Homework 6

Michael Anderson

May 11, 2011

CS517

Prof. Cull

1

If S has a primitive recursive acceptor, then it can be used to build a primitive recursive recognizer. By definition of primitive recursive, we can calculate the maximum number of steps that the acceptor could possibly run if it will ever halt by multiplying the maximum number of steps in its bounded loop(s) by the maximum number of iterations of its bounded loops for some input. So run such an acceptor for some input up to the calculated maximum number of steps, and if it halts at some point and returns YES than the input is in S. If the acceptor halts and returns NO, or if it has not yet halted, than the input is in $\bar{S}$. The recognizer runs in a boundable amount of time and is therefore in PRIM.

Very much like in the last paragraph, we can build an exponential time recognizer from an exponential time acceptor. By definition, an exponential time acceptor will halt and return YES on any input of size n in no greater than $c\alpha^n$ time for some constants c and α. So a recognizer should run the acceptor on n for $c\alpha^n$ steps and return YES if the acceptor halts and returns YES, or return NO if the acceptor halts and returns NO or if it has not yet halted after $c\alpha^n$ steps. The recognizer runs in no more than exponential time in the size of n.

Finally, one example of a class of sets that have have acceptors and not recognizers is the HALT set, or any other set in RE-COMPLETE. By definition of RE these sets have acceptors, but by Turing's diagonalization argument they cannot have recognizers.

2

Suppose we find that a graph has a Hamiltonian circuit. Then it must also have a Hamiltonian path, because we can remove the last edge in any Hamiltonian circuit to get a Hamiltonian path.

Suppose we find that a graph G has no Hamiltonian circuit. Perform the following algorithm to check for a Hamiltonian path in G:

```
FOR EACH vertex V1 in G:
    FOR EACH vertex V2 in G:
        if there is no edge in G between V1 and V2:
            make a graph G' with all of the vertexes and edges of G, but that
             also contains an edge between V1 and V2:
            IF there is a Hamiltonian circuit in G':
                RETURN YES
RETURN NO
```

This algorithm works because if there is such a G' with a Hamiltonian circuit then the corresponding path in G without the (V1,V2) edge is a Hamiltonian path. This algorithm requires at most n^2 (polynomial) calls to a Hamiltonian circuit finder, making Hamiltonian path no harder than Hamiltonian circuit.

3

Suppose we find that a graph has no Hamiltonian path. It cannot have a Hamiltonian circuit, since a Hamiltonian circuit is always a Hamiltonian path plus one additional edge traversal.

Suppose we find that a graph has a Hamiltonian path. Given some polynomial time algorithm that finds all such paths, we could then check each one to see if the start and end vertexes of such paths share an edge, and if any of them have this property then add this extra edge to make a Hamiltonian circuit. If none of them have the property then there is no Hamiltonian circuit. The problem is that I cannot figure out how to find all of the paths in polynomial time, using just the decision version of Hamiltonian path.

```
return NO
```

4

Starting at V_1 and applying Prim's algorithm gives the edges of the minimum spanning tree as:

$$(v_1, v_9), (v_1, v_3), (v_9, v_8), (v_3, v_2), (v_8, v_6), (v_6, v_7), (v_7, v_5), (v_5, v_4)$$

So for a path that the salesman might take based on this tree we have the following edges along with their weights:

$$(v_1, v_9, 1), (v_9, v_8, 3), (v_8, v_6, 4), (v_6, v_7, 3), (v_7, v_5, 1), (v_5, v_4, 4), (v_4, v_3, 6), (v_3, v_2, 4)$$

For a total cost of $1 + 3 + 4 + 3 + 1 + 4 + 6 + 4 = 26$. Upon inspection it looks as though in this example the algorithm has actually produced the optimal path.

5

6

Use n (polynomial) number of invocations of the oracle to guess the proper relabeling of each vertex in G_1 to G_2, and each edge in G_1 to an edge in G_2. This is a non-deterministic polynomial time algorithm in the number of vertexes and edges that solves the problem, therefore it is in NP.

Also, let $G_1 = (V_1, E_1)$ and let $G_2 = (V_2, E_2)$. Let $SOLUTION_V(v)$ and $SOLUTION_E(e)$ be the relabelings given by the oracle to V_1 and E_1 respectively. The solution is correct if the following algorithm returns YES, and incorrect if it returns NO:

```
IF SOLUTIONV is not 1-1 or onto from V1 to V2, or SOLUTIONE is not 1-1 or onto
 from E1 to E2
    RETURN NO
IF there exists some edge (v,v') in E1 such that
 SOLUTIONE((v,v')) != (SOLUTIONV(v), SOLUTIONV(v'))
    RETURN NO

RETURN YES
```

Since the steps of this algorithm are polynomial in either the number of the vertexes or the number of the edges of G, as they simply require iterating through each of them and performing constant time operations in each iteration, any solution is verifiable in polynomial time, the other formulation of NP.

7

This problem does not seem to be any harder than graph isomorphism, which is known to be in NP, but has not been proven to be NP-complete. Any regular expression (star-free or not) can be represented easily and exactly by a transition graph, where edges represent the reading of some string contained within the expression, and $\wedge$'s and $\vee$'s are represented by arcs. If edges are then labeled by the strings that must be read in to enable the transition, then simply compare the nodes, edges, and their labels of one regular expression transition graph to another, and if all three are equivalent then the regular expressions are equivalent, and vice versa for non- equivalence.