Planning as model checking

Omar El-khatib and Hung Viet Le

March 15, 2004

A classical plan is of the form: \( a_1, a_2, \ldots, a_n \), which is a sequence of actions. In planning as model checking, we consider plans - called situated plans - of the following form:

\[
\pi = \{(s, a) : s \in S \text{ and } a \in A\}
\]

Each \((s, a)\) in \(\pi\) is called a state-action pair.

To execute a situated plan plan \(\pi\), we use the following algorithm:

\[
\text{while } s \in \{S : (s, a) \in \pi\} \\
\quad \text{do execute a such that } (s, a) \in \pi \text{ and let } s := R(s, a)
\]

**Example 1** Consider example 1 from previous lecture, repeated here:

- \(F = \{\text{Loaded, Locked}\}\)
- \(A = \{\text{Lock, Load, wait, Unlock, Unload}\}\)
- \(S = \{\emptyset, \{\text{Locked}\}, \{\text{Loaded}\}, \{\text{Locked, Loaded}\}\}\)
- \(R = \{(s_0, \text{wait, } s_0), (s_0, \text{Lock, } s_1), (s_0, \text{Load, } s_2), (s_1, \text{Unlock, } s_0), (s_2, \text{Unlock, } s_0), (s_2, \text{Lock, } s_3), (s_3, \text{Unlock, } s_2)\}\)

![Diagram of system description]

Figure 1: System description
Figure 2: Example with \( \pi_3 = \{(1, a), (2, b), (1, c)\} \)

(See figure 1)

Let \( \pi_1 = \{(2, \text{wait}), (3, \text{lock})\}. \) Applying the previous algorithm, starting from the state 2, we will execute \text{wait} forever.

If \( \pi_2 = \{(2, \text{load}), (3, \text{lock})\} \) then, starting from the state 2, we will execute \text{load}, then \text{lock} and we will reach state \( \{4\} \) and stop.

This shows that the execution of a situated plan is different than the execution of a classical plan.

Another difference is illustrated in the next example.

**Example 2 (See figure 2)**

For \( \pi_3 = \{(1, a), (2, b), (1, c)\} \), since state \( \{1\} \) appears twice in \( \pi_3 \), we cannot decide whether we should execute \( a \) or \( c \).

A plan is called a goal preserving plan if it satisfies the following condition:

\[
\pi = \{(s, a) | s \not\in G\}
\]

where \( G \) is the set of states in which the goal is satisfied. The above formula says that the execution of a plan should stop when the goal is reached. This condition is a strong one. A weaker condition for a plan might needed, which is rather than not acting in a goal state, a plan can still act without removing the goal states. This condition is called a dynamic goal preserving plan which is stated formally:

\[
\pi = \{(s, a) : s \in S, a \in A, i f s \in G \text{ then } R(s, a) \in G\}
\]

A plan is called a fair goal preserving if it satisfies the following condition:

For every \( (s_0, a_0) \in \pi \) with \( s_0 \in G \), then \( \exists (s_1, a_1), (s_2, a_2), \ldots, (s_n, a_n) \) such that \( s_{i+1} = R(s_i, a_i) \) for \( 0 \leq i < n-1 \) and \( s_n \in G \).

A goal achieving plan is defined recursively as follows:

- \( \pi = \{(s, a) : s \in S, a \in A, R(s, a) \in G\} \) is a goal achieving plan. This represent reaching the goal in one step.

- If \( \pi' \) is a goal achieving plan, then \( \pi = \pi' \cup \{(s, a)\} \) such that \( R(s, a) \in \{s' : (s', a') \in \pi' \} \) is a goal achieving plan.
Situated plan is a goal preserving and a goal achieving plan.

Situated plan is different than classical plan in that it can handle unexpected action outcomes. Consider example 1, for the plan \( \pi_2 \), starting from the state 2. If the execution of the action load fails (i.e. no load happens), then we stay at state 2, and we need to execute load again. On the other hand, the classical plan load; lock will fail.

