Rintanen, Jussi

Planning and SAT

Published in:
Handbook of Satisfiability

DOI:
10.3233/FAIA201003

Published: 05/05/2021

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Rintanen, J. (2021). Planning and SAT. In *Handbook of Satisfiability: Second Edition* (pp. 765-789). (Frontiers in Artificial Intelligence and Applications). IOS Press. https://doi.org/10.3233/FAIA201003

Chapter preprint of the second edition of the Handbook of Satisfiability (2021)
Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh (Eds.)
Published by IOS Press. The print edition can be ordered at
https://www.iospress.nl/book/handbook-of-satisfiability-2/
The published electronic chapters (Version of Record) are available at
https://ebooks.iospress.nl/ISBN/978-1-64368-161-0/

Chapter 19

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.

19.1. 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].

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≠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 reduce the planning problem to SAT.

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 size n. Finding a plan reduces to testing the satisfiability of the formulae for different values of n.

The efficiency of the SAT-based approach 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 19.3 and 19.4.
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 19.5.

Section 19.3 describes the most basic representation of planning in SAT, and the most important improvements to it to achieve efficient planning.

Section 19.4 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 action at each time point. The length parameter n restricts the number of time points but not directly the number of actions. 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 19.5 presents different strategies for making the satisfiability tests for the bounded planning problem for different lengths.

Section 19.6 describes the extension of Kautz and Selman's ideas to temporal planning, in which actions have a real or rational valued duration and multiple actions can temporally each other.

Section 19.7 discusses possibilities to reduce more general planning problems, for example with nondeterminism and sensing, to SAT and its quantified extensions.

19.2. Classical Planning

We consider planning in a setting where the states of the world are represented in terms of a set X 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 x or $\neg x$ where $x \in X$ is a state variable. We define the *complements* of literals as $\overline{x} = \neg x$ and $\overline{\neg x} = x$ for all $x \in X$. A *clause* is a disjunction $l_1 \vee \cdots \vee l_n$ of one or more literals. We also use the constant atoms $\top$ and $\bot$ for denoting

true and *false*, respectively. Each *state* $s : X \to \{0, 1\}$ assigns each state variable in X a value 0 or 1.

Actions change the state of the world.

Definition 19.2.1. An *action* over a set of state variables X is a pair $\langle p, e\rangle$ where

1. p is a propositional formula over X (*the precondition*) and
2. e is a set of pairs $f \Rightarrow d$ (*the effects*) where f is a propositional formula over X and d is a set of literals over X. (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 action $a = \langle p, e\rangle$ its *active effects* in a state s are

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

The action is *executable* in s if $s \models p$ and $[a]_s$ is consistent (does not contain both x and $\neg x$ for any $x \in X$.) If this is the case, then we define $exec_a(s)$ as the unique state that is obtained from s by making $[a]_s$ true and retaining the values of the state variables not occurring in $[a]_s$. For sequences $a_1; a_2; \ldots; a_n$ of actions we define $exec_{a_1;a_2;\ldots;a_n}(s)$ as $exec_{a_n}(\cdots exec_{a_2}(exec_{a_1}(s))\cdots)$. For sets S of actions and states s we define $exec_S(s)$ as the result of simultaneously executing all actions in S. We require that $exec_a(s)$ is defined for every $a \in S$ and that the set $[S]_s = \bigcup_{a\in S}[a]_s$ of active effects of all actions in S is consistent. For actions $a = \langle p, e\rangle$ and atomic effects l of the form x and $\neg x$ (for $x \in X$) define the *effect precondition* $EPC_l(a) = \bigvee\{f | f \Rightarrow d \in e, l \in d\}$ where the empty disjunction $\bigvee\emptyset$ is defined as $\bot$. This formula represents those states in which l is an active effect of a.

Lemma 19.2.1. *For literals l, actions a and states s, $l \in [a]_s$ if and only if $s \models EPC_l(a)$.*

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

A (sequential) *plan* for π is a sequence $\sigma = a_1; \ldots; a_n$ of actions from A such that $exec_\sigma(I) \models G$. This means that executing the actions in the given order starting in the initial state is defined (the precondition of every action is true and the active effects are consistent when the action is executed) and produces a state that satisfies the goal formula. Sometimes we say that an action sequence is a plan for A and I when we simply want to say that the plan is executable starting from I without specifying the goal states.

19.3. Sequential Plans

Consider a problem instance $\langle X, I, A, G\rangle$. We define, for any $t \geq 0$, the set $X@t = \{x@t | x \in X\}$ of propositional variables which contains a variable $x@t$ for every state variable $x \in X$. The variable $x@t$ refers to the value of x in a state

at time t. The propositional variables refer to the values of x at different integer time points. We will later use these superscripts with formulae ϕ so that $\phi@t$ is the formula obtained from ϕ by replacing every $x \in X$ by the corresponding $x@t \in X@t$.

An action $a = \langle p, e \rangle$ can be represented as

$$\tau_a = p@0 \wedge \bigwedge_{x \in X} [(EPC_x(a)@0 \vee (x@0 \wedge \neg EPC_{\neg x}(a)@0)) \leftrightarrow x@1]. \qquad (19.1)$$

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

Lemma 19.3.1. *Let s and s' be states over X. Let $s_0 : X@0 \to \{0,1\}$ be a state over $X@0$ such that $s_0(x) = s(x)$ for all $x \in X$. Let $s_1 : X@1 \to \{0,1\}$ be a state over $X@1$ such that $s_1(x') = s'(x)$ for all $x \in X$. Then $s_0 \cup s_1 \models \tau_a$ if and only if $s' = exec_a(s)$.*

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

$$\mathcal{T}(0) = \bigvee_{a \in A} \tau_a. \qquad (19.2)$$

Later we will use this formula by replacing occurrences of the propositional variables in $X@0$ and $X@1$ by propositional variables referring to different integer time points.

