CS 188 Spring 2002 — Introduction to AI — Stuart Russell — Midterm Solutions

1. (20 pts.) True/False

(a) (2) *A system must think like a human in order to pass the Turing Test reliably.*
False. It only has to *act* like a human.

(b) (3) *An agent that senses only partial information about the state cannot be perfectly rational.*
False. Rationality and omniscience are different concepts.

(c) (3) *Every definite clause is a Horn clause.*
True. A definite clause has exactly one positive literal; a Horn clause at most one.

(d) (3) *The clauses $P(F(A,G(A)))$ and $P(F(x,y))$ resolve to produce the empty clause.*
False. Although they unify, they are not complementary (neither is negated).

(e) (3) *$h(n)=0$ is an admissible heuristic for the 8-puzzle.*
True. 0 is always an underestimate of the true cost for the 8-puzzle.

(f) (3) *There are STRIPS planning problems for which a partially ordered solution exists but no totally ordered solution.*
False. A partially ordered solution has the property that every linearization is a solution.

(g) (3) *The number of open conditions is an admissible heuristic for partial-order plans.*
False. Open conditions can be satisfied by existing steps (no extra cost), and some actions might satisfy more than one open condition because they have more than one useful effect.

2. (20+10 pts.) Search
A basic Brio$^{\text{TM}}$ railway set contains the pieces shown in Figure 1. The task is to connect all *these pieces into a* single *railway that has* no loose ends *where a train could run off onto the floor and no* overlapping *tracks.*

(a) (10) *Suppose that the pieces fit together* exactly *with no slack. Give a precise formulation of the task as a search problem.*
Initial state: one arbitarily selected piece (say a straight piece).
Successor function: for any open peg, add any piece type from remaining types. (You can add to open holes as well, but that isn't necessary as all complete tracks can be made by adding to pegs.) For a curved piece, add *in either orientation*; for a fork, add *in either orientation* and (if there are two holes) connecting *at either hole*. It's a good idea to disallow any overlapping configuration, as this terminates hopeless configurations early. (Note: there is no need to consider open holes, because in any solution these will be filled by pieces added to open pegs.)
Goal test: all pieces used in a single connected track, no open pegs or holes, no overlapping tracks.
Step cost: one per piece (actually, doesn't really matter).

(b) (5) *Identify a suitable uninformed search algorithm for this task and explain your choice.*
All solutions are at the same depth, so depth-first search would be appropriate. (One could also use depth-limited search with limit $n-1$, but strictly speaking it's not necessary to do the work of checking the limit because states at depth $n-1$ have no successors.) The space is very large, so uniform-cost and breadth-first would fail, and iterative deepening simply does unnecessary extra work. There are many repeated states, so it might be good to use a closed list.

(c) (5) *Explain briefly why removing any one of the "fork" pieces makes the problem unsolvable.*
A solution has no open pegs or holes, so every peg is in a hole, so there must be equal numbers of pegs and holes. Removing a fork violates this property. There are two other "proofs" that are acceptable: 1) a similar argument to the effect that there must be an even number of "ends"; 2) each fork creates two

tracks, and only a fork can rejoin those tracks into one, so if a fork is missing it won't work. The argument using pegs and holes is actually more general, because it also applies to the case of a three-way fork that has one hole and three pegs or one peg and three holes. The "ends" argument fails here, as does the fork/rejoin argument (which is a bit handwavy anyway).

(d) (5 extra credit) *Give an upper bound on the total size of the state space defined by your formulation.*
The maximum possible number of open pegs is 3 (starts at 1, adding a two-peg fork increases it by one). Pretending each piece is unique, any piece can be added to a peg, giving at most $12+(2\cdot16)+(2\cdot2)+(2\cdot2\cdot2) = 56$ choices per peg. The total depth is 32 (there are 32 pieces), so an upper bound is $168^{32}/(12!\cdot16!\cdot2!\cdot2!)$ where the factorials deal with permutations of identical pieces. One could do a more refined analysis to handle the fact that the branching factor shrinks as we go down the tree, but it is not pretty.

(e) (5 extra credit) *Now consider the real problem, in which pieces don't fit together exactly but allow for up to 10 degrees of rotation to either side of the "proper" alignment. Explain how to formulate the problem so it could be solved by simulated annealing.*
We need a complete-state formulation, so let the states be any configuration of the 32 pieces. We need discrete operators, which remove any piece and reconnect it anywhere else; and continuous operators, which change the angle of any joint by some real-valued angle. (Unfortunately, especially for closed loops, moving one joint necessarily moves some or all of the others, so we need to be careful how this is carried out.) We need a heuristic function, which could penalize open pegs, amount of overlapping track, sum of joint distortion angles, number of disjoint connected components.