The Algorithm for generating a situated plan is shown below:

Algorithm plan(P)
1 CS = φ
2 NS= φ
3 plan = φ
4 whileNS ≠ CS do
5 if I ∈ NS then return plan
6 compute OS = OSP(NS, D)
7 CS=NS
8 plan = plan ∪ prune(OS, NS)
9 NS = NS ∪ proj(OS)
10 end while
11 return plan
12 end_plan

where:

- NS is the next states set.
- CS is the current states set.
- plan is a list of actions that represents the required plan.
- OS=OSP: returns all state-action pairs that can reach any state in NS, i.e. OSP = \{ (s, a) : s ∈ S, a ∈ A, (∃ s’ : (s’ ∈ NS and s’ = R(s, a))) \}
- prune(OS, NS) = \{ (s, a) : (s, a) ∈ OS and s \∉ NS \}
- proj(OS) = \{ s : (s, a) ∈ OS \}

**Example 3** Consider example 1 (from the previous note). Let the goal be 4. Executing algorithm plan step by step is as follows:

step Execution
1 CS=φ
2 NS=4
3 plan =φ
4 while loop: CS ≠ NS not satisfied so enter the loop
5 I \notin NS
6 OS = { (3, lock) }
7 CS = {4}
Figure 3: System description

8 \( \text{plan} = \{(3, \text{lock})\} \)
9 \( \text{NS} = \{4\} \cup \{3\} = \{3, 4\} \)
4 while loop: \( \text{CS} \neq \text{NS} \) not satisfied, so continue in while loop
5 \( I \notin \text{NS} \)
6 \( OS = \{(2, \text{load}), (3, \text{lock}), (4, \text{unlock})\} \)
7 \( CS = \{3, 4\} \)
8 \( \text{plan} = \{(3, \text{lock})\} \cup \{(2, \text{load})\} = \{(2, \text{load}), (3, \text{lock})\}, \)
pruning will remove \((3, \text{lock})\) and \((4, \text{unlock})\) because they are in \( \text{NS} \).
9 \( \text{NS} = \{3, 4\} \cup \{2\} = \{2, 3, 4\} \)
4 while loop: \( \text{CS} \neq \text{NS} \) not satisfied, so continue in while loop
5 \( I \in \text{NS} \) satisfied return plan = \{(2, load), (3, lock)\}

The above algorithm returns a goal preserving plan and goal achieving plan.

Properties of the algorithm plan(P) are:

- Correctness: every plan returned by the algorithm plan(P) is a goal preserving plan.

- Completeness: If there is a situated plan, algorithm plan(p) will return that plan. It will stop if there is no situated plan. However the algorithm plan(P) returns only one plan, it can be modified to return multiplans plans.

1 Non–determinism

In classical planning we assume that actions are deterministic meaning that an action will change from a state to only one state. However, if we relax this assumption, by allowing actions to be non–determinism, i.e., execution of an action might result in different states. Adding a new state 5 to example 1 (from
the previous note) as shown in figure 3, with a new action \textit{adjust}.

This requires to change the function $R$ to a transition relation $R \subseteq S \times A \times S$.

In this example, we have

- $\pi_5 = \{(2, \text{load}), (3, \text{lock}), (5, \text{adjust})\}$ is called \textit{strong} plan meaning that it guarantees to reach a goal.

- $p_{i6} = \{(2, \text{load}), (3, \text{lock})\}$ is a \textit{weak} plan meaning that it might reach the goal.

We need an algorithm to generate situated plans in the presence of non-determinism. It is easy, just modify the algorithm plan($P$) by changing step 6 to:

\begin{equation*}
OS = OSP\left(\text{NS, D}\right) = \{(s, a) : \forall s' \ (s' \in \text{NS and } R(s, a, s'))\}
\end{equation*}

### 2 planning via symbolic model checking

The key idea is the following:

- Planning problem is represented as formulas.
- Plans are represented as formulas.
- Planning is searching through the set of states by evaluating the assignments to the variables in the formulas to satisfy the formulas.

