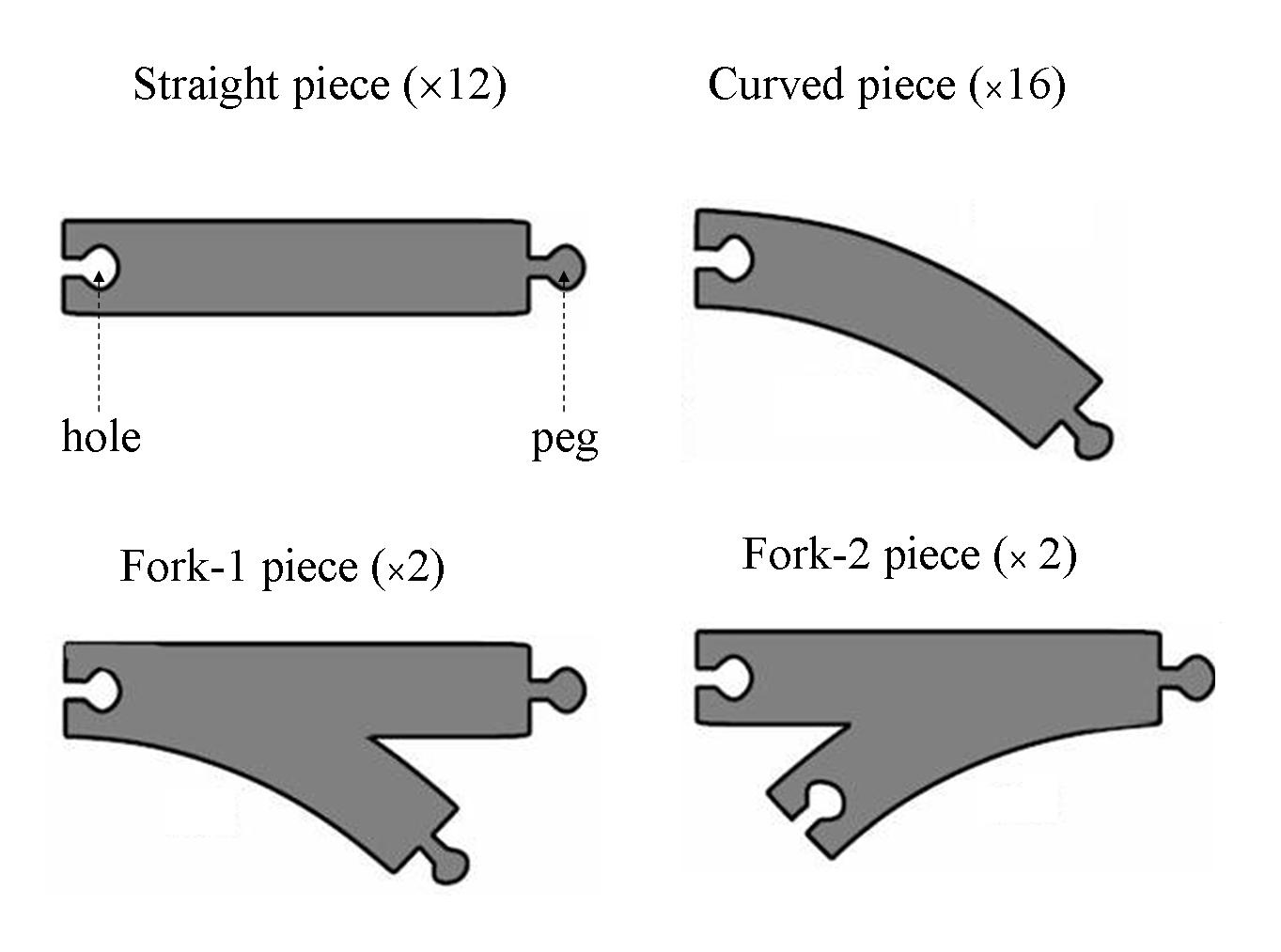
**B351 Midterm Review Questions**

**I. Toy railway**

A toy-railway set contains four types of track pieces shown in Figure 1. There are 12 straight pieces, 16 curved pieces, 2 fork-1 pieces and 2 fork-2 pieces. Each straight and curved piece has a *hole* at one end and a *peg* at the other end. A fork-1 piece has one hole at one end and one peg at each of the other two ends. A fork-2 piece has one peg at one end and one hole at each of the other two ends. Two pieces can be *connected* together by inserting a peg of one into a hole of another. The pieces fit together exactly with no slack. The curves and fork pieces can be flipped over; hence, they can curve in either direction. Each curve subtends a 45-degree arc.



**Figure 1.** The pieces of the toy-railway set

The task is to connect **all** those pieces into a **single** railway track that has no **open ends** where a train could run off and no **overlapping** tracks.

1. Give a precise formulation in words of the task as a search problem, that is, describe the state space, the successor function, the initial state, and the goal test.

