ARTIFICIAL INTELLIGENCE

Third Edition

Instructor's Manual

Elaine Rich
Microelectronics and Computer Technology Corporation

Kevin Knight
Carnegie Mellon University

Shivashankar B Nair
IIT Guwahati

Tata McGraw-Hill Publishing Company Limited
NEW DELHI

McGraw-Hill Offices
New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto

ABOUT THE AUTHORS

Elaine Rich is Director of the Artificial Intelligence Laboratory at the Microelectronics and Computer Technology Corporation (MCC). Formerly on the faculty in computer sciences at the University of Texas, she received her PhD from Carnegie Mellon University. Her research interests include natural language processing. knowledge representation, and machine translation.

Kevin Knight received his B.A. from Harvard University and is presently completing his PhD in computer science at Carnegie Mellon University. A regular consultant at MCC, his research interests include natural language processing, unification, and machine translation, and search.

Shivashankar B Nair received his Master's and Doctoral degrees in Engineering from Amravati University, Amravati, where he served as a faculty member from 1986 to 1998. He later joined the Indian Institute of Technology Guwahati, where he is currently an Associate Professor in Computer Science and Engineering. His major areas of interest include real-world Artificial Immune System Applications, Intelligent and Emotional Robotics, Natural Language Processing, Genetic Algorithms and Mobile Agent Systems.

Dr Nair has been the chief investigator for projects funded both by the Indian government and foreign agencies and is a member of several international and national journal and conference committees. Presently, he is on a sabbatical as a Visiting Professor (Korean Brain Pool) at the Human-Centred Advanced Research Education Centre, Hanbat National University, Daejeon, South Korea, where he is investigating new ways of instilling emotions into robots.

Contents

Chapter 2

Problems, Problem Spaces, and Search

2.1 Errata

- **Page 33.** In Figure 2.3, rule 12 should read:

12	$(x.2)$	$\rightarrow (0.2)$	Empty the 4-gallon jug on the ground

2.2 Solutions to Selected Exercises

1.
   - Chess
     - In general not decomposable, although there may be configurations that can be treated as though they were composed of mostly independent subparts.
     - Solution steps can be neither ignored nor undone.
     - The universe is not predictable since it's not possible to predict with certainty how the opponent will move. But it is possible to make a guess that will be right a lot of the time.
     - A good solution (a win) is absolute.
     - The desired solution is the path to the winning state.
     - The only necessary knowledge is the rules of the game. Other knowledge is important only because of the exponential growth of dumb, blind search.
     - The computer does not need to interact with a person to decide on its moves.
   - Missionaries and Cannibals
     - Not decomposable.
     - Solution steps can be neither ignored nor undone.
     - The universe is completely predictable (in the abstract problem; of course, in the real world it is not. For example, the boat could spring a leak.) So planning a complete solution is possible and steps can be ignored during planning.
     - A good solution is absolute unless we modify the problem to ask for the quickest way to accomplish the goal.
     - The solution is a path.
     - Only a small amount of knowledge is required. There is not even much additional knowledge that can be used to improve efficiency.
     - No interaction required.
   - Tower of Hanoi
     - The problem is decomposable. To move 64 disks, first move the top 63 to the spare peg, now move the bottom disk into place. Then move the other 63 on top of it.

- Solution steps can be undone.
- The universe is completely predictable.
- If we want the shortest solution, we may need to compare.
- Solution is a path.
- Little knowledge is necessary. There's not even much heuristically useful knowledge.
- No interaction required.

• Cryptarithmetic

- Not decomposable because of the constraints that connect the letters.
- Solution steps (assignments) can be undone.
- The universe is completely predictable.
- Good solution is absolute.
- Solution is a state.
- A small amount of knowledge is absolutely required. But there is a lot of additional knowledge that can speed up the process if it is available.
- No interaction is required.

2. • Missionaries and Cannibals

State space: An ordered pair of ordered pairs. The first pair describes the left side of the river; the second describes the right. The first element of each pair is the number of missionaries. The second is the number of cannibals. The sum of the two missionary numbers must be 3, 2, or 1. Same for the two cannibal numbers. The sum of all four numbers must be 6, 5, or 4. (The missing people are in the boat, which is being represented implicitly.)

Start state: ((3, 3), (0, 0))

Goal state: ((0, 0), (3, 3))

3. Branch-and-bound is a depth-first technique. For problems like the water jug where the optimal solution requires only a small number of steps and all steps have equal cost, breadth-first search is often better and it is guaranteed to find the shortest path, although it will require a nearly exhaustive search (you may be able to skip some nodes at the final level after a solution is found). Branch-and-bound will only cut off paths that are more expensive than the best path found so far, but breadth-first will never consider such paths either since they will be at deeper levels in the search tree/graph. So we could implement a depth-first, branch-and-bound search but it would be silly to do so. It could be a lot worse than breadth-first if it explores some long paths early on. Branch-and-bound is usually used for problems like traveling salesman in which not all moves have the same cost, so that some short paths are worse than some other longer ones and we'd like a way to prune them.

4. (a) Blocks world: See Section 3.2.2.

(b) Theorem proving: To do this requires appeal to knowledge about the domain of the theorems to be proved.

(c) Missionaries and cannibals: The number of people on the goal side of the river for all acceptable states. A large negative number for any state in which the missionaries are outnumbered by the cannibals.

5. Breadth-first works better than depth-first for water jug problems. Depth-first usually works better than breadth-first for theorem proving.

Chapter 3

Heuristic Search Techniques

3.1 Errata

- **Page 87.** In some printings, there is a mistake in the description of the AO* algorithm. It should read as follows:

Algorithm: AO*

1. Let *GRAPH* consist only of the node representing the initial state. (Call this node *INIT*.) Compute $h'(INIT)$.
2. Until *INIT* is labeled *SOLVED* or until *INIT*'s h' value becomes greater than *FUTILITY*, repeat the following procedure:
   (a) Trace the labeled arcs from *INIT* and select for expansion one of the as yet unexpanded nodes that occurs on this path. Call the selected node *NODE*.
   (b) Generate the successors of *NODE*. If there are none, then assign *FUTILITY* as the h' value of *NODE*. This is equivalent to saying that *NODE* is not solvable. If there are successors, then for each one (called *SUCCESSOR*) do the following:
      i. Add *SUCCESSOR* to *GRAPH*.
      ii. If *SUCCESSOR* is a terminal node, label it *SOLVED* and assign it an h' value of 0.
      iii. If *SUCCESSOR* is not a terminal node, compute its h' value.
   (c) Propagate the newly discovered information up the graph by doing the following: Let S be a set of nodes that have been labeled *SOLVED* or whose h' values have been changed and so need to have values propagated back to their parents. Initialize S to *NODE*. Until S is empty, repeat the following procedure:
      i. If possible, select from S a node none of whose descendants in *GRAPH* occurs in S. If there is no such node, select any node from S. Call this node *CURRENT*, and remove it from S.
      ii. Compute the cost of each of the arcs emerging from *CURRENT*. The cost of each arc is equal to the sum of the h' values of each of the nodes at the end of the arc plus whatever the cost of the arc itself is. Assign as *CURRENT*'s new h' value the minimum of the costs just computed for the arcs emerging from it.
      iii. Mark the best path out of *CURRENT* by marking the arc that had the minimum cost as computed in the previous step.
      iv. Mark *CURRENT SOLVED* if all of the nodes connected to it through the new labeled arc have been labeled *SOLVED*.
      v. If *CURRENT* has been labeled *SOLVED* or if the cost of *CURRENT* was just changed, then its new status must be propagated back up the graph. So add to S all of the ancestors of *CURRENT* whose h' is not already greater than FUTILITY.

There are two changes in this version. In step 2(b), we add all SUCCESSORs to the graph, regardless of whether they are also ancestors of NODE. In Step 2(c)v, we add a check to see whether the value of h' has already exceeded FUTILITY.

3.2 Solutions to Selected Exercises

1. If the path to the solution is short, breadth-first search will find it straightforwardly. Best-first search will also find it quickly if it has a good heuristic function. But if the heuristic function is bad, it may take the search down several, possibly long, blind alleys before the solution is eventually found. In this case, breadth-first search would be more efficient. This is especially true if we take into account the cost of evaluating the heuristic function.

2. Clearly, it makes a lot of difference where the duplicate nodes occur. If they occur near the top of the tree, they cause a greater problem (since everything below them must then be duplicated) than if they occur near the leaf nodes. We will assume that they are distributed evenly throughout the tree. We will also assume a uniform branching factor for the generation of unique nodes. Call it B. If N is the duplication factor, then we actually generate $B \times N$ nodes from each node we expand. Of course, we won't expand all the nodes that we generate, whether we use a tree or a graph; the better the heuristic function is the fewer paths will be pursued. Let E be the effective branching factor, without duplicates. Then $E \times N$ is the effective branching factor with duplicates. (Actually, it's $(E-1) \times N$, since once we find a solution, we won't explore the duplicates of it, but for simplicity we will ignore this constant factor here.) So the cost of the search at level P of a tree is proportional to $(E \times N)^P$. The cost of the search at level P of a graph, on the other hand is proportional to $E^P \times M$. For small values of P, the tree is probably better, particularly if E and N are low and M is high. But as it becomes necessary to explore longer and longer search paths, these two functions cross, since the factor of N contributed by the duplicates is growing exponentially, while the factor of M contributed by the cost of checking stays constant.

4. The way the algorithm is stated in the chapter, it won't move if it hits either a plateau or a local maximum, since it only resets current state if it finds a state to move to that is better than where it is now. With the proposed change, it will still try to move to the best possible state, but it will always move, even if it must make a move to a state that is worse than or the same as the current state. This can be a win since it's a way to avoid getting stuck on a local maximum or a plateau. But it may also waste effort if the previous state was genuinely the best we could do. There's also a chance that we'll miss the best state. This could happen if we find the global maximum, but don't know it, so we move off it. Then the algorithm could terminate when it reaches a state from which it can go nowhere. Then it will return that state as the answer. To solve this problem, we probably also want to add to the algorithm a step that keeps track of the best state found so far. Then it should return that state when it halts, regardless of where it was when it halted. Another problem that this change introduces is that it's possible that the algorithm will never halt if there are no states that have no successors. So we need to add some additional mechanism to cause it to halt. We might say that if we have gone N (for some reasonable value of N) iterations without making some improvement then it is time to give up.

5. (a) E

   (b) We can only guarantee that the best solution will be found if h' never overestimates h. It appears that in this case it may overestimate, though, if the value at E is right. If that is the case, then we can't be sure that it hasn't overestimated at D just as it did at B and thus caused us to miss a better solution.

6. In the graph shown, the path from G to F introduces a cycle that appears not to be necessary since (assuming nonnegative g) it is not a cheaper way to get to F (compared to the shorter path A-B-D-E-F). But this is not a property of G itself–it is only true because we got to G the way we did. If we could get to G from C, then (assuming constant g), the path A-C-G-F is shorter than the old A-B-D-E-F. So we need to record paths that appear cyclical because they may later become a part of a different, noncyclical path.

9. This proof depends on the assumption that all arc costs are positive. To prove that the algorithm terminates, we need to prove that S eventually becomes empty. We do that as follows. First, we observe that there are only a finite number of nodes in the graph. So if we can guarantee that each one can only enter S a finite number of times, S must eventually become empty. If there are no cycles, then the trace up the ancestor path must terminate at the root of the graph and no node will ever end up in S more than once. If there are cycles, however, we need to show that the algorithm doesn't just keep running around them. Let's examine the first case. If we add A as a successor of F, then we initialize S to F. We select F as CURRENT, the first time through the propagation loop. The cost of the only arc out of F is 29, so this becomes the h' value of F. The path from F to A is marked. Since the cost of F changed, we add its ancestor, C, to S. At the next step, we select C as CURRENT. The cost

of the path to E stays 31. The cost of the path through F is now 30 (29 + 1). So it is marked. Now the cost of C is 30, and its ancestor, A, goes in S. A gets selected as CURRENT at the next step. Since C's cost is now 30, A's cost is 47. So its ancestor, F, gets put in S. So we see F enter S a second time. This time we again compute the cost of F to be that of A plus 1, giving 48. We add C to S and select it as current. This time, though, E, becomes the marked path out of C since the path through F has a cost of 49. C's cost is now 31. We again update A, whose cost goes to (31 + 15 + 2) or 48. Now the cost of F becomes 49. So we again examine C. But this time the cost of C doesn't change since it no longer depends on F at all. So nothing new gets added to S and the process terminates. In general, if the path with the cycle is "guarded" at any point by an alternative path that does not involve a cycle, then the algorithm will terminate when the cost of the cyclic path exceeds the cost of the guarding path and thus that path gets taken out of consideration. But what happens if there are no alternative paths. Then the cost of the cyclic path will increase strictly monotonically each time through the loop. Eventually, it will be greater than FUTILITY, and the propagation will stop.

On the second graph, this works as follows: The cost of F becomes 29. The cost of C becomes 30. The cost of A becomes 47. The cost of F becomes 48. And so forth. Assume FUTILITY is 50. Then C becomes 49. A becomes 66. F becomes 67. C becomes 68. But we don't add A to S this time because it is already above FUTILITY. So the algorithm halts.

On the third graph, this works as follows: C becomes 3, so we add B and A to S. Both A and B have descendants in S (each other), so we pick one, let's say A. A's cost becomes 4. B's cost is 4, though, since the path through C is better. So we look at A again. Its cost (using the path through C) doesn't change, since it doesn't depend on the path through B, so the algorithm halts.