How to construct the planning domain ($D = \langle F, A, S, R \rangle$) as a formula:

- Fluent $F$ is represented by $\bar{x} = x_1, x_2, \ldots, x_n$.
- $S(\bar{x})$ is a formula in $\bar{x}$. This represents the state $S$. In addition, $s(\bar{x}) = T$ (tautology, i.e. always true).
- for every $Q \subseteq S$, $Q(\bar{x})$ is a formula represents $Q$.
- Actions $A$ is represented by $\hat{a} = a_1, a_2, \ldots, a_n$.
- $R$ is a transition relation which is represent by: $R(\bar{x}, \hat{a}, \bar{x}')$, where $\bar{x}' = \bar{x}$ but with a prime '$$'.

\textbf{Example 4} Consider example 1 previously.

- $\bar{x} = \text{Loaded, Locked}$.
- $s(\bar{x}) = (\neg \text{Loaded} \lor \neg \text{Locked}) \land (\text{Locked} \lor \neg \text{Loaded}) \land (\neg \text{Locked} \lor \text{Loaded}) \land (\text{Loaded} \lor \text{Locked})$
- $\hat{a} = \text{lock, unlock, load, unload, wait}$.
- $R(\bar{x}, \hat{a}, \bar{x}')$ is:
- \( R(1, \text{unlock}, 2) = (\text{Locked} \land \neg \text{Loaded}) \land \text{unlock} \rightarrow (\neg \text{Loaded}' \land \neg \text{Locked}') \)
- \( R(2, \text{lock}, 1) = (\neg \text{Locked} \land \neg \text{Loaded}) \land \text{lock} \rightarrow (\text{Locked}' \land \neg \text{Locked}') \)
- \( R(2, \text{load}, 3) = (\neg \text{Locked} \land \neg \text{Loaded}) \land \text{load} \rightarrow (\neg \text{Locked}' \land \text{Loaded}') \)
- \( R(3, \text{unload}, 2) = (\neg \text{Locked} \land \text{Loaded}) \land \text{load} \rightarrow (\text{Locked}' \land \neg \text{Loaded}') \)
- \( R(3, \text{lock}, 4) = (\neg \text{Locked} \land \text{Loaded}) \land \text{lock} \rightarrow (\text{Locked}' \land \text{Loaded}') \)
- \( R(4, \text{unlock}, 3) = (\text{Locked} \land \text{Loaded}) \land \text{unlock} \rightarrow (\text{Locked}' \land \neg \text{Loaded}') \)

The symbolic representation of a planning problem \( P=<D, I, G> \) is obtained from the symbolic representation of the planning domain \( D \), and from the boolean formulas \( I(\vec{x}) \) and \( G(\vec{x}) \). A symbolic plan for a symbolic planning domain \( D \) is a formula in \( \vec{x} \) and \( a \).

**Example 5** For a plan \( \{ (2, \text{load}), (3, \text{lock}) \} \), it is represented symbolically as:
\[
(\neg \text{Loaded} \lor \neg \text{Locked} \lor \text{load}) \land (\text{Loaded} \lor \neg \text{Locked} \lor \text{lock}).
\]

To change algorithm plan(\( P \)) to handle the new symbolic representation of the planning problem by changing:
\[
\text{OS} = \text{OSP} = (x, a) : \exists x'(x' \in \text{NS} \land R(x, a, x'))
\]

Planning via symbolic model checking can be implemented in several ways, but the most successful way is using Ordered Binary Decision Diagram (OBDD). OBDD is a compact representation of the assignments satisfying (and falsifying) a given boolean formula. OBDD is rooted, directed, binary, acyclic graph with one or two terminal nodes (labeled 0 or 1).