32002: AI (PO Planning, Language, Planning with Prop. Logic) **Spring 2014**

Lecture 37: April 16, 2014

Instructor: K.R. Chowdhary *: Professor of CS*

Note: *LaTeX template courtesy of IITJ, CSE dept.*

37.1 Partial Order Planning

The Forward-planner enforces a *total ordering* on actions at all the stages in the planning process. The idea of partial ordering between actions only commit to an ordering between actions when forced. A partial ordering is a '$\leq$' relation, that is *transitive* and *antisymmetric.* It is set of actions together with partial ordering, representing a "before relation" on actions, such that any total ordering of the actions, consistent with partial ordering will solve the goal from initial state. That is, $Act_0 < act_1$, if action Act_0 is before Act_1 in partial order; i.e. Act_0 must occur before Act_1.

For the sake of uniformity, the relations which are precondition to *start* state are true, and same is case of *finish* state, so that both the state can occur. We assume that these pseudo actions: *start*, *finish* are before and after each action, so that the algorithm does not require any preconditions.

In fact, a partial planner works as follows: Begin with actions *start* and *finish*, and the partial order $start < finish$. The planner maintains an agenda set of $\langle P, A\rangle$ pairs, where, A is action in plan, and P is *precondition* of A. First, $\langle G, finish\rangle$ is chosen, such that G is precondition for *Goal*. Then at each stage, a pair $\langle P, act_1\rangle$ is chosen from the agenda, where P is precondition for act_1. Subsequent to this, act_0 is chosen to achieve P, which is either already in the plan or it is *start* to achieve P, or it could be a new, that is added to the plan. The act_0 must occur before act_1, that adds a new causal link. Any action to delete P must happen after act_1 or before act_0. If act_0 is new action, its preconditions are added to the agenda, and process continues till the agenda is empty.

37.2 Planning Language

The STRIPS (*STanford Research Institute Problem Solver*)formulation for planning problems employs propositional logic as a language for describing states of the world, a simple but general format for specifying operators, and a clear semantics for change over time. In the STRIPS formulation, there is a set of conditions, for example, a set of Boolean variables, used to describe states. A complete assignment maps the set of conditions to possible values, for example, truth values, and a partial assignment maps a subset of conditions to values. A state is a *complete assignment.* An operator consists of two *partial assignments*; the first, called the *preconditions* or *results*, determines the states in which the operator can be applied, and the second, called the *postconditions*, determines the state that results from applying the operator in a particular starting state. Implicit frame axioms specify that the value of any condition not mentioned in an operator's postconditions is unchanged by the application of the operator.

A planning problem (instance) is specified by a set of operators, an *initial state*, and a goal, for example, a

Boolean formula. Generally speaking, a solution is a partially ordered *multiset* of operators such that any total ordering consistent with the given partial order transforms the assignment specified by the initial state into a assignment that satisfies the *goal* formula. The STRIPS formulation provides the semantic foundations for many extensions, including those handling external events, multiple agents, probabilistic transformations, and variants that allow the agent to observe aspects of the current state and choose actions conditioned on observations.

The STRIPS is action centric representation, which is based on the idea that most things are not effected by single action. It specifies: *action*, *precondition*, and *effect*. For example, for the goal: "Robo to pickup coffee" has,

$$\begin{aligned} &precondition : cs \wedge \overline{rhc} \\ &effect : rhc \end{aligned}$$

The other features are unaffected in above. The action of delivering coffee (dc) is:

$$\begin{aligned} &precondition : off \wedge rhc \\ &effect : \overline{rhc} \wedge \overline{swc} \end{aligned}$$

Apart from above, you need to specify the initial states and goal. The *action* or the *rule* has two formats:

- A *causal Rule*, when a feature gets a new value.
- A *Frame Rule*, When a feature keeps its value.

