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:

$$\pi = \{(s,a) : s \in S \textit{ and } a \in A\}$$

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

To execute a situated plan plan π, we use the following algorithm:

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

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

- F = {*Loaded, Locked* }
- A = {*Lock, Load, wait, Unlock, Unload* }
- S = {ϕ, {*Locked*}, {*Loaded*}, {*Locked, Loaded*}}
- R = {(s_0, *wait*, s_0), (s_0, *Loack*, s_1), (s_0, *Load*, s_2), (s_1, *Unlock*, s_0), (s_2, *Unload*, s_0), (s_2, *Lock*, s_3), (s_3, *Unlock*, s_2) }

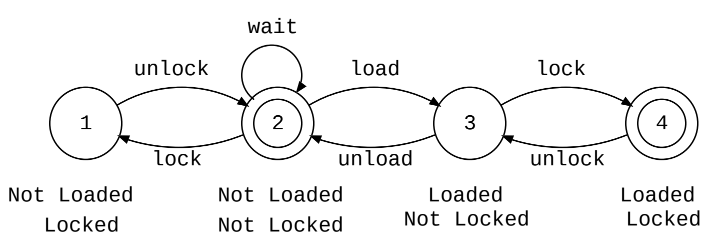

Figure 1: System description

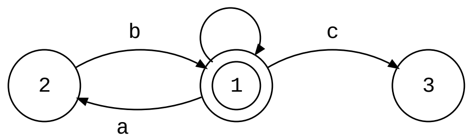

Figure 2: Example with π_3={(1, a), (2, b), (1, c)}

(See figure 1)

Let π_1 *= {(2, wait), (3, lock)}. Applying the previous algorithm, starting from the state 2, we will execute* **wait** *forever.*

If π_2*={(2, load), (3, lock)} then, starting from the state 2, we will execute* **load***, then* **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 π_3*={(1, a), (2, b), (1, c)}, since state {1} appears twice in* π_3*, we cannot decide whether we should execute a or c.*

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

$$\pi = \{(s,a)|s \notin G\}$$

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:

$$\pi = \{(s,a) : s \in S, a \in A, if s \in G\ then\ R(s,a) \in G\}$$

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

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

A *goal achieving plan* is defined recursivly as follows:

- π = {(s, a) : s $\in$ S, a $\in$ A, R(s, a) $\in$ G} is a goal acheiving plan. This represent reaching the goal in one step.
- If π' is a goal acheiving 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 π_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 fail
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:* $CS \neq NS$ *not satisfied so enter the loop*
5	*I* ∉*NS*
6	*OS* = {*(3, lock)*}
7	*CS* = {*4*}

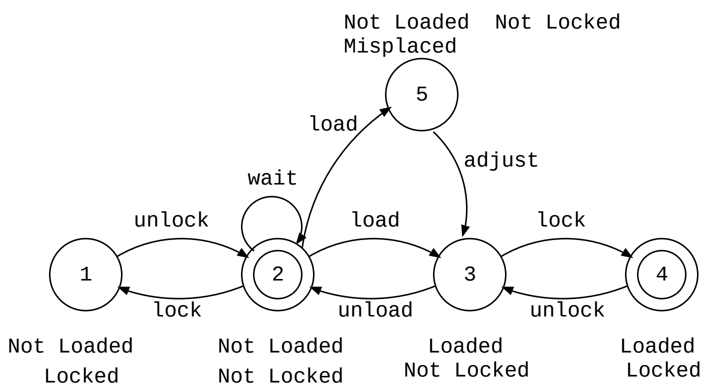

Figure 3: System description

8 *plan* $=\{(3, lock)\}$
9 $NS = \{4\} \cup \{3\} = \{3, 4\}$
4 *while loop:* $CS \neq NS$ *not satisfied, so continue in while loop*
5 $I \notin NS$
6 $OS=\{(2, load), (3, lock), (4, unlock)\}$
7 $CS=\{3, 4\}$
8 *plan*$=\{(3, lock)\} \cup \{(2, load)\} = \{(2, load), (3, lock)\}$,
pruning will remove (3, lock) and (4, unlock) because they are in NS.
9 $NS=\{3, 4\} \cup \{2\} = \{2, 3, 4\}$
4 *while loop:* $CS \neq NS$ *not satisfied, so continue in while loop*
5 $I \in NS$ *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.
- Completness: 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 determinstic 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 *adjust*.
This requires to change the function R to a transition relation $R \subseteq S \times A \times S$.

In this example, we have

- π_5 = {(2, load), (3, lock), (5, adjust)} is called *strong* plan meaning that it guarantees to reach a goal.
- pi_6={(2, load), (3, lock)} is a *weak* plan meaning that it might reach the gaol.

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:
OS = OSP(NS, D) = {(s, a) : ∀ s' (s' ∈ NS and R(s, a, s')}

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 = <F, A, S, R>) as a formula:

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

Example 4 *Consider example 1 previously.*

- *$\underline{x}$= Loaded, Locked.*
- *s($\underline{x}$)= (¬Loaded ∨ ¬Locked) ∧ (Locked ∨¬Loaded) ∧ (¬Locked ∨Loaded) ∧ (Loaded ∨Locked)*
- *$\underline{a}$ = lock, unlock, load, unload, wait.*
- *R($\underline{x}$, $\underline{a}$, $\underline{x}$') is:*

- $R(1, \mathit{unlock}, 2) = (\mathit{Locked} \wedge \neg \mathit{Loaded}) \wedge \mathit{unlock} \rightarrow (\neg \mathit{Loaded}' \wedge \neg \mathit{Locked}')$
- $R(2, \mathit{lock}, 1) = (\neg \mathit{Locked} \wedge \neg \mathit{Loaded}) \wedge \mathit{lock} \rightarrow (\mathit{Locked}' \wedge \neg \mathit{Locked}')$
- $R(2, \mathit{load}, 3) = (\neg \mathit{Locked} \wedge \neg \mathit{Loaded}) \wedge \mathit{load} \rightarrow (\neg \mathit{Locked}' \wedge \mathit{Loaded}')$
- $R(3, \mathit{unload}, 2) = (\neg \mathit{Locked} \wedge \mathit{Loaded}) \wedge \mathit{load} \rightarrow (\neg \mathit{Locked}' \wedge \neg \mathit{Loaded}')$
- $R(3, \mathit{lock}, 4) = (\neg \mathit{Locked} \wedge \mathit{Loaded}) \wedge \mathit{lock} \rightarrow (\mathit{Locked}' \wedge \mathit{Loaded}')$
- $R(4, \mathit{unlock}, 3) = (\mathit{Locked} \wedge \mathit{Loaded}) \wedge \mathit{unlock} \rightarrow (\mathit{Locked}' \wedge \neg \mathit{Loaded}')$

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($\underline{x}$) and G($\underline{x}$). A ymbolic plan for a symbolic planning domain D is a formula in $\underline{X}$ and $\underline{a}$.

Example 5 *For a plan* $\{(2, \mathit{load}), (3, \mathit{lock})\}$*, it is represented symboliclly as:* $(\neg \mathit{Loaded} \vee \neg \mathit{Locked} \rightarrow \mathit{load}) \wedge (\mathit{Loaded} \vee \neg \mathit{Locked} \rightarrow \mathit{lock})$.

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

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).