CS360 Homework 10– Solution

SAT-based Planning

1) In class, we have seen how to formulate a planning problem as a SAT (= satisfiablity) problem. The formulation allows the SAT solver to find plans that can execute actions in parallel, as long as all ways of sequentializing these parallel actions result in valid plans, that is, none of the actions deletes a precondition of another one, and none of the actions add a proposition that another action deletes. How would you modify the formulation if we want to find plans that execute actions in parallel as long as there exists at least one way of sequentializing theses parallel actions that results in a valid plan?

Answer:
We change the definition of conflicting actions and then ensure that conflicting actions cannot be executed in the same timestep. We say that a set of actions $A = \{X_1, \ldots, X_n\}$ is conflicting iff (=if and only if) there is an action $X_i \in A$ that deletes a proposition that an action $X_j \in A$ adds, or, for all orderings of the actions in A, there is an action that deletes the precondition of a later one. To ensure that conflicting actions cannot be executed in the same timestep, for each set $\{X_1, \ldots, X_n\}$ of conflicting actions and for each timestep t, we add the clause $\neg X_{1,t} \vee \cdots \vee \neg X_{n,t}$.

Note that, even though the SAT-based planner can execute more actions in parallel with this new formulation, it can still disallow some sets of actions from being executed in parallel, even if there is a way of sequentializing them. For instance, suppose there are three actions A, B, and C, that can all be individually executed in a state S. Suppose A deletes a precondition of C, but B adds the same precondition, and executing A, B, and C in this order results in a valid plan. Our new formulation would treat A, B, and C as a set of conflicting actions (since A deletes a proposition that B adds).

Another example would be that, in a timestep t in the beginning of which a proposition X is false, one can execute an action A that adds X followed by an action B that has X as a precondition but our formulation would not allow one to execute A and B in parallel because B has X as a precondition but X is not true in the beginning of timestep t.

A more general solution to this problem (that allows the parallel execution of all sets of actions that can be sequentialized to find a valid plan) would result in a very complicated formulation.

2) You want to find an operator sequence of length at most 2 that solves the following planning problem:

Start state:

```
In(SF),
Connected(LA,SF), Connected(SF,LA), Connected(LA,SD), Connected(SD,LA)
```

Goal state:

```
In(LA)
```

Operator:

```
Go(x,y):
Preconditions = {In(x), Connected(x,y)}
Effects = {-In(x), In(y)}
```

Formulate the corresponding propositional satisfiability problem by following the approach given in the lectures.

Answer:

[Set of Constraints 1]
$\text{In(SF,0)} \wedge \neg\, \text{In(LA,0)} \wedge \neg\, \text{In(SD,0)} \wedge$

[Set of Constraints 2]
$\text{In(LA,2)} \wedge$

[Set of Constraints 3]
$(\text{Go(SF,LA,0)} \Rightarrow \text{In(SF,0)}) \wedge$
$(\text{Go(LA,SD,0)} \Rightarrow \text{In(LA,0)}) \wedge$
$(\text{Go(SD,LA,0)} \Rightarrow \text{In(SD,0)}) \wedge$
$(\text{Go(LA,SF,0)} \Rightarrow \text{In(LA,0)}) \wedge$
$(\text{Go(SF,LA,1)} \Rightarrow \text{In(SF,1)}) \wedge$
$(\text{Go(LA,SD,1)} \Rightarrow \text{In(LA,1)}) \wedge$
$(\text{Go(SD,LA,1)} \Rightarrow \text{In(SD,1)}) \wedge$
$(\text{Go(LA,SF,1)} \Rightarrow \text{In(LA,1)}) \wedge$

[Set of Constraints 4]
$\neg(\text{Go(SF,LA,0)} \wedge \text{Go(LA,SD,0)}) \wedge$
$\neg(\text{Go(SF,LA,0)} \wedge \text{Go(LA,SF,0)}) \wedge$
$\neg(\text{Go(LA,SD,0)} \wedge \text{Go(SF,LA,0)}) \wedge$
$\neg(\text{Go(LA,SD,0)} \wedge \text{Go(SD,LA,0)}) \wedge$
$\neg(\text{Go(LA,SD,0)} \wedge \text{Go(LA,SF,0)}) \wedge$
$\neg(\text{Go(SD,LA,0)} \wedge \text{Go(LA,SD,0)}) \wedge$
$\neg(\text{Go(SD,LA,0)} \wedge \text{Go(LA,SF,0)}) \wedge$
$\neg(\text{Go(LA,SF,0)} \wedge \text{Go(SF,LA,0)}) \wedge$
$\neg(\text{Go(LA,SF,0)} \wedge \text{Go(LA,SD,0)}) \wedge$