We can demonstrate the change of Robo's location, from current $RLoc^0$ to new location $RLoc^1$ on account of action mcc, mc, and one which is neither mcc nor mc, as follows:

$$\begin{aligned} &(RLoc^1 = cs) \leftarrow (RLoc^0 = off) \wedge (Act = mcc) \\ &(RLoc^1 = cs) \leftarrow (RLoc^0 = mr) \wedge (Act = mc) \\ &(RLoc^1 = cs) \leftarrow (RLoc^0 = cs) \wedge (Act \neq mcc) \wedge (Act \neq mc) \end{aligned}$$

In the above, the first two are causal rules, and the last one is frame rule. Similarly, the state "Robo holds coffee" in the resulting state, it would depend on whether it was holding coffee in the previous state and its action:

$$\begin{aligned} &rhc^1 \leftarrow rhc^0 \wedge Act \neq dc \\ &rhc^1 \leftarrow Act^0 = puc \end{aligned}$$

where first is frame rule, and second is causal rule.

37.3 Planning Graphs

The planning graphs give better heuristics and consists of sequence of levels, for *time steps* in plan. Each level has literals (constant values) which have become true because of previous action, and each level has preconditions for the next action. The planning graphs represent the *actions* as well as *in-actions*. The following example demonstrates the application of planning graphs.

Example 37.1 *Problem of having a 'Biryani' and eating Biryani.*

$$
\begin{aligned}
&init(have(biryani))\\
&Goal(have(biryani) \wedge eaten(biryani))\\
&Action(eat(biryani)\\
&\qquad Precond : have(biryani)\\
&\qquad Effect : \neg have(biryani) \wedge eaten(biryani))\\
&Action(bake(biryani)\\
&\quad Precond : \neg have(biryani)\\
&\quad Effect : have(biryani))
\end{aligned}
$$

The planning graph for these actions is shown in figure 37.1. The box action in the figure indicate the *mutual exclusions* of actions.

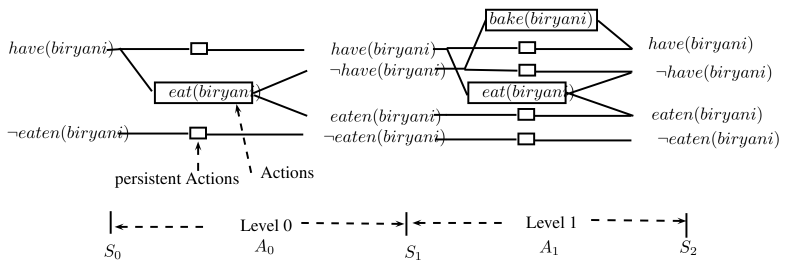

Figure 37.1: A Planning Graph for "Have a biryani and eat the biryani."

In the planning graph all the actions A_i at level i contains all actions that are applicable at state S_i, along with constraints saying which part of actions cannot be executed. Every S_i level contains all literals that could result from any choice of actions at A_{i-1}, along with constraint saying which part of actions cannot be executed.

The graph plan provides lesser complexity because it does not require choosing about all actions. It just records impossibility of certain choices using *mutex*, i.e., through mutual exclusion links. This results to complexity as a function of low polynomial complexity of actions and literals.

37.4 Planning with Propositional Logic

Consider the graph/tree shown in figure 37.2, with s_0 as start state, and $a_0 \ldots a_{n-1}$ as actions representing a path leading to the goal state s_n. If s_0, $a_0 \ldots a_{n-1}$, s_n are considered as propositional expressions, then

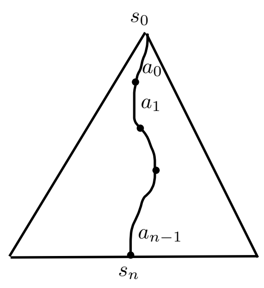

Figure 37.2: Propositional Planning.

