optical, and psr domains (i.e., most

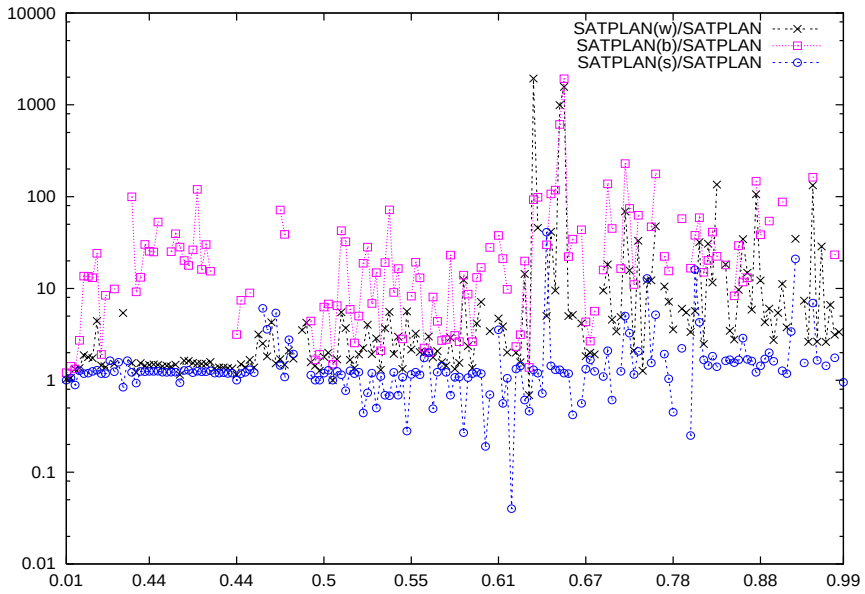

Figure 3.5: Performances of SATPLAN/(w)/(b)/(s): degradation in the CPU time.

of the problems having a ratio < 0.5 in the left plot): for each problem in these domains SATPLAN/(w)/(b)/(s) returned plans with the same sequential length. As it has been the case for the soft-goal, the quality of the plan returned by SATPLAN(s) is in most cases equal to that of SATPLAN(w), and in many cases both return plans of better quality than SATPLAN.

3.6 Conclusions and Related Work

It is not possible to give an an adequate account of all the related work in a few pages. We thus limit ourselves to the most recent papers on preferences in SAT and in planning, though we are aware that there are topics, like over-subscription planning, that are clearly related to this work and that we do not cover at all. As a partial solution to this problem we suggest to read the related work of the papers we cite here for a more complete picture. As usual, π_n is a planning problem with makespan n.

Considering the SAT literature, the problem of finding an optimal solution for π_n in the presence of a quantitative constraint can be formulated as a Pseudo-Boolean optimization problem. In the 2005 competition among Pseudo-Boolean solvers, MINISAT+ has been the most effective and is based on a reduction to SAT [MR06]. In [GM06] the authors consider the problem of optimally solving SAT problems with preferences on literals, show how this can be done by modifying DLL along the same lines used here, and show that the resulting system is competitive with MINISAT+ on MAXSAT and MINONE problems. This work differs from [GM06] both theoretically and experimentally. From a