3. (20 pts.) Propositional Logic
Consider the following sentence:

$$[(Food \Rightarrow Party) \lor (Drinks \Rightarrow Party)] \Rightarrow [(Food \land Drinks) \Rightarrow Party]$$

(a) (6) *Determine whether this sentence is valid, satisfiable (but not valid), or unsatisfiable, using enumeration. (You may abbreviate the proposition symbols in your answer.)*
A simple truth table has eight rows, and shows that the sentence is true for all models and hence valid.

(b) (6) *Convert the left-hand and right-hand sides of the main implication into CNF, showing each step, and explain how the results confirm your answer to (a).*
For the left-hand side we have:

$$(Food \Rightarrow Party) \lor (Drinks \Rightarrow Party)$$
$$(\neg Food \lor Party) \lor (\neg Drinks \lor Party)$$
$$(\neg Food \lor Party \lor \neg Drinks \lor Party)$$
$$(\neg Food \lor \neg Drinks \lor Party)$$

and for the right-hand side we have

$$(Food \land Drinks) \Rightarrow Party$$
$$\neg(Food \land Drinks) \lor Party$$
$$(\neg Food \lor \neg Drinks) \lor Party$$
$$(\neg Food \lor \neg Drinks \lor Party)$$

The two sides are identical in CNF, and hence the original sentence is of the form $P \Rightarrow P$, which is valid for any P.

(c) (8) *Prove your answer to (a) using resolution.*
To prove that a sentence is valid, prove that its negation is unsatisfiable. I.e., negate it, convert to CNF, use resolution to prove a contradiction. We can use the above CNF result for the LHS.

$$\neg[[(Food \Rightarrow Party) \lor (Drinks \Rightarrow Party)] \Rightarrow [(Food \land Drinks) \Rightarrow Party]]$$
$$[(Food \Rightarrow Party) \lor (Drinks \Rightarrow Party)] \land \neg[(Food \land Drinks) \Rightarrow Party]$$
$$(\neg Food \lor \neg Drinks \lor Party) \land Food \land Drinks \land \neg Party$$

Each of the three unit clauses resolves in turn against the first clause, leaving an empty clause.

4. (20 pts.) First-order logic
For each of the following sentences in English, decide if the accompanying first-order logic sentence is a good translation. If not, explain why not and correct it.

(a) "Any apartment in Berkeley has lower rent than some apartments in Palo Alto."

$$\forall x\ [Apt(x) \land In(x, Berkeley)] \Rightarrow \exists y\ [[Apt(y) \land In(x, PaloAlto)] \Rightarrow (Rent(x) < Rent(y))]$$

Incorrect.
Simple error: should say $In(y, PaloAlto)$ not $In(x, PaloAlto)$.
Classic error: using $\Rightarrow$ with $\exists$. Should be $\land$.

(b) "There is exactly one apartment in Palo Alto with rent below \$1000."

$$\exists x\ \ Apt(x) \land In(x, PaloAlto) \land \forall y\ [Apt(y) \land In(y, PaloAlto) \land (Rent(y) < Dollars(1000))] \Rightarrow (y = x)$$

Incorrect.
The sentence does not say that x has rent below \$1000; it is true if there are no such apartments.
There should be parentheses enclosing $\forall y\ \ldots (y = x)$.
We gave full credit for noticing either of these errors.

(c) "If an apartment is more expensive than all apartments in Berkeley, it must be in San Francisco."

$$\forall x\ \ Apt(x) \land [\forall y\ \ Apt(y) \land In(y, Berkeley) \land (Rent(x) > Rent(y))] \Rightarrow In(x, SanFrancisco)$$

Incorrect.
Classic error: using $\land$ with $\forall$. Should be $\Rightarrow (Rent(x) > Rent(y))$.

(d) (6) *Which of the first-order logic sentences above may be converted to definite clauses?*
Intuitively, all three have just one positive conclusion, so all three are (sets of) definite clauses. Just to make sure, convert to CNF:

$$\neg Apt(x) \lor \neg In(x, Berkeley) \lor \neg Apt(f(x)) \lor \neg In(x, PaloAlto) \lor (Rent(x) < Rent(f(x)))$$
$$Apt(g) \land In(g, PaloAlto) \land (\neg Apt(y) \lor \neg In(y, PaloAlto) \lor \neg(Rent(y) < Dollars(1000)) \lor (y = g))$$
$$\neg Apt(x) \lor \neg Apt(f(x)) \lor \neg In(f(x), Berkeley) \lor \neg(Rent(x) > Rent(f(x))) \lor In(x, SanFrancisco)$$

5. (20 pts.) Planning
Consider a robot whose operation is described by the following STRIPS operators:

$$Op(\textsc{Action}{:}Go(x,y), \textsc{Precond}{:}At(Robot,x), \textsc{Effect}{:}\neg At(Robot,x) \land At(Robot,y))$$
$$Op(\textsc{Action}{:}Pick(o), \textsc{Precond}{:}At(Robot,x) \land At(o,x), \textsc{Effect}{:}\neg At(o,x) \land Holding(o))$$
$$Op(\textsc{Action}{:}Drop(o), \textsc{Precond}{:}At(Robot,x) \land Holding(o), \textsc{Effect}{:}At(o,x) \land \neg Holding(o))$$

