

Part II

Applications and Extensions

Chapter 1

Planning and SAT

Jussi Rintanen

Planning in Artificial Intelligence is one of the earliest applications of SAT to solving generic search problems. The planning problem involves finding a sequence of actions that reaches a given goal. Such an action sequence and an associated state sequence correspond to a satisfying assignment of a propositional formula which can be easily constructed from the problem description.

1.1. Introduction

Planning in artificial intelligence is the process of finding a sequence of actions that reaches a predefined goal. The problem is usually expressed in terms of a description of an initial state, a set of actions, and a goal state. In the most basic form of planning, called classical planning, there is only one initial state and the actions are deterministic.

In their 1992 paper, Kautz and Selman suggested that the classical planning problem could be solved by translating it into a propositional formula and testing its satisfiability [KS92]. This approach was in strong contrast with earlier works on planning which viewed planning as a deductive problem or a specialized search problem. At first, this idea did not attract much interest, but in 1996 Kautz and Selman demonstrated that the best algorithms for SAT together with improved translations provided a competitive approach to classical planning [KS96].

The success of SAT in planning suggested the same approach to solving other similar problems, such as model-checking in computer-aided verification and validation. A SAT-based approach to model-checking properties expressed in the linear temporal logic LTL [Pnu77] was proposed soon after the success of planning as satisfiability became more widely known [BCCZ99]. More recent applications include diagnosis and diagnosability testing for discrete event systems [RG07, GARK07].

Following the work by Kautz and Selman, similar translations of planning into many other formalisms were proposed: nonmonotonic logic programs [DNK97], mixed integer linear programming [KW99, WW99, VBLN99], and constraint satisfaction problems CSP [vBC99, DK01].

The classical planning problem is PSPACE-complete [Byl94], so there are presumably (assuming $P \neq \text{PSPACE}$) no polynomial time translations from classical

planning into SAT. The formulae that represent the classical planning problem may, in the worst case, be exponential in the size of the problem instance. However, the formula size is determined by a natural parameter which tends not to be very high in practically interesting cases: the plan length. Most classes of planning problems are solvable with plans of a polynomial length, which means that the corresponding propositional formulae have a polynomial size. So in practice it is feasible to translate the planning problem into a SAT problem.

The underlying idea in using SAT for planning is to encode the bounded plan existence problem, i.e. whether a plan of a given length n exists, as a formula in the classical propositional logic. The formula for a given n is satisfiable if and only if there is a plan of length n . Finding a plan reduces to testing the satisfiability of the formulae for different values of n .

The efficiency of the SAT-based approach to AI planning is determined by three largely orthogonal components.

1. Efficient representations ϕ_n of the planning problem for a given plan length n . This is the topic of Sections 1.3 and 1.4.
2. Efficient algorithms for testing the satisfiability of ϕ_n .
3. Algorithms for choosing which values of n to consider to find a satisfiable ϕ_n as quickly as possible. This is the topic of Section 1.5.

Steps 1 and 2 can be combined by implementing specialized inference and search algorithms for planning [Rin98, BH99]. This approach may be very useful because it allows the incorporation of planning specific heuristics and inference rules. However, the use of general-purpose SAT algorithms is more flexible and has so far provided more benefits than the specialized approach.

Section 1.3 describes the most basic representation of planning in SAT, and the most important improvements to it to achieve efficient planning.

Section 1.4 introduces the notion of parallel plans [BF97, KS96] which is an important factor in the efficiency of planning as satisfiability. In a parallel plan there may be more than one operator at each time point. The length parameter n restricts the number of time points but not directly the number of operators. Different notions of parallel plans can be defined, and for maintaining the connection to sequential plans it is required that there is a parallel plan exactly when there is a sequential plan, and moreover, mapping a given parallel plan to a sequential plan should be possible in polynomial time.

Section 1.5 presents different strategies for making the satisfiability tests for the bounded planning problem for different lengths.

Section 1.7 describes an extension of planning as satisfiability that uses quantified Boolean formulae (QBF) for handling incomplete information. The additional power of QBF makes it possible to express plans which reach the goals starting from any of a number of initial states, and other extensions of the classical planning problem, for example, nondeterministic actions and partial observability.

1.2. Notation

We consider planning in a setting where the states of the world are represented in terms of a set A of Boolean state variables which take the value *true* or *false*.

Formulae are formed from the state variables with the connectives \vee and \neg . The connectives \wedge , \rightarrow and \leftrightarrow are defined in terms of \vee and \neg . A *literal* is a formula of the form a or $\neg a$ where $a \in A$ is a state variable. We define the *complements* of literals as $\bar{a} = \neg a$ and $\overline{\neg a} = a$ for all $a \in A$. A *clause* is a disjunction $l_1 \vee \dots \vee l_n$ of one or more literals. We also use the constant atoms \top and \perp for denoting *true* and *false*, respectively. Each *state* $s : A \rightarrow \{0, 1\}$ assigns each state variable in A a value 0 or 1.

We use *operators* for expressing the actions which change the state of the world.

Definition 1.2.1. An *operator* over a set of state variables A is a pair $\langle p, e \rangle$ where

1. p is a propositional formula over A (*the precondition*) and
2. e is a set of pairs $f \Rightarrow d$ (*the effects*) where f is a propositional formula over A and d is a set of literals over A . (In the planning literature, the case in which f is not \top is called a “conditional effect”.)

The meaning of $f \Rightarrow d$ is that the literals in d are made true if the formula f is true. For an operator $o = \langle p, e \rangle$ its *active effects* in a state s are

$$[o]_s = [e]_s = \bigcup \{d \mid f \Rightarrow d \in e, s \models f\}.$$

The operator is *executable* in s if $s \models p$ and $[o]_s$ is consistent (does not contain both a and $\neg a$ for any $a \in A$.) If this is the case, then we define $exec_o(s)$ as the unique state that is obtained from s by making $[o]_s$ true and retaining the values of the state variables not occurring in $[o]_s$. For sequences $o_1; o_2; \dots; o_n$ of operators we define $exec_{o_1; o_2; \dots; o_n}(s)$ as $exec_{o_n}(\dots exec_{o_2}(exec_{o_1}(s)) \dots)$. For sets S of operators and states s we define $exec_S(s)$ as the result of simultaneously executing all operators in S . We require that $exec_o(s)$ is defined for every $o \in S$ and that the set $[S]_s = \bigcup_{o \in S} [o]_s$ of active effects of all operators in S is consistent. For operators $o = \langle p, e \rangle$ and atomic effects l of the form a and $\neg a$ (for $a \in A$) define the *effect precondition* $EPC_l(o) = \bigvee \{f \mid f \Rightarrow d \in e, l \in d\}$ where the empty disjunction $\bigvee \emptyset$ is defined as \perp . This formula represents those states in which l is an active effect of o .

Lemma 1.2.1. For literals l , operators o and states s , $l \in [o]_s$ if and only if $s \models EPC_l(o)$.

Let $\pi = \langle A, I, O, G \rangle$ be a *problem instance* consisting of a set A of state variables, a state I over A (the initial state), a set O of operators over A , and a formula G over A (the goal formula).

