Homework 2 Solutions

1. (Ch. 6) Discuss how well the standard approach to game playing would apply to games such as tennis, pool, and croquet, which take place in a continuous physical state space. Include a discussion of how you would discretize the problem and how well that discretization would be expected to work. Consider whether these games can be modeled correctly as turn-taking games, and describe why or why not. Describe whether randomized strategies would be helpful for any of these games, and why or why not.

The space of actions of these new games is now continuous. For example, in pool, the cueing direction, angle of elevation, speed, and point of contact with the cue ball are all continuous quantities. The simplest solution for these types of games is to discretize the action space and then apply standard methods. This might work for tennis (modeled crudely as alternating shots with speed and direction), but for games such as pool and croquet it is likely to fail miserably because small changes in direction have large effects on action outcome. Instead, one must analyze the game to identify a discrete set of meaningful local goals, such as “potting the 4-ball” in pool or “laying up for the next hoop” in croquet. Then, in the current context, a local optimization routine can work out the best way to achieve each local goal, resulting in a discrete set of possible choices. Typically, these games are stochastic, so the backgammon model is appropriate provided that we use sampled outcomes instead of summing over all outcomes.

Whereas pool and croquet are modeled correctly as turn-taking games, tennis is not. While one player is moving to the ball, the other player is moving to anticipate the opponent’s return. In general, it may be reasonable to derive randomized strategies in tennis so that the opponent cannot anticipate where the ball will go.

2. (Ch. 7) Discuss what is meant by optimal behavior in the wumpus world. Show that the definition of the PL-WUMPUS-AGENT in Figure 7.19 is not optimal, and suggest ways to improve it.

Optimal behavior means achieving an expected utility that is as good as any other agent problem. The PL-WUMPUS-AGENT is clearly non-optimal when it chooses a random move (and may be non-optimal in other branches of its logic). One example: in some cases when there are many dangers (breezes and smells) but no safe move, the agent chooses at random. A more thorough analysis should show when it is better to do that, and when it is better to go home and exit the wumpus world, giving up on any chance of finding the gold. Even when it is best to gamble on an unsafe location, our agent does not distinguish degrees of safety – it should choose the unsafe square which contains a danger in the fewest number of possible models. These refinements are hard to state using a logical agent, but we will see later in this semester that a probabilistic agent can handle them.

3. (Ch. 9) Work problem 9.18 a-c (page 318, “Horses are animals…”).

a. \( \forall x \, \text{Horse}(x) \Rightarrow \text{Animal}(x) \)
\( \forall x, h \, \text{Horse}(x) \land \text{HeadOf}(h, x) \Rightarrow \exists y \, \text{Animal}(y) \land \text{HeadOf}(h, y) \)

b. In the following, A comes from the first sentence in part a., while B, C, and D come from the second sentence in part a. Below, \( G \) and \( H \) are Skolem constants.

A. \( \neg \text{Horse}(y) \lor \text{Animal}(x) \)
B. \( \text{Horse}(G) \)
C. \( \text{HeadOf}(H, G) \)
D. \( \neg \text{Animal}(y) \lor \neg \text{HeadOf}(H, y) \)

c. Resolve D and C to yield \( \neg \text{Animal}(G) \). Resolve this result with A to give \( \neg \text{Horse}(G) \). Resolve this result with B to obtain a contradiction.
4. (Ch. 9) Work problem 9.19 a-g (pages 318 – 319, “Here are two sentences…”).

   a. (A) translates to “For every natural number there is some other natural number that is smaller than or equal to it.” (B) translates to “There is a particular natural number that is smaller than or equal to any natural number.”

   b. Yes, (A) is true under this interpretation. You can always pick the number itself for the “some other” number.

   c. Yes, (b) is true under this interpretation. You can pick 0 for the “particular natural number”.

   d. No, (A) does not logically entail (B).

   e. Yes, (B) logically entails (A).

   f. To prove that (B) entails (A), we start with a knowledge base containing (B) and the negation of (A), which we will call (-A):

   
   
   \begin{align*}
   (-A) & \quad \neg F_1(x) \geq y \\
   (B) & \quad x \geq F_2(x)
   \end{align*}

   This time the resolution goes through, with the substitution \{x/ F_1, y/ F_2(F_1)\}, thereby yielding False, and proving that (B) entails (A).