$\neg(\text{Go(LA,SF,0)} \wedge \text{Go(SD,LA,0)}) \wedge$
$\neg(\text{Go(SF,LA,1)} \wedge \text{Go(LA,SD,1)}) \wedge$
$\neg(\text{Go(SF,LA,1)} \wedge \text{Go(LA,SF,1)}) \wedge$
$\neg(\text{Go(LA,SD,1)} \wedge \text{Go(SF,LA,1)}) \wedge$
$\neg(\text{Go(LA,SD,1)} \wedge \text{Go(SD,LA,1)}) \wedge$
$\neg(\text{Go(LA,SD,1)} \wedge \text{Go(LA,SF,1)}) \wedge$
$\neg(\text{Go(SD,LA,1)} \wedge \text{Go(LA,SD,1)}) \wedge$
$\neg(\text{Go(SD,LA,1)} \wedge \text{Go(LA,SF,1)}) \wedge$
$\neg(\text{Go(LA,SF,1)} \wedge \text{Go(SF,LA,1)}) \wedge$
$\neg(\text{Go(LA,SF,1)} \wedge \text{Go(LA,SD,1)}) \wedge$
$\neg(\text{Go(LA,SF,1)} \wedge \text{Go(SD,LA,1)}) \wedge$

(The following constraints are added only if we want to allow at most one action to be executed per time slice. If we do not add these constraints, we are allowing more actions to be executed in parallel, as long as they can be executed in any order.)

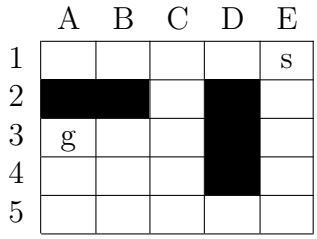
$\neg(\text{Go(SF,LA,0)} \wedge \text{Go(SD,LA,0)}) \wedge$
$\neg(\text{Go(SD,LA,0)} \wedge \text{Go(SF,LA,0)}) \wedge$
$\neg(\text{Go(SF,LA,1)} \wedge \text{Go(SD,LA,1)}) \wedge$
$\neg(\text{Go(SD,LA,1)} \wedge \text{Go(SF,LA,1)}) \wedge$

[Set of Constraints 5]
$(\text{In(SF,1)} \Rightarrow \text{Go(LA,SF,0)} \vee \text{In(SF,0)} \wedge \neg\ \text{Go(SF,LA,0)}) \wedge$
$(\text{In(LA,1)} \Rightarrow \text{Go(SF,LA,0)} \vee \text{Go(SD,LA,0)} \vee \text{In(LA,0)} \wedge \neg\ \text{Go(LA,SD,0)} \wedge \neg\ \text{Go(LA,SF,0)}) \wedge$
$(\text{In(SD,1)} \Rightarrow \text{Go(LA,SD,0)} \vee \text{In(SD,0)} \wedge \neg\ \text{Go(SD,LA,0)}) \wedge$
$(\text{In(SF,2)} \Rightarrow \text{Go(LA,SF,1)} \vee \text{In(SF,1)} \wedge \neg\ \text{Go(SF,LA,1)}) \wedge$
$(\text{In(LA,2)} \Rightarrow \text{Go(SF,LA,1)} \vee \text{Go(SD,LA,1)} \vee \text{In(LA,1)} \wedge \neg\ \text{Go(LA,SD,1)} \wedge \neg\ \text{Go(LA,SF,1)}) \wedge$
$(\text{In(SD,2)} \Rightarrow \text{Go(LA,SD,1)} \vee \text{In(SD,1)} \wedge \neg\ \text{Go(SD,LA,1)}) \wedge$

[Set of Constraints 6]
$(\neg\ \text{In(SF,1)} \Rightarrow \text{Go(SF,LA,0)} \vee \neg\ \text{In(SF,0)} \wedge \neg\ \text{Go(LA,SF,0)}) \wedge$
$(\neg\ \text{In(LA,1)} \Rightarrow \text{Go(LA,SD,0)} \vee \text{Go(LA,SF,0)} \vee \neg\ \text{In(LA,0)} \wedge \neg\ \text{Go(SF,LA,0)} \wedge \neg\ \text{Go(SD,LA,0)}) \wedge$
$(\neg\ \text{In(SD,1)} \Rightarrow \text{Go(SD,LA,0)} \vee \neg\ \text{In(SD,0)} \wedge \neg\ \text{Go(LA,SD,0)}) \wedge$
$(\neg\ \text{In(SF,2)} \Rightarrow \text{Go(SF,LA,1)} \vee \neg\ \text{In(SF,1)} \wedge \neg\ \text{Go(LA,SF,1)}) \wedge$
$(\neg\ \text{In(LA,2)} \Rightarrow \text{Go(LA,SD,1)} \vee \text{Go(LA,SF,1)} \vee \neg\ \text{In(LA,1)} \wedge \neg\ \text{Go(SF,LA,1)} \wedge \neg\ \text{Go(SD,LA,1)}) \wedge$
$(\neg\ \text{In(SD,2)} \Rightarrow \text{Go(SD,LA,1)} \vee \neg\ \text{In(SD,1)} \wedge \neg\ \text{Go(LA,SD,1)})$

Breadth-First Search

3) A 4-neighbor gridworld is given below. In which order does breadth-first search (with a sensible node pruning strategy) expand the cells when searching from s to g? Ties are broken in lexicographic order. That is, A1 is preferred over A2 and B1, and A2 is preferred over B1.



Answer:

The node pruning strategy of breadth-first search is to prune nodes whenever some node in the search tree is already labeled with the same state.

Expansion order: (E1), (D1, E2), (C1, E3), (B1, C2, E4), (A1, C3, E5), (B3, C4, D5), (A3). The parentheses group states based on their depth in the search tree. Depending on the tie-breaking strategy, the order of expansions can change, but only within a group.