A (sequential) *plan* for π is a sequence $\sigma = o_1; \dots; o_n$ of operators from O such that $exec_\sigma(I) \models G$. This means that executing the operators in the given order starting in the initial state is defined (the precondition of every operator is true and the active effects are consistent when the operator is executed) and produces a state that satisfies the goal formula. Sometimes we say that an operator sequence is a plan for O and I when we simply want to say that the plan is executable starting from I without specifying the goal states.

1.3. Sequential Plans

Consider a problem instance $\langle A, I, O, G \rangle$. We define the set $A' = \{a' | a \in A\}$ of propositional variables which contains a variable a' for every state variable $a \in A$. The variable a' refers to the value of a in a successor state, when we consider only two consecutive states. Similarly, we define $A^n = \{a^n | a \in A\}$ consisting of all variables in A superscripted with an integer $n \geq 0$. The propositional variables refer to the values of a at different integer time points. We will later use these superscripts with formulae ϕ so that ϕ^t is the formula obtained from ϕ by replacing every $a \in A$ by the corresponding $a^t \in A^t$.

An operator $o = \langle p, e \rangle$ can be represented as

$$\tau_o = p \wedge \bigwedge_{a \in A} [(EPC_a(o) \vee (a \wedge \neg EPC_{\neg a}(o))) \leftrightarrow a'].$$

This formula expresses the precondition of the operator and the new value of each state variable a in terms of the values of the state variables before the operator is executed: a will be true if it becomes true or it was true already and does not become false.

Lemma 1.3.1. *Let s and s' be states over A . Let $s'' : A' \rightarrow \{0, 1\}$ be a state over A' such that $s''(a') = s'(a)$ for all $a \in A$. Then $s \cup s'' \models \tau_o$ if and only if $s' = \text{exec}_o(s)$.*

The formula for representing the possibility of taking any of the actions is

$$\mathcal{T}(A, A') = \bigvee_{o \in O} \tau_o.$$

Later we will use this formula by replacing occurrences of the propositional variables in A and A' by propositional variables referring to different integer time points.

Lemma 1.3.2. *Let s and s' be states. Let $s'' : A' \rightarrow \{0, 1\}$ be a state over A' so that $s''(a') = s'(a)$ for all $a \in A$. Then $s \cup s'' \models \mathcal{T}(A, A')$ if and only if $s' = \text{exec}_o(s)$ for some $o \in O$.*

Since the formula $\mathcal{T}(A, A')$ expresses the changes in the state of the world between two time points by one action, the changes caused by a sequence of actions over a sequence of states can be expressed by iterating this formula.

Theorem 1.3.3. *Let $\pi = \langle A, I, O, G \rangle$ be a problem instance. Let $\iota = \bigwedge \{a \in A | s \models a\} \wedge \bigwedge \{\neg a | a \in A, I \not\models a\}$. The formula*

$$\iota^0 \wedge \mathcal{T}(A^0, A^1) \wedge \mathcal{T}(A^1, A^2) \wedge \dots \wedge \mathcal{T}(A^{t-1}, A^t) \wedge G^t$$

is satisfiable if and only there is a sequence o_0, o_1, \dots, o_{t-1} of t actions such that $\text{exec}_{o_{t-1}}(\dots \text{exec}_{o_1}(\text{exec}_{o_0}(I)) \dots) \models G$.

A satisfying assignment represents the sequence of states that is visited when a plan is executed. The plan can be constructed by identifying operators which correspond to the changes between every pair of consecutive states.

1.3.1. Improvements

This basic representation of planning in the propositional logic can be improved many ways. In the following we discuss some of the most powerful ones that are applicable to wide classes of planning problems.

1.3.1.1. Approximations of Reachability

Important for the efficiency of planning with SAT is that the search by a SAT algorithm focuses on sequences of states that are reachable from the initial state. Testing whether a state is reachable is as difficult as planning itself but there are efficient algorithms for finding approximate information about reachability, for example, in the form of dependencies between state variables. A typical dependence between some state variables a_1, \dots, a_n is that at most one of them has value 1 at a time, in any reachable state. Adding formulae $\neg a_i \vee \neg a_j$ for all $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, n\}$ such that $i \neq j$ may improve the efficiency of the satisfiability tests tremendously because no states need to be considered that are later found not to be reachable from the initial state. This kind of dependencies are called *invariants*. There are algorithms for finding invariants automatically [Rin98, GS98, Bra01, Rin08]. These algorithms run in polynomial time and find a subset of the invariants that are 2-literal clauses. Invariants were first introduced to planning in the form of *mutexes* in the planning graphs of Blum and Furst [BF97].

1.3.1.2. Control Knowledge

Domain-specific control knowledge can substantially improve the efficiency of satisfiability testing [HSK99]. Typically, domain-specific control is in the form of rules that tell something about the properties of plans, for example that certain actions in certain situations can or cannot contribute to reaching the goals. Control knowledge can be expressed in many ways, including as formulae of the Linear Temporal Logic LTL. The extension of the planning problem to executions satisfying an LTL formula has been investigated in connection with LTL model-checking and SAT [BCCZ99].

1.3.1.3. Factoring

Input languages for planning systems represent sets of actions as an action schema together with objects by which the variables in the action schema are instantiated in order to obtain the non-schematic (ground) actions of Section 1.2. The size of the set of ground actions may be exponential in the size of the schematic representation. Sometimes it is possible to reduce the size increase by representing part of the instantiation process directly as a propositional formula. For example, an action $\text{move}(x, y, z)$ for moving an object x from location y to location z can be represented by the parameters $x \in X$, $y \in Y$ and $z \in Y$ where X is the set of all objects and Y is the set of all locations. Instead of $|X| \times |Y|^2$ propositional variables for representing the ground actions, only $|X| + 2|Y|$ propositional variables are needed for representing the possible parameter values of the move action. This has been called a factored representation [KMS96, EMW97].

1.3.1.4. Symmetries

Symmetry, like the possibility of factored representations, often arises from the possibility of expressing planning problems as schemata which are instantiated with some set of objects. If some objects are interchangeable, any plan can be mapped to another valid plan by permuting the objects in an arbitrary way. For every class of plans that are equivalent modulo permutation it suffices to consider only one plan. For large but highly symmetric planning problems recognizing and utilizing the symmetry may be critical for efficient planning.

Symmetry is a general phenomenon in many types of search and reasoning problems. Generic SAT-based symmetry reduction techniques [CGLR96] can be applied to planning directly but they may be expensive due to the large size of the formulae. There are specialized algorithms for deriving symmetry-breaking constraints for planning [Rin03].

1.4. Parallel Plans

An important factor in the efficiency of SAT-based planners is the notion of parallel plans. Few plans are purely sequential in the sense that any two consecutive actions have to be in the given order. For example, the last two actions in a plan for achieving $a \wedge b$ could respectively make a and b true and be completely independent of each other. The ordering of the actions can be reversed without invalidating the plan. This kind of independence naturally leads to the notion of partially ordered plans, which was first utilized in the connection of the so-called non-linear or partial-order approach to AI planning [Sac75, MR91]. Techniques utilizing partial-orders and independence outside planning include partial-order reduction techniques [God91, Val91, ABH⁺97] for reachability analysis.

