General notes: Again, this is by no means representative of an actual final you might expect to see, in terms of length, difficulty, or content. But they are good practice for the type of questions you may actually encounter.

Search

Perform A* on the following tree, labeling the nodes in the order that they are expanded by the algorithm. Also fill in the f-value of each node that is put into the priority queue (and only those nodes). S indicates the start node, G indicates the goal node. Costs are labeled on the edges.

Note that in the very bottom node, the parent is not expanded, so the f value is not calculated.
STRIPS

There is a ferry that goes across the bay from Oakland to San Francisco. This ferry is capable of carrying one car from one side of the bay to the other. Ann’s car and Bob’s car are sitting on the Oakland side; each car must be loaded onto the ferry, which is empty and docked at the Oakland side, sailed across the bay, and unloaded in San Francisco.

We wish to formulate this problem in the language of STRIPS. We define the following objects:

¥ SF
¥ Oakland
¥ Ann’s car
¥ Bob’s car

Note that we have not defined a ferry object. Take this into account as you work through the problem.

a) What propositions would we have to define for this problem? (Hint: you will probably need six of them)

¥AUTO(x)
¥PLACE(x)
¥AT-FERRY(x)
¥ON-FERRY(x)
¥EMPTY-FERRY
¥AT(x, y)

b) What actions would we have to define for this problem? (Hint: you will need 3; it will probably be easier if you think of a car as not being any place if it is on the ferry). List the preconditions, adds, and deletes for each action.

BOARD(x, y)
Preconditions: AUTO(x), PLACE(y), AT(x, y), AT-FERRY(y), EMPTY-FERRY
Adds: ON-FERRY(x)
Deletes: EMPTY-FERRY, AT(x, y)

SAIL(x, y)
Preconditions: PLACE(x), PLACE(y), AT-FERRY(x)
(we might also want to say that x cant equal y...) 
Adds: AT-FERRY(y)
Deletes: AT-FERRY(x)
DEBARK(x, y)

Preconditions: AUTO(x), PLACE(y), ON-FERRY(x), AT-FERRY(y)

Adds: EMPTY-FERRY, AT(x, y)

Deletes: ON-FERRY(x)

c) Represent the initial state in terms of the propositions you defined. You should need 8 propositions.

PLACE(SF)
PLACE(Oakland)
AUTO(A)
AUTO(B)
AT(A, Oakland)
AT(B, Oakland)
AT-FERRY(Oakland)
EMPTY-FERRY

d) Represent the goal state in terms of the propositions you defined.

AT(A, SF)
AT(B, SF)

e) Give a valid STRIPS plan that achieves the goal, i.e.-- list the actions in the order that they should be performed.

BOARD(A, Oakland)
SAIL(Oakland, SF)
DEBARK(A, SF)
SAIL(SF, OAKLAND)
BOARD(B, Oakland)
SAIL(Oakland, SF)
DEBARK(B, SF)
PROLOG

Translate the following into logical formulas, then into Horn clauses. Use the predicates American, Weapon, Country, Hostile, Sells, Missile, Enemy, Owns, and Criminal.

Any American who sells weapons to hostile nations is a criminal. Nono is a country. Nono is an enemy of America. Nono has some missiles, and all of its missiles were sold to it by Colonel West. Colonel West is an American. Missiles are weapons. An enemy of America is considered "hostile."

Logical formulas:
\[ \forall x, y, z \ ((\text{American}(x) \land \text{Sells}(x, z, y) \land \text{Country}(z) \land \text{Hostile}(z) \land \text{Weapon}(y)) \Rightarrow \text{Criminal}(x)) \]

Country(Nono)
Enemy(Nono, America)
\exists x \ (\text{Owns}(Nono, x) \land \text{Missile}(x))
\forall x \ (\text{Owns}(Nono, x) \land \text{Missile}(x) \Rightarrow \text{Sells}(\text{CW}, \text{Nono}, x))
American(CW)
\forall x \ (\text{Missile}(x) \Rightarrow \text{Weapon}(x))
\forall x \ (\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x))

Horn clauses:
\[ \neg\text{American}(x) \lor \neg\text{Sells}(x, y, z) \lor \neg\text{Country}(z) \lor \neg\text{Hostile}(z) \lor \text{Weapon}(y) \lor \neg\text{Criminal}(x) \]

Country(Nono)
Enemy(Nono, America)
Owns(Nono, Sk1)
Missile(Sk1)
\neg\text{Owns}(Nono, x) \lor \neg\text{Missile}(x) \lor \text{Sells}(\text{CW}, \text{Nono}, x)
American(CW)
\neg\text{Missile}(x) \lor \text{Weapon}(x)
\neg\text{Enemy}(x, \text{America}) \lor \text{Hostile}(x) \]
Using the language of prolog, prove that Colonel West is a criminal.

