CHAPTER 3

Complexity of Classical Planning

3.1 Introduction

Much planning research has been motivated by the difficulty of producing complete and correct plans. We now examine how the decidability and complexity of classical planning varies depending on a variety of conditions. The results in this chapter can be summarized as follows.

1. For classical planning, plan existence is *decidable*, i.e., it is possible to write an algorithm that takes as input a statement $P = (A, s_0, g)$ of a planning problem $\mathcal{P}$ and returns yes if $\mathcal{P}$ is solvable and no otherwise. This is independent of whether the statement of the problem uses the set-theoretic, classical, or state-variable representations.
2. There are a number of syntactic restrictions we can place on the planning operators to reduce the complexity of the planning problem. The computational complexity of classical planning varies from constant time to EXPSPACE-complete,[1] depending on which representation we use and which restrictions we make.
3. The classical and state-variable representation schemes can be extended to allow the terms to contain function symbols (e.g., see Section 2.4.6). Such an extension puts them outside of the set of restricted state-transition systems discussed in Chapter 2 because it means that the set of possible states can be infinite. In this case, plan existence is *semidecidable*. It is possible to write a procedure that always terminates and returns yes if $\mathcal{P}$ is solvable and never returns yes if $\mathcal{P}$ is unsolvable. However, if $\mathcal{P}$ is unsolvable, then there is no guarantee that the procedure will terminate. It remains semidecidable even if we make several restrictions on the planning operators.

1. See Appendix A.

4. All of these results are independent of whether or not we extend the planning operators to have conditional effects.

In this chapter, Section 3.3 discusses the decidability and undecidability results, Section 3.4 discusses the complexity results, Section 3.5 discusses the significance and limitations of the results, and Section 3.6 contains discussion and historical remarks. The results in this chapter are based largely on work by Bylander [100], Erol *et al.*[176], and Bäckström and Nebel [40].

3.2 Preliminaries

In order to analyze the decidability and complexity of planning problems, they must be reformulated as language-recognition problems. Appendix A discusses the details of how to do this. Given a set D of statements of planning problems, we let:

- PLAN-EXISTENCE(D) be the set of all statements $P \in D$ such that P represents a solvable planning problem; and
- PLAN-LENGTH(D) be the set of all statements $P \in D$ such that P represents a planning problem for which there is a solution that contains no more than k actions.

For example, D may be the set of all classical planning problems, or the set of all planning problems that can be obtained by extending the representation scheme to include function symbols, or the set of all classical problems in which none of the operators have negative effects, etc. When it is obvious what D is, we will omit it and just write PLAN-EXISTENCE or PLAN-LENGTH instead.

In our theorems and proofs, we will concentrate primarily on classical planning (i.e., on planning problems expressed using the classical representation scheme). However, in most cases this is sufficient to establish results for set-theoretic and state-variable planning as well because of the equivalences among them described in Section 2.6. Classical planning and state-variable planning are essentially equivalent; and set-theoretic planning, ground classical planning, and ground state-variable planning are all essentially equivalent.

We also consider a variety of syntactic extensions and restrictions on classical planning, such as allowing the terms to contain function symbols (see Section 2.4.6), not allowing the operators to have negative preconditions, not allowing them to have negative effects, and so forth. These extensions and restrictions are also applicable to set-theoretic planning, with one exception: it makes no sense to allow function symbols in set-theoretic planning because the latter does not have terms.

With one exception, equivalent extensions and restrictions can also be made for state-variable planning. As examples, both classical planning and state-variable planning can be extended to allow function symbols, and requiring classical planning operators to have no negative preconditions is equivalent to requiring state-variable planning operators to have no inequality preconditions.

The exception is that although we can restrict classical planning operators from having negative effects, there is no sensible way to accomplish the same thing in state-variable planning. If $x(v_1, \ldots, v_n) = c$ and a state-variable planning operator assigns $x(v_1, \ldots, v_n) \leftarrow d$, this automatically removes the old value, c.

Whenever the extensions and restrictions can sensibly be applied to set-theoretic and state-variable planning, our results for classical planning can be generalized to include them as well. We will point this out as appropriate.