The highly influential GraphPlan planner [BF97] introduced a restricted notion of partial ordering which turned out to be very effective for SAT-based planning [KS96]: a plan is viewed as a sequence of sets of actions.

In this section we discuss two forms of partially ordered plans and present their translations into the propositional logic. The first definition generalizes that of Blum and Furst [BF97]. The idea of the second was proposed by Dimopoulos et al. [DNK97] and later formalized and applied to SAT-based planning [RHN06].

1.4.1. \forall -Step Plans

The first definition of parallel plans interprets parallelism as the possibility of ordering the operators to any total order.

Definition 1.4.1 (\forall -Step plans). For a set of operators O and an initial state I , a \forall -step plan for O and I is a sequence $T = \langle S_0, \dots, S_{l-1} \rangle$ of sets of operators for some $l \geq 0$ such that there is a sequence of states s_0, \dots, s_l (the execution of T) such that

1. $s_0 = I$, and
2. for all $i \in \{0, \dots, l-1\}$ and every total ordering o_1, \dots, o_n of S_i , $exec_{o_1; \dots; o_n}(s_i)$ is defined and equals s_{i+1} .

When active effects are independent of the current state and preconditions are conjunctions of literals, Definition 1.4.1 exactly corresponds to a syntactic definition of independence [RHN06]. In more general cases the correspondence breaks down, and syntactically independent sets of operators are only a subclass of sets of operators which can be ordered to any total order.

Testing whether a sequence of sets of operators is a \forall -step is in general intractable [RHN06] which justifies syntactic rather than semantic restrictions on parallel operators [BF97, KS96].

We define positive and negative occurrences of state variables in a formula.

Definition 1.4.2 (Positive and negative occurrences). We say that a state variable a *occurs positively* in ϕ if $\text{positive}(a, \phi)$ is true. Similarly, a *occurs negatively* in ϕ if $\text{negative}(a, \phi)$ is true.

$$\begin{aligned} \text{positive}(a, a) &= \text{true}, \text{ for all } a \in A \\ \text{positive}(a, b) &= \text{false}, \text{ for all } \{a, b\} \subseteq A \text{ such that } a \neq b \\ \text{positive}(a, \phi \vee \phi') &= \text{positive}(a, \phi) \text{ or } \text{positive}(a, \phi') \\ \text{positive}(a, \neg\phi) &= \text{negative}(a, \phi) \\ \\ \text{negative}(a, b) &= \text{false}, \text{ for all } \{a, b\} \subseteq A \\ \text{negative}(a, \phi \vee \phi') &= \text{negative}(a, \phi) \text{ or } \text{negative}(a, \phi') \\ \text{negative}(a, \neg\phi) &= \text{positive}(a, \phi) \end{aligned}$$

A state variable a *occurs in* ϕ if it occurs positively or negatively in ϕ .

Definition 1.4.3 (Affect). Let A be a set of state variables and $o = \langle p, e \rangle$ and $o' = \langle p', e' \rangle$ operators over A . Then o *affects* o' if there is $a \in A$ such that

1. $a \in d$ for some $f \Rightarrow d \in c$ and a occurs in f for some $f \Rightarrow d \in c'$ or occurs negatively in p' , or
2. $\neg a \in d$ for some $f \Rightarrow d \in c$ and a occurs in f for some $f \Rightarrow d \in c'$ or occurs positively in p' .

An operator o which affects another operator o' may prevent its execution or change its active effects. This means that replacing $o'; o$ by $o; o'$ in a plan may make the plan invalid or change its execution.

Definition 1.4.4 (Interference). Let A be a set of state variables. Operators o and o' *interfere* if o affects o' or o' affects o .

If operators o and o' do not interfere, then the sequences $o; o'$ and $o'; o$ are interchangeable. Non-interference is a sufficient but not a necessary condition for interchangeability.

1.4.2. \exists -Step Plans

A more relaxed notion of parallel plans was proposed by Dimopoulos et al. [DNK97] and formalized and investigated in detail by Rintanen et al. [RHN06].

Definition 1.4.5 (\exists -Step plans). For a set O of operators and an initial state I , a \exists -step plan is $T = \langle S_0, \dots, S_{l-1} \rangle \in (2^O)^l$ together with a sequence of states s_0, \dots, s_l (the execution of T) for some $l \geq 0$ such that

1. $s_0 = I$, and
2. for every $i \in \{0, \dots, l-1\}$ there is a total ordering $o_1 < \dots < o_n$ of S_i such that $s_{i+1} = \text{exec}_{o_1; \dots; o_n}(s_i)$.

The difference to \forall -step plans is that instead of requiring that each step S_i can be ordered to any total order, it is sufficient that there is one order that maps state s_i to s_{i+1} . Unlike for \forall -step plans, the successor s_{i+1} of s_i is not uniquely determined solely by S_i because the successor depends on the implicit ordering of S_i . For this reason the definition has to make the execution s_0, \dots, s_l explicit.

The more relaxed definition of \exists -step plans sometimes allows much more parallelism than the definition of \forall -step plans.

Example 1.4.1. Consider a row of n Russian dolls, each slightly bigger than the preceding one. We can nest all the dolls by putting the first inside the second, then the second inside the third, and so on, until every doll except the largest one is inside another doll.

For four dolls this can be formalized as follows.

$$\begin{aligned} o_1 &= \langle \text{out1} \wedge \text{out2} \wedge \text{empty2}, \{\top \Rightarrow \text{1in2}, \top \Rightarrow \neg \text{out1}, \top \Rightarrow \neg \text{empty2}\} \rangle \\ o_2 &= \langle \text{out2} \wedge \text{out3} \wedge \text{empty3}, \{\top \Rightarrow \text{2in3}, \top \Rightarrow \neg \text{out2}, \top \Rightarrow \neg \text{empty3}\} \rangle \\ o_3 &= \langle \text{out3} \wedge \text{out4} \wedge \text{empty4}, \{\top \Rightarrow \text{3in4}, \top \Rightarrow \neg \text{out3}, \top \Rightarrow \neg \text{empty4}\} \rangle \end{aligned}$$

The shortest \forall -step plan nests the dolls in three steps: $\langle \{o_1\}, \{o_2\}, \{o_3\} \rangle$. The \exists -step plan $\langle \{o_1, o_2, o_3\} \rangle$ nests the dolls in one step.

Theorem 1.4.1. (i) Every \forall -step plan is a \exists -step plan, and (ii) for every \exists -step plan T there is a \forall -step plan whose execution leads to the same final state as that of T .

Similarly to \forall -step plans, testing the validity of \exists -step plans is intractable in the worst case [RHN06]. One way of achieving tractability uses two restrictions. All parallel operators are required to be executable in the current state (unlike for \forall -step plans this does not follow from the definition), and parallel operators are required to fulfill an ordered dependence condition based on Definition 1.4.3: operators are allowed in parallel if they can be totally ordered so that no operator affects a later operator. This means that the operators preceding a given operator do not change its effects or prevent its execution.

