In this section, we describe a method of solving planning problems by reduction to SAT.

1 Propositional STRIPS Planning

A propositional STRIPS planning instance $\Pi$ is a tuple $\Pi = \langle F, I, A, G \rangle$ consisting of:

- **Set $F$ of “Facts”:** Possible states of the world are described in terms of a set $F$ of “state variables”, or “facts”, each of which may be true or false. We will take $F$ to be a set of propositional atoms, so each truth assignment to $F$ is a possible state of the world. We denote by $\text{lits}(F)$ be the set of literals over $F$, that is $\{ f, \neg f \mid f \in F \}$. The maximal satisfiable subsets of $\text{lits}(F)$ are one-to-one with the truth assignments for $F$, and thus with states of the world.

- **Initial state $I$:** The initial state is given by a truth assignment for $F$.

- **Set $A$ of “Actions”:** Each action $a \in A$ is defined by two satisfiable sets of literals from $\text{lits}(F)$:
  - The set $\text{pre}(a)$ of preconditions of $a$;
  - The set $\text{eff}(a)$ of effects of $a$.

- **Goal $G$:** A set of states, specified by a satisfiable set of literals from $\text{lits}(F)$. Any state which satisfies $G$ is a goal state.

An action $a$ is executable in a state $S$ if each literal in $\text{pre}(a)$ is true in $S$, that is, if $S \models \text{pre}(a)$. The result of executing $a$ in $S$ is the state $S'$ defined by:
\[
S'(f) = \begin{cases} 
  f & \text{if } f \in \text{eff}(a) \\
  \neg f & \text{if } \neg f \in \text{eff}(a) \\
  S(f) & \text{otherwise.}
\end{cases}
\]

In other words, \(S'\) is the same as \(S\), except in-so-far as necessary so that \(S' \models \text{eff}(a)\).

A plan \(P\) for \(\Pi\) of length \(L\) is a sequence of actions \(P = \langle a_1, a_2, \ldots, a_L \rangle\) such that there is a sequence of states \(S_P = \langle s_0, s_1, \ldots, s_L \rangle\) satisfying

1. \(s_0 = I;\)
2. For each \(1 \leq i \leq L\), action \(a_i\) is executable in \(s_{i-1};\)
3. For each \(1 \leq i \leq L\), state \(s_i\) is the result of executing action \(a_i\) in state \(s_{i-1};\)
4. \(s_L\) is a goal state, that is, \(s_L \models G.\)

## 2 Representing Planning in CNF

**Fact 1.** Given as input a propositional STRIPS instance \(\Pi\), deciding if \(\Pi\) has a plan is PSPACE-complete.

Intuitively, the reason is that the shortest plan may be of length exponential in the size of the planning instance. As a consequence, representing the set of plans in propositional logic requires formulas which are of size exponential in the size of the instance, which seems undesirable. Instead of doing this, we consider a more convenient task: representing plans of a given length.

**Fact 2.** Given as input a propositional STRIPS planning instance \(\Pi\) and a natural number \(L\), deciding existence of a plan of length at most \(L\) is NP-complete.

Actually, we will not model plans of bounded length, but rather plans which are fairly naturally modelled with a bounded number of time-steps. In particular, we will devise a family of formulas \(\Phi_{T}^{\Pi}\), parameterized by planning instance \(\Pi\) and positive integer \(T\), with the property that

\(\Phi_{T}^{\Pi}\) is satisfiable iff there is a plan for \(\Pi\) involving at most \(T\) time steps.

Often, we leave the planning instance \(\Pi\) implicit, writing simply \(\Phi_{T}\).

Notice the shift in perspective here: we defined plans as a sequence of actions, but the description of the formula refers to time steps rather than actions. In fact, the formula we will use:
1. allows time steps in which no action is performed;
2. allows multiple actions to be performed at one time step (with some constraints);
3. bounds the number of time steps, but not (at least directly) the number of actions.

These properties seem to make solving easier. The first allows finding plans without an exactly specified number of steps. Any “no-op” steps can be trivially eliminated in post-processing. The second property allows us to make the formula smaller. The size of the formula needed to represent the plans grows linearly with the number of steps involved, so allowing multiple actions per step reduces the size of formula needed to find plans for a given instance. We call these “parallel plans”, but they do not model concurrent actions in any serious way. We require that they be serializable, and therefore we need to add clauses to ensure this.

To write the formula, we will use two sets of atoms:

- **State Atoms**: For each fact \( f \in F \) and each time \( t \in \{0, \ldots, T\} \), we have atom \( f_t \). The intuitive meaning of \( f_t \) is that \( f \) is true at time \( t \).
- **Action Atoms**: For each action \( a \in A \) and each time \( t \in \{1, \ldots, T\} \), we have atom \( a_t \). The intuitive meaning of \( a_t \) is that action \( a \) is executed in the \( t^{th} \) time step, which is the transition from the state at time \( t - 1 \) to the state at time \( t \).

The formula will be the union of the following sets of clauses, each enforcing a particular constraint on plans. (In some cases, for improved readability, we do not write in clause form, but the translation to clauses is easy.)