Note that we are not doing this the way a Prolog prover would actually go about this — were taking informed steps.

1) :- Criminal(CW)
2) Criminal(x) :- American(x), Sells(x, z, y), Country(z), Hostile(z), Weapon(y)
3) Country(Nono) :-
4) Enemy(Nono, America) :-
5) Owns(Nono, Sk1) :-
6) Missile(Sk1) :-
7) Sells(CW, Nono, x) :- Owns(Nono, x), Missile(x)
8) American(CW) :-
9) Weapon(x) :- Missile(x)
10) Hostile(x) :- Enemy(x, America)
11) :- American(CW), Sells(CW, z, y), Country(z), Hostile(z), Weapon(y)
    unify 1, 2 substituting CW/x
12) :- American(CW), Sells(CW, Nono, y), Hostile(Nono), Weapon(y)
    unify 11, 3 substituting Nono/z
13) Sells(CW, Nono, Sk1) :-
    unify 5, 6, 7 substituting Sk1/x
14) :- American(CW), Hostile(Nono), Weapon(Sk1)
    unify 12, 13 substituting Sk1/y
15) :- Hostile(Nono), Weapon(Sk1)
    unify 8, 14
16) :- Hostile(Nono), Missile(Sk1)
    unify 9, 15 substituting Sk1/x
17) :- Hostile(Nono)
    unify 6, 16
18) :- Enemy(Nono, America)
    unify 17, 10 substituting Nono/x
19) :-
    unify 4, 18
Predicate logic representation in the real world

Try to translate these English sentences into first-order predicate logic. Some of these are probably more obscure than anything you’d see on the final; just use your judgement, have some fun, and compare your answers to the solution set...depending on your interpretation, you might come up with different formulations. But that’s ok...

1. Nothing compares to you.
\[\exists x \ (\text{Compares}(x, \text{YOU}))\]
There is nothing in the universe that compares to YOU.

2. Nobody knows the trouble I’ve seen.
\[\exists x \forall y \ ((\text{Trouble}(y) \land \text{Seen}(\text{ME}, y) \land \text{Person}(x)) \supset \text{Knows}(x, y))\]
No person knows every one of the troubles I’ve seen.

3. Every cloud has a silver lining.
\[\forall x \exists y \ (\text{Cloud}(x) \supset (\text{Has}(x, y) \land \text{SilverLining}(y)))\]
If it’s a cloud, then it has its own silver lining.

4. It’s all fun and games.
\[\forall x \ (\text{Fun}(x) \land \text{Games}(x))\]
Everything in the universe is both ‘fun’ and ‘games’.

5. We learned calculus in the days of yore.
Tricky. We should probably define some notion of what exactly ‘yore’ is. Let’s say that ‘yore’ is a time interval. \(y_0\) is some time point that is the beginning of ‘yore’ and \(y_f\) the end. Then for any time \(y\) such that \(\text{Before}(y_0, y)\) and \(\text{Before}(y, y_f)\) hold, we can say that \(y\) is a point of time in the days of yore.

But we don’t want to say that everybody learned calculus at the same point of time in the days of yore. So we could define a predicate \(\text{We}\) such that \(\text{We}(x)\) means that \(x\) is in the group of ‘We’. Thus,
\[\forall x \exists y \ (\text{We}(x) \supset \text{Before}(y_0, y) \land \text{Before}(y, y_f) \land \text{Learned}(x, \text{Calculus}))\]
To use this to prove anything, we might have to define some standard time axioms as well.

6. That’s life in the city.
Since we don’t know what ‘that’ refers to, we can do one of two things. Either define find out what ‘that’’s antecedent is and use that as a constant, or simply generalize and say that there is something in the universe that can be described as ‘life in the city’. :)
\[\exists x (\text{LifeInTheCity}(x))\]
Knowledge representation

Consider the following set of facts:

University classes are taught by teachers. Stanford classes last one quarter. Berkeley classes last one semester. English 3a is a class at Stanford. Computer science courses at Stanford are generally difficult, but they are a lot of fun. CS 105 is a computer science course at Stanford that is useful. CS 105 is an introductory course; introductory courses are university classes, are generally low-workload, and are easy, unless they are graduate-level introductory classes, in which case they are hard. CS 101 is an introductory class at Berkeley.

a) Draw a semantic network to represent this knowledge set. What can you conclude about CS 105?