3.3 Decidability and Undecidability Results

The decidability and undecidability results are summarized in Table 3.1. For ordinary classical planning, PLAN-EXISTENCE and PLAN-LENGTH are decidable. The same is true for both set-theoretic and state-variable planning. If we extend classical or state-variable planning to allow the terms to contain function symbols, then PLAN-LENGTH remains decidable, but PLAN-EXISTENCE becomes semidecidable. The latter is true even with several syntactic restrictions on the planning operators.

Table 3.1 Decidability of PLAN-EXISTENCE and PLAN-LENGTH.

Allow function symbols?	*Decidability of* PLAN-EXISTENCE	*Decidability of* PLAN-LENGTH
No[a]	Decidable	Decidable
Yes	Semidecidable[b]	Decidable

[a] This is ordinary classical planning.
[b] True even if we make several restrictions (see text).

PLAN-EXISTENCE becomes semidecidable. The latter is true even with several syntactic restrictions on the planning operators.

Proposition 3.1 *For classical, set-theoretic, and state-variable planning,* PLAN-EXISTENCE *is decidable.*

The above proposition is not hard to prove. For a classical planning problem, the number of possible states is finite, so it is possible to do a brute-force search to see whether a solution exists. The same is true of set-theoretic and state-variable planning.

Proposition 3.2 *If we extend classical planning or state-variable planning to allow the terms to contain function symbols, then* PLAN-LENGTH *is still decidable.*

Proof We present the proof only for classical planning; it is straightforward to generalize it to state-variable planning.

Let $P = (O, s_0, g)$ be the statement of a classical planning problem and k be a nonnegative integer. We can modify the Lifted-backward-search procedure of

Chapter 4 to exit with failure every time it reaches a plan of length k that is not a solution. It is not hard to see that this procedure is sound and is guaranteed to terminate. If we can also show that the procedure is complete, this will be sufficient to show that PLAN-LENGTH is decidable.

To show that the procedure is complete, let π be any solution for P of length k or less. If $|\pi| = 0$, then π is empty, so our procedure terminates immediately. Otherwise, let a_1 be the last action in π. Then a_1 is relevant for g, so it must be a substitution instance of some operator o_1 chosen in one of our procedure's nondeterministic traces. Thus, $\gamma^{-1}(g, a_1)$ is a substitution instance of $\gamma^{-1}(g, o_1)$. If $|\pi| = 1$, then s_0 satisfies $\gamma^{-1}(g, a_1)$, so it also satisfies $\gamma^{-1}(g, o_1)$, so the procedure terminates. Otherwise, let a_2 be the second-to-last action in π. Then a_2 is relevant for $\gamma^{-1}(g, a_1)$, so it must be a substitution instance of some operator o_2 chosen by one of our procedure's nondeterministic traces. Continuing in this manner, we can show that some execution trace of our procedure will terminate in $|\pi|$ iterations. ■

Proposition 3.3 *If we extend classical planning or state-variable planning to allow the terms to contain function symbols, then* PLAN-EXISTENCE *is semidecidable.*

Proof As before, we will present the proof just for classical planning.

Let $P = (O, s_0, g)$ be a statement of a planning problem. PLAN-EXISTENCE is no worse than semidecidable because **Lifted-backward-search**(P) will terminate if a solution exists for P.

To show that PLAN-EXISTENCE is not decidable, let $P = (O, s_0, g)$ be a statement of a planning problem such that each operator of O has no negative preconditions, no negative effects, and at most one positive effect. Then we can define a set of Horn clauses H_P as follows. Each atom $a \in s_0$ is also in H_P. Corresponding to the goal g, H_P includes the following Horn clause:

$$:\text{-}\ g$$

If $o \in O$ is an operator, there are atoms $p_1, \ldots, p_n, e$ such that $\text{precond}(o) = \{p_1, \ldots, p_n\}$ and $\text{effects}(o) = \{e\}$. Corresponding to this operator, H_P includes the following Horn clause:

$$e :\text{-}\ p_1, \ldots, p_n$$

It is straightforward to show that P and H_P are equivalent in the sense that P has a solution plan iff H_P is consistent. It is a well-known result of logic programming that it is not decidable whether H_P is consistent; thus it follows that it is not decidable whether P has a solution. ■

The same result holds regardless of whether or not the planning operators have negative preconditions, negative effects, more than one precondition, or conditional effects. The result is true even if the planning operators are fixed in advance (to

enable the use of a domain-specific planning algorithm) or given as part of the input.