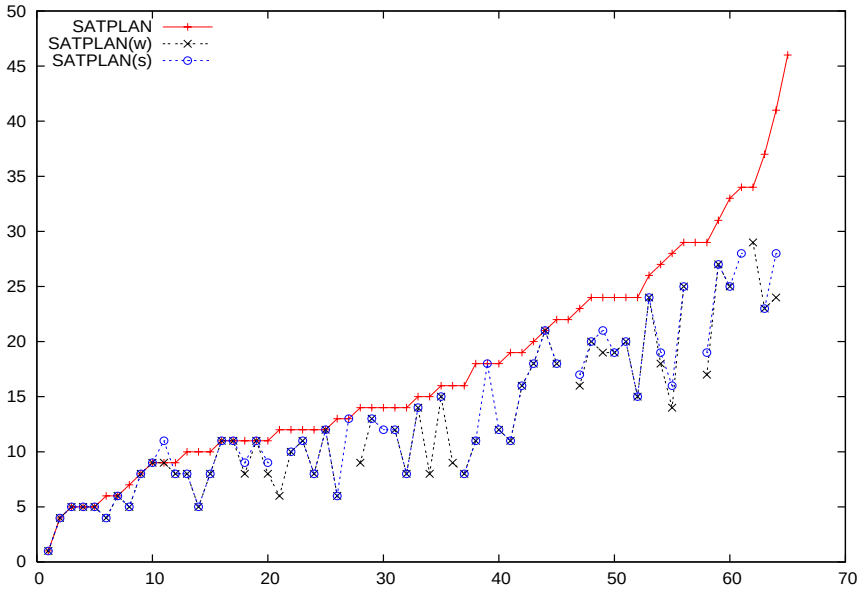

Figure 3.6: Performances of SATPLAN/(w)/(b)/(s): sequential length of the plan.

theoretical point of view, we provide a simpler and more general treatment of preferences: for us, a preference is a formula and not just a literal. Formulas allow, e.g., to model conditional preferences like "If my jacket and pants have different color then I prefer to wear a white shirt" [BBD$^+$04]. From an experimental point of view, here we focus on planning problems generated by SATPLAN, and analyze the impact on the performances of the planner depending on the number of preferences. We show that when there are a few preferences, the performances of SATPLAN are not affected, which in turn implies that the same holds for the performances of the underlying SAT solver. The number of preferences in all the problems considered in [GM06] was very high, equal to the number of clauses and variables for the MAXSAT and MINONE problems respectively.

Considering the literature on preferences in planning, three recent papers on planning with preferences are [BFM06, BC05, BR05], but see also the proceedings of the ICAPS'06 workshop on preferences and soft constraints. In the first paper, the authors define a simple language for expressing temporally extended preferences and implement a forward search planner, called PPLAN integrating them. For each $n \geq 0$ PPLAN is guaranteed to be n-optimal, where this intuitively means that there is no better plan of sequential length $\leq n$. The basic language for expressing preferences (called "basic desire formulas (BDFs)") is based on Linear Temporal Logic (LTL). BDFs are then ranked according to a total order to form "atomic preference formulas" which can then combined to form "general and aggregated preference formulas". It is well known how to compile LTL with a bounded makespan into propositional logic and thus in the language of π_n. It seems thus plausible that BDFs can be expressed as preferences in our setting, and we think the same holds for the preference formulas, but this is just a conjecture. In [BC05], the au-

thors show how to extend GP-CSP [DK01] in order to plan with preferences expressed as a TCP-net [BBD$^+$04]. In the Boolean case, TCP-net can be expressed as Boolean formulas. Though these two works are not based on satisfiability, the problem they consider is the same we deal with: find an optimal plan wrt the given preferences among the plans with makespan n. Thus, both [BFM06] and [BC05] reason with a bounded horizon as we do. However, all these approaches (including ours) can be easily adapted in order to work without a bounded horizon, by simply using an iterative deepening approach, i.e., by successively incrementing n, each time using the previously found solutions to bound the search space, up to a point in which we are guaranteed to have an optimal solution independent from the bound n. This is the approach followed in [BR05], where the problem considered is to extend the planning as satisfiability approach in order to find plans with optimal sequential length. Interestingly, the authors use a Boolean formula to encode the function representing the sequential length of the plan. In their approach, for a given n, the search for an optimal solution is done by iteratively calling the SAT solver, each time posting a constraint imposing a smaller value for the objective function: when the SAT formula becomes unsatisfiable, n is set to $n + 1$ and the process is iterated looking for a better plan than the one so far discovered. For a fixed n, the problem considered in [BR05] is exactly the same we deal with in Section 3.5: finding an optimal "minimal" plan for π_n using a quantitative approach. The fundamental difference between our approach and theirs is that we look for an optimal solution by imposing an ordering on the heuristic of the DLL solver, while they iteratively call the SAT solver as a black box. The disadvantage of their approach is that, e.g., the nogoods computed during a call are not re-used by the following calls for the same n. More in general, our approach can also deal with qualitative preferences.

Bibliography

[AGKS00] Dimitris Achlioptas, Carla Gomes, Henry Kautz, and Bart Selman. Generating satisfiable problem instances. In *Proc. AAAI*, 2000.

[BB03] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *Proc. CP*, pages 108–122, 2003.

[BBD+04] Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. Preference-based constrained optimization with CP-nets. *Computational Intelligence*, 20(2):137–157, 2004.