CS 105 is useful; it is fun, lasts 1 quarter, has a low workload, and is taught by teachers. CS 105 may be easy or hard, since from CS Classes it inherits being not easy, while from Introductory Classes it inherits being easy; depending on how we deal with multiply-inherited attributes, we would assign easy or hard to CS 105, but not both, and not neither.
b) We want to know whether what difficulty level CS 101 is. Create a truth-maintenance table for this task, including only those predicates that are necessary to determine whether a course is easy or not. Write out generally what you can conclude about the easiness of a class, given various states of INness and OUTness of the cells in the table, using the minus superscript to indicate OUT. Fill in the table for CS 101, then conclude whether CS 101 is easy or not.

Only a simple table is needed (filled in for CS 101):

<table>
<thead>
<tr>
<th>IntroClass</th>
<th>GradIntroClass</th>
<th>CSClass</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

CSClass is out because we don’t know whether CS 101 is a computer science course (CS could stand for cognitive science, for example).

You can define rules involving any combination of INness and OUTness; here are some of the interesting ones:

\[
\text{IntroClass}(x) \land \text{GradIntroClass}(x) \land \neg \text{CSClass}(x) \Rightarrow \neg \text{Easy}(x)
\]

If all we know about a class is that it’s an intro class, we can conclude its easy.

\[
\neg \text{IntroClass}(x) \land \neg \text{GradIntroClass}(x) \land \neg \text{CSClass}(x) \Rightarrow \neg \text{Easy}(x)
\]

If all we know is that it’s a CS class, we can conclude its hard.

\[
\neg \text{IntroClass}(x) \land \neg \text{GradIntroClass}(x) \land \neg \text{CSClass}(x) \Rightarrow \neg \text{Easy}(x)
\]

We do run into a problem again with the conflicting demands of IntroClass and CSClass. However, we can create rules that impose an ordering. In this case, we have specified that if it’s a CS Class, it is always hard. If it’s an IntroClass, its easy only if we know it’s not a CS Class (or a GradIntroClass).

Bayesian networks

Consider the following Bayesian network:

```
A -- B --> C
    |     |     |
    |     |     |
    v     v     v
     D --> C  
```

We are given:

\[\begin{align*}
P(A = \text{true}) &= 0.6 \\
P(B = \text{true} | A = \text{true}) &= 0.4 \\
P(B = \text{true} | A = \text{false}) &= 0.9 \\
P(C = \text{true} | A = \text{true}) &= 0.2 \\
P(C = \text{true} | A = \text{false}) &= 0.7 \\
P(D = \text{true} | B = \text{true}, C = \text{true}) &= 0.9 \\
P(D = \text{true} | B = \text{true}, C = \text{false}) &= 0.7 \\
P(D = \text{true} | B = \text{false}, C = \text{true}) &= 0.4 \\
P(D = \text{true} | B = \text{false}, C = \text{false}) &= 0.1
\end{align*}\]
Determine the following probabilities:

a) \( P(D = \text{true}) \)

\[
P(D) = P(D | B, C) P(B, C)
\]

\[
P(D = \text{true}) = P(D = \text{true} | B = \text{true}, C = \text{true}) P(B = \text{true}, C = \text{true}) + P(D = \text{true} | B = \text{true}, C = \text{false}) P(B = \text{true}, C = \text{false}) + P(D = \text{true} | B = \text{false}, C = \text{true}) P(B = \text{false}, C = \text{true}) + P(D = \text{true} | B = \text{false}, C = \text{false}) P(B = \text{false}, C = \text{false})
\]

\[
P(B, C) = P(B, C | A) P(A)
\]

We note that \( B \) and \( C \) are conditionally independent, given \( A \), by the structure of the bayes net. Thus

\[
P(B, C | A) = P(B | A) P(C | A)
\]

So, \( P(B = \text{true}, C = \text{true}) \)

\[
= P(B = \text{true} | A = \text{true}) P(C = \text{true} | A = \text{true}) P(A = \text{true}) + P(B = \text{true} | A = \text{false}) P(C = \text{true} | A = \text{false}) P(A = \text{false})
\]

\[
= (0.4)(0.2)(0.6) + (0.9)(0.7)(0.4) = 0.3
\]

Similarly,

\( P(B = \text{true}, C = \text{false}) = 0.3 \)

\( P(B = \text{false}, C = \text{true}) = 0.1 \)

\( P(B = \text{false}, C = \text{false}) = 0.3 \)

Thus, \( P(D = \text{true}) = (0.9)(0.3) + (0.7)(0.3) + (0.4)(0.1) + (0.1)(0.3) = 0.55 \)

b) \( P(A = \text{true} | D = \text{true}) \)

Applying Bayes Theorem,

\[
P(A = \text{true} | D = \text{true}) = P(D = \text{true} | A = \text{true}) P(A = \text{true}) / P(D = \text{true})
\]

We calculate \( P(D = \text{true} | A = \text{true}) \) in part c.