(a) (5) *The operators allow the robot to hold more than one object. Show how to modify them with an EmptyHand predicate for a robot that can hold only one object.*
EmptyHand is not affected by *Go*, so we modify just the *Pick* and *Drop* operators:

$$Op(\textsc{Action}{:}Pick(o), \textsc{Precond}{:}EmptyHand() \land At(Robot,x) \land At(o,x),$$
$$\textsc{Effect}{:}\neg EmptyHand() \land \neg At(o,x) \land Holding(o))$$
$$Op(\textsc{Action}{:}Drop(o), \textsc{Precond}{:}At(Robot,x) \land Holding(o),$$
$$\textsc{Effect}{:}EmptyHand() \land At(o,x) \land \neg Holding(o))$$

Notice that STRIPS does not allow negated preconditions, so we could not use $\neg Holding(p)$ as a precondition for $Pick(o)$; this is why we need *EmptyHand*. Also, we cannot use $\neg EmptyHand$ as a precondition of $Drop(o)$; but this is no problem because we already have $Holding(o)$ as a precondition.

(b) (5) *Assuming that these are the only actions in the world, write a successor-state axiom for EmptyHand.*
In English, the hand is empty after doing an action if it was empty before and the action was not a successful *Pick*; or if an object was dropped.

$$\begin{aligned} EmptyHand(Result(a,s)) \quad &\Leftrightarrow \quad [(EmptyHand(s) \land \neg\exists\, o,x\ \ (At(Robot,x) \land At(o,x) \land a = Pick(o))) \\ &\lor \quad (Holding(o,s) \land a = Drop(o))] \end{aligned}$$

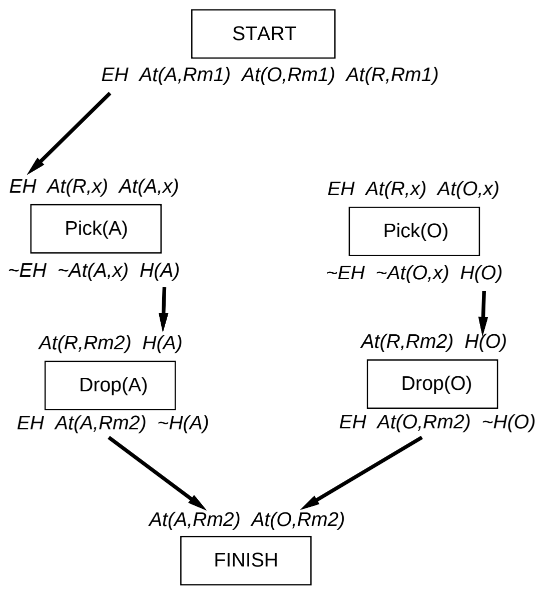

Fig. 2: Partial-order plan exhibiting a threat by $Pick(O)$ to the $EmptyHand$ link from $Start$ to $Pick(A)$.

(c) (10) *Suppose the initial state has*

$$At(Apple, Room_1) \wedge At(Orange, Room_1) \wedge At(Robot, Room_1)$$

and the goal is

$$At(Apple, Room_2) \wedge At(Orange, Room_2)$$

Consider the operation of the POP algorithm on this problem, for the STRIPS operators as modified in part (a). Diagram the partial plan that has been constructed at the point where the first threat to a causal link appears. Explain what the threat is and how it can be resolved (if at all).

One possibility is shown in Figure 2. From the empty plan, we add the two drop actions to put the apple and orange in Room 2. Each requires that the robot be holding the object to be dropped, so we add the two *Pick* actions. Each has a precondition that the hand be empty. As soon as we achieve one of those from the *Start* state, it is threatened by the other *Pick* action—essentially, the threat arises because the robot can hold only one object. We cannot order $Pick(O)$ before *Start*, so it is ordered after $Pick(A)$.

Another possible solution involves adding $Go(x, Room_2)$ actions instead of *Pick* actions. As soon as we satisfy the $At(Robot, x)$ precondition of one *Go* action using the $At(Robot, Room_1)$ effect of *Start*, we have a *possible* threat to this causal link from the $\neg At(Robot, x)$ of the other *go* action (see p.357 for a discussion of possible threats). It becomes a definite threat after we satisfy the $At(Robot, x)$ precondition of the second *Go* action using the $At(Robot, Room_1)$ effect of *Start*. Because the threats apply in both directions, there is no way to resolve them both and the search must backtrack.

It is important to remember that an *effect* of one action threatens a complementary *precondition* of another action that is protected by a *causal link* covering an interval into which the first action might fall. Many people wrote down a complete plan (which was not asked for) and described conflicts between two simultaneous *Pick* actions rather than mentioning any specific causal link. Many people constructed plans by forward search from the start, which is not how the POP algorithm works (although it is one possible way to construct plans).