10. The reason we want to select a node with no descendants in S is that if a node N has descendants in S, then after those descendants are processed, N will probably need to be reprocessed (assuming the values of the descendants have changed). So if we do N before its descendants we're probably wasting our time since we'll have to do it over anyway.

    The easiest way to implement S is as a first-in, first-out (FIFO) queue. But this does not always guarantee that every node will be processed before its ancestors, as shown in this example. S is initially E. Then it is either B, D or D, B (given that we add on the right and remove from the left). Assume it is the former. Then we remove B and process it. Then S becomes D, A. Then we remove D and S becomes A, C. But now if we simply remove A, we have taken a node with a descendant in S. We could check for that by looking through the queue and through A's successors, but then we'd have to check every time even though the FIFO queue usually does the right thing. It's an empirical question (the answer to which depends on how often structures like the one in this example occur) whether it is more efficient to do the check or not.

11. The third graph of Problem 9 is an example if the cost of C is updated. A and B go into S, but each is the ancestor of the other.

12. The answer is D = 1, O = 2, S = 3, E = 4, A = 5, R = 6, G = 7, N = 8, C = 9.

    We notice immediately that D is 1 and that R is even. R can't be 0 because then there could be no carry out to D. Try R = 2, but it fails. So does R = 4. R = 6 works, with the result shown.

3.3 Software

The software package includes programs for the different search strategies of Chapters 2 and 3 as well as different domains in which to apply them. The domains are:

- `n-puzzle.lisp`

  Code for manipulating the the 8-puzzle, 15-puzzle, etc.

- `travel.lisp`

  Code for a railroad travel domain.

- `generic-domain.lisp`

  User-implementable functions for a search domain.

The search programs are:

- `dfs.lisp`
  Depth-first search.
- `bfs.lisp`
  Breadth-first search.
- `hill.lisp`
  Hill climbing search.
- `a-star.lisp`
  A* search.

See also `dfid.lisp` (Chapter 12) and `rta.lisp` (Chapter 21).

Chapter 5

Using Predicate Logic

5.1 Errata

- **Pages 152-153.** In some printings the unification algorithm has some bugs. The correct version reads:

  Algorithm: Unify(L1, L2)

  1. If $L1$ or $L2$ is a variable or constant, then:
     (a) If $L1$ and $L2$ are identical, then return NIL.
     (b) Else if $L1$ is a variable, then if $L1$ occurs in $L2$ then return FAIL, else return $\{(L2/L1)\}$.
     (c) Else if $L2$ is a variable then if $L2$ occurs in $L1$ then return FAIL, else return $\{(L1/L2)\}$.
     (d) Else return FAIL.
  2. If the initial predicate symbols in $L1$ and $L2$ are not identical, then return FAIL.
  3. If $L1$ and $L2$ have a different number of arguments, then return FAIL.
  4. Set *SUBST* to NIL. (At the end of this procedure, *SUBST* will contain all the substitutions used to unify $L1$ and $L2$.)
  5. For $i \leftarrow 1$ to number of arguments in $L1$:
     (a) Call Unify with the ith argument of $L1$ and the ith argument of $L2$, putting result in S.
     (b) If S = FAIL then return FAIL.
     (c) If S is not equal to NIL then:
        i. Apply S to the remainder of both $L1$ and $L2$.
        ii. *SUBST* := APPEND(S, *SUBST*).
  6. Return *SUBST*.

- **Page 154.** In some printings, there is a mistake in the description of the algorithm for resolution in predicate logic. The correct algorithm description is:

  Algorithm: Resolution

  1. Convert all the statements of F to clause form.
  2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in 1.
  3. Repeat until either a contradiction is found, no progress can be made, or a predetermined amount of effort has been expended.
     (a) Select two clauses. Call these the parent clauses.
     (b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals $T1$ and $\neg T2$ such that one of the parent clauses contains $T1$ and the other contains $\neg T2$ and if $T1$ and $T2$ are unifiable, then neither $T1$ nor $\neg T2$ should appear

in the resolvent. We call *T1* and *T2* *complementary literals*. Use the substitution produced by the unification to create the resolvent. If there is more than one pair of complementary literals, only one pair should be omitted from the resolvent.

(c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

The change is in step 3(b), where two ¬'s were left out of the original description.

- **Page 156.** Axiom number 7 is missing a negation sign before *loyalto*. The axiom should read:

7. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee \neg loyalto(x_4, y_1)$

5.2 Solutions to Selected Exercises

1. If we try to prove $\neg hate(Marcus, Caesar)$, we immediately find that we have no axioms that enable us to conclude that someone doesn't hate someone. So that path fails. We can, however, successfully prove $hate(Marcus, Caesar)$:

$hate(Marcus, Caesar)$

↑ (5, substitution)

$Roman(Marcus) \wedge$
$\neg loyalto(Marcus, Caesar)$

↑ (3, substitution)

$Pompeian(Marcus) \wedge$
$\neg loyalto(Marcus, Caesar)$

↑ (2)

$\neg loyalto(Marcus, Caesar)$

↑ (7, substitution)

$person(Marcus) \wedge$
$ruler(Caesar) \wedge$
$tryassassinate(Marcus, Caesar)$

↑ (4)

$person(Marcus) \wedge$
$tryassassinate(Marcus, Caesar)$

↑ (8)

$person(Marcus)$

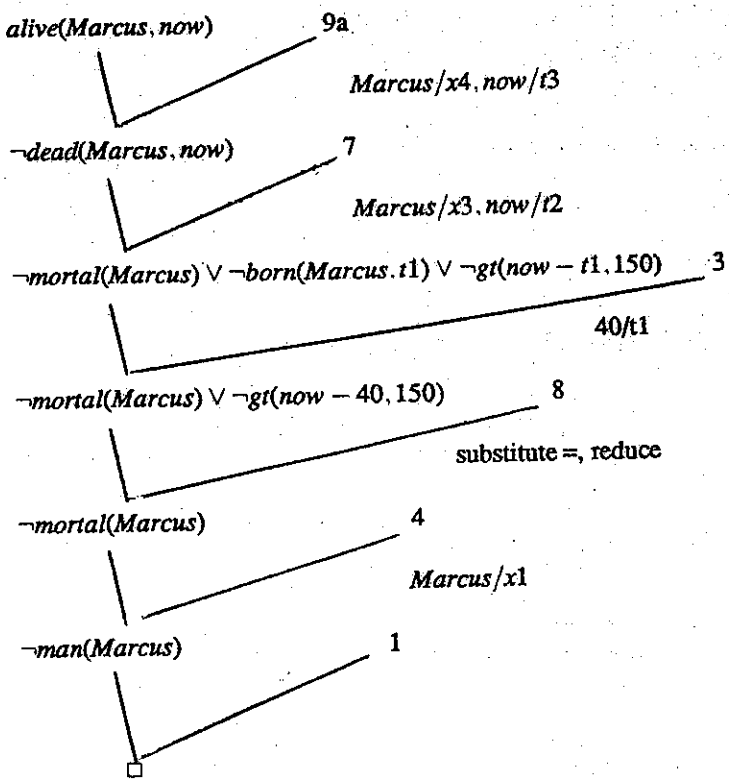
↑ (9)

$man(Marcus)$

↑ (1)

nil

2. Proof:



3. Unification yields:

   (a) FAIL

   (b) $(g(y)/x)$

   (c) Unifying the first pair of arguments produces: $(Marcus/x)$. If we make that substitution, then the next unification attempt is between

   $g(Marcus, y)$ and $g(Caesar, Marcus)$

   This returns FAIL.

4. (a) To do this we need to make one assumption. The fourth sentence is ambiguous. Does it mean that if everyone eats it and isn't killed then it's food, or does it mean if at least one person eats it and isn't killed, it's food? We'll choose the latter. We need to decide whether to represent apples, chicken, and peanuts as constants, or as predicates so that we can talk about individual instances of those general types. For this problem, it is easier to represent them as constants so we'll do it that way, but this could make some other problem solving difficult, so you might want to try it the other way also. We will break sentence 5 into two separate formulas (5 and 6). Also, we will need to add (as axiom 8) the fact that if you're still alive, then you haven't been killed by anything.

   1. $\forall x : food(x) \rightarrow likes(John, x)$
   2. $food(apples)$
   3. $food(chicken)$
   4. $\forall x : (\exists y : eats(y, x) \wedge \neg killedby(y, x)) \rightarrow food(x)$
   5. $eats(Bill, peanuts)$
   6. $alive(Bill)$
   7. $\forall x : eats(Bill, x) \rightarrow eats(Sue, x)$
   8. $\forall x : \forall y : alive(x) \rightarrow \neg killedby(x, y)$

(b) Backward chaining proof:

likes(John,peanuts)

↑ (1)

food(peanuts)

↑ (4)

$\exists y$: *eats(y,peanuts)* $\wedge$ $\neg$*killedby(y,peanuts)*

↑ (5)

$\neg$*killedby(Bill,peanuts)*

↑ (8)

alive(Bill)

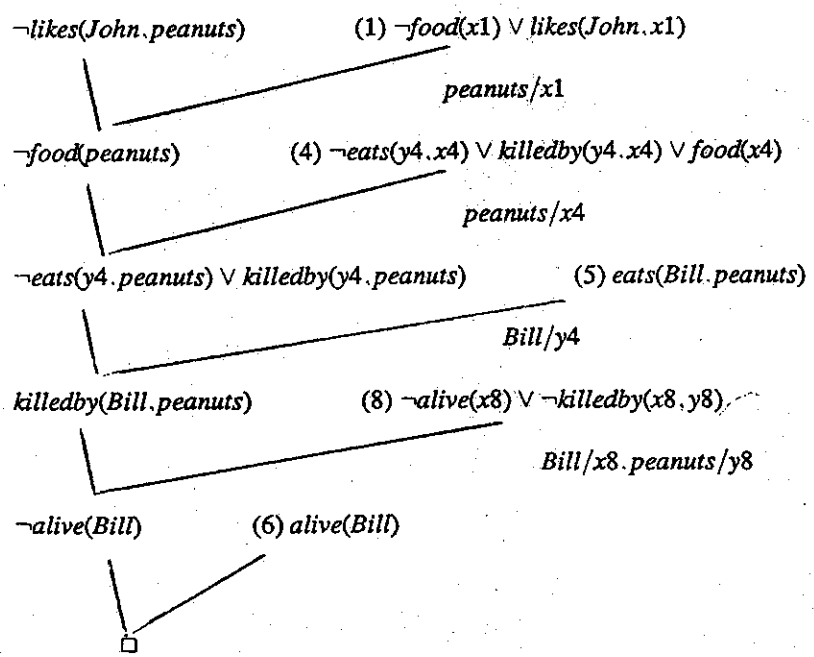
↑ (6)

nil

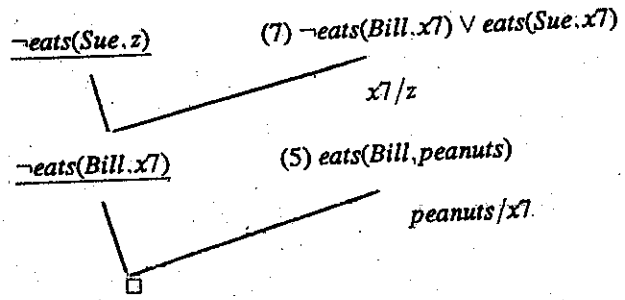
(c) Clause form:

1. $\neg$*food*(x1) $\vee$ *likes*(*John*,x1)
2. *food*(*apples*)
3. *food*(*chicken*)
4. $\neg$*eats*(y4,x4) $\vee$ *killedby*(y4,x4) $\vee$ *food*(x4)
5. *eats*(*Bill*,*peanuts*)
6. *alive*(*Bill*)
7. $\neg$*eats*(*Bill*,x7) $\vee$ *eats*(*Sue*,x7)
8. $\neg$*alive*(x8) $\vee$ $\neg$*killedby*(x8,y8)

(d) Resolution proof that John likes peanuts:



(e) We can answer the question of what Sue eats by the following resolution proof in which z is unified with x7 and then x7 is unified with peanuts:



8. (a) Either $(a/x)(father(a)/y)$ or $(x/a)(father(a)/y)$.

   (b) Either

   $loves(a, father(a))$

   or

   $loves(x, father(x))$

   depending on which unification was produced in step (a). But notice that these two clauses are equivalent. They just use different variable names.

   (c) The substitutions have to be applied in the same order in which they were generated. Otherwise, the variable substitutions won't be consistent. So, using our notation, they must be applied right to left. To see why, look at what happens if we apply $(a/x)(father(a)/y)$ in the other order. We get

   $loves(x, father(a))$

   which means that everyone likes everyone's father, not that everyone likes his or her own father, as we got when the substitution was applied correctly.

9. (a) At line 3, we use a premise clause (the one corresponding to the first premise) for the second time. It thus reintroduces the variable names it contains. So we end up unifying two clauses that start out with one or more of the same variable names (in this case just y), even though they are not actually the same variable. The unification procedure, though, doesn't know that and it collapses them into a single variable. The y that is to be substituted for z has no relationship to the y in $\neg gt(y, z)$, but since the resolution procedure doesn't know that, it produces a resolvent with the term $\neg gt(y, y)$, which is a tautology, so the resolution procedure terminates since it cannot get a contradiction out of a tautology.