Theorem 1.4.2. Let O be a set of operators, I a state, $T = \langle S_0, \dots, S_{l-1} \rangle \in (2^O)^l$, and s_0, \dots, s_l a sequence of states. If

1. $s_0 = I$,
2. for every $i \in \{0, \dots, l-1\}$ there is a total ordering $<$ of S_i such that if $o < o'$ then o does not affect o' , and
3. $s_{i+1} = \text{exec}_{S_i}(s_i)$ for every $i \in \{0, \dots, l-1\}$,

then T is a \exists -step plan for O and I .

Even though the class of \exists -step plans based on *affects* is narrower than the class sanctioned by Definition 1.4.5, much more parallelism is still possible in comparison to \forall -step plans. For instance, nesting of Russian dolls in Example 1.4.1 belongs to this class.

1.4.3. Optimality of Parallel Plans

The most commonly used measure for plan quality is the number of operators in the plan. So the problem of finding an optimal plan is the problem of finding the shortest sequential plan, or in terms of the satisfiability tests, finding an integer t such that the formula for plan length $t - 1$ is unsatisfiable and the formula for plan length t is satisfiable. For parallel plans the question is more complicated. It is possible to find an optimal parallel plan for a given parallel length by using cardinality constraints on the number of actions in the plans, but it seems that finding an optimal parallel plan always reduces to testing the existence of sequential plans [BR05]: if there is a parallel plan with t time points and n actions, the optimality test always requires testing whether there is a sequential plan with $n - 1$ time points and $n - 1$ actions. Tests restricting to formulae with t time points are not sufficient.

1.4.4. Representation in SAT

In this section we present translations of parallel plans into the propositional logic. A basic assumption is that for sets S of simultaneous operators executed in state s the state $exec_S(s)$ is defined, that is, all the preconditions are true in s and the set of active effects of the operators is consistent.

1.4.4.1. The Base Translation

Planning is performed by propositional satisfiability testing by producing formulae $\phi_0, \phi_1, \phi_2, \dots$ such that ϕ_l is satisfiable iff there is a plan of length l . The translations for different forms of parallel plans differ only in the formulae that restrict the simultaneous execution of operators. Next we describe the part of the formulae that is shared by both definitions of parallel plans.

Consider the problem instance $\pi = \langle A, I, O, G \rangle$. Similarly to the state variables in A , for every operator $o \in O$ we have the propositional variable o^t for expressing whether o is executed at time point $t \in \{0, \dots, l - 1\}$.

A formula $\Phi_{\pi, l}$ is generated to answer the following question. Is there an execution of a sequence of l sets of operators that reaches a state satisfying G when starting from the state I ? The formula $\Phi_{\pi, l}$ is conjunction of ι^0 with $\iota = \bigwedge \{a \in A \mid s \models a\} \wedge \bigwedge \{\neg a \mid a \in A, I \not\models a\}$, G^l , and the formulae described below, instantiated with all $t \in \{0, \dots, l - 1\}$.

First, for every $o = \langle p, e \rangle \in O$ there are the following formulae. The precondition p has to be true when the operator is executed.

$$o^t \rightarrow p^t \tag{1.1}$$

For every $f \Rightarrow d \in c$ the effects d will be true if f is true at the preceding time point.

$$(o^t \wedge f^t) \rightarrow d^{t+1} \tag{1.2}$$

Here we view sets d of literals as conjunctions of literals. Second, the value of a state variable does not change if no operator that changes it is executed. Hence

for every state variable a we have two formulae, one expressing the conditions for the change of a from true to false,

$$(a^t \wedge \neg a^{t+1}) \rightarrow ((o_1^t \wedge (EPC_{\neg a}(o_1))^t) \vee \cdots \vee (o_m^t \wedge (EPC_{\neg a}(o_m))^t)), \quad (1.3)$$

and another from false to true,

$$(\neg a^t \wedge a^{t+1}) \rightarrow ((o_1^t \wedge (EPC_a(o_1))^t) \vee \cdots \vee (o_m^t \wedge (EPC_a(o_m))^t)). \quad (1.4)$$

Here $O = \{o_1, \dots, o_m\}$.

The formulae $\Phi_{\pi,l}$, just like the definition of $exec_S(s)$, allow sets of operators in parallel that do not correspond to any sequential plan. For example, the operators $\langle a, \{\top \Rightarrow \neg b\} \rangle$ and $\langle b, \{\top \Rightarrow \neg a\} \rangle$ may be executed simultaneously resulting in a state satisfying $\neg a \wedge \neg b$, even though this state is not reachable by the two operators sequentially. But we require that all parallel plans can be executed sequentially. Further formulae that are discussed in the next sections are needed to guarantee this property.

Theorem 1.4.3. *Let $\pi = \langle A, I, O, G \rangle$ be a transition system. Then there is $T = \langle S_0, \dots, S_{l-1} \rangle \in (2^O)^l$ so that s_0, \dots, s_l are states so that $I = s_0$, $s_l \models G$, and $s_{i+1} = exec_{S_i}(s_i)$ for all $i \in \{0, \dots, l-1\}$ if and only if there is a valuation satisfying the formula $\Phi_{\pi,l}$.*

Proposition 1.4.4. *The size of the formula $\Phi_{\pi,l}$ is linear in l and in the size of π .*

Theorem 1.4.3 says that a sequence of operators fulfilling certain conditions exists if and only if a given formula is satisfiable. The theorems connecting certain formulae to certain notions of plans (Theorems 1.4.5 and 1.4.6) provide an implication only in one direction: whenever the formula for a given value of parameter l is satisfiable, a plan of l time points exists. The other direction is missing because the formulae in general only approximate the respective definition of plans and there is no guarantee that the formula for a given l is satisfiable when a plan with l time points exists. However, the formula with some higher value of l is satisfiable. This follows from the fact that whenever a \forall -step or \exists -step plan $\langle S_0, \dots, S_{l-1} \rangle$ with $n = |S_0| + \dots + |S_{l-1}|$ occurrences of operators exists, there is a plan consisting of n singleton sets, and the corresponding formulae $\Phi_{\pi,n} \wedge \Phi_{O,n}^x$ are satisfiable. The formulae $\Phi_{O,n}^x$ represent the parallel semantics x for formulae O .

1.4.4.2. Translation of \forall -Step Plans

The simplest representation of the interference condition in Definition 1.4.4 is by formulae

$$\neg(o_1^t \wedge o_2^t) \quad (1.5)$$

for every pair of interfering operators o_1 and o_2 . Define $\Phi_{O,l}^{\forall step}$ as the conjunction of the formulae (1.5) for all time points $t \in \{0, \dots, l-1\}$ and for all pairs of interfering operators $\{o, o'\} \subseteq O$ that could be executed simultaneously. There are $\mathcal{O}(ln^2)$ such formulae for n operators. This can be improved to linear [RHN06].

Theorem 1.4.5. *Let $\pi = \langle A, I, O, G \rangle$ be a transition system. There is a \forall -step plan of length l for π if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall step}$ is satisfiable.*

1.4.4.3. Translation of \exists -Step Plans

The simplest efficient representation of \exists -step plans imposes a fixed total ordering on the operators and allows the simultaneous execution of a subset of the operators only if none of the operators affects any operator later in the ordering. There are more permissive ways of guaranteeing the validity of \exists -step plans, but they seem to be difficult to implement efficiently, and furthermore, there are ways of identifying most permissive total orderings as a preprocessing step by finding strongly connected components of the graph formed by the *affects* relation on operators [RHN06].

