Problem Set 7: CS103

Katherine Cheng, Richard Davis, Marty Keil

June 3, 2015

Problem 1: Closure Properties of RE

i. bool inL1nL2(string w) {
return inL1(w) && inL2(w);
}

Will return true when true, returns false or loops infinitely otherwise.

ii. Evaluates left to right (confirm?), so if the first function is an infinite loop, will not evaluate the second function (which might be true)

iii. bool imConvincedIsInL1uL2(string w, string c) {
return imConvincedIsInL1(w, c) || imConvincedIsInL2(w, c);
}

Now won't get caught in an infinite loop.

Problem 2: Password Checking

i. We begin with the assumption that this program is a valid password checker. If this is the case, this means the program only accepts a single string p. However, because of the way this program is constructed, we see that if it is a valid password checker we will be able to find some certificate c such that the program will always accept no matter the input string. This is a contradiction.

ii. We begin with the assumption that this program is not a valid password checker. In this case, the program is constructed in a way that no matter what the input string is the program never accepts. This is consistent with the program not being a valid password checker.

iii. The modified program is as follows:

```
bool imConvincedIsPasswordChecker(string program, string certificate) {
/* ... some implementation ... */
}

int main() {
    string me = mySource();
    string input = getInput();

    bool actualAnswer = (input == p);

    for (int i = 0 to infinity) {
        for (each string c of length i) {
            if (imConvincedIsPasswordChecker(me, c)) {
                accept(); // Once the certificate is found, accept any input
            } else {
                if (actualAnswer) {
                    accept();
                } else {
                    reject();
                }
            }
        }
    }
}
```

This program leads to a contradiction regardless of whether it is a password checker. If the program is a password checker, by the same argument as in part i. we can show that this leads to a contradiciton. If the program is not a password checker, we know that `imConvincedIsPasswordChecker` returns false for all values of c. However, the program is constructed such that for all values of c, the program behaves like a password checker, only accepting input strings that are equal to p. In other words, if the program is not a password checker (as determined by `imConvincedIsPasswordChecker`) the program always behaves as a password checker. This is a contradiction.

iv.

Theorem. $L \notin \boldsymbol{RE}$.

Proof. By contradiction; assume that $L \in \mathbf{RE}$. Then there is some verifier V for L. This verifier has the property that if M is a TM that is a password checker, there is a certificate c such that V accepts $\langle M, c \rangle$, and if M is not a password checker, V will never accept $\langle M, c \rangle$ for any certificate c.

Given this, we could then construct the following TM:

M =On input w:

Have M obtain its own description, $\langle M \rangle$. For all strings c:

If V accepts $\langle M, c \rangle$, accept.

Otherwise, if V does not accept $\langle M, c \rangle$, accept if the input string is equal to p and return false otherwise.

Choose any string w and trace through the execution of the machine. If V ever accepts $\langle M, c \rangle$, we are guaranteed that M only accepts p, but in this case we find that M accepts any input, a contradiction. If V never accepts $\langle M, c \rangle$, then we are guaranteed that M is not a password checker, but in this case we find that M always behaves as a password checker, a contradiction.

In both cases we reach a contradiction, so our assumption must have been wrong. Therefore, $L \notin \mathbf{RE}$.

Problem 3: Equivalent TMs

```
bool imConvincedAreEqual(string p1, string p2, string certificate) {
  // Returns true if p1 and p2 are both turing machines that have the same language, false
}

// Creat a turing machine that does not accept any strings
int tm2(string inp) {
  return false;
}

int main() {
  string me = mySource();
  string m2 = "int tm2(string inp) {return false;}";
  string input = getInput();

  for (i = 0 to infinity) {
    for (each string c of length i) {
      if (imConvincedAreEqual(me, m2, c)) {
        return true;
      }
    }
  }
}
```

ii.

Theorem. $EQ_{TM} \notin \boldsymbol{RE}$.

Proof. By contradiction; assume that $EQ_{TM} \in \mathbf{RE}$. Then there is some verifier V for EQ_{TM}. This verifier has the property that if $M1$ and $M2$ are TMs and $L(M1) = L(M2)$, there is a certificate c such that V accepts $\langle M1, M2, c\rangle$, and if $M1$ and $M2$ are TMs and $L(M1) \neq L(M2)$, V will never accept $\langle M, M2, c\rangle$ for any certificate c.