   (b) To make sure this doesn't happen, the resolution procedure needs to be modified so that each time a clause is introduced, its variables are renamed with unique names. This is necessary despite the renaming step (9) in the algorithm that converts to clause form because of the possibility, as shown in this example, of using the same original clause more than once in a proof.

10. unify $p(x, f(x))$
    $p(f(a), a)$

    $(f(a)/x)$

    unify $f(f(a))$
    a

    FAIL

    But without the occur check, the substitution step would fail to terminate. (Important note: in this problem, a is not meant to be a constant. Both a and x are variables.)

11. The problem is that the first sentence does not describe a property that holds for each element of the set of men. This contrasts with other sentences, such as "Pompeians are Romans," that do make such an assertion about the individual elements of a set. Instead, this sentence asserts a property of the set itself. In this case, it is not valid to draw any conclusions about the individual members of the set.

To fix this problem, we need a way to make assertions about the class of men. We can't do that in first order logic if we represent the class as a predicate, the way we have been doing. But go back to Figure 5.3. If we use either the second or the third of the representations given there, then we have a representation of each set about which we can make assertions. So, using representation 2 in that figure, we could handle the current example by writing:

1. distributed(men, Earth)
2. instance(Socrates, men)

With no further axioms, no inferences, much less the one we are trying to avoid, can occur. If we wanted to be able to conclude something about Socrates by virtue of his being a man, then we'd have to add axioms like axiom 5 in the figure, but we'd add such axioms only for *distributive* properties (those that apply individually to the members of the class) and not for *collective* properties, such as being widely distributed, that apply only to the collection as a whole.

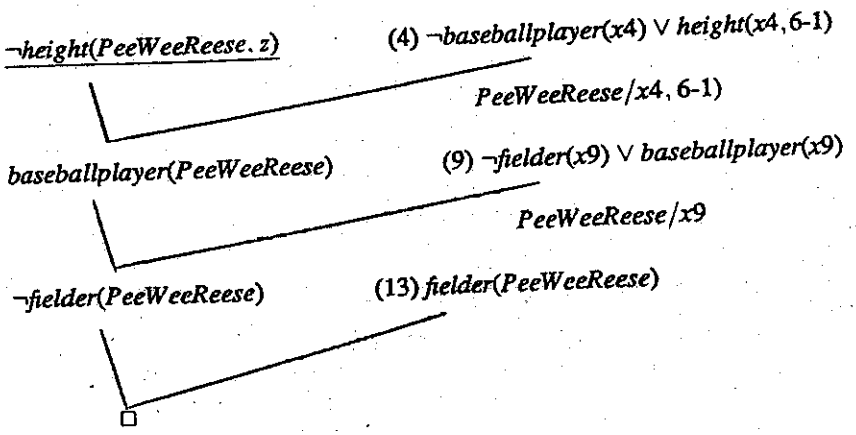
We address this issue of the difference between properties of sets and properties of elements of sets again in Chapter 9, where we present a solution to this problem in frame-based systems.

12. We need to make a decision about how to represent *isa* and *instance* relationships. You can do it in any of the ways shown in Figure 5.3. We'll do it here in the first of those ways, just as we did throughout this chapter.

We also have another problem. Recall that the notion of inheritable knowledge that we introduced in Section 4.2 allowed us to specify default values, which would be inherited if no more specific information were available, but that could be overridden by more specific facts if they were present. We can provide some of this ability using the logical mechanisms that we have described in this chapter. In particular, as we showed in Section 5.2, we can handle the case in which a more specialized class overrides the default value of the more general class. We'll do this below. (For example, *baseballplayer* will override the height value given for *adultmale*.) But we don't have any good way to handle the case in which we don't need to apply any default in a particular case because we know the exact value. For example, if we knew the height of Pee Wee Reese, we would want that value to override all the default values given in our knowledge base. The problem is that we need to be able to say, for example, that for all fielders, their batting average is .262 unless we know some other value. In Chapter 7, we will discuss some extensions to the logical framework we have been using that will allow us to handle that case. For now, we'll have to ignore it. But there will be some strange things in what we have to write as a result. For example, axiom 5 applies if you're a baseball player but not either a fielder or a pitcher. Clearly you have to be one or the other. What we really want is for axiom 5 to apply if you're a baseball player and you're not known to be either a fielder or a pitcher (i.e., you are one or the other but we don't know which).

1. $\forall x : person(x) \rightarrow handed(x, right)$
2. $\forall x : adultmale(x) \rightarrow person(x)$
3. $\forall x : adultmale(x) \wedge \neg baseballplayer(x) \rightarrow height(x, 5\text{-}10)$
4. $\forall x : baseballplayer(x) \rightarrow height(x, 6\text{-}1)$
5. $\forall x : baseballplayer(x) \wedge \neg pitcher(x) \wedge \neg fielder(x) \rightarrow battingaverage(x, .252)$
6. $\forall x : baseballplayer(x) \rightarrow \exists y : bats(x, y) \wedge \exists z : handed(x, z) \wedge y = z$
7. $\forall x : pitcher(x) \rightarrow baseballplayer(x)$
8. $\forall x : pitcher(x) \rightarrow battingaverage(x, .106)$
9. $\forall x : fielder(x) \rightarrow baseballplayer(x)$
10. $\forall x : fielder(x) \rightarrow battingaverage(x, .262)$
11. $pitcher(ThreeFingerBrown)$
12. $team(ThreeFingerBrown, ChicagoCubs)$
13. $fielder(PeeWeeReese)$
14. $team(PeeWeeReese, BrooklynDodgers)$

We can use these axioms to derive some of the results of Section 4.2. For example, we can use resolution, as we described in Section 5.4.7, to extract values for attributes. We can, for example, find Pee Wee Reese's height from the following proof in which z is unified with 6-1:



13. We take each sentence in turn:

- What do we really mean? The only movies John likes are French (most likely). But we could mean the only thing John likes is to see French movies; he does not like to eat or go fishing. Contrast, for example, "John only likes to go fishing (and not anything else)."
- To handle this, we need to be able to express a default that holds in the *absence* of some other information. As we discussed in the solution to problem 12, it's not possible to do that with only the mechanisms we have so far introduced. We'll return to this in Chapter 7.
- So far, the only techniques we have for describing how often a situation occurs are universal quantification (it holds all the time), universal quantification of the negation (it never holds), and existential quantification (it holds at least once). In Chapter 6, we will introduce some techniques that allow us more flexibility in describing the likelihood of some situation occurring.
- This is straightforward to represent in logic, except that we don't really mean it. What we really mean is something like, "People don't do things that will cause them to be in situations that they don't like unless they can't avoid it, in which case they'll try for the situation that they dislike least."
- To represent this, we again have the problem of representing likelihood. And because we didn't solve the representation problem for the other axioms either, we have no way to conclude it.

You might want to try this exercise again after you've completed chapters 6 and 7.

5.3 Software

- `unify.lisp`
  Unification for first-order terms.
- `resolve.lisp`
  Resolution engine for propositional logic.

Chapter 6

Representing Knowledge Using Rules

6.1 Solutions to Selected Exercises

1. (a) Horn clauses:
   1. $\neg cat(x) \vee \neg fish(y) \vee likes\text{-}to\text{-}eat(x, y)$
   2. $\neg calico(x) \vee cat(x)$
   3. $\neg tuna(x) \vee fish(x)$
   4. $tuna(Charlie)$
   5. $tuna(Herb)$
   6. $calico(Puss)$

   (b) PROLOG program:

```
likestoeat(X,Y) :- cat(X), fish(Y).
cat(X) :- calico(X).
fish(X) :- tuna(X).
tuna(charlie).
tuna(herb).
calico(puss).
```

   (c) Query:

```
?- likestoeat(puss,X).
```

   Answer: `charlie`

   (d) An alternative PROLOG program:

```
likestoeat(X,Y) :- cat(X), fish(Y).
cat(X) :- calico(X).
fish(X) :- tuna(X).
tuna(herb).
tuna(charlie).
calico(puss).
```

   This program will answer with `herb`. The order in which the assertions appear in the program matters since they are checked in the order in which they appear.

2. See pages 178–179. This is better as a test question than a homework assignment.

3. Backward vs. forward reasoning:

   (a) Water jug problem: The branching factor is about the same going in either direction. But there are more goal states (anything of the form (2, n)) than there are start states (just (0, 0)), so it is better to reason forward.

(b) Blocks world: There are the same number of start states as goal ones. The details of the way the available operators are described can have a big effect on the branching factor, but it is usually better to go backward from the goal state because the branching factor is usually lower in that direction.

(c) Natural language understanding: Here the start state is an utterance and the goal is a meaning representation of the utterance. Since we don't have any idea what the meaning representation ought to be until we do the understanding, we have to reason forward from the initial state. There are some circumstances, where this is not completely true, though, since sometimes one can predict, in a given context, what is likely to be said/written next. Then it may help to do some backward (usually called top-down) prediction to help guide the forward (bottom-up) process.

Chapter 7

Symbolic Reasoning under Uncertainty

7.1 Errata

- **Page 225.** In Figure 7.14, the justification on line [3] should read {A4}, not {4}; the one on line [5] should read {A6}, not {6}.
- **Page 228.** In some printings, there is a mistake in Exercise 4. The axioms should read:

$\forall x : Republican(x) \wedge \neg AB1(x) \rightarrow \neg Pacifist(x)$
$\forall x : Quaker(x) \wedge \neg AB2(x) \rightarrow Pacifist(x)$
$Republican(Dick)$
$Quaker(Dick)$

- **Page 245.** In the boxed table near the bottom of the page, in the third row and third column, {A, F,C} should read {A}.

7.2 Solutions to Selected Exercises

1. We can write:

$suspect(Abbott) \vee suspect(Babbitt) \vee suspect(Cabot)$
$\forall x : alibi(x) \rightarrow \neg suspect(x)$
$\forall x : inhotelregister(x) \rightarrow alibi(x)$
$\forall x : (\exists y : testifytovisit(y, x)) \rightarrow alibi(x)$
$\forall x : outoftown(x) \rightarrow alibi(x)$
$\forall x : onTVelsewhere(x) \rightarrow outoftown(x)$
$inhotelregister(Abbott)$
$testifytovisit(BIL, Babbitt)$

We can conclude $suspect(Cabot)$, since Abbott and Babbitt both have alibis. But that's because we just ignored Cabot's stated alibi. If we now add the fact that he was on TV, we end up with a contradiction and no good way to proceed because we have no explicit assumptions that we can decide to retract (e.g., that the register wasn't forged or that the brother in law didn't lie or that there are no other suspects out there.) We didn't write these assumptions as conditions in our rules because there is no way to prove them. So we had to ignore them to enable the system to be able to prove anything.

2. Using nonmonotonic logic:

- Axioms:
  1 $\forall x : \neg AB(x, 1) \wedge \mathrm{M}\neg fly(x) \rightarrow \neg fly(x)$
  2 $\forall x : bird(x) \rightarrow AB(x, 1)$
  3 $\forall x : bird(x) \wedge \neg AB(x, 2) \wedge \mathrm{M}fly(x) \rightarrow fly(x)$
  4 $\forall x : tooyoung(x) \rightarrow AB(x, 2)$
  5 $\forall x : dead(x) \rightarrow AB(x, 2)$
  6 $\forall x : brokenwing(x) \rightarrow AB(x, 2)$
  7 $\forall x : penguin(x) \rightarrow bird(x) \wedge AB(x, 2)$
  8 $\forall x : ostrich(x) \rightarrow bird(x) \wedge AB(x, 2)$
  9 $\forall x : ostrich(x) \wedge \neg AB(x, 3) \wedge \mathrm{M}\neg fly(x) \rightarrow \neg fly(x)$
  10 $\forall x : penguin(x) \wedge \mathrm{M}\neg fly(x) \rightarrow \neg fly(x)$
  11 $\forall x : ostrich(x) \wedge magical(x) \rightarrow AB(x, 3) \wedge fly(x)$
  12 $bird(Tweety)$
  13 $penguin(Chirpy) \vee ostrich(Chirpy)$
  14 $ostrich(Feathers) \wedge magical(Feathers)$
  15 $\forall x, y : \mathrm{M}\neg AB(x, y) \rightarrow \neg AB(x, y)$
- Can Tweety fly?

  We prove $fly(Tweety)$. The attempt to prove $\neg fly(Tweety)$ fails as described in note B below.

$$
\begin{array}{l}
fly(Tweety) \\
\qquad \uparrow \quad (3) \\
bird(Tweety) \wedge \\
\neg AB(Tweety, 2) \wedge \\
\mathrm{M}\, fly(Tweety) \\
\qquad \uparrow \quad (12) \\
\neg AB(Tweety, 2) \wedge \\
\mathrm{M}\, fly(Tweety) \\
\qquad \uparrow \quad \text{(note A)} \\
\mathrm{M}\, fly(Tweety) \\
\qquad \uparrow \quad \text{(note B)} \\
nil
\end{array}
$$

  A. Establishing $\neg AB(Tweety, 2)$ requires the use of axiom 15, plus a failed attempt to prove $AB(Tweety, 2)$. That attempt fails because none of the axioms (4, 5, 6, 7, 8) that can produce $AB(x, 2)$ have their antecedents met for Tweety.