Lemma 19.3.2. *Let s and s' be states over X. Let $s_0 : X@0 \to \{0,1\}$ be a state over $X@0$ such that $s_0(x) = s(x)$ for all $x \in X$. Let $s_1 : X@1 \to \{0,1\}$ be a state over $X@1$ such that $s_1(x') = s'(x)$ for all $x \in X$. Then $s_0 \cup s_1 \models \tau_a$ if and only if $s' = exec_a(s)$. Then $s_0 \cup s_1 \models \mathcal{T}(0)$ if and only if $s' = exec_a(s)$ for some $a \in A$.*

Since the formula $\mathcal{T}(0)$ 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 19.3.3. *Let $\pi = \langle X, I, A, G \rangle$ be a problem instance. Let $\iota = \bigwedge\{x@0 | x \in X | s \models x\} \wedge \bigwedge\{\neg x@0 | x \in X, I \not\models x\}$. The formula*

$$\iota \wedge \mathcal{T}(0) \wedge \mathcal{T}(1) \wedge \cdots \wedge \mathcal{T}(t-1) \wedge G@t \qquad (19.3)$$

is satisfiable if and only there is a sequence $a_0, a_1, \ldots, a_{t-1}$ of t actions such that $exec_{a_{t-1}}(\cdots exec_{a_1}(exec_{a_0}(I))\cdots) \models G$.

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

19.3.1. Improvements

Thes basic representation of planning in the propositional logic can be improved many ways, and effort in implementation technologies for the underlying SAT solving can be critical for the scalability [Rin12b, Rin12a]. Next we briefly discuss two powerful ways to improve the translations of planning into the propositional logic.

19.3.1.1. 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 $x_1, \ldots, x_n$ is that at most one of them has value 1 at a time, in any reachable state. Adding formulae $\neg x_i \vee \neg x_j$ for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, n\}$ 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].

19.3.1.2. 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 19.2. 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 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]. Factoring may reduce the potential for parallelism (Section 19.4) in plans: in the move action above, the action should be factored only with respect to the location parameters y and z, as different objects can be moved in parallel, but any given object can be moved only from one location to another. Factoring can speed up the planning process substantially [RGPS09].

19.4. 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 second was proposed by Dimopoulos et al. [DNK97] and later formalized and applied to SAT-based planning [RHN06].

19.4.1. ∀-Step Plans

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

Definition 19.4.1 (∀-Step plans)**.** For a set of actions A and an initial state I, *a ∀-step plan for A and I* is a sequence $T = \langle S_0, \ldots, S_{l-1} \rangle$ of sets of actions for some $l \geq 0$ such that there is a sequence of states $s_0, \ldots, s_l$ (the execution of T) such that

1. $s_0 = I$, and
2. $exec_{a_1;\ldots;a_n}(s_i)$ is defined and equals s_{i+1} for every $i \in \{0, \ldots, l-1\}$ and every total ordering $a_1, \ldots, a_n$ of S_i.

When active effects are independent of the current state and preconditions are conjunctions of literals, Definition 19.4.1 exactly corresponds to a syntactic definition of independence [RHN06]. In more general cases the correspondence breaks down, and syntactically independent sets of actions are only a subclass of sets of actions which can be ordered to any total order.

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

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

Definition 19.4.2 (Positive and negative occurrences)**.** We say that a state variable *a occurs positively in ϕ* if positive(a, ϕ) is true. Similarly, *a occurs negatively*

in ϕ if negative(a, ϕ) is true.

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

$$\begin{aligned}
\text{negative}(x, y) &= \text{false, for all } \{x, y\} \subseteq X \\
\text{negative}(x, \phi \vee \phi') &= \text{negative}(x, \phi) \text{ or negative}(x, \phi') \\
\text{negative}(x, \neg\phi) &= \text{positive}(x, \phi)
\end{aligned}$$

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

Definition 19.4.3 (Affect)**.** Let X be a set of state variables and $a = \langle p, e\rangle$ and $a' = \langle p', e'\rangle$ actions over X. Then a *affects* a' if there is $x \in X$ such that

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

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

Definition 19.4.4 (Interference)**.** Let X be a set of state variables. Actions a and a' *interfere* if a affects a' or a' affects a.

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

19.4.2. ∃-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 19.4.5 (∃-Step plans)**.** For a set A of actions and an initial state I, a *∃-step plan* is $T = \langle S_0, \ldots, S_{l-1}\rangle \in (2^A)^l$ together with a sequence of states $s_0, \ldots, s_l$ (the execution of T) for some $l \geq 0$ such that

1. $s_0 = I$, and
2. for every $i \in \{0, \ldots, l-1\}$ there is a total ordering $a_1 < \ldots < a_n$ of S_i such that $s_{i+1} = exec_{a_1;\ldots;a_n}(s_i)$.

The difference to ∀-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 at least one order that maps state s_i to s_{i+1}. Unlike for ∀-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 $s_0, \ldots, s_l$ explicit.