3.4 Complexity Results

As summarized in Table 3.2, the computational complexity for classical planning problems (and thus for most set-theoretic and state-variable planning problems)

Table 3.2 Complexity of classical planning.

Kind of representation	*How the operators are given*	*Allow negative effects?*	*Allow negative preconditions?*	*Complexity of* PLAN-EXISTENCE	*Complexity of* PLAN-LENGTH
Classical rep.	In the input	Yes	Yes/no	EXPSPACE-complete	NEXPTIME-complete
		No	Yes	NEXPTIME-complete	NEXPTIME-complete
			No	EXPTIME-complete	NEXPTIME-complete
			No[a]	PSPACE-complete	PSPACE-complete
	In advance	Yes	Yes/no	PSPACE [b]	PSPACE [b]
		No	Yes	NP [b]	NP [b]
			No	P	NP [b]
			No[a]	NLOGSPACE	NP
Set-theoretic or ground classical rep.	In the input	Yes	Yes/no	PSPACE-complete	PSPACE-complete
		No	Yes	NP-complete	NP-complete
			No	P	NP-complete
			No[a]/no[c]	NLOGSPACE-complete	NP-complete
	In advance	Yes/no	Yes/no	Constant time	Constant time
State-variable rep.	In the input	Yes[d]	Yes/no	EXPSPACE-complete	NEXPTIME-complete
	In advance	Yes[d]	Yes/no	PSPACE [b]	PSPACE [b]
Ground state-variable rep.	In the input	Yes[d]	Yes/no	PSPACE-complete	PSPACE-complete
	In advance	Yes[d]	Yes/no	Constant time	Constant time

[a] No operator has > 1 precondition.
[b] With PSPACE- or NP-completeness for some sets of operators.
[c] Each operator with > 1 precondition is the composition of other operators.
[d] There is no way to keep the operators from having negative effects.

varies from constant time to EXPSPACE-complete, depending on what kinds of restrictions we make. Rather than proving every entry in Table 3.2 (which would make for a very long chapter!), we will illustrate the proof techniques by proving the first entry in the table, which says that ordinary classical planning is EXPSPACE-complete.

The proof, which appears in Section 3.4.2, involves reducing a known EXPSPACE-complete language-recognition problem, the EXPSPACE-bounded Turing Machine problem, to classical planning. The proof depends on using function-free ground atoms to represent binary n-bit counters and function-free planning operators to increment and decrement these counters. Section 3.4.1 shows how this can be done.

3.4.1 Binary Counters

To represent a counter that can be incremented, we would like to have an atom $c(i)$ whose intuitive meaning is that the counter's value is i and an operator incr that deletes $c(i)$ and replaces it with $c(i+1)$. The problem is that without function symbols, we cannot directly represent the integer i nor the arithmetic operation on it. However, because we have the restriction $0 \leq i \leq 2^n - 1$ for some n, then we can achieve the same effect by encoding i in binary as

$$i = i_1 \times 2^{n-1} + i_2 \times 2^{n-2} + \cdots + i_{n-1} \times 2^1 + i_n,$$

where each i_k is either 0 or 1. Instead of the unary predicate $c(i)$, we can use an n-ary predicate $c(i_1, i_2, \ldots, i_n)$; and to increment the counter we can use the following operators:

$\mathsf{incr}_1(i_1, i_2, \ldots, i_{n-1})$
  precond: $c(i_1, i_2, \ldots, i_{n-1}, 0)$
  effects: $\neg c(i_1, i_2, \ldots, i_{n-1}, 0), c(i_1, i_2, \ldots, i_{n-1}, 1)$

$\mathsf{incr}_2(i_1, i_2, \ldots, i_{n-2})$
  precond: $c(i_1, i_2, \ldots, i_{n-2}, 0, 1)$
  effects: $\neg c(i_1, i_2, \ldots, i_{n-2}, 0, 1), c(i_1, i_2, \ldots, i_{n-2}, 1, 0)$

  $\vdots$

$\mathsf{incr}_n()$
  precond: $c(0, 1, 1, \ldots, 1)$
  effects: $c(0, 1, 1, \ldots, 1), c(1, 0, 0, \ldots, 0)$

