for us to move a block onto it. To fix this, we do two things. First, we introduce another action to move a block \( b \) from \( x \) to the table:

\[
\text{Action(Move To Table(b, x),}
\]

\[
\text{PRECOND: On(b, x) } \land \text{ Clear(}b\text{)}
\]

\[
\text{EFFECT: On(b, Table) } \land \text{ Clear(}x\text{) } \land \text{ On(b, x)''}
\]

Second, we take the interpretation of \( \text{Clear(x)} \) to be "there is a clear space on \( x \) to hold a block." Under this interpretation, \( \text{Clear(Table)} \) will always be true. The only problem is that nothing prevents the planner from using \( \text{Move(b, x, Table)} \) instead of \( \text{MoveToTable(b, x)} \).

We could live with this problem—it will lead to a larger-than-necessary search space, but will not lead to incorrect answers—or we could introduce the predicate \( \text{Block(b)} \) and add \( \text{Block(b)} \) \( \text{Block(y)} \) to the precondition of \( \text{Mom} \).

### 10.1.4 The complexity of classical planning

In this subsection we consider the theoretical complexity of planning and distinguish two decision problems. \( \text{PlanSAT} \) is the question of whether there exists any plan that solves a planning problem. Bounded \( \text{PlanSAT} \) asks whether there is a solution of length \( k \) or less; this can be used to find an optimal plan.

The first result is that both decision problems are decidable for classical planning. The proof follows from the fact that the number of states is finite. But if we add function symbols to the language, then the number of states becomes infinite, and \( \text{PlanSAT} \) becomes only semidecidable: an algorithm exists that will terminate with the correct answer for any solvable problem, but may not terminate on unsolvable problems. The Bounded \( \text{PlanSAT} \) problem remains decidable even in the presence of function symbols. For proofs of the assertions in this section, see Ghallab et al. (2004).

Both \( \text{PlanSAT} \) and Bounded \( \text{PlanSAT} \) are in the complexity class \( \text{PSPACE} \), a class that is larger and hence more difficult than \( \text{NP} \) and refers to problems that can be solved by a deterministic Turing machine with a polynomial amount of space. Even if we make some rather severe restrictions, the problems remain quite difficult. For example, if we disallow negative effects, both problems are still \( \text{NP-hard} \). However, if we also disallow negative preconditions, \( \text{PlanSAT} \) reduces to the class \( \text{P} \).

These worst-case results may seem discouraging. We can take solace in the fact that agents are usually not asked to find plans for arbitrary worst-case problem instances, but rather are asked for plans in specific domains (such as blocks-world problems with \( n \) blocks), which can be much easier than the theoretical worst case. For many domains (including the blocks world and the air cargo world), Bounded \( \text{PlanSAT} \) is \( \text{NP-complete} \) while \( \text{PlanSAT} \) is in \( \text{P} \); in other words, optimal planning is usually hard, but sub-optimal planning is sometimes easy. To do well on easier-than-worst-case problems, we will need good search heuristics. That’s the true advantage of the classical planning formalism: it has facilitated the development of very accurate domain-independent heuristics, whereas systems based on state axioms in first-order logic have had less success in coming up with good heuristics.
10.2 ALGORITHMS FOR PLANNING AS STATE-SPACE SEARCH

Now we turn our attention to planning algorithms. We saw how the description of a planning problem defines a search problem: we can search from the initial state through the space of states, looking for a goal. One of the nice advantages of the declarative representation of action schemas is that we can also search backward from the goal, looking for the initial state. Figure 10.5 compares forward and backward searches.

10.2.1 Forward (progression) state-space search

Now that we have shown how a planning problem maps into a search problem, we can solve planning problems with any of the heuristic search algorithms from Chapter 3 or a local search algorithm from Chapter 4 (provided we keep track of the actions used to reach the goal). From the earliest days of planning research (around 1961) until around 1998 it was assumed that forward state-space search was too inefficient to be practical. It is not hard to come up with reasons why.

First, forward search is prone to exploring irrelevant actions. Consider the noble task of buying a copy of *Al: A Modern Approach* from an online bookseller. Suppose there is an

![Figure 10.5](image)

Figure 10.5  Two approaches to searching for a plan. (a) Forward (progression) search through the space of states, starting in the initial state and using the problem’s actions to search forward for a member of the set of goal states. (b) Backward (regression) search through sets of relevant states, starting at the set of states representing the goal and using the inverse of the actions to search backward for the initial state.