  B. This step depends on a failed attempt to prove $\neg fly(Tweety)$ (since that is what we must check to make sure it would be consistent to assume $fly(Tweety)$). The failure happens because none of the axioms (1, 9, 10) that can show $\neg fly(x)$ apply. Axiom 1 fails because axiom 2 says $AB(Tweety, 1)$, so axiom 1 cannot generate $\neg AB(Tweety, 1)$. Axioms 9 and 10 fail because Tweety cannot be shown to be either an ostrich or a penguin.
- Can Chirpy fly?

  We prove $\neg fly(Chirpy)$. The attempt to prove $fly(Chirpy)$ fails as described in note B below.

$\neg fly(Chirpy)$

↑ (9, 10, logical manipulation)

$M\neg fly(Chirpy) \wedge$
$[[ostrich(Chirpy) \wedge \neg AB(Chirpy, 3)] \vee penguin(Chirpy)]$

↑ (note A)

$M\neg fly(Chirpy) \wedge$
$[ostrich(Chirpy) \vee penguin(Chirpy)]$

↑ (13)

$M\neg fly(Chirpy)$

↑ (note B)

nil

A. Establishing $\neg AB(Chirpy, 3)$ depends on axiom 15 and the failure of the attempt to prove $AB(Chirpy, 3)$, which happens because the only axiom (11) that could establish it fails to apply since there is no way to show $magical(Chirpy)$.

B. This step depends on a failed attempt to prove $fly(Chirpy)$. Only axioms 3 and 11 can be used to conclude $fly(x)$. Axiom 3 cannot apply since, by axiom 8, $AB(Chirpy, 2)$. Axiom 11 cannot apply as described in note A.

- Can Feathers fly?

  We prove $fly(Feathers)$. The attempt to prove $\neg fly(Feathers)$ fails because none of the axioms that can produce $fly(x)$ (1, 9, 10) can apply. Axiom 1 is blocked because Chirpy is an ostrich, thus also a bird (by axiom 8) and thus abnormal in aspect 1 (by axiom 2). Axiom 9 fails because, by axiom 11, Feathers is abnormal in aspect 3. Axiom 10 fails because Feathers is not a penguin.

$fly(Feathers)$

↑ (11)

$ostrich(Feathers) \wedge$
$magical(Feathers)$

↑ (14)

nil

- Can Paul fly?

  We prove $\neg fly(Paul)$. The attempt to prove $fly(Paul)$ fails as described in note B below.

$\neg fly(Paul)$

↑ (1)

$\neg AB(Paul, 1) \wedge$
$M\neg fly(Paul)$

↑ (note A)

$M\neg fly(Paul)$

↑ (note B)

nil

A. Establishing $\neg AB(Paul, 1)$ depends on axiom 15 and the failure to prove $AB(Paul, 1)$, which occurs because the only relevant axiom (2) does not apply since Paul is not a bird.

B. This step depends on the failure of the attempt to prove $fly(Paul)$. This occurs because neither of the axioms (3, 11) that could produce it apply. Axiom 3 fails because Paul is not a bird. Axiom 11 fails

because Paul is not an ostrich (or magical).

3. We assumed that many predicates were false because we did not know them to be true. Examples of such predicates include:

   bridge-across-river
   leaky-boat
   missing-oars
   paralyzed
   impassable-rapids

4. The problem of Dick, the Quaker and Republican.

   (a) Model 1:
   Republican(*Dick*)
   Quaker(*Dick*)
   *AB*1(*Dick*)
   ¬*AB*2(*Dick*)
   pacifist(*Dick*)

   Model 2:
   Republican(*Dick*)
   Quaker(*Dick*)
   *AB*2(*Dick*)
   ¬*AB*1(*Dick*)
   ¬*pacifist*(*Dick*)

   (b) Yes, Model 2 is smaller than Model 1 since fewer positive statements are true. But this is a very syntactic way to resolve the problem and is probably not what we want.

   (c) We could define the order in which the *AB* predicates should be minimized. This order could be based on knowledge about the relative likelihoods of the two kinds of exceptions.

5. (a)

		IN-list	OUT-list
1.	wearjeans		(2,3)
2.	dirtyjeans		
3.	jobinterview		
4.	sweater	(5)	
5.	cold	(6)	(8)
6.	winter		(9)
7.	sandals	(8)	
8.	warm	(9)	(5)
9.	summer		(6)

   (b) i. We begin by giving a premise justification to node 9 (summer). Then nodes 9 (summer), 8 (warm), and 7 (sandals) become IN, and nodes 6 (winter), 5 (cold), and 4 (sweater) are OUT. Nodes 2 (dirtyjeans) and 3 (jobinterview) are OUT since they have no justifications. So node 1 (wearjeans) is IN. We wear jeans and sandals.

   ii. If we now add a premise justification to node 5 (cold), then nodes 8 (warm) and 7 (sandals) go OUT. Node 4 (sweater) comes IN. We wear jeans and a sweater.

   iii. If we now add a premise justification to node 3 (jobinterview), then node 1 (wearjeans) goes out. We're down to a sweater.

Chapter 8

Statistical Reasoning

8.1 Errata

- **Page 245.** In some printings, there is a mistake in the last paragraph, which should begin:

  But there is now a total belief of 0.54 associated with $\emptyset$; only 0.46 is associated with outcomes ...

8.2 Solutions to Selected Exercises

1. Let $P(G) =$ probability that my initial guess is right.
   $P(O) =$ probability that one of the other 2 shells has the pea.
   $P(T) =$ Probability that a shell without a pea will be turned over.

   $P(G) = \frac{1}{3}$

   $P(O) = \frac{2}{3}$

   $P(T) = 1$

   The important fact here is that $P(T) = 1$, i.e., it is certain that a shell without a pea will be turned over. So the turning over provides no new information. So:

   $P(G|T) = P(G) = \frac{1}{3}$

   $P(O|T) = P(O) = \frac{2}{3}$

   But O now contains only one shell that hasn't been turned over. So the probability that that shell contains the pea is $\frac{2}{3}$. You should change your guess to that other shell.

   To see what is really going on here, we can contrast this problem with one in which, after you make your guess, the other person turns over one of the other shells are random (i.e., without knowing where the pea is). Then:

   $P(G) = \frac{1}{3}$

   $P(O) = \frac{2}{3}$

   $P(T) = \frac{2}{3}$

   $P(G|T) = \frac{P(G \wedge T)}{P(T)} = \frac{\frac{1}{3}}{\frac{2}{3}} = \frac{1}{2}$

   We got this by using the simplest form of Bayes rule and observing that if your guess was right, then the other person is certain to pick a shell without a pea, so $P(G \wedge T)$ is just $P(G)$.

2. $MB[h_1, o_1] = 0.5$

$MB[h_1, o_1 \wedge o_2] = 0.5 + (0.3 \times (1 - 0.5)) = 0.65$

$MB[h_1, o_1 \wedge o_2 \wedge o_3] = 0.65$

$MD[h_1, o_1] = 0$

$MD[h_1, o_1 \wedge o_2] = 0$

$MD[h_1, o_1 \wedge o_2 \wedge o_3] = 0.2$

$CF[h_1, o_1 \wedge o_2 \wedge o_3] = 0.65 - 0.2 = 0.45.$

3. • Commutativity: We show that $MB[h, s_1 \wedge s_2] = MB[h, s_2 \wedge s_1]$.

$MB[h, s_1 \wedge s_2] =$
$MB[h, s_1] + MB[h, s_2] \times (1 - MB[h, s_1]) =$
$MB[h, s_1] + MB[h, s_2] - (MB[h, s_2] \times MB[h, s_1]) =$
$MB[h, s_2] + MB[h, s_1] - (MB[h, s_1] \times MB[h, s_2]) =$
$MB[h, s_2] + MB[h, s_1] \times (1 - MB[h, s_2]) =$
$MB[h, s_2 \wedge s_1]$

• Additional confirming evidence increases MB:

Let $MB[h, s_1]$ be the measure of belief in h after some amount of positive evidence s_1. Assuming that we have not yet reached certainty, we know that

$$0 < MB[h, s_1] < 1$$

So (multiplying by -1) we have

$$-1 < -MB[h, s_1] < 0$$

and (adding 1)

$$0 < 1 - MB[h, s_1] < 1$$

Let $MB[h, s_2]$ be the measure of belief in h given just the new piece of evidence s_2. Since this is a confirming piece of evidence, it is positive, so we have

$$0 < MB[h, s_2] < 1$$

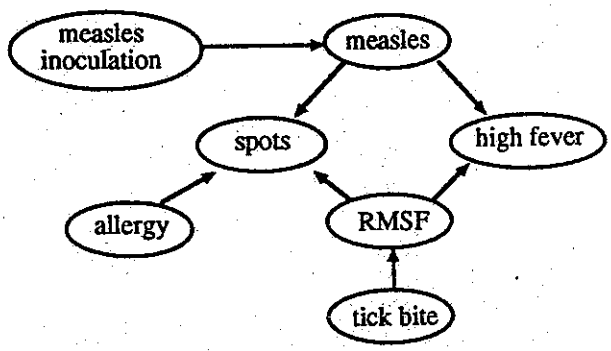
Multiplying these two inequalities, we get

$$0 < MB[h, s_2] \times (1 - MB[h, s_1]) < 1$$

This is the amount that is to be added to $MB[h, s_1]$ to get $MB[h, s_1 \wedge s_2]$. Since this is positive, the combined MB must increase.

4. Bayesian network.

(a)



(b) (These are example numbers. I have no idea of the true values.)

Attribute	*Probability*
$p(innoculation)$	0.4
$p(tickbite)$	0.0001
$p(allergy)$	0.1
$p(measles \mid innoculation)$	0.000001
$p(measles \mid \neg innoculation)$	0.0001
$p(RMSF \mid tickbite)$	0.1
$p(fever \mid RMSF, \neg measles)$	0.9
$p(fever \mid RMSF, measles)$	0.91
$p(fever \mid \neg RMSF, \neg measles)$	0.001
$p(fever \mid \neg RMSF, measles)$	0.8
$p(spots \mid allergy, RMSF, \neg measles)$	0.85
$p(spots \mid allergy, RMSF, measles)$	0.95
$p(spots \mid allergy, \neg RMSF, \neg measles)$	0.1
$p(spots \mid allergy, \neg RMSF, measles)$	0.85
$p(spots \mid \neg allergy, RMSF, \neg measles)$	0.8
$p(spots \mid \neg allergy, RMSF, measles)$	0.9
$p(spots \mid \neg allergy, \neg RMSF, \neg measles)$	0.001
$p(spots \mid \neg allergy, \neg RMSF, measles)$	0.8

5. • $\Theta = \{M, A, R\}$:

M: measles
A: allergy
R: RMSF

• In all of these, remember that we are assuming that Θ contains the universe of possibilities, so our patient must have one of the three diseases. Also, notice that these numbers aren't the same as the numbers in problem 4. There the numbers described probabilities of symptoms given diseases. Here we're talking about likelihoods of diseases given symptoms.

m_1 (belief given just spots):

$\{M, A, R\}$	(0.9)
$\{\Theta\}$	(0.1)

m_2 (belief given just fever):

$\{M, R\}$	(0.9)
$\{\Theta\}$	(0.1)

m_3 (belief given just measles shot):

$\{A, R\}$	(0.95)
$\{\Theta\}$	(0.05)

m_4 (belief given just tick bit):

$\{R\}$	(0.95)
$\{\Theta\}$	(0.05)

Combining m_1 and m_2 to form m_5 (belief given spots and fever):

		$\{M,R\}$	(0.9)	Θ	(0.1)
$\{M,A,R\}$	(0.9)	$\{M,R\}$	(0.81)	$\{M,A,R\}$	(0.09)
Θ	(0.1)	$\{M,R\}$	(0.09)	Θ	(0.01)

The total $Bel(\{M,R\})$ at this point is .9.

Combining m_5 and m_4 (belief given spots, fever, and tick bite):

		$\{R\}$	(0.95)	Θ	(0.05)
$\{M,R\}$	(0.9)	$\{R\}$	(0.855)	$\{M,R\}$	(0.045)
$\{M,A,R\}$	(0.09)	$\{R\}$	(0.0855)	$\{M,A,R\}$	(0.0045)
Θ	(0.01)	$\{R\}$	(0.0095)	Θ	(0.0005)

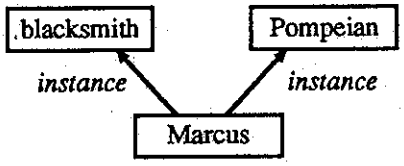
So our overall $Bel(\{R\})$ is 0.95.

Chapter 9

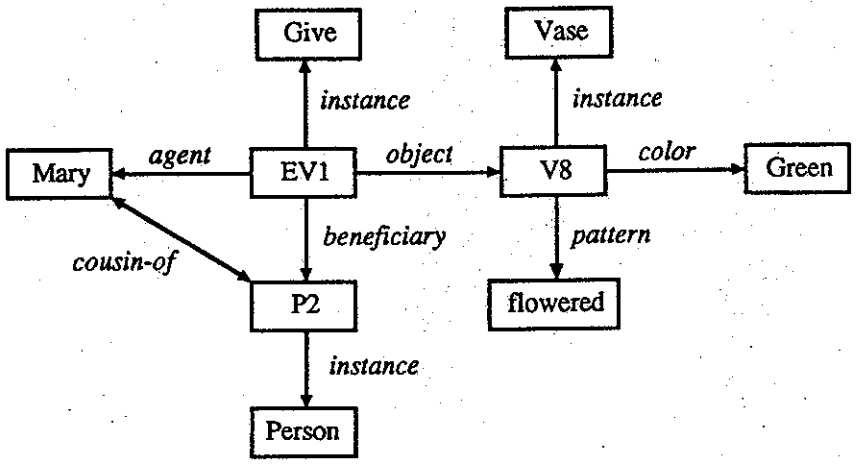
Weak Slot-and-Filler Structures

9.1 Solutions to Selected Exercises

1. (a) Marcus:



(b) Mary:



3.
   - As in Figure 9.4(b), substituting Batters for Dogs, Hit for Bite, Ball for Mail-carrier, agent for assailant, and object for victim.
   - As in Figure 9.4(c), eliminating Town-Dogs, and substituting Batters for Dogs, Like for Bite, Pitchers for constables, agent for assailant, and object for victim.