For each $i < 2^n - 1$, exactly one of the incr_j will be applicable to $c(i_1, i_2, \ldots, i_n)$, and it will increment i by 1. If we also wish to decrement the counter, then similarly we can define a set of operators $\{\mathsf{decr}_k : k = 1, \ldots, n\}$ as follows:

$\mathsf{decr}_k(i_1, i_2, \ldots, i_{n-k+1})$
  precond: $c(i_1, i_2, \ldots, i_{n-k+1}, 1, 0, \ldots, 0)$

effects: $\neg c(i_1, i_2, \ldots, i_{n-k+1}, 1, 0, \ldots, 0)$,
$c(i_1, i_2, \ldots, i_{n-k+1}, 0, 1, \ldots, 1)$

For each $i > 0$, exactly one of the decr_k will be applicable to $c(i_1, i_2, \ldots, i_n)$, and it will decrement i by 1.

Suppose we want to have two n-bit counters having values $0 \leq i \leq 2^n$ and $0 \leq j \leq 2^n$ and an operator that increments i and decrements j simultaneously. If we represent the counters by n-ary predicates $c(i_1, i_2, \ldots, i_n)$ and $d(j_1, j_2, \ldots, j_n)$, then we can simultaneously increment i and decrement j using a set of operators $\{\mathsf{shift}_{hk} : h = 1, 2, \ldots, n,\ k = 1, 2, \ldots, n\}$ defined as follows:

$\mathsf{shift}_{hk}(i_1, i_2, \ldots, i_{n-h+1}, j_1, j_2, \ldots, j_{n-k+1})$
precond: $c(i_1, i_2, \ldots, i_{n-h+1}, 0, 1, 1, \ldots, 1)$,
$d(j_1, j_2, \ldots, j_{n-k+1}, 1, 0, 0, \ldots, 0)$
effects: $\neg\, c(i_1, i_2, \ldots, i_{n-h+1}, 0, 1, 1, \ldots, 1)$,
$\neg\, d(j_1, j_2, \ldots, j_{n-k+1}, 1, 0, 0, \ldots, 0)$,
$c(i_1, i_2, \ldots, i_{n-h+1}, 1, 0, 0, \ldots, 0)$,
$d(j_1, j_2, \ldots, j_{n-k+1}, 0, 1, 1, \ldots, 1)$

For each i and j, exactly one of the shift_{hk} will be applicable, and it will simultaneously increment i and decrement j.

For notational convenience, instead of explicitly defining a set of operators such as the set $\{\mathsf{incr}_h : h = 1, \ldots, n\}$ defined previously, we sometimes will informally define a single abstract operator such as:

$\mathsf{incr}(\underline{i})$
precond: $c(\underline{i})$
effects: $\neg c(\underline{i}), c(\underline{i+1})$

where $\underline{i}$ is the sequence $i_1, i_2, \ldots, i_n$ that forms the binary encoding of i. Whenever we do this, it should be clear from context how a set of actual operators could be defined to manipulate $c(i_1, i_2, \ldots, i_n)$.

3.4.2 Unrestricted Classical Planning

Proposition 3.4 *For classical planning,* PLAN-EXISTENCE *is* EXPSPACE*-complete.*

Proof In order to prove Proposition 3.4, it is necessary to prove (1) that PLAN-EXISTENCE is in EXPSPACE, and (2) that PLAN-EXISTENCE is EXPSPACE-hard. Here is a summary of each of those proofs.

The number of ground instances of predicates involved is exponential in terms of the input length. Hence the size of any state cannot be more than exponential. Starting from the initial state, we nondeterministically choose an operator and

apply it. We do this repeatedly until we reach the goal, solving the planning problem in NEXPSPACE. NEXPSPACE is equal to EXPSPACE, hence our problem is in EXPSPACE.

To see that PLAN-EXISTENCE is EXPSPACE-hard, we define a polynomial reduction from the EXPSPACE-bounded Turing Machine problem, which is defined as follows:

> Given a Turing Machine M that uses at most an exponential number of tape cells in terms of the length of its input and an input string x, does M accept the string x?

