Answer key for 2nd Midterm Exam

Tuesday November 18
75 minutes == 75 points

1. *15 points*

   Show the backed-up values for all the nodes in the following game tree and show the branches that are pruned by alpha-beta. For each branch pruned, explain briefly why alpha-beta prunes it. Follow the convention used in the textbook to examine the branches in the tree from left to right.

   Above shows the backed up values for each node that is examined. No value is shown for nodes that were pruned since we did not need to look at those nodes.

   Upward pointing nodes are the results of your actions where, presumably, you picked the best (highest valued) option available to you. Downward pointing nodes are the results of actions made by your opponent who, presumably, picked the option that was best for him (and, consequently, worst for you). So the up nodes should show the largest number directly under it and the down nodes should show the smallest value directly below it.

   There were three branches that were pruned, marked A, B and C.

   A is pruned because 3 is less than 5. This is worth explaining in more detail. The leftmost max node at level three has two choices, 5 and 3. Given these two choices, Max will pick 5. We did not look at the node on level five below cut-point A. Why?

   Consider two examples. If that node was 1, that's even smaller so Min would have picked it instead of 3. But it doesn't matter to us - we already know one of our options is 5 so anything less than that isn't worth thinking about. So if the node we didn't evaluate is less than 3, it doesn't change our decision because we'll pick the 5.

   On the other hand, suppose the value in the pruned node is 10. That's a lot better than the 5 we currently have. But it doesn't matter - Min gets to pick first and would have to pick between 3 and 10, so they'll pick 3. Meaning our choice would just be between 3 and 5, of which we'd pick 5. So if the node we didn't evaluate is greater than 3, it doesn't change our decision because Min will pick 2.

   The second prune point is at B, right below level three. This is Max's choice. Max can either pick 8 or whatever is in the branch we pruned. If the number is less than 8, we won't pick it. If it's higher than 8 (say, 11), we'll pick it but it won't matter because that would make Min's choice at level 2 between 5 and 11, and they'll pick 5. So if it's less than 8, it won't change the value at level two, and it's more, it still won't change it.

   The reasoning for prune point C is the same. Min already has 8 as its first option. Its second option is either 9 or what's in the branch we

depth-first search with backtracking. Why? In earlier classes, we said that DFS was often considered one of the worst search algorithms, so why use it here? Why not use breadth-first search or A*? There is a reason but you needed to explain it. DFS works well due to certain properties of a CSP (i.e., guaranteed depth limit) but no better than any other algorithm (and potentially worse) unless you use a good CSP-oriented heuristic (namely MRV).

(b) Why is a complete-state formulation a good choice for CSP when using local search? explain briefly.

The complete state formulation is required for local search algorithms such as hill climbing, simulated annealing, and genetic algorithms. These algorithms explore (often randomly) the state space and thus need the entire state description to be available. This is true in the complete state formulation but not in the incremental formulation.

You use complete-state formulations because local search algorithms (hill climbing, simulated annealing, beam search, genetic algorithms, etc.) won't work otherwise. These algorithms explore the state space (often randomly and with doubling back) and thus need the entire state description to be available. This is true in the complete state formulation but not in the incremental formulation.

A few people gave the answer that complete-state formulations let you use the min-conflicts heuristic but did not explain why. To know how many constraints have been violated, you need assignments for every variable, which is only true for complete-state formulations. Note that this doesn't actually answer why it is good for local search problems - you'd still need to say that the min-conflicts heuristic has been shown to work well on many CSPs.

Many people answered a different question, why would you want to use local search algorithms. This was not the question asked. Local search algorithms can solve n-queen problems quickly and can do online learning with solution patching but these are not reasons why local search algorithms want complete-state formulations.

3. *5 points*
Show the results of applying the unification algorithm to the following expressions:

(a) UNIFY[p(x, f(u,x)), p(f(y,A), f(z,f(B,z))]

x/f(y,A), u/z, y/B, z/A

(b) UNIFY [p(g(A), h(u,v)), p(g(w), h(w,B))]
w/A, u/A, v/B

4. *5 points*

Convert to Conjunctive Normal Form the following expression:

$\forall p[pet(p) \wedge \exists c[owner(c,p) \vee feeds(c,p)]] \Rightarrow happy(p)$

(a) implication elimination:
$\forall p \neg[pet(p) \wedge \exists c[owner(c,p) \vee feeds(c,p)]] \vee happy(p)$

(b) de Morgan:
$\forall p \neg pet(p) \vee \forall c \neg[owner(c,p) \vee feeds(c,p)] \vee happy(p)$

(c) de Morgan:
$\forall p \neg pet(p) \vee \forall c[\neg owner(c,p) \wedge \neg feeds(c,p)] \vee happy(p)$