5. We make two changes to the algorithm:
   - Step 2(a)—Don't terminate a branch when a value is found.
   - Steps 3 and 4 are not necessary. Instead, just return the set *CANDIDATES*.

Chapter 10

Strong Slot-and-Filler Structures

10.1 Errata

- **Page 280.** Line 3 should be PP <=> PP, not PP <=> PA.

10.2 Solutions to Selected Exercises

5. No, because CD is designed for representing actions. Much of *National Geographic* is made up of physical descriptions (of places, people, animals, etc.) and CD offers very little to enable us to represent such descriptions.

6.
   - **cat**: The class of cats is an instance of a collection. It is specialization of the class of animals. It is probably also a specialization of the class of agents.
   - **court case**: The class of court cases is an instance of a collection. It is a specialization of the class of intangible things (and probably other, more specific classes).
   - **New York Times**: is an instance of the class of newspapers, which is a specialization of the class of composite objects, since a newspaper is both a physical thing and an abstract set of statements.
   - **France**: is an instance of the class of countries, which is a specialization of physical region.
   - **glass of water**: depending on what we mean here, this could be either an instance of water, which is a specialization of substance. In this case, we would indicate quantity by reference to the glass. Alternatively, we really want to represent a glass, whose contents are water. Then it is an instance of the class of glasses, which is a specialization of the class of tangible objects.

Chapter 11

Knowledge Representation Summary

11.1 Solutions to Selected Exercises

1. • We can use simple predicate logic:
   1. $\forall x : fruit(x) \rightarrow likes(John, x)$
   2. $fruit(kumquats)$
   3. $\forall x, y : person(x) \wedge likes(x, y) \rightarrow eats(x, y)$
   4. $person(John)$

   Notice that we had to add axiom 4. Now we prove that John eats kumquats using backward chaining:

   $eats(John, kumquats)$
   ↑ (3)
   $person(John) \wedge likes(John, kumquats)$
   ↑ (4)
   $likes(John, kumquats)$
   ↑ (1)
   $fruit(kumquats)$
   ↑ (2)
   nil

   • We can use nonmonotonic logic:
   1. $\forall x : candy(x) \wedge \neg AB(x, 1) \rightarrow containssugar(x)$
   2. $\forall x : diebetic(x) \rightarrow AB(x, 1)$
   3. $candy(M\&Ms)$
   4. $\forall x, y : diabetic(x) \wedge containssugar(y) \rightarrow \neg shouldeat(x, y)$
   5. $diabetic(Bill)$
   6. $\forall x, y : M\neg AB(x, y) \rightarrow \neg AB(x, y)$

   Notice that we have not made a distinction between "shouldn't eat" and "not necessary to eat" (i.e., "not should eat").

   Then we can prove that Bill should not eat M&Ms using backward chaining:

$\neg shouldeat(Bill.M\&Ms)$

↑ (4)

$diabetic(Bill) \wedge$
$containssugar(M\&Ms)$

↑ (5)

$containssugar(M\&Ms)$

↑ (1)

$candy(M\&Ms) \wedge$
$\neg AB(M\&Ms.1)$

↑ (3)

$\neg AB(M\&Ms.1)$

↑ (6, note A)

nil

Note A: This step depends on the proof that it is consistent to assume $\neg AB(M\&Ms.1)$. The attempt to prove $AB(M\&Ms.1)$ fails since the only way to do that is with axiom 2, whose antecedent is not provable.

- We need to use some numeric technique.
- We need to appeal to a stored representation of what normally happens at movie theaters. A script could represent such facts.

2. `It usually rains in the spring.`
   `The electricity is normally on.`
   `If you have a baby, you have about equal chances of having a boy or a girl.`
   `Most people can see.`
   `80% of the students passed.`

3. It depends on how we have chosen to represent Instance and Isa relationships. As we showed n Figure 5.3, some representations require a general propagation axiom (axiom 6). So we must have that if we chose the corresponding representation.

   We also need to describe the exceptions as part of the more general statements higher up the tree, as we discussed in Section 7.2.1. So we need to write:

   $$\forall x : adultmale(x) \wedge \neg baseballplayer(x) \rightarrow height(x, 5\text{-}10)$$

   But recall that this still presents a problem since not knowing that x is a baseball player is not the same as knowing that he is not a baseball player. So to make this work without some form of nonmonotonic reasoning will require that we specify the things x is not as well as the things x is.

4. - The JTMS system that we have described so far has a problem here since the statements that constitute its nodes must be ground instances of formulas (i.e., they contain no variables). But what we want to write here are relationships among rules that apply to individuals that are mammals, cows, and platypuses. The difficulty in extending the system to include formulas with variables is that the correct propagations of variable bindings must be recorded as part of the justifications so that it is clear how instances of the general case depend on each other.

   What is usually done for problems like this is to encode the general cases as rules and to use the JTMS just to keep track of the dependencies among ground instances that are generated by the rules. So, in this case, we have (in the notation of nonmonotonic logic):

   $\forall x : cow(x) \rightarrow mammal(x)$
   $\forall x : platypus(x) \rightarrow mammal(x)$
   $\forall x : mammal(x) \rightarrow birthmode(x, live)$

   Then, for the specific object Flossie, we have:

		IN-list	OUT-list
1.	*cow(Flossie)*	(premise)	
2.	*birthmode(Flossie, live)*	(1)	

- Now the rules must be changed to

  $\forall x : cow(x) \rightarrow mammal(x)$

  $\forall x : platypus(x) \rightarrow mammal(x)$

  $\forall x : mammal(x) \wedge \mathrm{M}\neg platypus(x) \rightarrow birthmode(x, live)$

  $\forall x : platypus(x) \rightarrow birthmode(x, eggs)$

  (Alternatively, we could have used the AB notation, but that clutters the TMS with extra nodes.)

  Now for Flossie we have:

		IN-list	OUT-list
1.	*cow(Flossie)*	(premise)	
2.	*birthmode(Flossie, live)*	(1)	(3)
3.	*platypus(Flossie)*		

Chapter 12

Game Playing

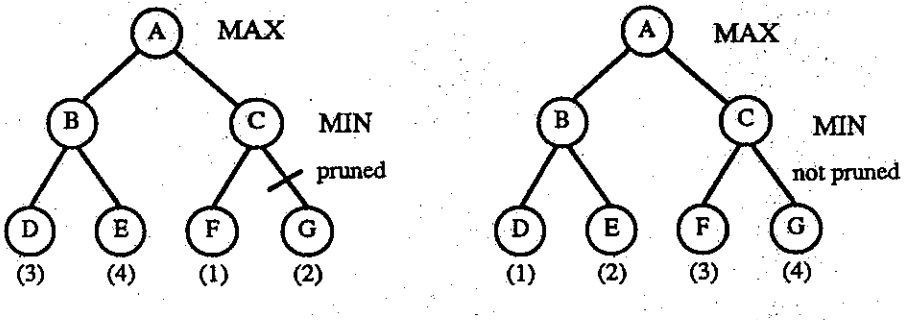
12.1 Errata

- **Page 313.** Footnote 3 is missing in some printings. It should read:

  [3]OPPOSITE is a function that returns PLAYER-TWO when given PLAYER-ONE, and PLAYER-ONE when given PLAYER-TWO.

12.2 Solutions to Selected Exercises

1. Move D (expected value of 8 at node W).
2. Nodes O, I, T, U, and Y.
3. There are many goal states, but only one start state.
4. Depth-first, with a cutoff.
5. This is a good idea if the evaluation of a node is very expensive, or if the identical board position will be generated a great many times.
6. Luckhardt and Irani (Proc. AAAI-86, p. 158–162) describe a multiplayer minimax algorithm in which leaf node evaluations are N-tuples, where N is the number of players. Korf (AI Journal 48(1), 1991) proves that alpha-beta pruning is only effective for two-player games; surprisingly, it does not help much in the multiplayer case.
7. Move ordering does matter. If moves are ordered such that better nodes tend to be evaluated before worse nodes, alpha-beta pruning will be more effective. For example, notice the difference between the following two trees:



8. These programs are included in the software package (`minimax.lisp`, `tictactoe.lisp`).

12.3 Software

- `minimax.lisp`

  Programs for doing two-player minimax search, with and without alpha-beta pruning.

- `tictactoe.lisp`

  Functions that implement the game of tic-tac-toe for minimax search.

- `generic-game.lisp`

  Function stubs that a user can fill out to create a new board game for minimax search.

- `dfid.lisp`

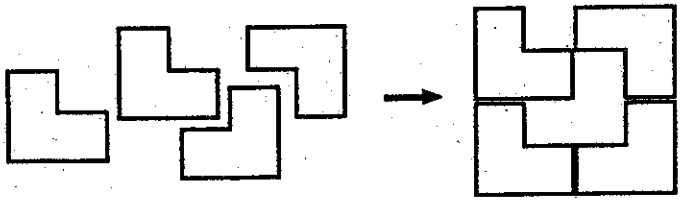
  DFID and IDA* search algorithms. Sample domains are: `n-puzzle.lisp` and `travel.lisp`. See Section 3.3 for more details.

Chapter 14

Understanding

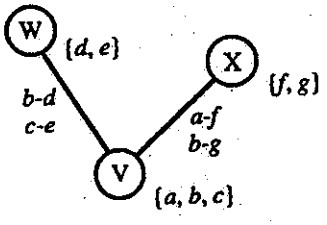
14.1 Solutions to Selected Exercises

1. A speech example is "Kevin Knight," in which the two "n" sounds are merged into one. A vision example is pictured here, where four L-shaped tiles form what looks like a fifth L-shaped tile, but which is actually just part of the background:



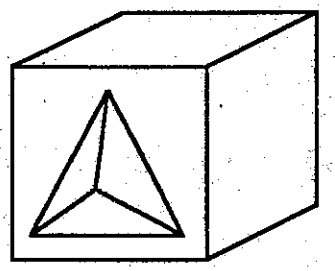
2. The trihedral figures are (*b*), (*c*), and (*d*).

5. Suppose vertex V has possible labelings *a*, *b*, and *c*. Suppose that one of its two neighboring vertices, W, has possible labelings *d* and *e*; the other neighboring vertex X, has possible labelings *f* and *g*. The only compatible labelings between V and W are *b*-*d* and *c*-*e*, and the only compatible labelings between V and X are *a*-*f* and *b*-*g*.



V is our current vertex. If we look to W, we can constrain V's possibilities to *b* and *c*. This does not constrain W any further, however. But if we look to X, we can constrain V down to *b* alone. (*a* was already ruled out by W). The constraints on V provided by X *and* W are strong enough to allow us to constrain W to *d* alone. This is why we must look at all of V's neighboring vertices before propagating the constraints.