The representation does not allow all the possible parallelism but it leads to small formulae and is efficient in practice. In the translation for \exists -step plans the set of formulae constraining parallel execution *is a subset* of those for the less permissive \forall -step plans. One therefore receives two benefits simultaneously: possibly much shorter parallel plans and formulae with a smaller size / time points ratio.

The idea is to impose beforehand an (arbitrary) ordering on the operators o_1, \dots, o_n and to allow parallel execution of two operators o_i and o_j such that o_i affects o_j only if $i \geq j$. The formula $\Phi_{O,l}^{\exists step}$ is the conjunction of

$$\neg(o_i^t \wedge o_j^t) \text{ if } o_i \text{ affects } o_j \text{ and } i < j$$

for all o_i and o_j and all time points $t \in \{0, \dots, l-1\}$. Of course, this restriction to one fixed ordering may rule out many sets of parallel operators that could be executed simultaneously according to some other ordering than the fixed one.

The formula $\Phi_{O,l}^{\exists step}$ (similar to the translation for \forall -step plans in Section 1.4.4.2) has a quadratic size because of the worst-case quadratic number of pairs of operators that may not be simultaneously executed. This can be improved to linear [RHN06].

Theorem 1.4.6. *Let $\pi = \langle A, I, O, G \rangle$ be a transition system. There is a \exists -step plan of length l for π if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\exists step}$ is satisfiable.*

1.5. Finding a Satisfiable Formula

Given a sequence ϕ_0, ϕ_1, \dots of formulae representing the bounded planning problem for different plan lengths, a satisfiability algorithm is used for locating a satisfiable formula. The satisfying assignment can be translated into a plan. In this section we describe three high-level algorithms that use the formulae ϕ_i for finding a plan. We call them Algorithms S, A and B.

Algorithm S is the obvious sequential procedure for finding a satisfiable formula: first test the satisfiability of ϕ_0 , then ϕ_1 , ϕ_2 , and so on, until a satisfiable formula is found. It has been used by virtually all works that reduce planning to the fixed-length planning problem [KS92, BF97]. This algorithm has the property that it always find a plan with the smallest number of time points. It can be used for finding plans with the minimum number of actions if used in connection with the sequential encoding of planning from Section 1.3.

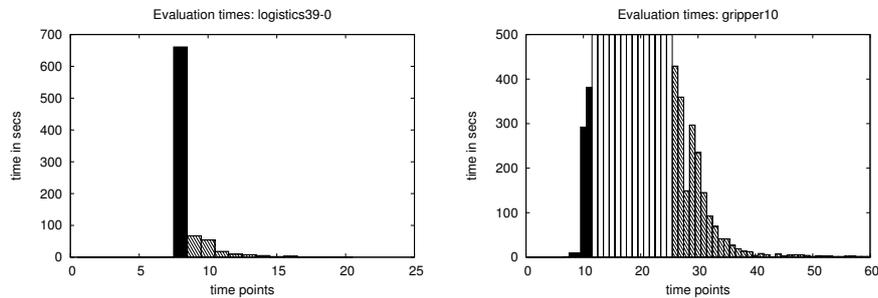


Figure 1.1. SAT solver runtimes for two problem instances and different plan lengths

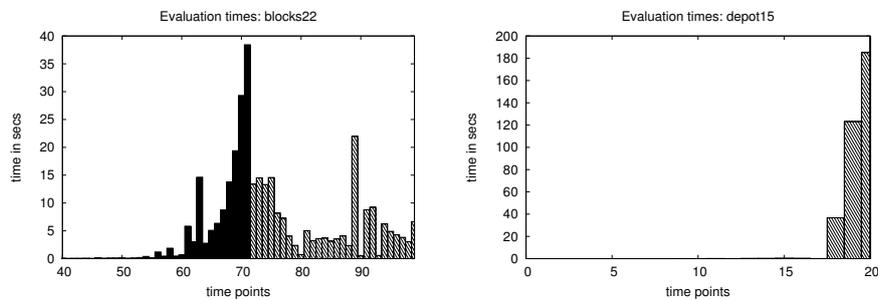


Figure 1.2. SAT solver runtimes for two problem instances and different plan lengths

If the objective is to find a plan with not necessarily the smallest number of actions or time points, the Algorithm S is often inefficient because the satisfiability tests for the last unsatisfiable formulae are often much more expensive than for the first satisfiable formulae. Consider the diagrams in Figures 1.1, 1.2 and Figure 1.3. Each diagram shows the cost of detecting the satisfiability or unsatisfiability of formulae that represent the existence of plans of different lengths. Grey bars depict unsatisfiable formulae and black bars satisfiable formulae. For more difficult problems the peak formed by the last unsatisfiable formulae is still much more pronounced.

Except for the rightmost diagram in Figure 1.2 and the leftmost diagram in Figure 1.3, the diagrams depict steeply growing costs of determining unsatisfiability of a sequence of formulae followed by small costs of determining satisfiability of formulae corresponding to plans.

We would like to run a satisfiability algorithm with the satisfiable formula for which the runtime of the algorithm is the lowest. Of course, we do not know which formulae are satisfiable and which has the lowest runtime. With an infinite number of processes we could find a satisfying assignment for one of the formulae in the smallest possible time: let each process $i \in \{0, 1, 2, \dots\}$ test the satisfiability of ϕ_i . However, an infinite number of processes running at the same speed cannot be simulated by any finite hardware. Algorithms A and B attempt to approximate

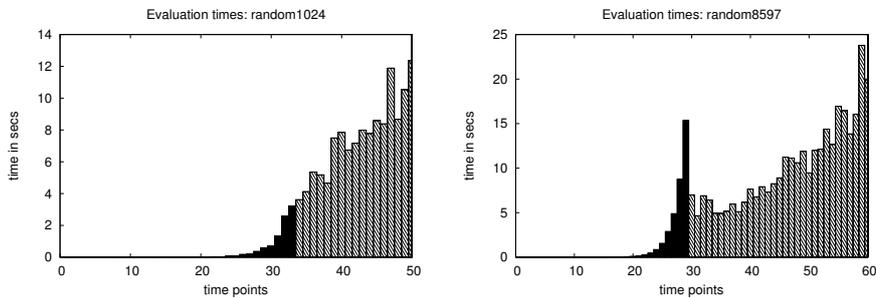


Figure 1.3. SAT solver runtimes for two problem instances and different plan lengths

```

1: procedure AlgorithmS()
2:    $i := 0$ ;
3:   repeat
4:     test satisfiability of  $\phi_i$ ;
5:     if  $\phi_i$  is satisfiable then terminate;
6:      $i := i + 1$ ;
7:   until 1=0;

```

Figure 1.4. Algorithm S

this scheme by using a finite number processes. Algorithm A uses a fixed number n of processes. Algorithm B uses a finite but unbounded number of processes which are run at variable rates.