The more relaxed definition of ∃-step plans sometimes allows much more parallelism than the definition of ∀-step plans.

Example 19.4.1. 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.

$$a_1 = \langle \text{out1} \wedge \text{out2} \wedge \text{empty2}, \{\top \Rightarrow \text{1in2}, \top \Rightarrow \neg\text{out1}, \top \Rightarrow \neg\text{empty2}\}\rangle$$
$$a_2 = \langle \text{out2} \wedge \text{out3} \wedge \text{empty3}, \{\top \Rightarrow \text{2in3}, \top \Rightarrow \neg\text{out2}, \top \Rightarrow \neg\text{empty3}\}\rangle$$
$$a_3 = \langle \text{out3} \wedge \text{out4} \wedge \text{empty4}, \{\top \Rightarrow \text{3in4}, \top \Rightarrow \neg\text{out3}, \top \Rightarrow \neg\text{empty4}\}\rangle$$

The shortest $\forall$-step plan nests the dolls in three steps: $\langle\{a_1\}, \{a_2\}, \{a_3\}\rangle$. The $\exists$-step plan $\langle\{a_1, a_2, a_3\}\rangle$ nests the dolls in one step.

Theorem 19.4.1. *(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 actions are required to be executable in the current state (unlike for $\forall$-step plans this does not follow from the definition), and parallel actions are required to fulfil an ordered dependence condition based on Definition 19.4.3: actions are allowed in parallel if they can be totally ordered so that no action affects a later action. This means that the actions preceding a given action do not change its effects or prevent its execution.

Theorem 19.4.2. *Let A be a set of actions, I a state, $T = \langle S_0, \ldots, S_{l-1}\rangle \in (2^A)^l$, and $s_0, \ldots, s_l$ a sequence of states. If*

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

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

Even though the class of $\exists$-step plans based on *affects* is narrower than the class sanctioned by Definition 19.4.5, much more parallelism is still possible in comparison to $\forall$-step plans. For instance, nesting of Russian dolls in Example 19.4.1 belongs to this class.

19.4.3. Optimality of Parallel Plans

The most commonly used measure for plan quality is the number of actions 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 in the worst case 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 may require 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.

19.4.4. 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 actions 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 actions is consistent.

19.4.4.1. The Base Translation

Planning is performed by propositional satisfiability testing by producing formulae $\phi_0, \phi_1, \phi_2, \ldots$ 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 actions. Next we describe the part of the formulae that is shared by both definitions of parallel plans.

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

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

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

$$a@t \rightarrow p@t \tag{19.4}$$

Then we define, for every $x \in X$, the condition under which x becomes true or false at step t

$$causes(x)@t = \bigvee_{a \in A} (a@(t-1) \wedge EPC_x(a)@(t-1)) \tag{19.5}$$

$$causes(\neg x)@t = \bigvee_{a \in A} (a@(t-1) \wedge EPC_{\neg x}(a)@(t-1)) \tag{19.6}$$

These formulas are used in defining when a state variable becomes true or false at a given time point.

$$causes(x)@t \rightarrow x@t \tag{19.7}$$

$$causes(\neg x)@t \rightarrow \neg x@t \tag{19.8}$$

Second, the value of a state variable does not change if no action that changes it is executed. Hence for every state variable x we have two formulae, one expressing the conditions for the change of x from true to false,

$$(x@t \wedge \neg x@(t+1)) \rightarrow causes(\neg x)@(t+1) \tag{19.9}$$

and another from false to true,

$$(\neg x@t \wedge x@(t+1)) \rightarrow causes(x)@(t+1). \tag{19.10}$$

The formulae $\Phi_{\pi,l}$, just like the definition of $exec_S(s)$, allow sets of actions in parallel that do not correspond to any sequential plan. For example, the actions $\langle x, \{\top \Rightarrow \neg y\}\rangle$ and $\langle y, \{\top \Rightarrow \neg x\}\rangle$ may be executed simultaneously resulting in a state satisfying $\neg x \wedge \neg y$, even though this state is not reachable by the two actions 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 19.4.3. *Let $\pi = \langle X, I, A, G\rangle$ be a transition system. Then there is $T = \langle S_0, \ldots, S_{l-1}\rangle \in (2^A)^l$ so that $s_0, \ldots, s_l$ are states so that $I = s_0$, $s_l \models G$, and $s_{i+1} = exec_{S_i}(s_i)$ for all $i \in \{0, \ldots, l-1\}$ if and only if there is a valuation satisfying the formula $\Phi_{\pi,l}$.*

Proposition 19.4.4. *The size of the formula $\Phi_{\pi,l}$ is linear in l and in the size of π.*

Theorem 19.4.3 says that a sequence of actions fulfilling certain conditions exists if and only if a given formula is satisfiable. The theorems connecting certain formulae to certain notions of plans (Theorems 19.4.5 and 19.4.6) 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 $\langle S_0, \ldots, S_{l-1}\rangle$ with $n = |S_0| + \cdots + |S_{l-1}|$ occurrences of actions exists, there is a plan consisting of n singleton sets, and the corresponding formulae $\Phi_{\pi,n} \wedge \Phi^x_{A,n}$ are satisfiable. The formulae $\Phi^z_{A,n}$ represent the parallel semantics z for actions A.

