CS 541 Midterm exam, Fall 2000, with model student answers

1) *What* are *the characteristics of problem domains that would be most appropriate for using description logics and plan ontologies?*

Description logics are particularly useful in domains that need reasoning about types of goals and plans, where these generalizations are based on hierarchal organizations of objects in the domain.

Plan ontologies are useful to represent plans in a shareable and reusable manner that is independent of the implementation of particular systems or approaches.

2) *What does it mean for* a *planning graph to level off? Under what circumstances does this happen and why?*

a. A planning graph levels off at the point when two adjacent levels are identical, in terms of propositions and mutexes.
b. This happens when we reach a proposition level Q such that all future levels have the same set of propositions as Q. This should happen eventually because:
- the number of propositions that can be created by STRIPS-like operators is finite, and
- if a proposition appears at some level in the graph it will appear in all subsequent levels, and
- a pair of propositions that are not mutually exclusive in
some level will not be marked as mutually exclusive in any future level.

3) *Give an example of how a linear inequality could be used to encode a temporal plan constraint*

Let V _i(t) and V ~(t) represent two different actions A_i and A~ respectively,
V _i(t) + V ~(t) <= 1 means that A_i and A~ cannot happen at the same time "t".

4) *Some researchers have* a *domain-independent algorithm for plan existence that they believe runs in quadratic time for planning problems in which no functions are allowed in the operators. They have analyzed the behavior of the algorithm on the blocks world and towers of hanoi. What* is *your advice to them?*

1. There is always a plan for those two domains, so an algorithm that is tested on just
those two could conceivably run in constant time. This leads us to suspect that the good performance seen may disappear when the algorithm is tested on more challenging problems.
2. The currently known computational complexity for plan existence varies from constant time to EXPSP ACE-complete, depending on the following conditions:
a. delete lists?
b. negative preconditions?
c. propositional predicates?
d. fixed planning operator or giving them as part of input.
The researchers should test on more problems that are hard according to these conditions before publishing.

5) *What kinds of planners use critics? Describe their role in the planning algorithm.*

HTN planners use critics after expanding the plan at the next level in order to find and fix problems in the expanded plan. HTN planners use many kinds of critics, including variable binding and redundancy critics, temporal constraints critics, and resource critics.

6) *How would a skeletal planner like MOLGEN detect that a plan is not feasible?*

a. In finding a skeletal plan, it checks whether the skeletal plan's goals include the goal of the current problem. If not, it is not feasible.

b. In plan refinement, a sub-plan is not feasible if either:
1. The goal under consideration does not match its utility.
(e.g. in MOLGEN this is done by checking the SEL-RULE and APPLICATION-USEFULNESS slots).
2. It doesn't satisfy the environmental suitability conditions.
This is done by checking the rules that relates its properties with necessary or forbidden properties of the environment (e.g. the properties of molecules and sample).
3. In addition, it may use heuristics that estimate the time,
cost, etc, of a sub-plan, and compare them with the current requirements.

7) *What is the standard approach for making a causal link planner deal with conditional effects? Describe briefly how it works.*

Conditional effects are effects of the form When <antecedent> <consequent>

To implement this, we modify the standard causal link planner in two ways:
a. When selecting an action to satisfy a goal, we also consider unifying the goal with a consequent of a conditional effect. If they do unify, the consequent's antecedent is put into the agenda of propositions to be satisfied.
b. If the consequent threatens another action, there is an
additional alternative for resolution (besides promotion and demotion), called confrontation: add the negation of the antecedent to the agenda.

8) *If we can guarantee that a search control rule prunes nodes from a planner's search space, does this guarantee that the time (in seconds) to generate a plan will also be reduced? Why or why not?*

No. The evaluation of the search control rule may cost more time than it saves. For example, I propose the following search control rule; *prune all nodes that do not lead to a successful plan*, which we would evaluate by attempting to generate a plan. Clearly, evaluating this search control rule will require at least as much time as an approach that uses no pruning at all, since it amounts an exhaustive search of the plan space.

9) *Suppose that a case-based planner is given the following problem:*

```
GOAL: (at-airport Joe)
INITIAL STATE: (tall Joe) (handsome Joe)
(single Joe) (has-car Joe)
```

and suppose it has two past cases in its library:

```
CASE 1
GOAL: (at-airport Jack)
INITIAL STATE: (has-car Jack)

CASE 2
```

```
GOAL: (at-airport Joe)
INITIAL STATE: (tall Joe) (handsome Joe)
(single Joe) "
```

How would such planner know that CASE 1 *is the right past case to retrieve even though CASE* 2 *has much more commonality with the new problem?*

By the indexing mechanism, the initial state is pruned to the set of features relevant to the solution. The system identifies the set of weakest preconditions necessary to achieve that goal, and project back its weakest preconditions into the literals in the initial state (foot- printing). In the given example, in case 2, the foot-printed initial state for the goal is nil because all the literals in the initial state are not relevant. The total match value is 1 (O for initial state, 1 for goal state). In case 1, the foot printed initial state is (has-car Jack), and the match value to the new problem is 2 (1 for initial state, 1 for goal state). Thus, the planner can know that case 1 should be retrieved because it has a higher match value than case 2.

10) Why would a heuristic search planner that uses a non-admissible heuristic be faster than a planner that uses an admissible heuristic?

The use of non-admissible heuristics jeopardizes the optimality of the search. However, the most optimal solution is not always required and a stronger, non-admissible heuristics can be used to quickly find suitable solutions. HSP uses two heuristics. One is the additive search, which provides an estimate of the number of steps required to get the goal. It is non-admissible. The other is max heuristics that focuses only on the most difficult subgoals, ignoring all others. Although the first one is not admissible it is more informative than the other and tends to drive the search in a good direction. That is why HSP with a non-admissible heuristics could be faster than a planner that uses an admissible heuristics.

11)) *How can Dynamic Relevance be used to prune the search space of forward chaining planners? Why are some plans with irrelevant sequences still considered in TLPlan?*

Dynamic Relevance looks at a sequence of actions R and tries to locate a subsequence R' which, when executed alone, can achieve the complete final state of the entire sequence. The complement of such a subsequence, R ", consists of all of the actions in R that are not in R'. All of the actions in R" are irrelevant, since the final state could be achieved without them. We can then prune the search space by ignoring nodes that lead us to create plans with irrelevant subsequences.

There are two sources of irrelevant sequences in TLPlan. First, TLPlan cannot identify a subsequence R’ that must be reordered to be equivalent to the supersequence R. Some irrelevant sequences can sneak in this way. Second, TLPlan uses a greedy algorithm that cannot detect subsequences that are irrelevant because of multiple "roots." The best way to describe this is an example: Initial state: -PI, -P2, -P3 Al asserts PI A2 asserts P2 A3 asserts P3 A4 asserts -PI, ":'P2 Final state: P3

Clearly, only A3 is relevant. However, TLPlan will try to construct a subsequence by leaving out AI. It will then also have to leave out A4, which depends on the execution of AI. Then the final state will be (P2 P3), not (P3), making Al appear to be relevant. The same thing will happen when TLPlan's greedy algorithm tries to construct a subsequence without A2. Bacchus and Teh claim that this can be corrected, but that the time cost grows exponentially with the number of "roots" that can be accounted for.