   g. We want to try to prove via resolution that (A) entails (B). To do this, we set our knowledge base to consist of (A) and the negation of (B), which we will call (-B), and try to derive a contradiction. First we have to convert (A) and (-B) to canonical form. For (-B), this involves moving the \(\neg\) in past the two quantifiers. For both sentences, it involves introducing a Skolem function:

   \begin{align*}
   (A) & \quad x \geq F_1(x) \\
   (-B) & \quad \neg F_2(y) \geq y
   \end{align*}

   Now we can try to resolve these two together, but the occur check (see page 277) rules out the unification. At first glance, it looks like the substitution should be \{x/ F_2(y), y/ F_1(x)\}, but that is equivalent to \{x/ F_2(y), y/ F_1(F_2(y))\}, which fails because \(y\) is bound to an expression containing \(y\). So the resolution fails, there are no other resolution steps to try, and therefore (B) does not follow from (A).

5. (Ch. 11) Work problem 11.4 a-d (pages 412 – 413, “Monkey-and-bananas…”).

   a. The initial state is:

   \[
   At(Monkey,A) \land At(Bananas,B) \land At(Box,C) \land Height(Monkey,Low) \land Height(Box,Low) \land Height(Bananas,High) \land Pushable(Box) \land Climbable(Box)
   \]

   b. The actions are:

   \[
   Action(Action: Go(x,y), Precond: At(Monkey,x), \\
   \text{Effect: } At(Monkey,y) \land \neg At(Monkey,x))
   \]

   \[
   Action(Action: Push(b,x,y), Precond: At(Monkey,x) \land Pushable(b), \\
   \text{Effect: } At(b,y) \land At(Monkey,y) \land \neg At(b,x) \land \neg At(Monkey,x))
   \]

   \[
   Action(Action: ClimbUp(b), Precond: At(Monkey,x) \land At(b,x) \land Climbable(b), \\
   \text{Effect: } On(Monkey,b) \land \neg Height(Monkey,High))
   \]

   \[
   Action(Action: Grasp(b), Precond: Height(Monkey,h) \land Height(b,h) \land At(Monkey,x) \land At(b,x), \\
   \text{Effect: } Have(Monkey,b))
   \]
Action(ACTION: ClimbDown(b), PRECOND: On(Monkey,b) ∧ Height(Monkey,High),
EFFECT: ¬On(Monkey,b) ∧ ¬Height(Monkey,High) ∧ Height(Monkey,Low))

Action(ACTION: UnGrasp(b), PRECOND: Have(Monkey,b)),
EFFECT: ¬Have(Monkey,b))

c. In situation calculus, the goal is a state s such that:

\[\text{Have(Monkey,Bananas,s)} \land (\exists x \text{At(Box,x,s,)} \land \text{At(Box,x,s)})\]

In STRIPS, we can only talk about the goal state; there is no way of representing the fact that there must be
some relation (such as equality of location of an object) between two states within the plan. So there is no way
to represent this goal.

d. Actually, we did include the Pushable precondition. This is an example of the qualification problem.

6. Using the ‘truth-table’ function, verify the last 6 equivalences in Figure 7.11 (page 210).

In all of the following, note that the last column is all ‘T’, which verifies these equivalences.

\[(\alpha \Rightarrow \beta) \equiv (\neg \alpha \lor \beta)\]

(truth-table "A => B <=> (not A) | B")

\[
\begin{array}{cccc}
A & B & A \Rightarrow B & (\neg A) \lor (A \Rightarrow B) \\
T & T & T & T \\
T & F & F & T \\
F & T & T & T \\
F & F & F & T \\
\end{array}
\]

NIL

\[(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))\]

(truth-table "(A <=> B) <=> ((A => B) ^ (B => A))")

\[
\begin{array}{cccccccc}
A & B & A \Leftrightarrow B & A \Rightarrow B & B \Rightarrow A & (A \Rightarrow B) \land (B \Rightarrow A) & (A \Leftrightarrow B) \Leftrightarrow ((A \Rightarrow B) \land (B \Rightarrow A)) \\
T & T & T & T & T & T & T \\
T & F & F & T & F & F & T \\
F & T & T & F & T & T & T \\
F & F & F & F & F & F & F \\
\end{array}
\]

NIL

\[\neg(\alpha \land \beta) \equiv (\neg \alpha \lor \neg \beta)\]