19.4.4.2. Translation of $\forall$-Step Plans

The simplest representation of the interference condition in Definition 19.4.4 is by formulae

$$\neg(a_1@t \wedge a_2@t) \tag{19.11}$$

for every pair of interfering actions a_1 and a_2. Define $\Phi^{\forall step}_{A,l}$ as the conjunction of the formulae (19.11) for all time points $t \in \{0, \ldots, l-1\}$ and for all pairs of interfering actions $\{a, a'\} \subseteq A$ that could be executed simultaneously. There are $\mathcal{O}(ln^2)$ such formulae for n actions. This can be improved to linear [RHN06].

Theorem 19.4.5. *Let $\pi = \langle X, I, A, G\rangle$ be a transition system. There is a $\forall$-step plan of length l for π if $\Phi_{\pi,l} \wedge \Phi^{\forall step}_{A,l}$ is satisfiable.*

19.4.4.3. Translation of ∃-Step Plans

The simplest efficient representation of ∃-step plans imposes a fixed total ordering on the actions and allows the simultaneous execution of a subset of the actions only if none of the actions affects any action later in the ordering. There are more permissive ways of guaranteeing the validity of ∃-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 actions [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 ∃-step plans the set of formulae constraining parallel execution *is a subset* of those for the less permissive ∀-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 actions $a_1, \ldots, a_n$ and to allow parallel execution of two actions a_i and a_j such that a_i affects a_j only if $i \geq j$. The formula $\Phi_{A,l}^{\exists step}$ is the conjunction of

$$\neg(a_i@t \wedge a_j@t) \text{ if } a_i \text{ affects } a_j \text{ and } i < j$$

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

The formula $\Phi_{A,l}^{\exists step}$ (similar to the translation for ∀-step plans in Section 19.4.4.2) has a quadratic size because of the worst-case quadratic number of pairs of actions that may not be simultaneously executed. This can be improved to linear [RHN06].

Theorem 19.4.6. *Let $\pi = \langle X, I, A, G\rangle$ be a transition system. There is a ∃-step plan of length l for π if $\Phi_{\pi,l} \wedge \Phi_{A,l}^{\exists step}$ is satisfiable.*

19.5. Finding a Satisfiable Formula

Given a sequence $\phi_0, \phi_1, \ldots$ 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 19.3.

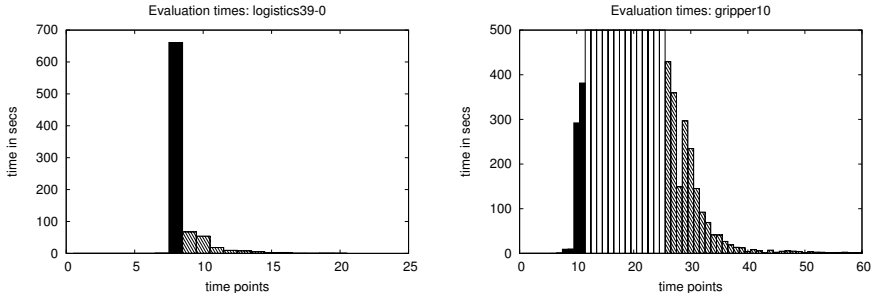

Figure 19.1. SAT solver runtimes for two problem instances and different plan lengths

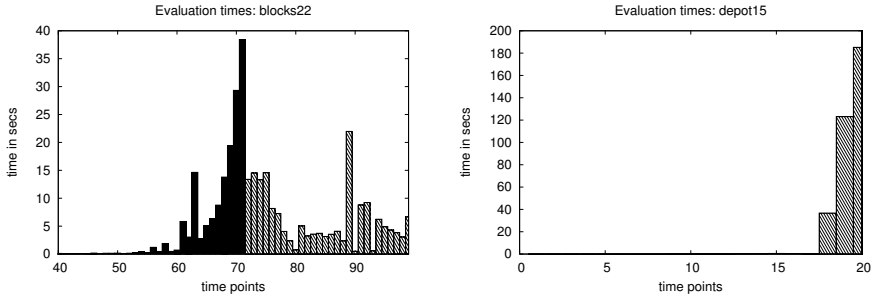

Figure 19.2. SAT solver runtimes for two problem instances and different plan lengths

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 19.1, 19.2 and Figure 19.3. 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 19.2 and the leftmost diagram in Figure 19.3, 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 $i \in \{0, 1, 2, \ldots\}$ 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

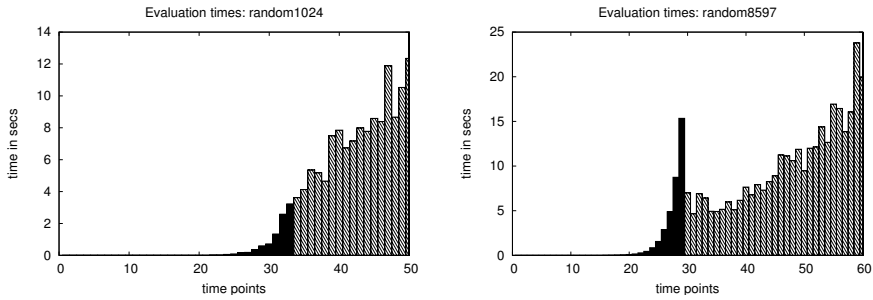

Figure 19.3. SAT solver runtimes for two problem instances and different plan lengths

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

Figure 19.4. 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.

19.5.1. 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 19.4.