6. In the figure below, the triangle may be a protrusion from the cube, or an indentation:



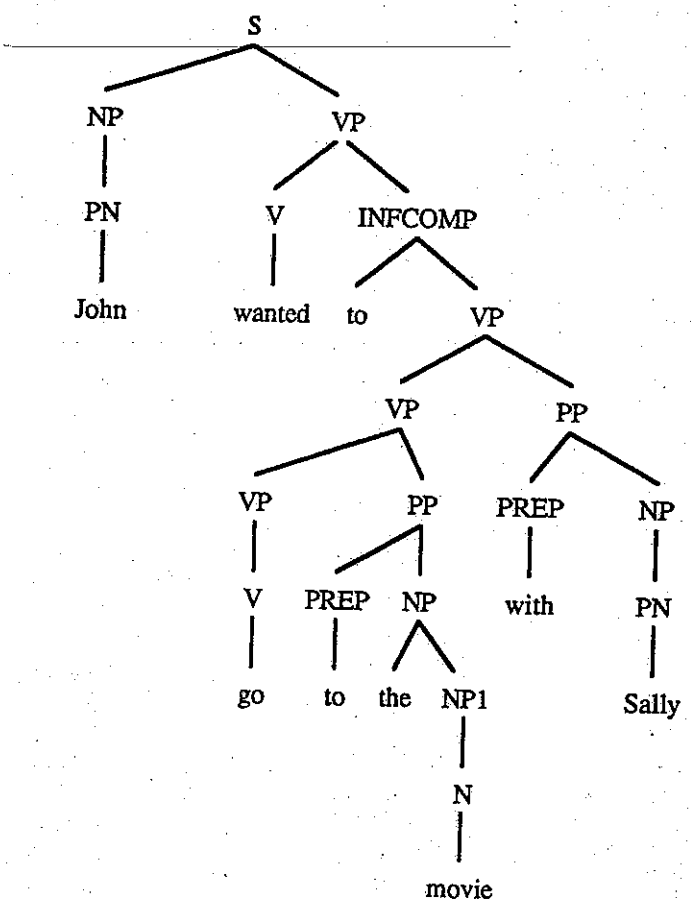
Chapter 15

Natural Language Processing

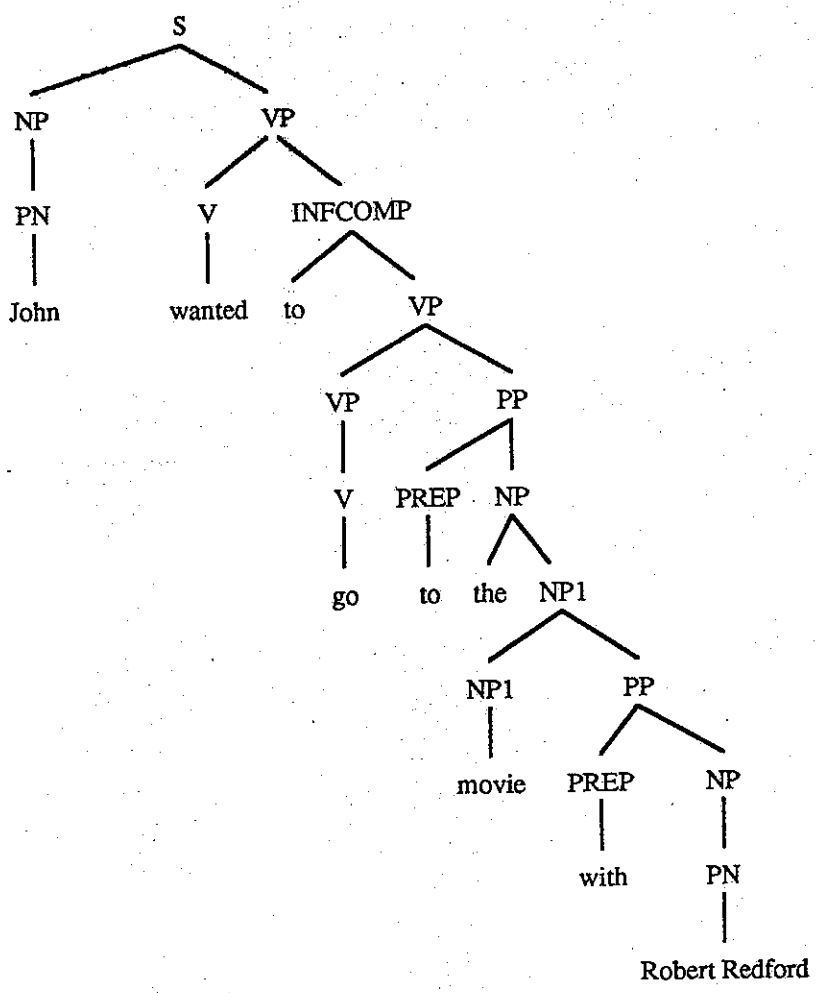
15.1 Solutions to Selected Exercises

1. The main predicate of the sentence is "were filled with". So we look to see what kinds of objects can participate in the relation that corresponds to this predicate. The thing that is filled must be a container of some kind. That causes us to choose the "glass you drink from" meaning of "glasses". Although it's possible to fill eyeglasses (since they're curved, you could set them down, curved side up and fill them), that's only because you can fill anything that is not completely flat. But we don't expect to. So we would only choose that interpretation if there were no better one or if we were sure we were already talking about eyeglasses. Also, we are more likely to choose this interpretation for something that is filled with something one doesn't normally expect to drink, but when the filling is a drink, there's a stronger disposition to choose the "drinking glass" meaning. Here we do that despite the fact that the glasses belong to an old man and the fact that people often only have one drinking glass, not more than one, whereas eyeglasses are always plural.

2. • "John wanted to go to the movie with Sally." Additional grammar rules:
VP → V INFCOMP
INFCOMP → to VP [We ignore the issue of whether VP is inflected.]
VP → VP PP
PP → PREP NP



- "John wanted to go to the movie with Robert Redford." Further new grammar rules:
  NP1 → NP1 PP



The issue in choosing between the structures shown for these two sentences is the attachment of the prepositional phrase, "with ...". In one case it modifies going, in the sense of there being a co-agent. In the other case, it modifies "movie" in the sense of an actor in the movie. Simple type checking isn't enough here since both roles are filled by people. So instead we need to look at knowledge about particular individuals to determine whether the individual is more likely to be someone who is in a movie or someone John knows whom he might like as a companion.

3. John went to the store to buy a shirt. The sales clerk asked him$_1$ if he$_2$ could help him$_3$. He$_4$ said he$_5$ wanted a blue shirt. The salesclerk found one$_6$ and he$_7$ tried it$_8$ on. He$_9$ paid for it$_{10}$ and left.

   For each numbered reference:

   1. John; The only candidates are John and the salesclerk. The salesclerk can be ruled out on syntactic grounds; it would have required a reflexive pronoun.
   2. The salesclerk; Now we need to appeal to a model (perhaps stored in a script-like structure) of what happens in a store and who helps whom.
   3. John; This must be different from 2 or a reflexive would have been used, so it must be John.

4. John; Again we must appeal to knowledge about stores.
5. John; Same knowledge.
6. a blue shirt; "one" picks out an element of a previously mentioned set. The best candidate is the set of blue shirts mentioned in the previous sentence.
7. John; Same knowledge about stores.
8. the same blue shirt that was found; That is the only candidate.
9. John; Same knowledge about stores.
10. same blue shirt.

4. (a) Put [the red block] on [the blue block [on the table.]]
   Put [the red block on [the blue block]] on [the table.]
   (b) If there is no blue block on the table, the first interpretation can be rejected. If there is no red block on any blue block, the second interpretation can be rejected.

5. • "Everyone doesn't know everything" could mean
   $\forall x, y : \neg know(x, y)$
   or
   $\neg\forall x, y : know(x, y) \equiv \exists x, y : \neg know(x, y)$
   • "John saw Mary and the boy with a telescope" could mean:

```
P1
    instance:  Seeing
    agent:  John
    object:  {MARY, Boy1}
    instrument:  Telescope 1

Boy 1
    instance:  Boy
```

or

```
P2
    instance:  Seeing
    agent:  John
    object:  {Mary, Boy 2}

Boy 2
    instance:  Boy
    owns:  Telescope1
```

• "John flew to New York" could mean:

```
P1
    instance:  Piloting
    agent:  John
    destination:  New York
```

or

```
P2
    instance:  TravelByPlane
    object:  John
    destination:  New York
```

8. [CONSTITUENT1: [{1} CAT: NP
NUMBER: {2}]
CONSTITUENT2: [{3} CAT: VP
NUMBER: {2}]
BUILD: [S: [NP: {1}
VP: {3}]]]

Notice that here we label graphs that we also care about some of the contents of. We labeled the entire NP {1} and the VP {3} so that they could be inserted into the structure for the resulting S and we also had to check that they are of the correct category to match this rule. We also need to verify that their NUMBER fields match. We do this by binding NUMBER of the NP to {2}. Then, by using {2} as the required value for NUMBER of the VP, we take advantage of the fact that unification will only succeed if the two values match. Because both NPs and VPs can be ambiguous with respect to number, we need to be able to use disjunction here (as described on page 396).

10. fly: object: plane
locative: above the clouds

(a) fly: dative: John
goal: New York

(b) fly: agent: co-pilot
object: plane

13. (a) B having a comb is a precondition to his lending it to A. So the sincerity conditions explain why A would need to know this. There is little other reason A would care about this, so the reasonableness conditions suggest that this is not just a request.

(b) On the surface, it appears that B's response violates the appropriateness conditions because he didn't talk about Jones's technical skills. But there is a politeness constraint in many circumstances such as this that says that one should not say anything negative. So if we assume that B is being as appropriate as he can subject to the other constraints, we conclude that Jones's technical skills aren't very good because if they were B would have said so.

(c) If B assumes that A is sincere and reasonable, then he can conclude that A must be asking if B has money since that would be a precondition to A borrowing the money, which he would only do if he intended to spend it. So B can assume that A wants to buy something. Then it makes sense for him to ask what.

15.2 Software

- `graph-unify.lisp`

  Program for doing graph unification, of the type used in unification-based parsing. The file includes several examples.

Chapter 16

Parallel and Distributed AI

16.1 Solutions to Selected Exercises

2. Q should make move *d*.

Chapter 17

Learning

17.1 Errata

- **Page 467.** There is an error in Figure 17.12. The *color* of the fourth example (in which *mfr* = *USA*) should be *Blue*, not *Red*.

17.2 Solutions to Selected Exercises

3. Here are four training examples:

   (*Japan*, *Honda*, *Blue*, 1980, *Economy*) (+)
   (*Japan*, *Toyota*, *Green*, 1970, *Economy*) (+)
   (*Japan*, *Toyota*, *Green*, 1970, *Sports*) (−)
   (*USA*, *Chrysler*, *Green*, 1970, *Economy*) (−)

4. Individual features can be generalized and specialized, just like sets of features. For example, the generalization of *Germany* and *Britain* is *European*; the generalization of *Italy* and *Japan* is *Imported*. The result of specializing *Imported* in order to exclude *Italy* is *Japan*. Another way of looking at this is that the partial ordering shown on page 465 becomes more complex: in between (x_1, x_2, x_3, x_4, *Economy*) and (*Japan*, x_2, x_3, x_4, *Economy*) is the new concept description (*Imported*, x_2, x_3, x_4, *Economy*).

17.3 Software

- `vs.lisp`

  Candidate elimination learning algorithm for version spaces.

Chapter 18

Connectionist Models

18.1 Errata

- **Page 510.** There is a minus sign (−) missing in the formula for Boltzmann updating. The formula:

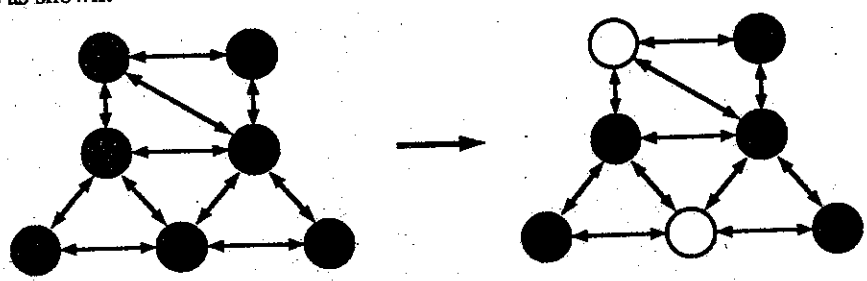
$$p = \frac{1}{1 + e^{\Delta E/T}}$$

Should read:

$$p = \frac{1}{1 + e^{-\Delta E/T}}$$

18.2 Solutions to Selected Exercises

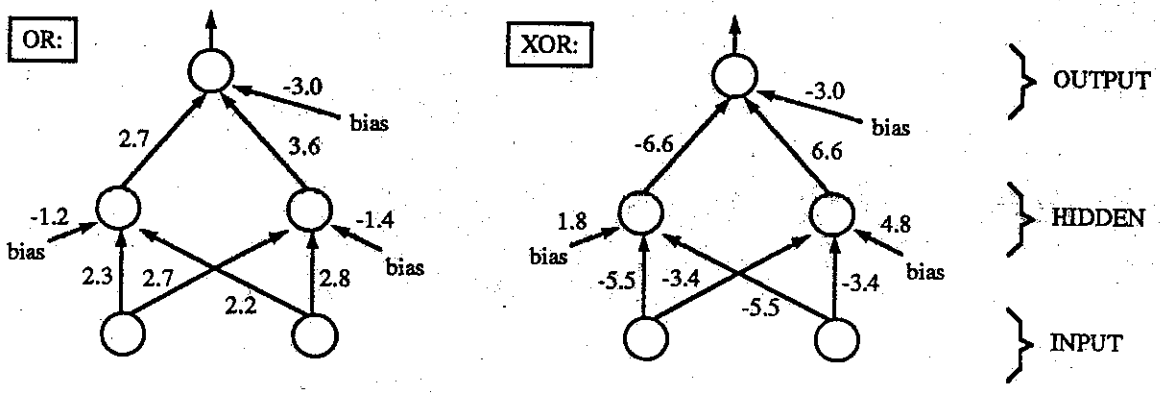
1. It will settle as shown:



Here is an example three-feature problem that is linearly separable: A bank must decide whether to grant a loan to an applicant, and to do so it considers the applicant's age, income, and credit history. The bank has a database of successful and unsuccessful loans it has made in the past. That database can be used to train a perceptron to output either 1 (grant the loan, it will probably be repaid) or 0 (do not grant the loan, it probably will not be repaid).