[BC05] Ronen I. Brafman and Yuri Chernyavsky. Planning with goal preferences and constraints. In *ICAPS*, pages 182–191, 2005.

[BCCZ99] 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.

[BCP+01] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: A Model Based Planner. Technical Report 01-1101, IRST, 2001.

[BCR01] 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 (IJCAI'2001)*, 2001.

[BCRT01] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In *Proc. 7^{th} International Joint Conference on Artificial Intelligence (IJCAI-01)*. AAAI Press, August 2001.

[BF95] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proc. of IJCAI-95*, pages 1636–1642, 1995.

[BFM06] M. Bienvenu, C. Fritz, and S. McIlraith. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR06)*, Lake District, UK, June 2006.

[BG00] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. AIPS*, 2000.

[BOP03] Josh Buresh-Oppenheim and Toniann Pitassi. The complexity of resolution refinements. In *Proc. LICS*, pages 138–147, 2003.

[BR05] Markus Büttner and Jussi Rintanen. Satisfiability planning with constraints on the number of actions. In *ICAPS*, pages 292–299, 2005.

[Bry92] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[BS97] 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.

[BST04] Daniel Le Berre, Laurent Simon, and Armando Tacchella. Challenges in the QBF arena: the SAT'03 evaluation of QBF solvers. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 468–485. Springer, 2004.

[CDLS96] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. Comparing space efficiency of propositional knowledge representation formalisms. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 364–373, San Francisco, November 5–8 1996. Morgan Kaufmann.

[CFG+01] 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.

[CGGT97] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: a decision procedure for AR. In *Lecture Notes in Computer Science*, volume 1348, 1997.

[CR99] 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.

[CR00] Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[CRT98] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proceeding of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin, 1998. AAAI-Press. Also IRST-Technical Report 9801-10, Trento, Italy.

[Dec90] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, January 1990.

[DK01] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artif. Intell.*, 132(2):151–182, 2001.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem proving. *Communication of ACM*, 5(7):394–397, 1962.

[Eme90a] Allen Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *The Handbook of Theoretical Computer Science*, pages 997–1072. Elsevier Science Publishers, 1990.

[Eme90b] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter 16, pages 995–1072. Elsevier, 1990.

[EMW97] Michael Ernst, Todd Millstein, and Daniel Weld. Automatic SAT-compilation of planning problems. In *Proc. IJCAI-97*, 1997.

[ES03] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518, 2003.

[FG00] Paolo Ferraris and Enrico Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. AAAI-2000*, 2000.

[Fin82] Jeffrey Finger. *Synthesis of Communicating Processes from Temporal Logic Specifications*. PhD thesis, Stanford University, 1982. PhD thesis.

[FPR00] Alberto Finzi, Fiora Pirri, and Ray Reiter. Open world planning in the situation calculus. In *Proc. AAAI*, 2000.

[Giu00] Enrico Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. In *Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, 2000.

[GKL97] Enrico Giunchiglia, G. Neelakantan Kartha, and Vladimir Lifschitz. Representing action: indeterminacy and ramifications. *Artificial Intelligence*, 95:409–443, 1997.

[GL95] Enrico Giunchiglia and Vladimir Lifschitz. Dependent fluents. In *Proc. IJCAI-95*, pages 1964–1969, 1995.

[GL98] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI*, pages 623–630, 1998.

[GL06] Alfonso Gerevini and Derek Long. Plan constraints and preferences in PDDL3. Manuscript, University of Brescia, 2006.

[GLL+04] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1-2):49–104, 2004.

[GM05] Enrico Giunchiglia and Marco Maratea. On the relation between sat and asp procedures (or, between smodels and cmodels). In *Proc. of the 21th International Conference on Logic Programming (ICLP)*, Lecture Notes in Computer Science 3668, pages 37–51. Springer, 2005.

[GM06] E. Giunchiglia and M. Maratea. Solving optimization problems with DLL. In *Proc. ECAI*, 2006.

[GML04] Enrico Giunchiglia, Marco Maratea, and Yuliya Lierler. Sat-based answer set programming. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*. AAAI Press / The MIT Press, 2004.

[GMS98] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proc. AAAI*, 1998.

[GMTZ01] 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*, pages 347–363, 2001.

[GNT04a] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Monotone literals and learning in QBF reasoning. In *Tenth International Conference on Principles and Practice of Constraint Programming, CP 2004*, pages 260–273, 2004.