[Hints and comments:

* If this helps, you may call a peg that has not been inserted yet into a hole, an *open* peg.
* Choose your successor function to limit the branching factor; if your branching factor is above 10, you will not get full credit. State explicitly how many successors are contributed by each piece of a given type.
* You do not need to provide all details of the goal test; just state what needs to be tested.
* Ignore step cost.]

1. Among the following blind search methods – breadth-first, depth-first, depth-limited, iterative deepening – which one would you choose? Why?
2. Explain briefly why removing any one of the fork pieces from the railway set of Figure 1 would make the problem unsolvable. Derive a general *necessary*, but not *sufficient*, condition for any given railway set (using the 4 types of pieces shown in Figure 1) to have a solution.
3. Give an upper bound on the number of nodes generated by either breadth-first or depth-first search for your formulation at Question 1. [This question will earn you points only if your formulation at Question 1 is correct. Your bound need not be very tight, but your grade will still depend on how tight it is.]
4. The state space consists of all connected, non-overlapping partial tracks, of up to 32 pieces. The initial state is a single piece (it can be chosen arbitrarily). Goal states are those for which all 32 pieces are placed, and there are no open pegs.   
     
   A successor function picks the first open peg of the track, and then produces a successor for all combinations of 1) a remaining piece from one of the four classes, 2) a hole from that piece, and 3) an orientation (either CCW or CCW) for a curved or a fork piece. (If a class has no pieces remaining, then no, then that class generates no successors) The successor function places the hole of the chosen piece at the first open peg of the state, and rejects candidate states that overlap. Straight pieces contribute at most 1 successor, curve pieces contribute 2 successors, fork-1 pieces contribute 2 successors, and fork-2 pieces contribute 4 successors.
5. Depth first search would be preferable here because 1) the problem is pathless, and 2) the depth of the tree is limited by 32, and 3) there will be few revisited states.
6. Each peg needs to be connected to a hole, and removing 1 fork piece would cause the number of holes and pegs to be unequal. A necessary condition is that the total number of holes and the total number of pegs among all pieces must be equal.
7. An upper bound is 9^32 – the branching factor is at most 9, and the depth is at most 32.

**II. CSPs for Cryptarithmetic**

Consider the following cryptarithmetic problem:

S E V E N  
 S E V E N  
+ S I X  
\_\_\_\_\_\_\_\_\_\_\_  
T W E N T Y

Each letter represents one of the digits 0-9, and each letter must represent a unique digit. S and T are constrained to be nonzero. Your job is to find an assignment of the digits such that the algebraic equation is satisfied.

1. Formulate this problem as a CSP. Give the variables, domains, and constraints. Hint: add new variables c1, …, c5 to represent the carry terms.

c1c2 c3 c4c5 S E V E N  
 S E V E N  
+ S I X  
\_\_\_\_\_\_\_\_\_\_\_  
T W E N T Y

2. Which variable would a CSP backtracking algorithm assign first using the most-constrained-variable heuristic? What would the most-constraining-variable heuristic pick?

3. Suppose that we extend AC3 to properly handle constraints that relate more than two variables. Which values of which variables’ domains would AC3 eliminate?

4. Suppose the backtracking algorithm chose the partial assignment S=2, N=0, c3=0. Would forward checking or AC3 detect that this assignment is inconsistent? Why?

1. Variables:

* c1, c2, …, c5 in {0,…,9}
* S, T in {1,…,9},
* W, E, V, N, E, I, X, Y in {0,…,9}

Constraints:

* Alldiff(S, T, W, E, V, N, E, I, X, Y)
* c1 = T
* c2 + S + S = W + 10\*c1
* c3 + E + E = E + 10\*c2
* c4 + V + V + S = N + 10\*c3
* c5 + E + E + I = T + 10\*c4
* N + N + X = Y + 10\*c5

2. The most-constrained-variable heuristic would pick either S or T because they have the smallest domains. The most-constraining-variable heuristic would pick either S or T arbitrarily, because they are both involved in 2 constraints.

3. Picking variables in arbitrary order…  
a) c2,…,c5 would have 3-9 eliminated, leaving {0,1,2}. c1 would have 0 eliminated, leaving {1,2}  
b) T would have 3-9 eliminated, leaving {0,1,2}  
c) E would have 1-7 eliminated, leaving {0,8,9}  
d) c2 would have 2 eliminated, leaving {0,1}  
e) c1 would have 2 eliminated, leaving {1}, leaving {1} for T  
f) S would have 0-3 eliminated, leaving 4-9

4. Forward checking would not detect that it is inconsistent because it only works with binary constraints. AC3, correctly modified so that it handles non-binary constraints, would detect that N=0 causes a contradiction with the constraint N+N+X=Y+10\*c5, because no values of X, Y, and c5 could make the constraint hold.