1. **Initial State**: The state at time 0 corresponds to the initial state.
   
   For each fact \( f \in F \), include unit clause \( (f_0) \) if \( f \) is in \( I \), and \( (\neg f_0) \) otherwise.

2. **Goal States**: The state at time \( T \) satisfies the goal conditions.
   
   For each fact \( f \in F \), include unit clause \( (f_T) \) if \( f \) is in \( G \), and the unit clause \( (\neg f_T) \) if \( \neg f \) is in \( G \).

3. **Action Preconditions**: If action \( a \) is executed in time step \( t \), then the preconditions of \( a \) hold a time \( t - 1 \).
   
   For each action \( a \), and each time \( t \in \{1, \ldots, T\} \), include clauses equivalent to
   
   \[
   a_t \rightarrow \bigwedge_{l \in \text{pre}(a)} l_{t-1}.
   \]

4. **Action Effects**: If action \( a \) is executed in time step \( t \), then its effects hold at time \( t \).
For each action \( a \), and each time \( t \in \{1, \ldots, T\} \), include clauses equivalent to
\[
a_t \rightarrow \bigwedge_{l \in \text{eff}(a)} l_t.
\]

5. **Explanatory Frame Axioms**: These are to ensure that the state only changes as a result of actions being executed. In particular, if a “fact” changes truth value during some time step, then it must be the effect of an action executed during that step.

For each fact \( f \in F \), and each time \( t \in \{1, \ldots, T\} \) include clauses equivalent to:
\[
(f_{t-1} \land \neg f_t) \rightarrow \bigvee_{\{a \mid \neg f \in \text{eff}(a)\}} a_t,
\]
and
\[
(\neg f_{t-1} \land f_t) \rightarrow \bigvee_{\{a \mid f \in \text{eff}(a)\}} a_t.
\]

6. **Serializability of Actions**: If multiple actions \( \{a_1, a_2, \ldots, a_k\} \) are executed during time step \( t \), then we require there to be an ordering of the actions which constitutes a (sequential) plan (because we are using “parallel” plans as a convenience, not to model truly concurrent actions). We may enforce this simply by requiring the actions \( a_1, \ldots, a_k \) to be pairwise non-conflicting, in the sense that the execution of one does not preclude the other being executed in the resulting state.

For each pair \( a, b \) of distinct actions, if \( \text{pre}(a) \cup \text{eff}(b) \) is unsatisfiable, then for each time \( t \in \{1, \ldots, T\} \), include the clause
\[
(\neg a_t \lor \neg b_t).
\]

3 **“Optimum” Planning via Satisfiability**

We now have a family of formulas which allow us to use a SAT solver to find plans bounded by some number of time steps. Our goal is to find the shortest plans possible. While finding optimum-length plans would be ideal, we will be satisfied with finding “parallel” plans using a minimum number of time steps. Let \( T^* \) denote the minimum number of time steps for which a plan exists. To establish that we have an optimum plan, we will need (at least) two calls to the SAT solver: one to show that \( \Phi_{T^*} \) is satisfiable, and one to show that \( \Phi_{T^*-1} \) is unsatisfiable.
Unless we are extremely lucky and guess $T^*$, we will need to call the solver with a sequence of formulas generated using a sequence of time bounds $\sigma = \langle T_1, \ldots, T_k \rangle$. We would like to choose this sequence to minimize the total time required to find an optimum plan (or perhaps the best plan we can find within some allotted amount of time). The easiest scheme is to use $\sigma = \langle 1, 2, 3, \ldots, T^* - 1, T^* \rangle$. While this seems wasteful, since in most cases $T^*$ is not very close to 1, generating and testing the formulas when $T$ is very small tends to be very fast. This method has the obvious advantage that no guessing is required, and in fact it works quite well in practice, provided $T^*$ is not too large.

Another natural idea is to use binary search. If we know an upper bound $T_{UB}$ on $T^*$, then we perform a binary search in the interval $[0, T_{UB}]$, which will find $T^*$ in about $\log T_{UB}$ calls to the solver. If we don’t know such a bound, we can make a sequence of calls with time bounds $\langle 1, 2, 4, \ldots, 2^i \rangle$, where $2^i$ is the smallest power of 2 with $2^i \geq T^*$. We know when we reach $i$ because it is the first call to the solver that returns a plan. The last call in this sequence gives us $T_{UB} = 2^i$, after which we perform binary search in the interval $(2^{i-1}, 2^i)$ for $T^*$, so we find $T^*$ in time $O(\log T^*)$. Unfortunately, minimizing the number of solver calls may not (and typically will not) minimize time, because the running time for the call varies dramatically with $T$. The typical pattern is, roughly, that time to solve $\Phi_1$ is trivial; times increase dramatically as $T^* - 1$ is approached; solving time drops moderately just above $T^* - 1$, and then increases further (due primarily to the large size of the formula). Thus, the time to establish the optimum value tends to dominate, except in the case that poor guesses well beyond the optimum are not made.