1.5.1. Algorithm S: Sequential Testing

The standard algorithm for finding plans in the satisfiability and related approaches to planning tests the satisfiability of formulae for plan lengths 0, 1, 2, and so on, until a satisfiable formula is found [BF97, KS96]. This algorithm is given in Figure 1.4.

1.5.2. Algorithm A: Multiple Processes

Algorithm A (Figure 1.5) distributes plan search to n concurrent processes and initially assigns the first n formulae to the n processes. Whenever a process finds its formula satisfiable, the computation is terminated. Whenever a process finds its formula unsatisfiable, the process is given the next unassigned formula. This strategy can avoid completing the satisfiability tests of many of the expensive unsatisfiable formulae, thereby saving a lot of computation effort.

Algorithm S is the special case with $n = 1$. The constant ϵ determines the coarseness of CPU time division. The *for each* loop in this algorithm and in the next algorithm can be implemented so that several processes are run in parallel.

```

1: procedure AlgorithmA( $n$ )
2:    $P := \{\phi_0, \dots, \phi_{n-1}\}$ ;
3:    $\text{next} := n$ ;
4:   repeat
5:      $P' := P$ ;
6:     for each  $\phi \in P'$  do
7:       continue computation with  $\phi$  for  $\epsilon$  seconds;
8:       if  $\phi$  was found satisfiable then terminate;
9:       if  $\phi$  was found unsatisfiable then
10:          $P := P \cup \{\phi_{\text{next}}\} \setminus \{\phi\}$ ;
11:          $\text{next} := \text{next} + 1$ ;
12:       end if
13:     end do
14:   until 0=1

```

Figure 1.5. Algorithm A

```

1: procedure AlgorithmB( $\gamma$ )
2:    $t := 0$ ;
3:   for each  $i \geq 0$  do  $\text{done}[i] = \text{false}$ ;
4:   for each  $i \geq 0$  do  $\text{time}[i] = 0$ ;
5:   repeat
6:      $t := t + \delta$ ;
7:     for each  $i \geq 0$  such that  $\text{done}[i] = \text{false}$  do
8:       if  $\text{time}[i] + n\epsilon \leq t\gamma^i$  for some maximal  $n \geq 1$  then
9:         continue computation for  $\phi_i$  for  $n\epsilon$  seconds;
10:        if  $\phi_i$  was found satisfiable then terminate;
11:         $\text{time}[i] := \text{time}[i] + n\epsilon$ ;
12:        if  $\phi_i$  was found unsatisfiable then  $\text{done}[i] := \text{true}$ ; end if
13:      end if
14:    end do
15:  until 0=1

```

Figure 1.6. Algorithm B

1.5.3. Algorithm B: Geometric Division of CPU Use

Algorithm B (Figure 1.6) uses an unbounded but finite number of processes. The amount of CPU given to each process depends on the index of its formula: if formula ϕ_k is given t seconds of CPU during a certain time interval, then a formula ϕ_i , $i \geq k$ is given $\gamma^{i-k}t$ seconds. This means that every formula gets only slightly less CPU than its predecessor, and the choice of the exact value of the constant $\gamma \in]0, 1[$ is less critical than the choice of n for Algorithm A.

Variable t , which is repeatedly increased by δ , characterizes the total CPU time $\frac{t}{1-\gamma}$ available so far. As the computation for ϕ_i proceeds only if it has been going on for at most $t\gamma^i - \epsilon$ seconds, CPU is actually consumed less than $\frac{t}{1-\gamma}$, and there will be at time $\frac{t}{1-\gamma}$ only a finite number $j \leq \log_{\gamma} \frac{\epsilon}{t}$ of formulae for which computation has commenced.

The constants n and γ respectively for Algorithms A and B are roughly related by $\gamma = 1 - \frac{1}{n}$: of the CPU capacity $\frac{1}{n} = 1 - \gamma$ is spent in testing the first unfinished formula. Algorithm S is the limit of Algorithm B when γ goes to 0.

Algorithms A and B have two very useful properties [RHN06]. First, both al-

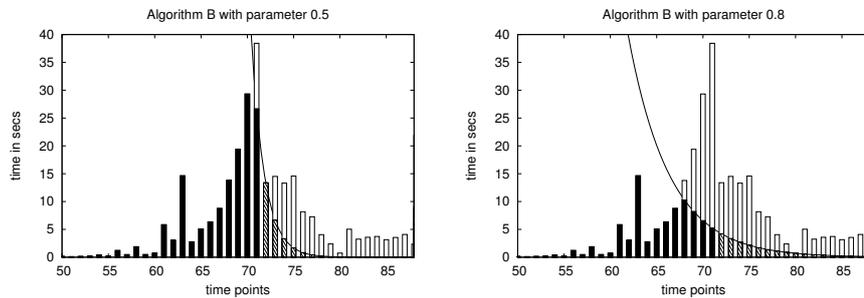


Figure 1.7. Illustration of two runs of Algorithm B. When $\gamma = 0.5$ most CPU time is spent on the first formulae, and when the first satisfiable formula is detected also the unsatisfiability of most of the preceding unsatisfiable formulae has been detected. With $\gamma = 0.8$ more CPU is spent for the later easier satisfiable formulae, and the expensive unsatisfiability tests have not been completed before finding the first satisfying assignment.

gorithms can be arbitrarily much faster than Algorithm S. This is because testing the satisfiability of some of the unsatisfiable formulae may be arbitrarily difficult in comparison to the easiest satisfiable formulae. Second, both algorithms are at most a constant factor slower than Algorithm S. This constant is n for Algorithm A and $\frac{1}{1-\gamma}$ for Algorithm B. So even in cases in which these algorithms do not improve on Algorithm S, the efficiency loss is bounded.

1.5.4. Algorithms for Optimal Planning

When trying to find a plan of minimum length it may be possible to find an arbitrary plan with a non-optimal planner and then tighten this upper bound. Sophisticated algorithms for doing this have been proposed by Streeter and Smith [SS07].

1.6. Improved SAT Solving for Planning

The regular structure of the formulae representing planning problems can sometimes be utilized to achieve much more efficient SAT solving.

When using Algorithm S, some of the information obtained from the unsatisfiability proof of the formula for i time points can be reused when testing the satisfiability of the formula for $i + 1$ time points. SAT solvers that use clause learning are particularly suitable for implementing this idea [Str01, ES03]: clauses that were learned for the formula with i points without using the goal formula are logical consequences of the formula for $i + 1$ time points as well.

Similar idea is also applicable for speeding up SAT solving of one formula because of the regular structure of clause sets [Str00]: under some conditions, several copies of a learned clause can be made for different time points.

1.7. QBF and Planning with Nondeterministic Actions

As pointed out in the introduction, the classical planning problem, even though PSPACE-complete, is in NP when restricted to polynomial length plans. There are, however, natural generalizations of the classical planning problem which are not known to be in NP even for polynomial length plans. Instead of one initial state, there may be several possible initial states. Instead of deterministic actions, actions may be nondeterministic in the sense that the successor state of a state is not unique. Both of these generalizations lift the complexity of the planning problem outside NP. Different variations of the nondeterministic planning problem are obtained by different observability restrictions. One can consider full observability, partial observability, and planning without observability. We will discuss the last case only, even though the same kind of solution methods are also applicable to the first two.