To transform this into a planning problem, a Turing Machine M is normally denoted by $M = (K, \Sigma, \Gamma, \delta, q_0, F)$. $K = \{q_0, \ldots, q_m\}$ is a finite set of states. $F \subseteq K$ is the set of final states. Γ is the finite set of allowable tape symbols; we use # to represent the "blank" symbol. $\Sigma \subseteq \Gamma$ is the set of allowable input symbols. $q_0 \in K$ is the start state. δ, the next-move function, is a mapping from $K \times \Gamma$ to $K \times \Gamma \times \{\text{Left}, \text{Right}\}$.

Suppose we are given M and an input string $x = (x_0, x_2, \ldots, x_{n-1})$ such that $x_i \in \Sigma$ for each i. To map this into a planning problem, the basic idea is to represent the machine's current state, the location of the head on the tape, and the contents of the tape by a set of atoms.

The transformation is as follows.

Predicates: contains($\underline{i}, c$) means that c is in the ith tape cell, where $\underline{i} = i_1, i_2, \ldots, i_n$ is the binary representation of i. We can write c on cell i by deleting contains($\underline{i}, d$) and adding contains($\underline{i}, c$), where d is the symbol previously contained in cell i.

$State_F(q)$ means that the current state of the Turing Machine is q. $h(\underline{i})$ means that the current head position is i. We can move the head to the right or left by deleting $h(\underline{i})$ and adding $h(\underline{i+1})$ or $h(\underline{i-1})$. counter($\underline{i}$) is a counter for use in initializing the tape with blanks. start() denotes that initialization of the tape has been finished.

Constant symbols: $\Gamma \cup K \cup \{0, 1\}$.

Operators: Each operator in this subsection that contains increment or decrement operations (such as mapping i to $i+1$ or $i-1$) should be expanded into n operators as described in Section 3.4.1.

Whenever $\delta(q, c)$ equals (s, c', Left), we create the following operator:

$L^{s,c'}_{q,c}(\underline{i})$
precond: $h(\underline{i})$, $State_F(q)$, contains($\underline{i}, c$), start()
effects: $\neg h(\underline{i})$, $\neg State_F(q)$, $\neg$ contains($\underline{i}, c$),
$h(\underline{i-1})$, $State_F(s)$, contains($\underline{i}, c'$)

Whenever $\delta(q, c)$ equals (s, c', Right), we create the following operator:

$R^{s,c'}_{q,c}(\underline{i})$
precond: $h(\underline{i})$, $State_F(q)$, contains($\underline{i}, c$), start()
effects: $\neg h(\underline{i})$, $\neg State_F(q)$, $\neg$ contains($\underline{i}, c$),
$h(\underline{i+1})$, $State_F(s)$, contains($\underline{i}, c'$)

We have the following operator that initializes the tape with blank symbols:

$I(\underline{i})$
  precond: counter($\underline{i}$), $\neg$ start()
  effects: counter($\underline{i+1}$), contains($\underline{i}$, #)

The following operator ends the initialization phase:

$I^*()$
  precond: counter($\underline{2^n - 1}$), $\neg$ start()
  effects: contains($\underline{2^n - 1}$, #), start()

Finally, for each $q \in F$ we have this operator:

$F_q()$
  precond: $State_F(q)$
  effects: done()

Initial state: {counter($\underline{n}$), $State_F(q_0)$, $h(\underline{0})$} $\cup$ {contains($\underline{i}, x_i$) : $i = 0, \ldots, n-1$}.

Goal condition: done().

The transformation is polynomial both in time and space. It directly mimics the behavior of the Turing Machine. Thus PLAN-EXISTENCE is EXPSPACE-hard. ■

3.4.3 Other Results

Using techniques like the ones in the previous sections, complexity results can be proved for PLAN-EXISTENCE and PLAN-LENGTH in a large variety of situations. As summarized in Table 3.2, the computational complexity varies from constant time to EXPSPACE-complete, depending on a wide variety of conditions: whether or not negative effects are allowed; whether or not negative preconditions are allowed; whether or not the atoms are restricted to be ground (which is equivalent to set-theoretic planning); and whether the planning operators are given as part of the input to the planning problem or instead are fixed in advance. Here is a summary of how and why these parameters affect the complexity of planning.

