CSCI 291 — Spring 2015

Final Exam

This exam should have seven problems and contain seven pages. Feel free to write helper functions as part of any of your solutions. Comments are not required for full credit. Closed book and notes. No calculators allowed or necessary.

Problem 1: [10 points]

We could use a notation in Haskell to describe circuits, similar to the one we used on the final Prolog assignment. In this notation, a circuit would be represented by a list of “gates”, such as:

```
["and","a","b","ab"],["not","ab","nab"],["or","c","nab","out"]
```

Define the function `containsNot` that takes a list of gates in the format shown above and returns true if it contains one or more not gates. For example:

```haskell
> containsNot [["and","a","b","ab"]]
False
> containsNot [["and","a","b","ab"],["not","ab","nab"]]
True
> containsNot [["and","a","b","ab"],["or","c","nab","out"]]
False
```
Problem 2: [15 points]

Define the Haskell function `findMinimum`, which takes a one-argument function and two integers (denoting the endpoints of a range of values) as its inputs. It should return the smallest output from the one-argument function that occurs on the specified range of integers. For example, the first test below asks Haskell to find the smallest value produced by the identity function on the range from 5 to 10. The second test passes in the “square” function, and the smallest output from that function on inputs ranging from -5 to 5 is 0. For full credit, avoid explicit recursion in your solution.

```haskell
> findMinimum (\x->x) 5 10
 5
> findMinimum (\x->x*x) (-5) 5
 0
> findMinimum (\x->10-x) 5 15
 -5
```
Problem 3: [15 points]

Define the Prolog predicate `reverse`, that takes two lists as arguments and succeeds if one is the reverse of the other.

?- reverse([a,b,c],X).
X = [c, b, a].

?- reverse(X,[a,b,c,d]).
X = [d, c, b, a] .

?- reverse([x,y,z],[z,x,y]).
false.
Problem 4: [20 points]

Define the Prolog predicate `connected` that determines whether two lines in a circuit are connected. Assume that the program uses gate facts like those shown below to define circuits, and that a collection of facts have been asserted as part of your program.

```prolog
  gate(and,a,b,and_out).
  gate(or,and_out,c,or_out).
  gate(not,or_out,out).
```

Now define `connected(In,Out)`, which is true if there’s a path from In to Out in the circuit. The tests below show how it would behave given the circuit description above.

```prolog
?- connected(a,b).
false.

?- connected(a,and_out).
true ;
false.

?- connected(b,out).
true ;
false.
```
Problem 5: [20 points]

The questions below refer to this Prolog program:

r1) parent(brad,charlie).
r2) parent(brad,flora).
r3) parent(durkee,brad).
r4) parent(durkee,trevor).

f1) ancestor(Old,Young) :- parent(Old,Young).
f2) ancestor(Old,Young) :- parent(Old,Mid), ancestor(Mid,Young).

a) On the back side of the previous page, draw the full search tree corresponding to the query ancestor(durkee,flora). Label each edge of the tree with the fact and rule labels used above, and label the leaves in the tree with true and false to denote successful and unsuccessful solutions.

b) Assume that a cut was introduced between the two goals in the right-hand side of the last rule. Would this cut remove any unique solutions from the meaning of the program? Explain.
Problem 6: [20 points]

Answer the questions below that ask you to reflect on the final Prolog assignment. Full credit requires thoughtful and insightful responses.

a) Would you have preferred to implement the circuit assignment in Haskell? Which aspects of project would’ve been easier in Haskell? Which would’ve been harder?

b) Which aspects of the project would’ve been easier if you’d implemented it in Java? Which would’ve been harder?