19.5.2. Algorithm A: Multiple Processes

Algorithm A (Figure 19.5) 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 := {φ_0, ..., φ_{n-1}};
3:  next := n;
4:  repeat
5:     P' := P;
6:     for each φ ∈ P' do
7:        continue computation with φ for ε seconds;
8:        if φ was found satisfiable then terminate;
9:        if φ was found unsatisfiable then
10:          P := P ∪ {φ_next}\{φ};
11:          next := next + 1;
12:       end if
13:    end do
14: until 0=1
```

Figure 19.5. Algorithm A

```
1:  procedure AlgorithmB(γ)
2:  t := 0;
3:  for each i ≥ 0 do done[i] = false;
4:  for each i ≥ 0 do time[i] = 0;
5:  repeat
6:     t := t + δ;
7:     for each i ≥ 0 such that done[i] = false do
8:        if time[i] + nε ≤ tγ^i for some maximal n ≥ 1 then
9:           continue computation for φ_i for nε seconds;
10:          if φ_i was found satisfiable then terminate;
11:          time[i] := time[i] + nε;
12:          if φ_i was found unsatisfiable then done[i] := true; end if
13:       end if
14:    end do
15: until 0=1
```

Figure 19.6. Algorithm B

19.5.3. Algorithm B: Geometric Division of CPU Use

Algorithm B (Figure 19.6) 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 $\phi_i, i \geq k$ 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-

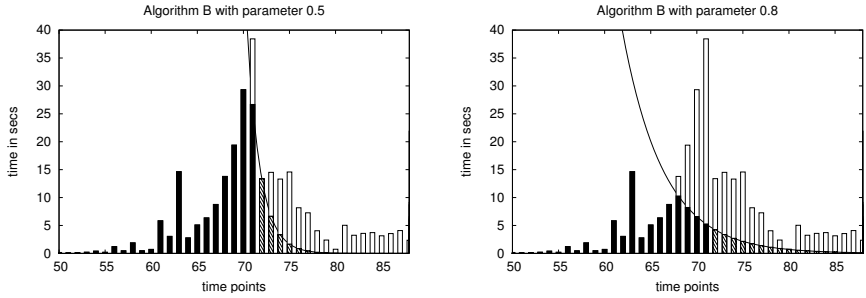

Figure 19.7. Illustration of two runs of Algorithm B. When $\gamma = 0.5$ most CPU time is spent on the first formulae, and when the first satisfiable formula is detected also the unsatisfiability of most of the preceding unsatisfiable formulae has been detected. With $\gamma = 0.8$ more CPU is spent 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.

19.6. Temporal Planning

The SAT modulo Theories (SMT) [WW99, ABC+02] framework, which supports numerical and other constraints in addition to Boolean ones, can be applied to the solution of more general planning problems.

The first application of SAT extended with numerical constraints was by Wolfman and Weld [WW99] to the classical planning problem with numerical state variables. The notion of plans and executions are the same as in classical planning. The difference is that in addition Boolean state variables, there are numerical state variables which can be assigned values expressed as arithmetic expressions.

Shin and Davis gave a systematic translation of an expressive planning language into the SMT framework [SD05]. This is a major generalization of the representations of classical planning as propositional formulas. Instead of a sequence of states, with a number of actions taking between any two consecutive states, in temporal planning there can be multiple actions temporally overlapping each other in different ways, and the execution of one action can span multiple steps of the plan. This framework can be applied to a range of modelling languages [Rin17], and as the number of steps of a plan is bounded similarly to classical planning, the same solving strategies (Section 19.5) are applicable.

Here we sketch the main ideas of the type of encodings first proposed by Shin and Davis [SD05] which have later been used for solving planning problems by using SMT solvers.

We use a simple formalization of temporal planning, with a set X of state variables and actions as triples (p, d, e), where p is a propositional formula over X just like in classical planning, and $d > 0$ is a rational *duration* of the action, and e is a set of literals over X made true by the action. The precondition p has to be true at the time point in which the action is taken, and the effects e become true after time d.

Unlike in classical planning, timelines in temporal planning are continuous, either rational or real valued, but it is still sufficient to represent explicitly only a finite sequence of time points in which an action is taken or an effect of an action takes place, just like in classical planning.

As before, l denotes the index of the last step so that the $l + 1$ steps of a plan are indexed from 0 to l. We use the same atomic propositions $a@i$ and $x@i$ to indicate whether an action is taken or a state variable is true at a given step i. Additionally, we associate an absolute time $\tau@i$ with each step. The numeric inequalities

$$\tau@i > \tau@j \tag{19.12}$$

for all i and j such that $l \geq j > i > 0$, forces the absolute time to be monotonically increasing. Next we express the properties of actions and state variables at different steps of a plan.

We have the formula

$$a@i \rightarrow p@i \tag{19.13}$$

for the precondition p of action a, for every $i \in \{0, \ldots, l-1\}$.

For an action $a = (p, d, e)$, define $EPC_l(a) = \top$ iff $l \in e$ and otherwise $EPC_l(a) = \bot$. Like earlier, we define $causes(l)@i$ to denote the conditions under which literal l becomes true at step i. For temporal planning this can refer to multiple earlier steps, as the last step might not be the one in which an action affecting l took place.

$$causes(x)@i = \bigvee_{a \in A} \bigvee_{j=0}^{i-1} (a@j \wedge EPC_x(a)@j \wedge (\tau@i - \tau@j = d_a)) \tag{19.14}$$

we denote the disjunction of all the conditions under which x becomes true at step $i \in \{0, \ldots, l\}$. These formulas refer to atomic propositions for earlier steps. Similarly $causes(\neg x)@i$ for x becoming false. Hence when $causes(l)@i$ is true, the effect l takes place.

$$causes(x)@i \rightarrow x@i \tag{19.15}$$

$$causes(\neg x)@i \rightarrow \neg x@i \tag{19.16}$$

Frame axioms allow inferring that the value of a state variable remains unchanged.

$$(x@i \wedge \neg x@(i-1)) \rightarrow causes(x)@i \tag{19.17}$$

$$(\neg x@i \wedge x@(i-1)) \rightarrow causes(\neg x)@i \tag{19.18}$$

When $t > 0$, to guarantee that there is a step for every effect, we need

$$a@i \rightarrow \bigvee_{j=i+1}^{l} (\tau@j - \tau@i = t). \tag{19.19}$$

Most temporal planning problems restrict the possibilities to overlap two actions in various ways. Assume that no overlap for actions a and a' is allowed, and these actions respectively have durations d and d'. Now, if a is taken at t, then a' cannot be started before time $t + d$, and if a' is taken at t, then a cannot be started before time $t + d'$, expressed by