1. If no restrictions are put on the planning problem $\mathcal{P}$, then as shown in Section 3.4.2, an operator instance might need to appear many times in the same plan, forcing us to search through all the states. Because the size of any state is at most exponential, PLAN-EXISTENCE can be solved in EXPSPACE.
2. If the planning operators are restricted to have no negative effects, then any predicate instance asserted remains true throughout the plan, hence no operator instance needs to appear in the same plan twice. Because the number of operator instances is exponential, this reduces the complexity of PLAN-EXISTENCE to NEXPTIME.
3. If the planning operators are further restricted to have no negative preconditions, then no operator can ever delete another's preconditions. Thus the

order of the operators in the plan does not matter, and the complexity of PLAN-EXISTENCE reduces to EXPTIME.

4. In spite of the restrictions above, PLAN-LENGTH remains NEXPTIME. Because we try to find a plan of length at most k, which operator instances we choose and how we order them makes a difference.
5. If each planning operator is restricted to have at most one precondition, then we can do backward search, and because each operator has at most one precondition, the number of the subgoals does not increase. Thus both PLAN-EXISTENCE and PLAN-LENGTH with these restrictions can be solved in PSPACE.
6. If we restrict all atoms to be ground, the number of operator instances and the size of each state reduce from exponential to polynomial. The complexity results for this case (and hence for set-theoretic planning) are usually one level lower than the complexity results for the nonground case.

   We could get the same amount of reduction in complexity by placing a constant bound on the arity of predicates and the number of variables in each operator. Set-theoretic planning corresponds to the case where the bound is zero.
7. If the operator set is fixed in advance, then the arity of predicates and the number of variables in each operator are bound by a constant. Thus, the complexity of planning with a fixed set of operators is the same as the complexity of set-theoretic planning. For classical planning with a fixed set of operators, the complexity of planning is at most in PSPACE, and there exist such domains for which planning is PSPACE-complete.

Examination of Table 3.2 reveals several interesting properties.

1. Extending the planning operators to allow conditional effects does not affect our results. This should not be particularly surprising because conditional operators are useful only when we have incomplete information about the initial state of the world or the effects of the operators, so that we can try to come up with a plan that would work in any situation that is consistent with the information available. Otherwise, we can replace the conditional operators with a number of ordinary operators to obtain an equivalent planning domain.
2. Comparing the complexity of PLAN-EXISTENCE in the set-theoretic case with the classical case reveals a regular pattern. In most cases, the complexity in the classical case is exactly one level harder than the complexity in the corresponding set-theoretic case. We have EXPSPACE-complete versus PSPACE-complete, NEXPTIME-complete versus NP-complete, EXPTIME-complete versus polynomial.
3. If negative effects are allowed, then PLAN-EXISTENCE is EXPSPACE-complete but PLAN-LENGTH is only NEXPTIME-complete. Normally, one would not expect PLAN-LENGTH to be easier than PLAN-EXISTENCE. In this case, it

happens because the length of a plan can sometimes be doubly exponential in the length of the input. In PLAN-LENGTH we are given a bound k, encoded in binary, which confines us to plans of length at most exponential in terms of the input. Hence finding the answer is easier in the worst case of PLAN-LENGTH than in the worst case of PLAN-EXISTENCE.

We do not observe the same anomaly in the set-theoretic case because the lengths of the plans are at most exponential in the length of the input. As a result, giving an exponential bound on the length of the plan does not reduce the complexity of PLAN-LENGTH.

4. PLAN-LENGTH has the same complexity regardless of whether or not negated preconditions are allowed. This is because what makes the problem hard is how to handle *enabling-condition interactions*, which are situations where a sequence of actions that achieves one subgoal might also achieve other subgoals or make it easier to achieve them. Although such interactions do not affect PLAN-EXISTENCE, they do affect PLAN-LENGTH because they make it possible to produce a shorter plan. It is not possible to detect and reason about these interactions if we plan for the subgoals independently; instead, we have to consider all possible operator choices and orderings, making PLAN-LENGTH NP-hard.
5. Negative effects are more powerful than negative preconditions. Thus, if the operators are allowed to have negative effects, then whether or not they have negated preconditions has no effect on the complexity.