2. Backpropagation learning is included in the software package (`backprop.lisp`). Here are two networks created by backpropagation (the one on the left solves the OR problem; the one on the right solves the XOR

problem):



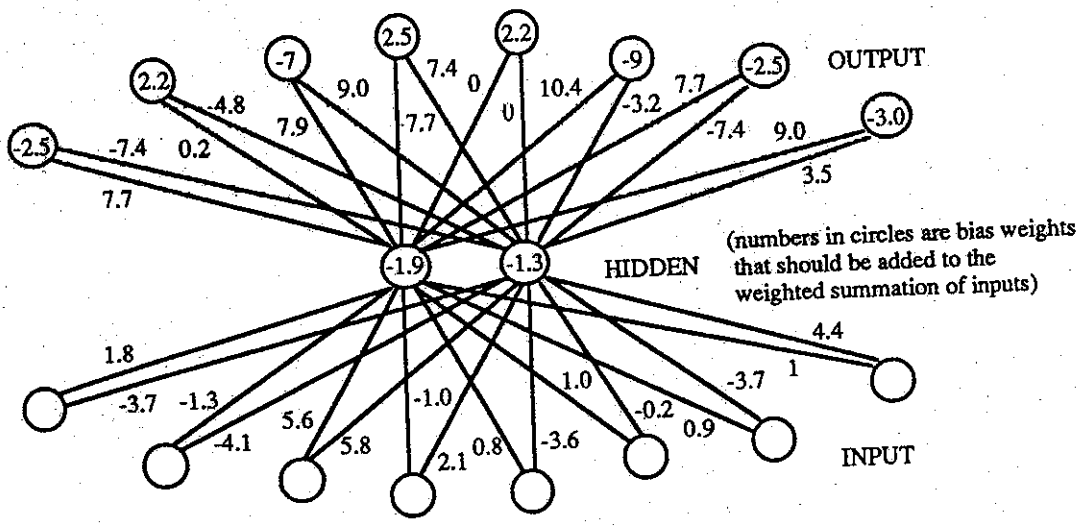
The momentum factor should greatly increase the speed of learning.

3. Here is a sample set of input/output vectors (see also `politics.txt` in the software package):

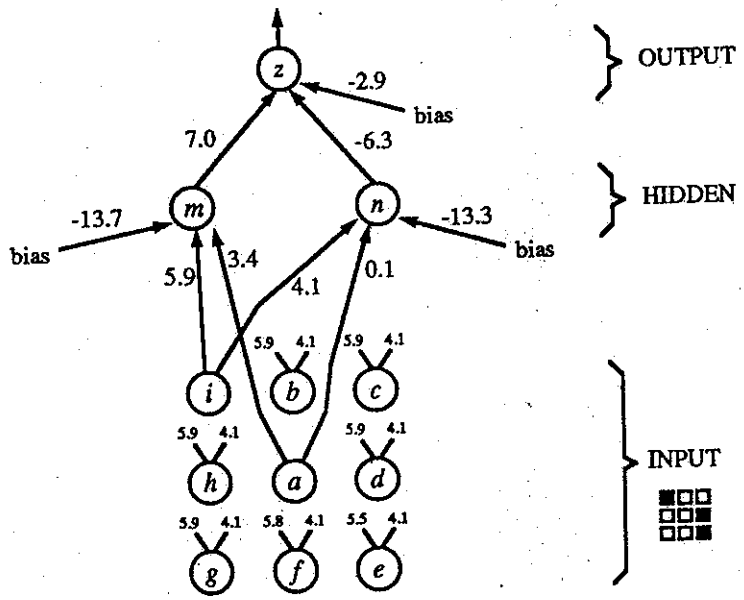
```
Training data:

1 1 0 0 1 0 1 1 --> 1 1 0 0 1 0 1 1
1 1 1 0 1 0 1 1 --> 1 1 0 0 1 0 1 1
1 1 0 1 1 0 1 1 --> 1 1 0 0 1 0 1 1
1 1 0 0 0 0 1 1 --> 1 1 0 0 1 0 1 1
1 1 0 0 1 1 1 1 --> 1 1 0 0 1 0 1 1
1 1 0 0 1 0 0 1 --> 1 1 0 0 1 0 1 1
1 1 0 0 1 0 1 0 --> 1 1 0 0 1 0 1 1
0 1 0 1 1 0 0 0 --> 0 1 0 1 1 0 0 0
1 1 0 1 1 0 0 0 --> 0 1 0 1 1 0 0 0
0 0 0 1 1 0 0 0 --> 0 1 0 1 1 0 0 0
0 1 0 1 0 0 0 0 --> 0 1 0 1 1 0 0 0
0 1 0 1 1 1 0 0 --> 0 1 0 1 1 0 0 0
0 1 0 1 1 0 1 0 --> 0 1 0 1 1 0 0 0
0 1 0 1 1 0 0 1 --> 0 1 0 1 1 0 0 0
0 0 1 1 1 0 0 1 --> 0 0 1 1 1 0 0 1
1 0 1 1 1 0 0 1 --> 0 0 1 1 1 0 0 1
0 1 1 1 1 0 0 1 --> 0 0 1 1 1 0 0 1
0 0 0 1 1 0 0 1 --> 0 0 1 1 1 0 0 1
0 0 1 0 1 0 0 1 --> 0 0 1 1 1 0 0 1
0 0 1 1 0 0 0 1 --> 0 0 1 1 1 0 0 1
0 0 1 1 1 1 0 1 --> 0 0 1 1 1 0 0 1
0 0 1 1 1 0 1 1 --> 0 0 1 1 1 0 0 1
0 0 1 1 1 0 0 0 --> 0 0 1 1 1 0 0 1
1 1 1 0 1 1 1 1 --> 1 1 1 0 1 1 1 1
0 1 1 0 1 1 1 1 --> 1 1 1 0 1 1 1 1
1 0 1 0 1 1 1 1 --> 1 1 1 0 1 1 1 1
1 1 0 0 1 1 1 1 --> 1 1 1 0 1 1 1 1
1 1 1 0 0 1 1 1 --> 1 1 1 0 1 1 1 1
1 1 1 0 1 0 1 1 --> 1 1 1 0 1 1 1 1
1 1 1 0 1 1 0 1 --> 1 1 1 0 1 1 1 1
1 1 1 0 1 1 1 0 --> 1 1 1 0 1 1 1 1

Testing data:

0 1 0 0 1 0 1 1 --> 1 1 0 0 1 0 1 1
1 0 0 0 1 0 1 1 --> 1 1 0 0 1 0 1 1
0 1 1 1 1 0 0 0 --> 0 1 0 1 1 0 0 0
0 1 0 0 1 0 0 0 --> 0 1 0 1 1 0 0 0
1 1 1 1 1 1 1 1 --> 1 1 1 0 1 1 1 1
```

Two hidden units should be sufficient to learn this mapping, since there are essentially four output patterns (11001011, 01011000, 00111001, 11101111). That is, neither hidden unit should be activated when the input is 11001011, the left hidden unit should be activated by 00111001 or 11101111, and the right unit by 01011000 or 11101111. So each input creates a unique pattern of activation across the hidden units. That pattern can then be mapped to the output units.

Sometimes backpropagation learning gets stuck in local minima. The following network almost solves the problem, but not quite. This is because 11001011 and 11101111 map to similar hidden unit configurations. (Also note the zero weights learned for the fifth output unit—it always fires, and this is correct, given the data).



4. Here is a sketch of what one backpropagation network learned (see also `life.txt` in the software package):



Hidden unit *m* fires if at least three of the input units (*a* through *i*) are active. This is because backprop has learned a highly negative bias weight for unit *m* (-13.7).

Hidden unit *n* also has a negative bias weight. This unit will fire only if at least four of the "neighbor" units (*b* through *i*) are active. Note that the center unit *a* has no effect on whether unit *n* will fire (i.e., overcome its bias).

Output unit *z* fires if *m* fires and *n* does not.

We can state what backprop has learned as follows: IF at least three inputs are active AND less than four neighbors are active, THEN output 1; OTHERWISE output 0. This rule is equivalent to the two rules introduced in the exercise, but is formulated substantially differently.

18.3 Software

- `hopfield.lisp`

  Simulator of parallel relaxation in a Hopfield network.

- `backprop.lisp`

  Program for backpropagation learning. Geared toward fully connected, multilayer networks with a single binary output.

- `backprop2.lisp`

  Program for backpropagation learning. Geared toward networks with multiple outputs.

- `xor.txt`, `xor.train`, `xor.test`

  Data for XOR problem as input for backpropagation program. (Exercise 18.3).

- `or.txt`, `or.train`, `or.test`

  Data for OR problem as input for backpropagation program. (Exercise 18.3).

- `politics.txt`, `politics.train`, `politics.test`

  Data for political party problem as input for backpropagation program. (Exercise 18.4).

- `life.txt`, `life.train`, `life.test`

  Data for Life problem as input for backpropagation program. (Exercise 18.7).

Chapter 19

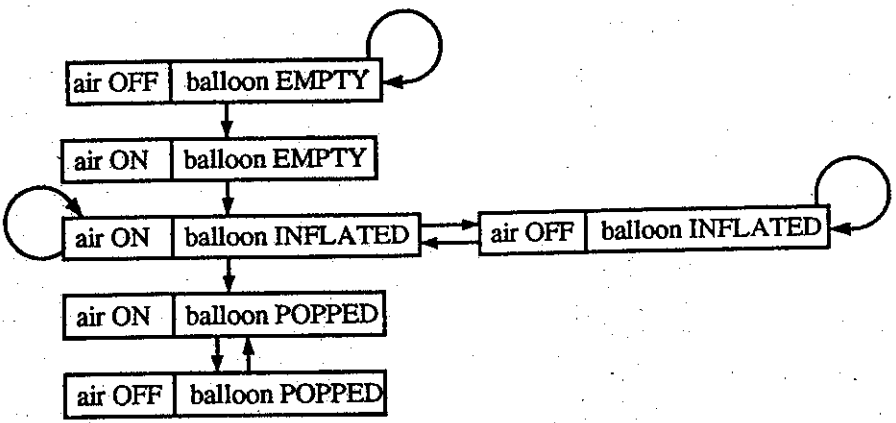
Common Sense

19.1 Errata

- **Page 546.** Exercise 4 should read "... axioms in Section 19.2.3, ..." and not "... axioms in Section 19.2.2, ..."

19.2 Solutions to Selected Exercises

1. These are two variables: air (ON or OFF) and balloon (EMPTY, INFLATED, POPPED). Here is one environment:



One possible history is: (OFF, EMPTY) (ON, EMPTY) (ON, INFLATED) (ON, INFLATED) (ON, INFLATED) (ON, POPPED).

2. Twelve temporal relations expressed in terms of MEETS:

i IS-MET-BY $j \equiv (j \text{ MEETS } i)$

i IS-BEFORE $j \equiv \exists k : (i \text{ MEETS } k) \wedge (k \text{ MEETS } j)$

i IS-AFTER $j \equiv \exists k : (j \text{ MEETS } k) \wedge (k \text{ MEETS } i)$

i EQUALS $j \equiv \exists k, m : (k \text{ MEETS } i) \wedge (k \text{ MEETS } j) \wedge (i \text{ MEETS } m) \wedge (j \text{ MEETS } m)$

i ENDS $j \equiv \exists k, m, p : (k \text{ MEETS } j) \wedge (m \text{ MEETS } i) \wedge (k \text{ MEETS } m) \wedge (i \text{ MEETS } p) \wedge (j \text{ MEETS } p)$

i IS-ENDED-BY $j \equiv \exists k, m, p : (k \text{ MEETS } i) \wedge (m \text{ MEETS } j) \wedge (k \text{ MEETS } m) \wedge (j \text{ MEETS } p) \wedge (i \text{ MEETS } p)$

i OVERLAPS $j \equiv \exists k, m, p, n : (k \text{ MEETS } i) \wedge (k \text{ MEETS } m) \wedge (m \text{ MEETS } j) \wedge$

$$(j \text{ MEETS } n) \wedge (p \text{ MEETS } n) \wedge (i \text{ MEETS } p)$$

i IS-OVERLAPPED-BY $j \equiv \exists k, m, p, n : (k \text{ MEETS } j) \wedge (k \text{ MEETS } m) \wedge (m \text{ MEETS } i) \wedge (i \text{ MEETS } n) \wedge (p \text{ MEETS } n) \wedge (j \text{ MEETS } p)$

i CONTAINS $j \equiv \exists k, m, p, n : (k \text{ MEETS } i) \wedge (k \text{ MEETS } m) \wedge (m \text{ MEETS } j) \wedge (i \text{ MEETS } m) \wedge (p \text{ MEETS } n) \wedge (j \text{ MEETS } p)$

i IS-DURING $j \equiv \exists k, m, p, n : (k \text{ MEETS } j) \wedge (k \text{ MEETS } m) \wedge (m \text{ MEETS } i) \wedge (j \text{ MEETS } n) \wedge (p \text{ MEETS } n) \wedge (i \text{ MEETS } p)$

i STARTS $j \equiv \exists k, m, p : (k \text{ MEETS } i) \wedge (k \text{ MEETS } j) \wedge (j \text{ MEETS } p) \wedge (m \text{ MEETS } p) \wedge (i \text{ MEETS } m)$

i IS-STARTED-BY $j \equiv \exists k. m, p : (k \text{ MEETS } j) \wedge (k \text{ MEETS } i) \wedge (i \text{ MEETS } p) \wedge (m \text{ MEETS } p) \wedge (j \text{ MEETS } m)$

3. Let f denote the time interval of the Franco-Prussian War, w denote World War I, and v denote the Battle of Verdun. The assumptions are IS-BEFORE(f, w) and IS-DURING(v, w). These can be written:

   (1) $\exists k : (f \text{ MEETS } k) \wedge (k \text{ MEETS } w)$

   (2) $\exists a, b, c, d : (a \text{ MEETS } w) \wedge (b \text{ MEETS } v) \wedge (a \text{ MEETS } b) \wedge (v \text{ MEETS } d) \wedge (d \text{ MEETS } c) \wedge (w \text{ MEETS } c)$

   We want to prove IS-BEFORE(f, v), which can be written:

   (3) $\exists p : (f \text{ MEETS } p) \wedge (p \text{ MEETS } v)$

   By (1), there is a k that meets w, and by (2), there is an a that meets w. Since a also meets b (2), then it must be the case that k meets b. This follows from the uniqueness of meeting points (p. 534). If k and b meet, then there is an interval kb that starts where k starts and ends where b ends. This follows from the axiom of the existence of continuous union intervals (p. 535). That axiom states that there is an interval that meets both k and kb (and another that is met by both b and kb). By (1), f meets k. We can apply the uniqueness of meeting points again to conclude that f meets kb. By similar reasoning, we can prove that kb meets v (b meets v by (2)). We are finished, since f meets kb, which meets v. That is, we have proved that an interval exists between the end of the Franco-Prussian War (f) and the beginning of World War I (w).