The planning problem without observability involves several initial states with the objective to find a sequence of (possibly nondeterministic) actions that reaches a goal state starting from any of the initial states. This problem is EXPSPACE-complete [HJ00], but with the restriction to polynomial length plans it is in Σ_2^P [Rin99] and Σ_2^P -hard [BKT00]. The complexity class Σ_2^P is presumed to properly include NP, and hence this planning problem presumably cannot be translated into SAT in polynomial time. Furthermore, the exponentiality of the translation easily shows up in practice. This suggests the use of quantified Boolean formulas (QBF) [SM73] for solving the problem.

The complexity class Σ_2^P corresponds to QBF with the prefix $\exists\forall$. However, the most natural translation of the planning problem into QBF has the prefix $\exists\forall\exists$, corresponding to problems in Σ_3^P . This discrepancy can be interpreted as the power of QBF with prefix $\exists\forall\exists$ to handle a generalized planning problem in which the possibility of taking an action in a given state and computing the unique successor state cannot be performed in polynomial time [Tur02]. In this section we present the most obvious translation that uses the prefix $\exists\forall\exists$. Translation with prefix $\exists\forall$ is less intuitive [Rin07].

The planning problem with several initial states is similar to the problem of finding homing or reset or synchronizing sequences of transition systems [PB91].

Definition 1.7.1. Let A be a set of state variables. A *nondeterministic operator* is a pair $\langle p, e \rangle$ where p is a propositional formula over A (the *precondition*), and e is an *effect* over A . Effects over A are recursively defined as follows.

1. Any set of pairs $f \Rightarrow d$ where f is a formula over A and d is a set of literals over A is an effect over A .
2. If e_1 and e_2 are effects over A , then also $e_1|e_2$ is an effect over A .

$e_1|e_2$ denotes a nondeterministic choice between e_1 and e_2 : when an action with effect $e_1|e_2$ is taken, then one of e_1 and e_2 is randomly chosen. If the chosen effect is nondeterministic, then random choice is made recursively until a deterministic effect is obtained.

Definition 1.7.2 (Operator execution). Let $\langle p, e \rangle$ be a nondeterministic operator over A . Let s be a state (a valuation of A). The operator is *executable in s* if

$s \models p$ and every set $E \in [e]_s^{nd}$ is consistent. The set $[o]_s^{nd} = [e]_s^{nd}$ of possible sets of active effects is recursively defined as follows.

1. $[e_1|e_2]_s^{nd} = [e_1]_s^{nd} \cup [e_2]_s^{nd}$
2. $[e]_s^{nd} = [e]_s$ if e is deterministic.

The alternative successor states of a state s when operator $\langle p, e \rangle$ is executed are those that are obtained from s by making the literals in some $E \in [e]_s^{nd}$ true and retaining the values of state variables not occurring in E .

Analogously to the deterministic planning problem in Section 1.2, a problem instance is defined $\langle A, I, O, G \rangle$ where A is a set of state variables, I and G are formulae over A (respectively representing the sets of initial and goal states), and O is a set of nondeterministic operators over A . There is a solution plan for this problem instance if there is a sequence o_1, \dots, o_n of operators such that $\{o_1, \dots, o_n\} \subseteq O$ and for all states s such that $s \models I$, the operator sequence is executable starting from s , and every execution terminates in a state s' such that $s' \models G$.

The condition for the atomic effect l to be executed when e is executed is $EPC_l^{nd}(e, \sigma)$. This expresses the active effects of e under some choice of nondeterministic outcomes expressed by auxiliary variables. The sequence σ of integers is used for deriving unique names for the auxiliary variables x_σ that control nondeterminism. The sequences start with the index of the operator.

$$\begin{aligned} EPC_l^{nd}(e, \sigma) &= EPC_l(e) \text{ if } e \text{ is deterministic} \\ EPC_l^{nd}(e_1|e_2, \sigma) &= (x_\sigma \wedge EPC_l^{nd}(e_1, \sigma 1)) \\ &\quad \vee (\neg x_\sigma \wedge EPC_l^{nd}(e_2, \sigma 1)) \end{aligned}$$

Nondeterminism is encoded by making the effects conditional on the values of the auxiliary variables x_σ . Different valuations of the auxiliary variables correspond to different nondeterministic outcomes.

The following frame axioms express the conditions under which state variables $a \in A$ may change from true to false and from false to true. Let e_1, \dots, e_n be the effects of o_1, \dots, o_n respectively. Each operator $o \in O$ has a unique integer index $oi(o)$.

$$\begin{aligned} (a \wedge \neg a') &\rightarrow \bigvee_{i=1}^n ((o_i \wedge EPC_a^{nd}(e_i, oi(o_i))) \\ (\neg a \wedge a') &\rightarrow \bigvee_{i=1}^n ((o_i \wedge EPC_a^{nd}(e_i, oi(o_i))) \end{aligned}$$

For $o = \langle p, e \rangle \in O$ there is a formula for describing values of state variables in the predecessor and successor states when the operator is executed.

$$\begin{aligned} &(o \rightarrow p) \wedge \\ &\bigwedge_{a \in A} (o \wedge EPC_a^{nd}(e, oi(o)) \rightarrow a') \wedge \\ &\bigwedge_{a \in A} (o \wedge EPC_{\neg a}^{nd}(e, oi(o)) \rightarrow \neg a') \end{aligned}$$

Constraints for parallel operators can be defined like for \forall -step plans in Section 1.4.4.2.

Let X be the set of auxiliary variables x_σ in all of the above formulae. The conjunction of all the above formulae is denoted by $\mathcal{R}(A, A', O, X)$.

The translation of the planning problem into QBF is the following.

$$\exists P \forall C \exists E (I^0 \rightarrow (\mathcal{R}(A^0, A^1, O^0, X^0) \wedge \dots \wedge \mathcal{R}(A^{t-1}, A^t, O^{t-1}, X^{t-1}) \wedge G^t))$$

The propositional variables $P = \bigcup_{i=0}^{t-1} O^i$ represent the operators in a plan, $C = A^0 \cup \bigcup_{i=0}^{t-1} X^i$ consists of the propositional variables that express different contingencies (initial states and nondeterminism), and $E = \bigcup_{i=1}^t A^i$ is the set of all remaining propositional variables which represent the executions of a plan expressed by P under the contingencies expressed by C .

The QBF says that *there is* a plan such that *for all* possible contingencies *there is* an execution that leads to a goal state. Since there is no observability, the plan is a sequence of operators that cannot depend on the initial state or any events that happen during plan execution.

It is possible to represent a much more general class of planning problems with QBF than the one discussed in this section, with partial observability and branching program-like plans [Rin99]. A probabilistic extension of QBF, stochastic satisfiability or SSAT, makes it possible to express transition probabilities associated with nondeterministic actions and noisy observations [ML03].

QBFs are implicitly present in some works that do not explicitly use QBF but use SAT as a subproblem [CGT03, BH04, PBDG05]. The search algorithms in these works can be viewed as implementing specialized algorithms for restricted classes of QBF.