**III. Minimax Algorithm**

1. Prove the following assertion: For every game tree, the utility obtained by MAX using minimax decisions against a suboptimal MIN will never be lower than the utility obtained by playing against an optimal MIN. (hint: use a recursive argument)
2. Come up with a game tree in which MAX can do better using a suboptimal (non-minimax) strategy against a suboptimal MIN.
3. Let V(N) be the minimax values at the game tree rooted at N. Let VSUB(N) be the backed up values at N assuming that MIN plays suboptimally. We want to show that VSUB(N) >= V(N) for all N.  
     
   At the base case, N is a terminal state. Then, clearly VSUB(N) >= V(N) because equality holds.  
     
   In the inductive case, suppose the inductive assumption holds on all of N’s children. We now need to show that it’s the case at N. If N is a MAX node, then let C be the child of N chosen by MAX. VSUB(N) = VSUB(C) >= V(C) by the inductive assumption . If it’s a MIN node, then let C be the child of N chosen by MIN, and let C\* be the minimax choice. Then VSUB(N) =VSUB(C) >= VSUB(C\*) because MIN might make a suboptimal choice. Finally, VSUB(C\*) >= V(C\*) = V(N) by the inductive assumption. Since VSUB(N) >= V(N) in all cases, it must always hold true.
4. Recall the 2-level binary game tree shown in class, where a MAX root has 2 MIN children, and each MIN child has two terminal children. The terminal states in the left branch have value 0, while the terminal states in the right branch have values +1 and -1. The minimax value at the root is 0, but if MIN were to play suboptimally and pick the +1 child, then MAX could do better by picking the right branch.

**IV. Action Planning: Riding an Elevator**

A robot is on the 2nd floor of a building and wishes to go to the 5th floor. The initial state is described by the following conjunction of propositions:

ON(Robot,2) ∧ WORKING(Elevator3) ∧ ON(Elevator3,8).

The goal of the robot is defined by: ON(Robot,5).

The action of the robot calling an elevator *e* from floor *f*1, so that the elevator eventually arrives at *f*1, is described by the following STRIPS action schema:

**Call**(*e*,*f*1)

P = ON(Robot, *f*1) ∧ ON(*e*, *f*2) ∧ WORKING(*e*)

D = ON(*e*, *f*2)

A = ON(*e*, *f*1)

Note that the location of the elevator is updated.

1. Write the STRIPS action schema **Ride**(*e*,*f*1*,f*2) that represents the action of the robot riding an elevator *e* from floor *f*1 to floor *f*2.
2. Is **Call**(Elevator3,5) relevant to achieving ON(Robot,5)? Is **Ride**(Elevator3,2,5) relevant to achieving ON(Robot,5)?
3. Express in English what the regression of a goal through a STRIPS action is. Is it a condition, or an action, or something else? If it is a condition, what is its meaning?
4. Compute the regression of the goal ON(Robot,5) through **Ride**(Elevator3,2,5).
5. Compute the regression of the goal ON(Robot,5)∧ON(Elevator3,4) through **Ride**(Elevator3,2,5)? Is the obtained condition achievable? If not, why can’t the planner detect that immediately? What additional knowledge would have to be given to the planner to make this detection possible? Is this knowledge implicit in the descriptions of **Ride** and **Call**?
6. Let the initial state be the one given at the beginning of this problem and the goal of the robot be ON(Robot,5). Let the robot’s planner be a breadth-first backward planner. So, the root of the search tree will be labeled by ON(Robot,5). The arcs of the tree will be labeled by actions and the other nodes will be labeled by regressed conditions. When level 2 is completed, there will be a path in the tree whose first arc is labeled by **Ride**(Elevator3,2,5) and second arc is labeled by **Call**(Elevator3,2). What is the regressed condition labeling the node at the end of this path? Is this condition satisfied in the initial state? How?
7. **Ride**(*e*,*f*1,*f*2)

P = ON(Robot, *f*1) ∧ ON(*e*, *f*1) ∧ WORKING(*e*)

D = ON(Robot, *f*1) ∧ ON(*e*, *f*1)

A = ON(Robot, *f*2) ∧ ON(*e*, *f*2)

1. No, and yes.
2. The regression of a goal G through an action A is the minimum set of proposition that must hold in a state S such that all the goals in G will hold if A is applied to S. It is a condition.
3. R[ON(Robot,5),Ride(Elevator3,2,5)] = ON(Robot,2) ∧ ON(Elevator3,2) ∧ WORKING(Elevator3)
4. R[ON(Robot,5)∧ON(Elevator3,4),Ride(Elevator3,2,5)] = ON(Robot,2) ∧ ON(Elevator3,2) ∧ WORKING(Elevator3) ∧ ON(Elevator3,4). Obviously this is a contradiction because an elevator cannot be ON floor 2 and floor 4 at once. We would have to tell the planner that elevators cannot be ON two floors at once, perhaps using a state constraint. This constraint is implicit in the definitions of Ride and Call, because they always delete an ON proposition and add a new one. However, more logic would be needed to be added to the planner to be able to detect that.
5. This question requires some logic machinery not discussed in class, but is described in more detail in the book. Don’t worry about this one.