Given this, we could then construct the following TM:

M =On input w:

Have M obtain its own description, $\langle M1\rangle$.

Let $M2$ be a TM that accepts $\varnothing$. Have M obtain its description, $\langle M2\rangle$.

For all strings c:

If V accepts $\langle M1, M2, c\rangle$, accept.

Choose any string w and trace through the execution of the machine. If V ever accepts $\langle M1, M2, c\rangle$, we are guaranteed that $M1$ and $M2$ are TMs and $L(M1) = L(M2)$, but in this case we find that $M1$ accepts any input while $M2$ accepts no inputs, a contradiction. If V never accepts $\langle M1, M2, c\rangle$, then we are guaranteed that $L(M1) \neq L(M2)$, but in this case we find that $M1$ loops on all inputs. This means it does not accept any inputs, which means that $L(M1) = L(M2)$, a contradiction.

In both cases we reach a contradiction, so our assumption must have been wrong. Therefore, $EQ_{TM} \notin \mathbf{RE}$.

Problem 4: The Big Picture

1. REG
2. ALL
3. REG
4. REG
5. REG (regular languages closed under union)
6. R (M-N so not REG, but CFG so R)
7. RE (?)
8. ALL (there is a TM where L={$\emptyset$} will loop forever, can't be RE)
9. RE (n is the certificate) (what happens when it loops? don't worry about that)

10. ALL
11. ALL
12. RE

if in RE, if I give you a machine, deep down I know this is in the machine, machine will halt. R if not in language, will halt, which is not guaranteed. Not in RE, I can give you (M,n) that's in the language and you can't conclusively tell me it's in the language. ex) at most length 5, run on all strings on at most length 5, if in language, will accept it. I'm giving you something in the language, if you halt

Problem 5: 4-Colorability

i. 3COLOR $\in$ NP because it is NP-complete and NP-complete is a subset of NP. Therefore if 3COLOR can be reduced to 4COLOR, then 4COLOR is also an element of NP. Take every 3COLOR graph, in which there exists a way to color each of the nodes one of 3 colors, such that no two nodes of the same color are connected by an edge. Each of these graphs is also by definition 4-colorable, just one of the 4 colors will go unused. Therefore, there is polynomial time reduction that shows that 4COLOR must also be in NP.

I'm am not sure about this explanation. I actually think they might be looking for:

We can guess different 4Colorings of a graph using a nondeterministic turing machine. We can then deterministically check whether the coloring option is a legal 4-coloring of G. This non-deterministic turing machine us in polynomial time, because it would only take the number of nodes, n, to check if this was a valid coloring for the graph.

ii. Take an arbitrary graph G and contrast Graph G' by adding a new node that is connected to all other nodes.

1. If the original graph G is 3-colorable, then the new graph G' must be 4-colorable because in the original graph G at most three colors were used so that no two nodes of the same color were connected by an edge. If a new node is added that connects to all other nodes, then this node must be of a new color. This new graph would require one additional color to prevent the same color from being connected by an edge, bringing the total to at most 4 colors, the definition of a 4-colorable graph.
2. If we take G' and complete the transformation that removes the node that connects to all other nodes, we have then reduced the colorable number by 1. This is because that node must have been a different color than all other nodes, by definition of colorability. Since G' was 4-colorable(had at most 4 colors needed), G is therefore 3-colorable(has at most 3 colors needed).
3. This reduction can be completed in polynomial time n+1. The reduction must add one node to graph G, then make n (the number of nodes in graph G) edges between the new node and all previous nodes in G.

Problem 6: Resolving P $\stackrel{?}{=}$ NP

1. neither
2. neither
3. P = NP
4. P = NP
5. P $\neq$ NP
6. P $\neq$ NP
7. neither
8. neither
9. neither
10. neither
11. neither
12. neither
13. P = NP
14. P = NP
15. neither
16. neither

Problem 7: The Big Picture

i. you can make a DFA

ii. prove context free? provide grammar. Prove not regular? M-H

iii. give a regular language, since regular languages $\subseteq$ P

iv. traveling salesman, 3SAT, we don't know if it's in P because we don't know if P=NP

v. A_{TM}, can't be a decider

vi. looping, can't be a verifier