References

- [ABH⁺97] Rajeev Alur, Robert K. Brayton, Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. Partial-order reduction in symbolic state space exploration. In *Computer Aided Verification, 9th International Conference, CAV'97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 340–351. Springer-Verlag, 1997.
- [BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In W. R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, 1999.
- [BF97] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.
- [BH99] Ronen I. Brafman and Holger H. Hoos. To encode or not to encode - linear planning. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 988–995. Morgan Kaufmann Publishers, 1999.
- [BH04] Ronen I. Brafman and Jörg Hoffmann. Conformant planning via heuristic forward search: A new approach. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 355–364. AAAI Press, 2004.

- [BKT00] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122(1):241–267, 2000.
- [BR05] Markus Büttner and Jussi Rintanen. Satisfiability planning with constraints on the number of actions. In Susanne Biundo, Karen Myers, and Kanna Rajan, editors, *ICAPS 2005. Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, pages 292–299. AAAI Press, 2005.
- [Bra01] R. I. Brafman. On reachability, relevance, and resolution in the planning as satisfiability approach. *Journal of Artificial Intelligence Research*, 14:1–28, 2001.
- [Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.
- [CGLR96] James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*, pages 148–159. Morgan Kaufmann Publishers, 1996.
- [CGT03] Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. SAT-based planning in complex domains: concurrency, constraints and nondeterminism. *Artificial Intelligence*, 147(1–2):85–117, 2003.
- [DK01] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182, 2001.
- [DNK97] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in nonmonotonic logic programs. In S. Steel and R. Alami, editors, *Recent Advances in AI Planning. Fourth European Conference on Planning (ECP'97)*, number 1348 in Lecture Notes in Computer Science, pages 169–181. Springer-Verlag, 1997.
- [EMW97] Michael Ernst, Todd Millstein, and Daniel S. Weld. Automatic SAT-compilation of planning problems. In Martha Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1169–1176. Morgan Kaufmann Publishers, 1997.
- [ES03] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.
- [GARK07] Alban Grastien, Anbulagan, Jussi Rintanen, and Elena Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 305–310. AAAI Press, 2007.
- [God91] P. Godefroid. Using partial orders to improve automatic verification methods. In E. M. Clarke, editor, *Proceedings of the 2nd International Conference on Computer-Aided Verification (CAV '90)*, Rutgers, New Jersey, 1990, number 531 in Lecture Notes in Computer Science, pages 176–185. Springer-Verlag, 1991.
- [GS98] Alfonso Gerevini and Lenhart Schubert. Inferring state constraints

- for domain-independent planning. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 905–912. AAAI Press, 1998.
- [HJ00] Patrik Haslum and Peter Jonsson. Some results on the complexity of planning with incomplete information. In Susanne Biundo and Maria Fox, editors, *Recent Advances in AI Planning. Fifth European Conference on Planning (ECP'99)*, number 1809 in Lecture Notes in Artificial Intelligence, pages 308–318. Springer-Verlag, 2000.
- [HSK99] Yi-Cheng Huang, Bart Selman, and Henry A. Kautz. Control knowledge in planning: Benefits and tradeoffs. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99) and the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 511–517, 1999.
- [KMS96] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*, pages 374–385. Morgan Kaufmann Publishers, 1996.
- [KS92] Henry Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363. John Wiley & Sons, 1992.
- [KS96] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201. AAAI Press, August 1996.
- [KW99] Henry Kautz and Joachim Walser. State-space planning by integer optimization. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99) and the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 526–533. AAAI Press, 1999.
- [ML03] Stephen M. Majercik and Michael L. Littman. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, 147(1-2):119–162, 2003.
- [MR91] David A. McAllester and David Rosenblitt. Systematic nonlinear planning. In T. L. Dean and K. McKeown, editors, *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 634–639. AAAI Press / The MIT Press, 1991.
- [PB91] C. Pixley and G. Beihl. Calculating resettability and reset sequences. In *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference*, pages 376–379, 1991.
- [PBDG05] Héctor Palacios, Blai Bonet, Adnan Darwiche, and Héctor Geffner. Pruning conformant plans by counting models on compiled d-DNNF representations. In Susanne Biundo, Karen Myers, and Kanna Rajan, editors, *ICAPS 2005. Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, pages 141–150.

- AAAI Press, 2005.
- [Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE, 1977.
- [RG07] Jussi Rintanen and Alban Grastien. Diagnosability testing with satisfiability algorithms. In Manuela Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 532–537. AAAI Press, 2007.
- [RHN06] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.
- [Rin98] Jussi Rintanen. A planning algorithm not based on directional search. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, pages 617–624. Morgan Kaufmann Publishers, June 1998.
- [Rin99] Jussi Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.
- [Rin03] Jussi Rintanen. Symmetry reduction for SAT representations of transition systems. In Enrico Giunchiglia, Nicola Muscettola, and Dana Nau, editors, *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, pages 32–40. AAAI Press, 2003.
- [Rin07] Jussi Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1045–1050. AAAI Press, 2007.
- [Rin08] Jussi Rintanen. Regression for classical and nondeterministic planning. In Malik Ghallab, Constantine D. Spyropoulos, and Nikos Fakotakis, editors, *ECAI 2008. Proceedings of the 18th European Conference on Artificial Intelligence*, pages 568–571. IOS Press, 2008.
- [Sac75] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 206–214, 1975.
- [SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 1–9, 1973.
- [SS07] Matthew Streeter and Stephen F. Smith. Using decision procedures efficiently for optimization. In Mark Boddy, Maria Fox, and Sylvie Thiébaux, editors, *ICAPS 2007. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 312–319, 2007.
- [Str00] Ofer Strichman. Tuning SAT checkers for bounded model checking. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 480–494. Springer-Verlag, 2000.
- [Str01] Ofer Strichman. Pruning techniques for the SAT-based bounded

- model checking problem. In Tiziana Margaria and Thomas F. Melham, editors, *Correct Hardware Design and Verification Methods, 11th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2001, Livingston, Scotland, UK, September 4-7, 2001, Proceedings*, volume 2144 of *Lecture Notes in Computer Science*, pages 58–70. Springer-Verlag, 2001.
- [Tur02] Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *Logics in Artificial Intelligence, European Conference, JELIA 2002*, number 2424 in *Lecture Notes in Computer Science*, pages 111–124. Springer-Verlag, 2002.
- [Val91] Antti Valmari. Stubborn sets for reduced state space generation. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1990. 10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany*, number 483 in *Lecture Notes in Computer Science*, pages 491–515. Springer-Verlag, 1991.
- [vBC99] Peter van Beek and Xinguang Chen. CPlan: A constraint programming approach to planning. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99) and the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 585–590. AAAI Press, 1999.
- [VBLN99] Thomas Vossen, Michael Ball, Amnon Lotem, and Dana Nau. On the use of integer programming models in AI planning. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 304–309. Morgan Kaufmann Publishers, 1999.
- [WW99] Steven A. Wolfman and Daniel S. Weld. The LPSAT engine & its application to resource planning. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 310–315. Morgan Kaufmann Publishers, 1999.