(d) remove $\forall$
$\neg pet(p) \vee [\neg owner(c,p) \wedge \neg feeds(c,p)] \vee happy(p)$

(e) regroup/move terms around
$\neg pet(p) \vee happy(p) \vee [\neg owner(c,p) \wedge \neg feeds(c,p)]$

(f) distributivity of $\vee$
$(\neg pet(p) \vee happy(p) \vee \neg owner(c,p)) \wedge (\neg pet(p) \vee happy(p) \vee -feeds(c,p))$

(g) drop $\wedge$
$(\neg pet(p) \vee happy(p) \vee \neg owner(c,p))$
$(\neg pet(p) \vee happy(p) \vee -feeds(c,p))$

The purpose of this question was to see if you could remove the conjunctions ($\wedge$) from the statement. Many people simply removed the implication and stopped. Some of those people would have gotten the answer had they continued working on it, some would not because they had their parentheses in the wrong place (in terms of logical errors, dropping the parentheses too early was the main problem).

A few people got all the way to the next-to-last step and then left the statement as two CNF statements joined by a conjunction. This is not CNF form and cannot be used in resolution. The statement needs to be broken into two statements.

5. *20 points*

(a) Write the following statements in predicate calculus, using the following predicates:

Alone(x) : x is alone
Friend(x,y) : x is a friend of y
Grumpy(x) : x is grumpy
Helpful(x,y) : x is helpful to y
Sibling(x,y) : x is a sibling of y

i. Anyone who has any sibling is not alone.
$\forall x \forall y \; sibling(x, y) \Rightarrow \neg alone(x)$

ii. If someone is grumpy, nobody is helpful to him.
$\forall w \; grumpy(w) \Rightarrow \neg \exists r \; helpful(w, r)$

or, equivalenty,
$\forall w \forall r \; grumpy(w) \Rightarrow \neg helpful(w, r)$

iii. Friends are helpful.
$\forall x \forall y \; friend(x, y) \Rightarrow helpful(x, y)$

iv. John has a sibling or a friend.
$\exists f \; sibling(John, f) \vee friend(John, f)$

(b) Convert them to conjunctive normal form.

i. $\neg sibling(z, y) \vee \neg alone(z)$

ii. $\neg grumpy(w) \vee \neg helpful(w, r)$

iii. $\neg friend(x, y) \vee helpful(x, y)$

iv. $sibling(John, S0) \vee friend(John, S0)$

(c) Prove by resolution that "If John is grumpy, John is not alone."

$grumpy(John) \Rightarrow \neg alone(John)$

We negate it and add to the KB

$\neg[\neg grumpy(John) \vee \neg alone(John)]$

which produces

5. $grumpy(John)$

6. $alone(John)$

We then do resolution:

5. with 2.	$\{w/John\}$	to obtain	7. $\neg helpful(John, r)$
7. with 3.	$\{x/John\}$	to obtain	8. $\neg friend(John, y)$
8. with 4.	$\{y/S0\}$	to obtain	9. $sibling(John, S0)$
9. with 1.	$\{John/z, S0/y\}$	to obtain	10. $\neg alone(John)$
10. with 6.		to obtain	NIL

6. *20 points*
For each of the following sentences, decide if the logic sentence given is a correct translation of the English sentence or not. If not explain briefly why not and correct it. There can be more than one mistake in each expression.

   (a) There is only one red mushroom.
   $\exists x\ mushroom(x) \wedge red(x) \wedge \neg\ [\exists y\ mushroom(y) \wedge red(y) \wedge x = y]$
   Error: It should be $\neg x = y$

   (b) Any red mushroom costs less than any purple mushroom.
   $\forall x\ [mushroom(x) \wedge red(x)] \Rightarrow [\exists y\ mushroom(y) \wedge purple(y) \wedge cost(x) < cost(y)]$
   Error: y should be universally quantified

   (c) Some purple mushrooms are tastier than some red mushrooms.
   $\exists x \forall y\ [mushroom(x) \wedge purple(x) \wedge mushroom(y) \wedge red(y)] \Rightarrow tastier(x, y)$
   Error: instead of $\forall y$ it should be $\exists y$. Also $\Rightarrow$ should be *wedge* instead.

   (d) There is a mushroom that is tastier than any other mushroom.
   $\forall x \exists y\ [mushroom(x) \wedge mushroom(y)] \Rightarrow tastier(y, x)]$
   Error: $\exists y$ should be outside $\forall x$. $mushroom(y)$ should be outside the premise.

   (e) All mushrooms contain at least one vitamin.
   $\forall x\ [mushroom(x) \wedge \exists y\ vitamin(y)] \Rightarrow in(x, y)$
   Error: instead of $\wedge$ it should be $\Rightarrow$ and instead of $\Rightarrow$ it should be $\wedge$