$$a@i \rightarrow \bigwedge_{j=i}^{l} (a'@j \rightarrow (\tau@j - \tau@i \geq d)) \tag{19.20}$$

$$a'@i \rightarrow \bigwedge_{j=i}^{l} (a@j \rightarrow (\tau@j - \tau@i \geq d')) \tag{19.21}$$

For different limits on the concurrency of actions see the literature [SD05, Rin17].

Temporal planning can be reduced to plain SAT [RGS15], but as numerical durations are not represented, a satisfying assignment is not guaranteed to correspond to a valid plan, and multiple calls, with tigther Boolean constraints, may be required to find one. Also, minimization of the durations of plans is harder.

19.7. Contingent Planning

Classical planning and temporal planning assume the world to be completely predictable. Nondeterminism and unpredictability, as well as limited observability, complicate planning and reduce the possibilities of effective reductions to SAT.

With a polynomial bound on the number of actions in a plan, the complexity of classical planning decreases from PSPACE-complete to NP-complete. More general planning problems have a complexity far exceeding PSPACE: planning with nondeterministic actions and full observability is EXP-complete [Lit97], with no observability EXPSPACE-complete [HJ00], and with partial observability 2-EXP-complete [Rin04]. Merely bounding plans to polynomial size will not decrease the complexity to NP. There are two options: either use a formalism harder than NP, or impose further restrictions. Both options have been considered.

In planning with full or partial observability, a plan may have to handle different contingencies in order to achieve its objectives (which is why these problems are sometimes called contingent planning.) Plans can no longer be limited to sequences of actions, as different observations made during the execution may lead to alternative courses of action, represented for example as a branching plan, or a mapping from observations (and the execution state) to actions.

If both plan sizes and executions are restricted to be polynomial, the planning problem is in Σ_2^p [Rin99]. Problems belonging to this complexity class correspond to quantified Boolean formulas (QBF) [SM73] with the prefix $\exists\forall$ (Section 19.7.1).

Even stricter restrictions can reduce the complexity to NP. Chatterjee et al. [CCD16] have shown that planning with partial observability, when the state space is represented simply as a graph with states as nodes, is NP-complete, and can therefore be reduced to SAT (Section 19.7.2).

For problems with very large state spaces, the explicit representation of all states is not feasible. Geffner and Geffner [GG18] have shown that, for a relatively broad class of contingent planning problems with full observability and actions

and states represented in terms of state variables, reduction to SAT can still be a competitive solution method (Section 19.7.3).

19.7.1. Contingent Planning in Σ_2^p with QBF

In this section we discuss the use of quantified Boolean formulas for solving simple contingent planning problems with a polynomial length bound, and more expressive contingent planning problems with a size bound and other, more restrictive conditions to bring the complexity down to NP.

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].

The uncertainty about the initial state as well as the nondeterminism of actions and the environment can be represented as state variables that can assume arbitrary values and different ones at different steps of the executions. We call these state variables the *contingency variables*.

Actions cannot change the values of the contingency variables, but the actions can otherwise depend on them: contingency variables can occur in the actions' preconditions and the condition parts of the effects, thereby affecting the applicability of actions and the effects of actions.

We define a contingent planning problem as a tuple $\langle X, C, I, A, G\rangle$, where X is the set of all state variables, $C \subseteq X$ is the contingency variables, I is a formula for the initial states, A is the set of actions, and G is the formula for the goal states. Here X, A, and G are like for classical in Section 19.2. The contingency variables and the definition of I as a formula are where these definitions diverge.

Next we define a quantified Boolean formula that represents the problem of finding a sequence of actions that leads from any of the initial state to any of the goal states, under any possible combination of contingencies.

Define $P = \{a@i | a \in A, 0 \leq i \leq l\}$ for the set of atomic propositions representing the action and thereby the plan. Define $Z = \{x@i | x \in C, 0 \leq i \leq l-1\} \cup \{x@0 | x \in X \backslash C\}$ for the values of the contingency variables at all points of the execution, and also the values of all other state variables at the first step of the execution (the initial state). Define $E = \{x@i | x \in X, 1 \leq i \leq l\}$ for the values of non-contingency state variables at all steps of the execution after the initial state. The values of these state variables are determined unique by the action variables and the values of the contingency variables on the previous steps.

Now the translation of the planning problem into QBF is as follows.

$$\exists P \forall Z \exists E\, (I@0 \rightarrow (\mathcal{T}(0) \wedge \cdots \wedge \mathcal{T}(l-1) \wedge G@l)) \tag{19.22}$$

The formula says that there is a plan (expressed by P) such that for all combinations of contingencies (expressed by Z), there is an execution (expressed by E) that leads to a state that satisfies the goal formula G.

This QBF evaluates to *true* if and only such a plan exists. For a true QBF the valuation of the outermost variables P indicates what the plan is.

The above formula representing planning *without* observability: a single sequence of actions has to reach the goals independently of the contingencies. In more general cases, observations made during the execution can determine how the plan execution is continued. This corresponds to plans with branching, and the variables P above could represent such a plan, together with a more complex body for the QBF to express how such a branching plan would determine which actions are taken during the execution [Rin99].