4. The assumptions of the problem are as follows:

   (1) FULL(river, water)

   (2) INSIDE(robot, river)

   Since the river is not wholly contained within any solid object,

   (3) FREE(river)

   To prove that the robot IS-WET-ALL-OVER, by the definition on page 539, we need to show that for every outer space d of the robot, there exists some liquid l such that WET-BY(d, l). Consider any such d. Since d is not wholly contained inside a solid object:

   (4) FREE(d)

   By the axiom $\forall d. o : \text{OUTER}(d, o) \rightarrow \text{INSIDE}(d. \text{SURROUND}(o))$,

   (5) INSIDE(d, SURROUND(robot))

   Combining (2) and (3) with the axiom $\forall s. o : \text{FREE}(s) \wedge \text{INSIDE}(o, s) \rightarrow \text{INSIDE}(\text{SURROUND}(o), s)$ we get:

   (6) INSIDE(SURROUND(robot), river)

   By the transitivity of INSIDE applied to (5) and (6):

(7) INSIDE(d, river)

Now we can use the last axiom on page 539, which states that INSIDE(s_1, s_2) $\wedge$ FREE(s_1) $\wedge$ FULL(s_2, l) $\rightarrow$ FULL(s_1, l). Using (7), (4), and (1), we get:

(8) FULL(d, water)

which means that the outer face d is full of water. The definition of FULL from page 538 tells us that the amount of water in d equals its capacity. This satisfies the definition of WET-BY on page 539, so that we can conclude:

(9) WET-BY(d, water)

and the proof is complete.

Chapter 21

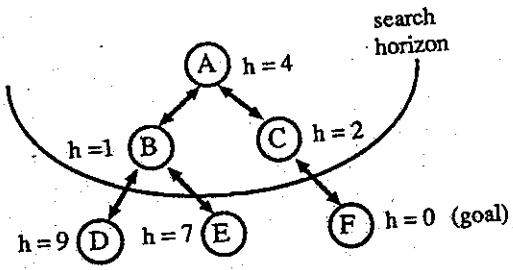
Perception and Action

21.1 Solutions to Selected Exercises

1. Consider a robot bartender. The robot must be reactive: drink orders may be changed or cancelled by shouting customers. It must be robust: it should ask for clarification if a drink order is slurred or drowned out. And it must be able to recover: if another robot bartender has taken the last bottle of gin, it must abandon its simple plan in favor of a more complex gin-acquisition strategy.

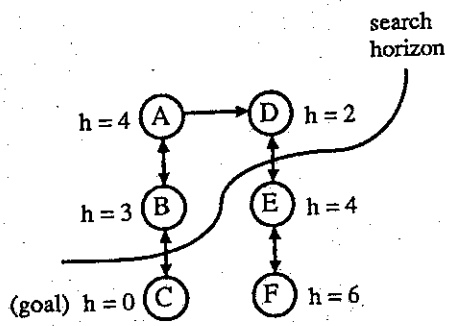
   The planning techniques described in Chapter 13 assume a complete and correct description of the world. They are helpless if some of the information required for the plan is not available. They make no provision for interleaving planning and execution.

4. Consider the following search graph:



   Node A is the start node. Assume a search horizon of only one move. Based on local search, RTA* chooses to move to node B. Now there are three choices: A, D, and E. A looks most promising, so the algorithm moves there. But if the algorithm does not remember that it has been at node A before, it will again choose B. It will keep oscillating between A and B, never reaching the goal.

5. Consider the following search graph:



   It is possible to move from A to D, but not vice-versa—perhaps the move involves mixing two liquids together. In any case, it is impossible to reach nodes A, B, and C from any of D, E, or F. RTA*, on the basis of local

search, will choose node D over B. Once it makes that move in the real world, it is impossible to reach the goal node F.

21.2 Software

- `rta.lisp`

  RTA* search algorithm. Works with domains like `n-puzzle.lisp` and `travel.lisp`. See Section 3.3 for more details.

Chapter 22

Hints to Exercises:

1. Now that a fuzzy cooler is in place, design a simple fuzzy washing machine that can decide the speed of its motor based on the dirt level, type and the weight of the cloth.

The Instructor should motivate the student to think on similar lines as the cooler described.

2. Visit the library of your institution, this time, for a different purpose. Inspect what kind of books are placed in each rack. Write a fuzzy logic program that can suggest the rack into which a new arrival (book) can go.
One simple method is suggested below. The first task is to process the title of a book and extract significant key words from it. Membership of these keywords to a an area/field (Computer Science, Chemistry, Electronics,…) could now be computed based on their occurrence in respective corpora related to these areas.
Fuzzy rules like: If book is Close Computer Science and Average to Electronics then it belongs to Computer Science could be used.
Defuzzification can be performed by the maximum truth value.

3. Based on the fuzzy cooler, model the exposure time of a typical camera. Assume the camera has sensors to detect ambient light, controls for flash and picture type (landscape, close, etc.) settings. Also assume that exposure times are based on values reported by these sensors.

Here too, the Instructor should motivate the student to think on similar lines as the cooler described. Parameters and profiles of course are different and need to be discussed accordingly.

Chapter 23

Hints to Exercises:

1. Generating a time table for an academic curriculum is a complex problem if we consider the priorities (preferred time of class, course, etc.) of faculty members and available resources (class rooms, free time, etc.). Several attempts have been made to apply Genetic algorithms for the generation of an optimal or near optimal time table. Try formulating this problem to suit the genetic algorithm. Decide on the representation of the chromosome, the crossover points and how both crossover and mutation should be realized in the search for the optimal goal.

Time table generation is known to be a complex problem. Though a GA may not be able to arrive at the most optimal table, it can definitely be used to generate near optimal solutions that could be manually tweaked, thus saving a lot of nerve racking work!
The Instructor could highlight to the students, the nature in which the time table in his/her academic institution is prepared. The different ways in which the chromosome can be structured may be discussed.

One simple way is to form an m x n matrix that resembles the contents of a typical time table. A few constraints, for instance, are cited below:
Professor X wishes to take a class on AI on either Mondays, Tuesdays and Fridays and would prefer to lecture at 10:00AM except on Fridays when he wishes to do so at 4:00PM.

A table of such constraints could be formed for faculty members. (Maybe senior Professors could be given preference!)
Likewise room availability constraints could cite the number and their size (maximum capacity).
Lunch and tea breaks too have to be accounted for as also the number of courses and lectures and labs. per week.

Crossover could be either random or controlled. For instance a crossover may cause allocation of one Professor to two lectures at the same time. This could be detected immediately and the crossover effect nullified.

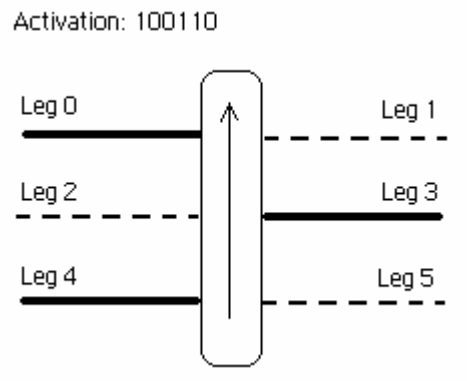
Mutation could be realized by randomly moving a lecture to another free area. Some thought should be given to the mutation rate and the manner of mutation.

A fitness function to evaluate each time table generated could take into consideration parameters like:

How well does it satisfy all the Professors' preferences?

Is there a proper distribution of courses? (It should not be that all the lectures of a course are held on Monday and Tuesday or students opting for some set of courses do not have to attend classes at all on some days).
The above parameters could be quantified and a fitness function formed.

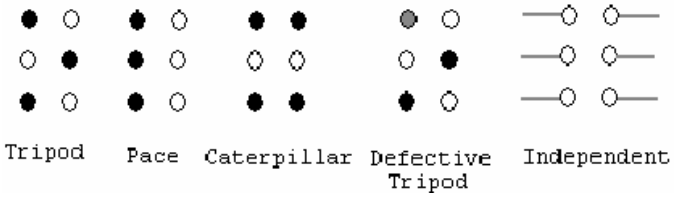
2. In the animal world, the ant, grasshopper, the centipede and the millipede move using different gaits and stances. These probably evolved after several generations. Can we make a multi-legged robot learn to find the best and most stable gait? Use a Genetic Algorithm to simulate and aid this robot to learn how to achieve optimal locomotion.
In the real world this problem requires a fitness function that could be calculated by actually trying out each solution. A 6-legged insect-like robot for instance could look something like this –



Students could be asked to look at *Stiquito* (http://www.stiquito.com/) the insect like robot and its versions.

The more the number of legs, the more is the number of possible gaits. This number also increases with the degrees of freedom of the legs. The above insect-like robot has six legs. Each can move forward when activated or remain in the position shown. The possible number of activation signals can go up to $2^6 = 64$. By applying different sequences of activation signals several gaits can be generated. The idea is to find the best gait using GAs.

Some of the possible gaits for a hexapod insect are shown below.



Evaluation could be performed by a fitness function that could take into consideration the actual distance traversed by the robot after several repeated cycles of a particular gait. In short if we consider the leg activation sequence as 10010 (1 means activated; 0 means stationary) then this sequence forms a gait. While such a sequence (solution) could be

generated using GAs, the robot's legs could be activated say around a 100 times with this sequence repeatedly and the distance traversed by it could form the fitness of this gait.

3. How would you go about embedding the above genetic algorithm into a real multi-legged robot (imagine a millipede or a centipede-like robot) so that it will evolve and learn to find the best gait in the real world?

The crucial part as mentioned earlier is the evaluation of the distance traveled (fitness of the gait). In the real world there is no better way than the *hard* way – make the robot walk physically using every gait several times. The solutions obtained from the simulation phase could be used as the initial starting population. Measuring the distance of course requires some ingenuity!

Chapter 24

1. A smart house needs an efficient burglar detection and alarm system. Can you model these using AIS concepts?

You could imagine numerous strategically located burglar detectors each comprising of cameras and proximity sensors that can trigger alarms. Cameras could sense movement.

Immune system	Approx. Metaphors
Lymphocyte (B-cell,T-cell)	Burglar Alarm
Antibody variable region	Alarm tone/intensity
Memory cell	A vulnerable area (in the fence) detected earlier by one of the detectors (camera).
Pathogen	Burglar
Circulation	Detectors could be networked and signal others of a possibly intrusion causing increase their sensitivity levels.
MHC	Representation parameters/Image and proximity processing
Cytokines	Sensitivity level
Signals	Activation threshold, human owner
Pathogen detection	Detection event
Pathogen elimination	Response from alarm and human intervention

The student could be asked to provide his/her own viewpoints about the model being tailored to suit AIS concepts within.

2. Many events can be looked upon as a defence mechanism. For instance the simple fact that you can balance on two legs is a mechanism to defend (or prevent) your centre of gravity from moving away from a stable point. Self preservation can be viewed, likewise, as the capability to *defend* your body. At every stage your actions (be it moving, talking, thinking, …) are all measures taken to make you feel stable (comfortable). With this in mind try modelling the following into an AIS based system:

(i) Riding a cycle
(ii) Spam detection and elimination

The Instructor is urged to motivate the students to think of each of the above problems as one that has an Antigen and a Pathogen. Rest of the metaphors could then be discussed.

(i) This is hypothetical unless we actually model an AIS based robot to actually ride the cycle. There have been attempts to balance a pole using a numerous sensors, each acting on the system to stabilize it. Riding a cycle could also be looked upon as one such system wherein sensors could find the tilt, the movement of the centre of gravity and take corrective action using AIS concepts. Destabilization/shift in the centre of gravity could be viewed as an antigen. Its detection could trigger antibodies in the form of motor (muscle) activations that

counteract this change in an attempt to balance the cycle. Initially generated antibodies may not be all that successful but a clonal selection mechanism will allow those successful to become memory cells and result in stable and safe riding over a period of time.

(ii) Spam has become a perennial problem. While many attempts have been made to circumvent it, the fact that spammers change their modes of spamming is highly frustrating.. An initial repertoire of words (naïve B/T-cells) could be extracted from an existing 100% spam database (similar to the bone marrow model). These could be used as starting material to counter spam. These words could undergo somatic mutation/hypermutation to catch the ever changing camouflaged words within the incoming email and trigger a detection. For instance the word – Viagra may have been spelt as Vi$agra. Mutation can help generate such non-words from real words. Students could be encouraged to use WordNet (http://wordnet.princeton.edu/) to find words related to spam words (such as synonyms) so as to make the word population diverse.