Assignment Chapter 6

1. Exercise 6-5. Using the `move` and `path` definitions for the knight's tour of Section 6.2.2, trace the execution of `pattern_search` on the goals:
   (a) `path(1,9)`
   (b) `path(1,5)`
   (c) `path(1,6)`
   When the `move` predicates are attempted in order, there is often looping in the search. Discuss loop detection and backtracking in this situation.
2. Exercise 4-8. Using the goal and start states of Figure 6.3, hand run the production system solution to the 8-puzzle:
   (a) In goal-driven fashion
   (b) In data-driven fashion
3. Exercise 4-9. Consider the financial advisor problem discussed in Chapters 2,3,4. Using predicate calculus as a representation language:
   (a) Write the problem explicitly as a production system.
   (b) Generate the state space and stages of working memory for the data-driven solution to the example in Chapter 3.

Solutions

Exercise 6-5. Using the `move` and `path` definitions for the knight's tour of Section 6.2.2, trace the execution of `pattern_search` on the goals:

(a) `path(1,9)`

Iteration	Current State	Goal State	Conflict Rules	Rule Fired
0	1	9	1,2	1
1	8	9	13,14	13
2	3	9	5,6	5
3	4	9	7,8	7
4	9	9		halt

(b) `path(1,5)`
(c) `path(1,6)`

Iteration	Current State	Goal State	Conflict Rules	Rule Fired
0	7	6	11,12	11
1	2	6	3,4	3
2	9	6	15,16	15
3	2	6	3,4	3
4	9	6	15,16	15
...	...	6	...	...

When the `move` predicates are attempted in order, there is often looping in the search. Discuss loop detection and backtracking in this situation.

Clearly we need loop detection in the production system search algorithm; using one of the search algorithms discussed in Chapter 3, we have it. Note also that the original query has a direct solution, missed by the algorithm. Best-first search can eliminate this problem.

Exercise 4-8. Using the goal and start states of Figure 6.3, hand run the production system solution to the 8-puzzle:

(a) In goal-driven fashion
(b) In data-driven fashion

Exercise 4-9. Consider the financial advisor problem discussed in Chapters 2,3,4. Using predicate calculus as a representation language:

Refer to the rules presented in Section 2-4.

(a) Write the problem explicitly as a production system.
Put rules 1-8 in production memory.
Put the functions minsavings and minincome in memory also, to be used whenever a rule requires these calculations to be performed.
Facts 9, 10, 11 will be used as needed.
Run the production system either goal- or data-driven.

(b) Generate the state space and stages of working memory for the data-driven solution to the example in Chapter 3.
Go through the rules in order until you produce the result "savings(X)" for some X. When no premise is yet known, as in the first three rules of the first iteration, go on to the next rule. Finally, in rule 4 we are asked about savings and dependents. There are no rules that can conclude these results, so the system queries the user for this information. When it is supplied, minsavings is calculated and either rule 4 or rule 5 fires to conclude about the adequacy of the savings account. This new fact is added.
At this point, if search is breadth-first/data-driven, continue through the remaining 3 rules, obtaining information on earnings and calculating minincome. If search is depth-first we take the result of rule 4 or 5 and return to rule 1 again.