The QBF-based representation can be generalized also in other ways. A probabilistic extension of QBF, stochastic satisfiability or SSAT, makes it possible to express transition probabilities associated with nondeterministic actions and noisy observations [ML03].

19.7.2. Planning with Partial Observability in NP

Chatterjee et al. [CCD16] observe that contingent planning with partial observability is in NP if limiting the memory available during the execution, and considering an enumerative – rather than a state variable based – representation of the state space. A graph represents all possible executions of all possible plans. A given plan selects a subset of the arcs of the graph, and the plan must be chosen so that the goals are reached on all execution of the given plan.

Despite the explicit representation of the state space as a graph, the problem is still NP-hard because the actions (outgoing arcs) for each state cannot be chosen independently of other states, and, instead, all states for which the observation (and possibly memory) are the same, must have the same action. This problem of choosing a suitable action for every observation so that goals are reached, makes the problem harder than the corresponding fully observable problem which could be solved in polynomial time.

First we formalize the planning problem.

Definition 19.7.1. A transition system $P = (S, A, O, \delta, Z, I, G)$ consists of

- S is a finite set of states,
- A is a finite set of actions,
- O is a finite set of observations,
- $\delta \subseteq S \times A \times S$ is the transition relation,
- $Z : S \to O$ is the observation function,
- $I \subseteq S$ is the set of initial states, and
- $G \subseteq S$ is the set of goal states.

The transition relation $\delta(s, a's')$ expresses whether a transition from s to s' is possible with action a. The function Z associates an observation $Z(s) \in O$ with every state s.

Plans associate an action with every observation. Executions under a given plan include an arc from s to s' if the observation $Z(s)$ in s is associated with

the action a, and $(s, a, s') \in \delta$. A valid plan is one that guarantees that for any state reachable from an initial state, it is always possible to reach one of the goal states.

We will next describe the reduction of this problem to SAT. A core part of it is – similarly to the problem considered in Section 19.7.3 – the representation of constraints expressing the constraint that a given node is (or is not) reachable from another node. For simplicity, we only correspond the memory-free case: generalization to multiple memory states is obtained with a straightforward extension of the formulas given here [CCD16].

The encodings rely on implicitly representing all possible plans and all possible execution graphs. The choice of a plan – the choice of actions and memory updates for all observation and memory state combinations – induces one particular execution graph. A plan is determined by a truth-value assignment to variables π below, and the remaining variables represent the corresponding execution graph (through the variables Arc) and its properties.

A number of propositional variables are defined to encode contingent planning problem with bounded memory into SAT. The first two represent the functions π of bounded-memory plans.

- $\pi(o, a)$ means that action a is taken on observation o.
- $Arc(s, s')$ means that there is a transition from state s to state s', for all $(s, s') \in Y$ where $Y = \{(s, s') \in S \times S | (s, a, s') \in \delta \text{ for some } a \in A\}$.
- $R^I(s)$ means that s is reachable from an initial node under the current plan.
- $R^G(s)$ means that a goal node is reachable from s under the current plan.

The arc variables determine whether a given arc (s, s') is considered to be present in the graph representing all executions of a given plan expressed by the valuation of the variables $\pi(o, a)$.

The value of $\pi(o, a)$ has to be unique for all o and a:

$$\neg(\pi(o, a_1) \wedge \pi(o, a_2)) \tag{19.23}$$

for all $o \in O, a_1 \in A, a_2 \in A$ such that $a_1 \neq a_2$.

The function π has to be defined for states that are reachable: for all $s \in S$, if s is reachable from an initial state, then at least one of the actions executable in s should be enabled in s and hence part of the plan.

$$R^I(s) \rightarrow \bigvee_{a \in X(s)} \pi(Z(s), a) \tag{19.24}$$

Here, $X(s) = \{a \in A | (s, a, s') \in \delta, \text{ for some } s'\}$. Whether an arc is included in the graph of possible executions is determined as follows.

$$Arc(s, s') \leftrightarrow \left(\bigvee_{a \in Y(s,s')} \pi(Z(s), a) \right) \tag{19.25}$$

Here, $Y(s, s') = \{a \in A | (s, a, s') \in \delta\}$. Finally, it is required that a goal state is reachable from any state that is reachable from an initial state.

$$R^I(s) \rightarrow R^G(s) \tag{19.26}$$

There are alternative ways of encoding these reachability constraints, including a set of clauses and using a specialized graph constraint for reachability or for acyclicity, and as these constraints dominate the encoding, they can have a strong impact on the scalability of the approach [PR18].

19.7.3. Planning with Full Observability in NP

Earlier works had suggested that contingent planning in general either requires a formalism that is harder than NP, or must be dramatically limited in terms of the size of the plans, the length of the executions, and the compactness of the transition system representation.

However, Geffner and Geffner [GG18] demonstrated that for a class of contingent planning problems with full observability, plans and their executions can be represented as a propositional formula even with the actions and states represented compactly in terms of state variables.

In this formula, each node of a plan is associated with atomic propositions that describe the relevant features of the state when the execution is in that node. This allows each satisfying assignment of the formula to represent both a branching plan and all of its relevant executions, thereby bringing the complexity down to NP and enabling reductions to SAT.