$$p = s_0 \wedge (a_0 \wedge a_1 \wedge \cdots \wedge a_{n-1}) \wedge s_n \tag{37.1}$$

is a propositional expression representing the path to goal from start state. If there is only one goal then p is the only path to goal, and all other propositions $p^{'}$ are false. The equation 37.1 can always be transformed into a *CNF* (conjuctive normal form) or *SAT* expression. Thus, planning through propositional logic is to find a satisfiability expression, comprising start state, path and goal state, i.e. logical sentence equal to,

$$\textit{initial state} \wedge \textit{all proposition action descriptions} \wedge \textit{goal}.$$

In other words, a model that satisfies a sentence will assign true to all actions that are part of a correct plan. If the planning problem is unsolvable then there is no sentence that is satisfiable. Following example demonstrates this planning.

Example 37.2 *Flight Planning.*

Initial plan: Plane p_1 at *DEL* (Delhi) and P_2 at *CAL* (Calcutta). The actions should correspond to flying these planes, so that the goal: "p_1 at CAL, and p_2 at DEL", is satisfied.

The initial state (0) is represented by

$$At(p_1, DEL)^0 \wedge At(p_2, CAL)^0$$

Since the propositional logic has no *closed world* (all the propositions not specified in the world are false), it is also necessary to show that initially the planes p_1, p_2 are not CAL, DEL, respectively, i.e.,

$$\neg At(p_1, CAL)^0 \wedge \neg At(p_2, DEL)^0.$$

The goal also need to be specified with a particular time step. Since, is not known how many time steps it would consume to iterate the action, a worst case time limit need to be specified so that either the goal is reached with in that time limit, else the solution is terminated as failure.

We test the following assertion for goal at time $T = 0$ (at start),

$$At(p_1, CAL)^0 \wedge At(p_2, DEL)^0.$$

If that fails, apply certain actions and again try it at time $T = 1$, and so on, up to time $T = T_{max}$. This is shown by the algorithm 1.

Algorithm 1 Planning with Propositional Logic.

```
1: $INPUT : Problem$; a planning problem
2:          $T_{max}$; an upper limit for plan length
3: for $T = 0$ to $T_{max}$ do
4:    $cnf, mapping \leftarrow translate\text{-}to\text{-}SAT(problem, T)$
5:    $assignment \leftarrow SAT\text{-}Solver(cnf)$
6:    if Assignment is not null then
7:       Return $Extract\text{-}solution(asignment, mapping)$
8:    end if
9: end for
10: Return Failure
11: End
```

37.4.0.1 Encoding Action Descriptions

We have propositional symbol for each occurrence. For the plane p_1 to be at CAL, there is proposition:

$$At(p_1, CAL) \Leftrightarrow (At(p-1, CAL)^0 \wedge \neg fly(p_1, CAL, DEL)) \\ \vee (fly(p_1, DEL, CAL)^0 \wedge At(P1, DEL)^0) \quad (37.2)$$

There ought to be a plan that achieves the goal at $T = 1$, now suppose CNF is:

Initial state $\wedge$ *successor state axioms* $\wedge$ *goal*[1]

that is, goal is true at $T = 1$, we check and verify that:

$$fly(P_1, DEL, CAL)^0 \wedge fly(p_2, CAL, DEL)^0$$

is model, and other assignments are false.

The proposition fly having general form as $fly(p, a_1, a_2)$, and time T, and let number of propositions are represented by $|p|$, and objects (a_1, a_2, etc.) are O, then complexity expression is:

$$\begin{aligned} fly(p, a_1, a_2) &\Rightarrow T \times |p| \times |a_1| \times |a_2| \\ &\Rightarrow T \times p \times |a|^2 \\ &\Rightarrow T \times |Act| \times |o|^p \end{aligned}$$

here p is *arity* of function, o is number of objects a_1, a_2, etc. We note that the complexity is exponential.

References

[1] Chowdhary K.R. (2020) Logic and Reasoning Patterns. In: Fundamentals of Artificial Intelligence. Springer, New Delhi. `https://doi.org/10.1007/978-81-322-3972-7_15`