3.5 Limitations

The results in this chapter say that the worst-case computational complexity of classical planning is quite high, even if we make severe restrictions. However, since these results are worst-case results, they do not necessarily describe the complexity of any particular classical planning domain.

As an example, consider the DWR domain. Since the planning operators for this domain have both negative preconditions and negative effects, this puts the domain into the class of planning problems in the first line of Table 3.2, i.e., the class for which PLAN-EXISTENCE is EXPSPACE-complete and PLAN-LENGTH is NEXPTIME-complete. We can get a much more optimistic measure of the complexity by observing that the operators are fixed in advance: this puts the DWR domain into the class of domains given in the fifth line of the table, where PLAN-EXISTENCE and PLAN-LENGTH are both in PSPACE.

Furthermore, by devising a domain-specific planning algorithm, we can get an even lower complexity. In the DWR domain, PLAN-EXISTENCE can be determined in polynomial time, and PLAN-LENGTH is only NP-complete. Chapter 4 gives a fast domain-specific planning algorithm for a restricted case of the DWR domain.

3.6 Discussion and Historical Remarks

What we call set-theoretic planning is usually called *propositional planning*, and its complexity has been heavily studied. The best-known works on this topic and the sources of the set-theoretic planning entries in Table 3.2 are Bylander [101] and Erol *et al.* [176]. Bylander [101] has also studied the complexity of propositional planning extended to allow a limited amount of inference in the domain theory. His complexity results for this case range from polynomial time to PSPACE-complete.

Littman *et al.* [365, 366] have analyzed the complexity of propositional planning if it is extended to allow probabilistic uncertainty (see Chapter 16). Their complexity results range from NP-complete to EXPTIME-complete depending on what kinds of restrictions are imposed.

Most of our results for classical planning come from Erol *et al.* [176]. They also describe some restrictions under which planning remains decidable when the planning formulation is extended to allow function symbols.

Our results on state-variable planning come from Bäckström *et al.* [38, 39, 40, 286]. They also describe several other sets of restrictions on state-variable planning. Depending on the restrictions, the complexity ranges from polynomial to PSPACE-complete.

3.7 Exercises

3.1 Let $\mathcal{P}$ be a set-theoretic planning problem in which the number of proposition symbols is r, the number of actions is a, and each operator has at most p preconditions and e effects. Write an upper bound on the number of states in $\mathcal{P}$.

3.2 Let $\mathcal{P}$ be a classical planning problem in which the number of predicate symbols is r, each predicate takes at most a arguments, the number of constant symbols is c, the number of operators is o, and each operator has at most p preconditions and e effects. Write an upper bound on the number of states in $\mathcal{P}$.

3.3 Suppose we extend classical planning to include function symbols. Write an algorithm that, given the statement of a planning problem P, returns **yes** if P has a solution of length k or less and **no** otherwise.

3.4 PLAN-EXISTENCE is EXPSPACE-complete for classical planning. However, every classical planning problem can be rewritten as a set-theoretic planning problem, and PLAN-EXISTENCE is only PSPACE-complete for set-theoretic planning. Resolve the apparent contradiction.

3.5 A *blocks-world planning problem* is a classical planning problem whose statement is as follows: The operators are those of Exercise 2.1 (see page 50), and the initial state and the goal are arbitrary sets of ground atoms in which every atom has one of the following forms: clear(x), holding(x), on(x, y), ontable(x). Let B be the set of all blocks-world planning problems.

(a) Write a polynomial-time algorithm to solve PLAN-EXISTENCE(B), thereby showing that PLAN-EXISTENCE(B) is in P.

(b) Write a polynomial-time nondeterministic algorithm to solve PLAN-LENGTH (B), thereby showing that PLAN-LENGTH(B) is in NP.

(c) Which lines of Table 3.2 include PLAN-EXISTENCE(B) and PLAN-LENGTH(B)? Resolve the apparent contradiction between this information and the results you proved in parts a and b of this exercise.

3.6 Write the exact set of operators produced by the transformation in Section 3.4.2 for the case where $n = 3$. If $s_0 = \{c(0,0,0)\}$ and $g = \{c(1,1,1)\}$, then what is the size of the state space?

3.7 If we rewrite the planning problem of Section 3.4.2 as a set-theoretic planning problem, how many actions will there be?