This approach resembles the preceding one (Section 19.7.2) in identifying executions with a graph that is represened in the satisfying assignments of the formula, and by relying on graph constraints to express the conditions for the validity of a plan.

The approach seems to be limited by the requirement that all states in a given node of the plan have to agree on the values of some of the state variables, but many practical problems are still effectively representable in this way.

References

[ABC+02] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A SAT based approach for solving formulas over Boolean and linear mathematical propositions. In A. Voronkov, editor, *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings*, number 2392 in Lecture Notes in Computer Science, pages 195–210. Springer-Verlag, 2002.

[ABH+97] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. 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] A. Biere, A. Cimatti, E. M. Clarke, and Y. 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] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[BR05] M. Büttner and J. Rintanen. Satisfiability planning with constraints on the number of actions. In S. Biundo, K. Myers, and K. 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] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[CCD16] K. Chatterjee, M. Chmelik, and J. Davies. A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 3225–3232. AAAI Press, 2016.

[DK01] M. B. Do and S. Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182, 2001.

[DNK97] Y. Dimopoulos, B. Nebel, and J. 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] M. Ernst, T. Millstein, and D. S. Weld. Automatic SAT-compilation of planning problems. In M. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1169–1176. Morgan Kaufmann Publishers, 1997.

[GG18] T. Geffner and H. Geffner. Compact policies for non-deterministic fully observable planning as sat. In *ICAPS 2018. Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling*, pages 88–96. AAAI Press, 2018.

[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 Verific ation (CAV '90), Rutgers, New Jersey, 1990*, number 531 in Lecture Notes in Computer Science, pages 176–185. Springer-Verlag, 1991.

[GS98] A. Gerevini and L. 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] P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In S. Biundo and M. 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.

[KMS96] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*, pages 374–385. Morgan Kaufmann Publishers, 1996.

[KS92] H. Kautz and B. Selman. Planning as satisfiability. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363. John Wiley & Sons, 1992.

[KS96] H. Kautz and B. 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] H. Kautz and J. 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.

[Lit97] M. L. Littman. Probabilistic propositional planning: Representations and complexity. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pages 748–754. AAAI Press, 1997.

[ML03] S. M. Majercik and M. L. Littman. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, 147(1-2):119–162, 2003.

[MR91] D. A. McAllester and D. 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.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[PR18] B. Pandey and J. Rintanen. Planning for partial observability by SAT and graph constraints. In *ICAPS 2018. Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling*, pages 190–198. AAAI Press, 2018.

[RGPS09] N. Robinson, C. Gretton, D.-N. Pham, and A. Sattar. SAT-based parallel planning using a split representation of actions. In *ICAPS 2009. Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, pages 281–288. AAAI Press, 2009.

[RGS15] M. F. Rankooh and G. Ghassem-Sani. ITSAT: an efficient SAT-based temporal planner. *Journal of Artificial Intelligence Research*, 53:541–632, 2015.

[RHN06] J. Rintanen, K. Heljanko, and I. Niemelä. Planning as satisfiability:

parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

[Rin98] J. 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] J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[Rin04] J. Rintanen. Complexity of planning with partial observability. In *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 345–354. AAAI Press, 2004.

[Rin07] J. 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] J. Rintanen. Regression for classical and nondeterministic planning. In M. Ghallab, C. D. Spyropoulos, and N. Fakotakis, editors, *ECAI 2008. Proceedings of the 18th European Conference on Artificial Intelligence*, pages 568–571. IOS Press, 2008.

[Rin12a] J. Rintanen. Engineering efficient planners with SAT. In *ECAI 2012. Proceedings of the 20th European Conference on Artificial Intelligence*, pages 684–689. IOS Press, 2012.

[Rin12b] J. Rintanen. Planning as satisfiability: heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[Rin17] J. Rintanen. Temporal planning with clock-based SMT encodings. In *IJCAI 2017, Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 743–749. AAAI Press, 2017.

[Sac75] E. D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 206–214, 1975.

[SD05] J.-A. Shin and E. Davis. Processes and continuous change in a SAT-based planner. *Artificial Intelligence*, 166(1):194–253, 2005.

[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.

[Tur02] H. 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] A. Valmari. Stubborn sets for reduced state space generation. In G. 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] P. van Beek and X. Chen. CPlan: A constraint programming approach to planning. In *Proceedings of the 16th National Conference on Arti-*

ficial Intelligence (AAAI-99) and the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99), pages 585–590. AAAI Press, 1999.

[VBLN99] T. Vossen, M. Ball, A. Lotem, and D. Nau. On the use of integer programming models in AI planning. In T. Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 304–309. Morgan Kaufmann Publishers, 1999.

[WW99] S. A. Wolfman and D. S. Weld. The LPSAT engine & its application to resource planning. In T. Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 310–315. Morgan Kaufmann Publishers, 1999.