[GNT04b] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QuBE++: An efficient QBF solver. In *5th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2004*, pages 201–213, 2004.

[GT99] Fausto Giunchiglia and Paolo Traverso. Planning as model checking. In *Lecture Notes in Computer Science, Proc. of the 5th European Conference on Planning (ECP-99)*. Springer, September 1999.

[GT00] 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.

[HE05] J. Hoffmann and S. Edelkamp. The deterministic part of IPC-4: An overview. *JAIR*, 24:519–579, 2005.

[HG00a] Patrick Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *Proc. AIPS*, pages 140–149, 2000.

[HG00b] Patrick Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *Proc. AIPS*, pages 140–149, 2000.

[Kam00] Subbarao Kambhampati. Planning Graph as (Dynamic) CSP: Exploiting CBL, DDB and other CSP search techniques in Graphplan. *JAIR*, 12:1–34, 2000.

[KMS96] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proc. KR-96*, pages 374–384, 1996.

[KNS02] 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.

[KS92] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI*, pages 359–363, 1992.

[KS96] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic and stochastic search. In *Proc. AAAI-96*, pages 1194–1201, 1996.

[KS98] 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.

[KS99] Henry A. Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 318–325, Stockholm, Sweden, July 31-August 6 1999. Morgan Kaufmann.

[LA97] 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.

[LF99] Derek Long and Maria Fox. The efficient implementation of the plan-graph. *JAIR*, 10:85–115, 1999.

[Lif90] Vladimir Lifschitz. Frames in the space of situations. *Artificial Intelligence*, 46:365–376, 1990.

[Lif97] Vladimir Lifschitz. On the logic of causal explanation. *Artificial Intelligence*, 96:451–465, 1997.

[Lif02] Vladimir Lifschitz. Answer set programming and plan generation. *Artif. Intell.*, 138(1-2):39–54, 2002.

[LR94] Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation*, 4:655–678, 1994.

[McD87] Drew McDermott. A critique of pure reason. *Computational Intelligence*, 3:151–160, 1987.

[MM93] D. S. Moore and G. P. McCabe. *Introduction to the Practice of Statistics*. W. H. Freeman and Co., 1993.

[MR06] Vasco Miguel Manquinho and Olivier Roussel. The first evaluation of pseudo-Boolean solvers (PB05). *Journal on Satisfiability, Boolean Modeling and Computation*, 2006.

[MS88] Karen Myers and David Smith. The persistence of derived information. In *Proc. AAAI-88*, pages 496–500, 1988.

[MT97] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proc. AAAI-97*, pages 460–465, 1997.

[MT98] 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.

[MT99] Victor W. Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. In Krysztof R. Apt, Victor W. Marek, Mirek Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm: A Twenty-Five Year Perspective*. Springer-Verlag, 1999.

[Nie99] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[Nil80] Nils Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA., 1980.

[NPT06] M. Narizzano, L. Pulina, and A. Tacchella. The third QBF solvers comparative evaluation. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:145–164, 2006. Available on-line at `http://jsat.ewi.tudelft.nl/`.

[PBT01] M. Pistore, R. Bettin, and P. Traverso. Symbolic Techniques for Extended Goals in Non-deterministic Domains. Technical Report 01-1328, IRST, 2001.

[PG86] D.A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proc. 18th Ann. IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[Pro93] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[PT01] M. Pistore and P. Traverso. Planning as Model Checking for Extended Goals in Non-deterministic Domains. In *Proc. 7^{th} International Joint Conference on Artificial Intelligence (IJCAI-01)*. AAAI Press, August 2001.

[Rin99a] Jussi Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[Rin99b] 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.

[Rin01] 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.

[Sht00] O. Shtrichman. Tuning SAT checkers for bounded model-checking. In *Proc. 12th International Computer Aided Verification Conference (CAV)*, 2000.

[SLM92] 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.

[SW83] Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 1-2. Springer-Verlag, 1983.

[SW98] David Smith and Daniel Weld. Conformant graphplan. In *Proc. AAAI-98*, pages 889–896, 1998.

[Tse70] G. Tseitin. On the complexity of proofs in propositional logics. *Seminars in Mathematics*, 8, 1970. Reprinted in [SW83].

[Tur02] Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *Proc. JELIA*, pages 111–124, 2002.

[War98] Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Inf. Process. Lett.*, 68(2):63–69, 1998.

[Zha97] 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.