Lab 8 Planning

SUSTC

PDDL and Air Cargo Transportation Problem

- PDDL:
  - Planning Domain Definition Language
- PDDL description of an air cargo transportation planning problem:

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Coding Example of PDDL

```
class PDLL:
    """
    PDLL used to define a search problem
    It stores states in a knowledge base consisting of first order logic statements
    The conjunction of these logical statements completely define a state
    """

    def __init__(self, initial_state, actions, goal_test):
        self.kb = FolKB(initial_state)
        self.actions = actions
        self.goal_test_func = goal_test

    def goal_test(self):
        return self.goal_test_func(self.kb)

    def act(self, action):
        """
        Performs the action given as argument
        Note that action is an Expr like expr('Remove(Glass, Table)') or expr('Eat(Sandwich)')
        """
        action_name = action.op
        args = action.args
        list_action = first(a for a in self.actions if a.name == action_name)
        if list_action is None:
            raise Exception("Action '{}' not found".format(action_name))
        if not list_action.check_precond(self.kb, args):
            raise Exception("Action '{}' pre-conditions not satisfied".format(action))
        list_action(self.kb, args)
```

Coding Example of PDDL

```
def air_cargo():
    init = [expr('At(C1, SFO)'),
            expr('At(C2, JFK)'),
            expr('At(P1, SFO)'),
            expr('At(P2, JFK)'),
            expr('Cargo(C1)'),
            expr('Cargo(C2)'),
            expr('Plane(P1)'),
            expr('Plane(P2)'),
            expr('Airport(JFK)'),
            expr('Airport(SFO)')]

    def goal_test(kb):
        required = [expr('At(C1 , JFK)'), expr('At(C2 ,SFO)')]
        for q in required:
            if kb.ask(q) is False:
                return False
        return True

    ## Actions
    #  Load
    precond_pos = [expr("At(c, a)"), expr("At(p, a)"), expr("Cargo(c)"), expr("Plane(p)"), expr("Airport(a)")]
    precond_neg = []
    effect_add = [expr("In(c, p)")]
    effect_rem = [expr("At(c, a)")]
    load = Action(expr("Load(c, p, a)"), [precond_pos, precond_neg], [effect_add, effect_rem])

                                        ⋮

    unload = Action(expr("Unload(c, p, a)"), [precond_pos, precond_neg], [effect_add, effect_rem])

                                        ⋮

    fly = Action(expr("Fly(p, f, to)"), [precond_pos, precond_neg], [effect_add, effect_rem])

                                        ⋮

    return PDLL(init, [load, unload, fly], goal_test)
```

Planning with Search

- Revisit tree search and graph search

```
def tree_search(problem, frontier):
    """Search through the successors of a problem to find a goal.
    The argument frontier should be an empty queue.
    Don't worry about repeated paths to a state. [Figure 3.7]"""
    frontier.append(Node(problem.initial))
    while frontier:
        node = frontier.pop()
        if problem.goal_test(node.state):
            return node
        frontier.extend(node.expand(problem))
    return None


def graph_search(problem, frontier):
    """Search through the successors of a problem to find a goal.
    The argument frontier should be an empty queue.
    If two paths reach a state, only use the first one. [Figure 3.7]"""
    frontier.append(Node(problem.initial))
    explored = set()
    while frontier:
        node = frontier.pop()
        if problem.goal_test(node.state):
            return node
        explored.add(node.state)
        frontier.extend(child for child in node.expand(problem)
                        if child.state not in explored and
                        child not in frontier)
    return None
```

Coding Example of Planning with Tree Search 1/2

```
from planning import *
from utils import *
import copy

airc=air_cargo()
aira=airc.actions

cset=['C1','C2']
pset=['P1','P2']
lset=['SFO','JFK']

aload=[]
for c in cset:
    for p in pset:
        for loc in lset:
            aload.append(expr('Load'+'('+c+','+p+','+loc+')'))

aunload=[]
for c in cset:
    for p in pset:
        for loc in lset:
            aunload.append(expr('Unload'+'('+c+','+p+','+loc+')'))

afly=[]
for p in pset:
    for loc1 in lset:
        for loc2 in lset:
            afly.append(expr('Fly'+'('+p+','+loc1+','+loc2+')'))

myacts= aload[:]
myacts.extend(aunload[:])
myacts.extend(afly[:])

# ##############################################################
```

Initialization: generate all possible actions

Coding Example of Planning with Tree Search 2/2

```
# ###########################################################
def plan_search(pdllp,myacts,frontier):

    frontier.append(pdllp)

    for i in range(64):
        print(i)

        node=frontier.pop()

        if node.goal_test():
            print('Succeed')
            return node

        def pexpand(prob,myacts):
            cnodes=[]
            for ia in myacts:
                try:
                    ipro=copy.deepcopy(prob)
                    ipro.act(ia)
                except:
                    print('',end='')
                else:
                    cnodes.append(ipro)

            return cnodes

        childnode=pexpand(node,myacts)
        frontier.extend(childnode)

    return node

mysol=plan_search(airc,myacts,FIFOQueue())
print(mysol.kb.clauses)
```

Tree search: search all states resulted from possible actions. This one cannot solve the problem. Why?