(truth-table "(not (A ^ B) <=> ((not A) | (not B)))")

\[
\begin{array}{cccccccc}
A & B & A \land B & (\neg A) \land (\neg A) & (\neg B) \land (\neg B) & (\neg (A \land B)) \Leftrightarrow ((\neg A) \land (\neg B)) \\
T & T & T & T & T & T \\
T & F & F & T & T & T \\
F & T & T & F & T & T \\
F & F & F & F & F & F \\
\end{array}
\]
\[
\neg(\alpha \lor \beta) \equiv (\neg \alpha \land \neg \beta)
\]

(truth-table "not(\(A \mid B\)) <=> ((not A) \lor (not B))")

| A | B | A | B | \neg(| A B) | \neg A | \neg B | (\neg (| A B)) <=\neg ((\neg A) \lor (\neg B)) |
|---|---|---|---|---|---|---|---|
| F | F | T | T | T | T | T | T |
| T | F | T | T | F | T | F | T |
| F | T | T | F | F | T | F | T |
| T | T | T | F | F | T | F | T |

NIL

\[(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))\]

(truth-table "((\alpha \land (B \lor G)) <=\lor ((\alpha \land B) \lor (\alpha \land G))")

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>G</th>
<th>B \lor G</th>
<th>A \lor (B \lor G)</th>
<th>A \lor B</th>
<th>A \lor G</th>
<th>(A \lor B) \land (A \lor G) &lt;=\lor ((A \lor B) \lor (A \lor G))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

NIL

\[(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))\]

(truth-table "((\alpha \lor (B \land G)) <=\land ((\alpha \lor B) \land (\alpha \lor G))")

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>G</th>
<th>B \land G</th>
<th>A \lor (B \land G)</th>
<th>A \lor B</th>
<th>A \lor G</th>
<th>(A \lor B) \lor (A \lor G) &lt;=\lor ((A \lor B) \land (A \lor G))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

NIL

7. Using the ‘validity’ function, decide whether the sentences in problem 7.8 (page 237) are valid, unsatisfiable, or both.

a. (validity "Smoke => Smoke")

VALID

b. (validity "Smoke => Fire")
SATISFIABLE

c. (validity "(Smoke => Fire) => ((not Smoke) => (not Fire))")
SATISFIABLE

d. (validity "Smoke | Fire | (not Fire)")
VALID

e. (validity "((Smoke ^ Heat) => Fire) <==> ((Smoke => Fire) | (Heat => Fire))")
VALID

f. (validity "(Smoke => Fire) => ((Smoke ^ Heat) => Fire)")
VALID

g. (validity "Big | Dumb | (Big => Dumb)")
VALID

h. (validity "(Big ^ Dumb) | (not Dumb)")
SATISFIABLE

8. You are given the following statements:

If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.

a. Use ‘setf’, ‘tell’, and ‘ask’, to generate axioms representing the above statements.

(setf kb (make-prop-kb))
#<a PROP-KB>
> (tell kb "Mythical => Immortal")
T
> (tell kb "~Mythical => ~Immortal ^ Mammal")
T
> (tell kb "Immortal | Mammal => Horned")
T
> (tell KB "Horned => Magical")
T

b. Is the unicorn mythical?

> (ask kb "Mythical")
NIL
> (ask kb "~Mythical")
NIL

We can’t deduce anything about whether it is mythical.

c. Is the unicorn magical?

> (ask kb "Magical")
T

Yes, the unicorn is magical.
d. Is the unicorn horned?

> (ask kb "Horned")
T

Yes, the unicorn is horned.

9. Use first-order logic to write axioms describing the following predicates: Grandchild, GreatGrandparent, Brother, Sister, Daughter, Son, Aunt, Uncle, BrotherInLaw, SisterInLaw, and FirstCousin. Enter these axioms into the provided LISP reasoning system, as well as the facts depicted in the family tree in Figure 8.5 (page 270). Debug your axioms by asking the LISP reasoning system who are Elizabeth’s grandchildren, Diana’s brothers-in-law, and Zara’s great-grandparents. Turn in the definitions of the axioms, as well as the LISP output to the above questions.

The axioms you define should use $\leftrightarrow$ rather than $\Rightarrow$, or define both directions of the implication in two statements, since otherwise you’re only imposing constraints, not writing a real definition. The sentences below use First Order Logic to define these axioms. We should be able to use ‘ask-patterns’ to pose queries to this knowledge base. Unfortunately, there appears to be an error (or at least incomplete documentation) on using ‘ask-patterns’ with a FOL knowledge base. I’m checking into this, and will get back with you.