\( P(D = \text{true}) = 0.55, \text{ from part a} \)

\( P(A = \text{true}) = 0.6, \text{ as given} \)

\[
P(A = \text{true} | D = \text{true}) = (0.392)(0.6) / (0.55) = 0.428
\]
c) $P(D = \text{true} | A = \text{true})$

By the definition of conditional probability,

$$P(D | A) = \frac{P(D, A)}{P(A)}$$

and

$$P(D, A) = P(D, A | B, C) P(B, C)$$

We know that $A$ and $D$ are conditionally independent given $B$ and $C$, so

$$P(D, A | B, C) = P(D | B, C) P(A | B, C)$$

Applying Bayes' theorem,

$$P(A | B, C) = \frac{P(B, C | A) P(A)}{P(B, C)}$$

we calculated $P(B, C)$ in part a; we also found that

$$P(B, C | A) = P(B | A) P(C | A)$$

because $B$ and $C$ are conditionally independent given $A$.

Thus,

$$P(A = \text{true} | B = \text{true}, C = \text{true}) = (0.4)(0.2) * (0.6) / (0.3) = 0.16$$
$$P(A = \text{true} | B = \text{true}, C = \text{false}) = (0.4)(0.8) * (0.6) / (0.3) = 0.64$$
$$P(A = \text{true} | B = \text{false}, C = \text{true}) = (0.6)(0.2) * (0.6) / (0.1) = 0.72$$
$$P(A = \text{true} | B = \text{false}, C = \text{false}) = (0.6)(0.8) * (0.6) / (0.3) = 0.96$$

$$P(D = \text{true}, A = \text{true} | B = \text{true}, C = \text{true}) = (0.9)(0.16) = 0.144$$
$$P(D = \text{true}, A = \text{true} | B = \text{true}, C = \text{false}) = (0.7)(0.64) = 0.448$$
$$P(D = \text{true}, A = \text{true} | B = \text{false}, C = \text{true}) = (0.4)(0.72) = 0.288$$
$$P(D = \text{true}, A = \text{true} | B = \text{false}, C = \text{false}) = (0.1)(0.96) = 0.096$$

$$P(D = \text{true}, A = \text{true}) = (0.144)(0.3) + (0.448)(0.3) + (0.288)(0.1) + (0.096)(0.3) = 0.235$$

$$P(D = \text{true} | A = \text{true}) = (0.235) / (0.6) = 0.392$$

d) $P(C = \text{true} | A = \text{true}, B = \text{false})$

Note that $B$ and $C$ are conditionally independent given $A$. So

$$P(C | A, B) = P(C | A)$$

$$P(C = \text{true} | A = \text{true}) = 0.2$$
Neural networks

Consider the following neural network with one hidden node and two output nodes. Each node implements a threshold function — i.e., each node outputs a 1 if the dot product of its weights and its inputs is greater than or equal to the threshold value stored in that node. Let’s say we want to model the following function: given an input of (1, 0), output (0, 1); given an output of (0, 1), output (1, 0); we don’t care about any other inputs. Provide a set of values for the weights and thresholds that would allow the network to behave in this manner. Show that the correct output is achieved for each of the two input vectors.

One possible weight setting would be the following:

Weights in layer 1:

\[ w_1 = 1 \]
\[ w_2 = 0.5 \]
\[ \text{threshold} = 0.9 \]

Weights in layer 2:

For O2, \[ w = -1 \]
\[ \text{threshold} = -0.5 \]

For O3, \[ w = 1 \]
\[ \text{threshold} = 0.5 \]

For an input of (1, 0), H1 yields a dot product of \((1)(1) + (0)(0.5) = 1\), which is greater than 0.9; thus, H1 will output a 1.

Given an input of 1, O2 will produce a dot product of \((1)(-1) = -1\), which is less than the threshold of -0.5, so O2 will output a 0.

O3 will produce a dot product of \((1)(1) = 1\), which is greater than 0.5, thus O3 outputs a 1.

Thus, the output for the entire network is (0, 1), which is what we wanted.
For an input of (0, 1), H1 yields a dot product of (0)(1) + (1)(0.5) = 0.5, which is less than the threshold of 0.9, thus H1 outputs a 0.

Given an input of 0, O2 will produce a dot product of 0, which is greater than the threshold of -0.5, so O2 will output a 1.

O3 will produce a dot product of 0, which is less than 0.5, so O3 outputs 0.

Thus the output for the entire network is (1, 0), which is what we wanted.

Vision

Perform scene analysis on the following drawing, such that a consistent labeling for each line is provided.

The following is one example of a consistent labeling.