The ‘ask-patterns’ horn clause version does work, where you define your kb as a horn knowledge base (i.e., (setf kb (make-horn-kb))). In this case, all the axioms must be in horn clause mode. In this case, the LISP statement to query your database is (ask-patterns kb '(Female $x)'), and it will print out all possible substitutions for x.

Also, note that you can input, edit, and store all of your axioms in a file (say, “myfile.lisp”), and then within clisp, just enter (load “myfile”), and the sentences will be loaded into LISP. This way, you don’t have to type everything from scratch in each working session.

Here are the clauses in FOL:

(setf kb (make-fol-kb))
tell kb "GrandChild(c,a) <-> exists(b,Child(c,b) ^ Child(b,a))")
tell kb "GreatGrandParent(a,d) <-> exists(b,exists(c,Child(d,c) | Child(c,b) | Child(b,a)))")
tell kb "Brother(x,y) <-> Male(x) ^ Sibling(x,y)")
tell kb "Sister(x,y) <-> Female(x) ^ Sibling(x,y)")
tell kb "Daughter(d,b) <-> Female(d) ^ Child(d,b)")
tell kb "Son(s,p) <-> Male(s) ^ Child(s,p)")
tell kb "AuntOrUncle(a,c) <-> exists(p,Child(c,p) ^ Sibling(a,p))")
tell kb "Aunt(a,c) <-> Female(a) ^ AuntOrUncle(a,c)")
tell kb "Uncle(u,c) <-> Male(u) ^ AuntOrUncle(a,c)")
tell kb "BrotherInLaw(b,x) <-> exists(m,Spouse(x,m) ^ Brother(b,m))")
tell kb "SisterInLaw(s,x) <-> exists(m,Spouse(x,m) ^ Sister(s,m))")
tell kb "FirstCousin(c,k) <-> exists(p,AuntOrUncle(p,c) ^ Parent(p,k))")
(tell kb "Child(William,Diana)")
tell kb "Child(William,Charles)")
tell kb "Child(Harry,Diana)")
tell kb "Child(Harry,Charles)")
tell kb "Child(Peter,Anne)")
tell kb "Child(Peter,Mark)")
tell kb "Child(Zara,Anne)")
tell kb "Child(Zara,Mark)")
tell kb "Child(Beatrice,Andrew)")
tell kb "Child(Beatrice,Sarah)")
(tell kb "Child(Eugenie,Andrew)")
(tell kb "Child(Eugenie,Sarah)")
(tell kb "Child(Diana,Spencer)")
(tell kb "Child(Diana,Kydd)")
(tell kb "Child(Charles,Elizabeth)")
(tell kb "Child(Charles,Philip)")
(tell kb "Child(Aanne,Elizabeth)")
(tell kb "Child(Aanne,Philip)")
(tell kb "Child(Andrew,Elizabeth)")
(tell kb "Child(Andrew,Philip)")
(tell kb "Child(Edward,Elizabeth)")
(tell kb "Child(Edward,Philip)")
(tell kb "Child(Elizabeth,Mum)")
(tell kb "Child(Margaret,George)")
(tell kb "Child(Margaret,Mum)")
(tell kb "Male(William)")
(tell kb "Male(Harry)")
(tell kb "Male(Peter)")
(tell kb "Female(Zara)")
(tell kb "Female(Beatrice)")
(tell kb "Female(Diana)")
(tell kb "Male(Charles)")
(tell kb "Female(Anne)")
(tell kb "Male(Mark)")
(tell kb "Male(Andrew)")
(tell kb "Female(Sarah)")
(tell kb "Male(Edward)")
(tell kb "Male(Spencer)")
(tell kb "Female(Kydd)")
(tell kb "Female(Elizabeth)")
(tell kb "Male(Philip)")
(tell kb "Female(Margaret)")
(tell kb "Male(Philip)")
(tell kb "Female(Mum)")
(tell kb "Spouse(Diana,Charles)")
(tell kb "Spouse(Aanne,Mark)")
(tell kb "Spouse(Andrew,Sarah)")
(tell kb "Spouse(Spencer,Kydd)")
(tell kb "Spouse(Elizabeth,Philip)")
(tell kb "Spouse(John,Mum)")