Problem Set 7: CS103

Katherine Cheng, Richard Davis, Marty Keil

May 29, 2015

Problem 1

Richard Davis (rldavis) submitted this turing machine.

Problem 2

Richard Davis (rldavis) submitted these turing machines.

Problem 3

i

Proof. Let the turing machine M reject any input. It is then a decider, because M halts on all inputs by rejecting. Also for any string w, the statement if M accepts w, then x $\in \Sigma^*$ is always a true statement because M never accepts w. A truth table is always true when the antecedent of an implication is false. □

ii

Proof. Let the turing machine M accept any input. It is then a decider, because M halts on all inputs by accepting. Also for any string w, the statement if M rejects w, then x $\notin \Sigma^*$ is always a true statement because M accepts all w. A truth table is always true when the consequent of an implication is true. □

iii

Proof. Let the turing machine M infinitely loop on any input. Both Statements 2 and 3 are always true in this case because both antecedents in the implications are always false. M never accepts any string w and M

never rejects any string w. As before, the truth table is always true when the antecedent of an implication is false. □

iv If L satisfies all 3 statements then we know the language is decidable. We then also know that L ∈ R. And since R is a subset of RE, we can also state that L is recognizable.

Problem 4

i. We can create a TM M that accepts an arbitrary string w and loops infinitely on all other inputs. M is constructed such that $\mathscr{L}(M) = \{w\}$. Because M does not halt on all inputs, M is not a decider.

   We can create a second TM N that accepts the string w and rejects all other inputs. The language of N is $\mathscr{L}(N) = \{w\}$. Since N halts on all inputs, N is a decider. By definition, a language L is called decidable if there is a decider N such that $\mathscr{L}(N) = \text{L}$, so $\mathscr{L}(N)$ is decidable.

   Because we defined $\mathscr{L}(M) = \mathscr{L}(N) = \{\text{w}\}$, we know that $\mathscr{L}(M)$ is must be decidable. Thus, we have described a TM M where $\mathscr{L}(M) \in \mathbf{R}$, but M is not a decider.

ii.

   Theorem. *For every string w, there's an **R** and an **RE** language containing w.*

   Proof. Take the second TM, N, described in part i of this problem. For any arbitrary string w, N accepts w and rejects all other inputs. N halts on all inputs, so N is a decider. Let $\text{L} = \mathscr{L}(N)$. Since N is a decider, L is decidable, which means that $L \in \mathbf{R}$.

   We also know that $\mathbf{R} \subseteq \mathbf{RE}$, and by definition of a subset, all elements of **R** are also elements of **RE**. As such, we know that $L \in \mathbf{RE}$. Thus, we have shown that for every string w, there exists an **R** language containing w and an **RE** language containing w, namely L. □

Problem 5

If we wanted to build a TM password checker, entering your password would correspond to starting up the TM on some string, and gaining access would mean that the TM accepts your string. Let $p \in \Sigma^*$ be your password. A TM that would work as a valid password checker would be a TM M where $L(M) = p$; the TM accepts your string, and it doesn't accept anything else.

Given a TM, is there some way you could tell whether the TM was a valid password checker? Let $p \in \Sigma^*$ be your password and consider the following language:

$$L = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \{p\}\}.$$

Prove that L is undecidable. This means there is no algorithm that can mechanically check whether TM is suitable as a password checker.

i. Suppose there is a function `bool isPasswordChecker(string program)` that accepts as input a program and returns whether or not that program only accepts the string p. Using the programs from lecture as a template, write the pseudocode for a self-referential program that uses the `isPasswordChecker` method to obtain a contradiction.

```
isPasswordChecker(isPasswordChecker)
```

ii.

Problem 6

In class we learned about the halting problem, which tells us that it is not possible to write a decider that will determine if a program halts or loops forever. This means it is not possible for the operating system to detect when a program has gone into an infinite loop, because that would require it to have access to this impossible decider. So, the best the operating system can do is to pop up a dialog asking if it should terminate a program.

Problem 7

Consider the program:

```
int main() {
    string me = mySource();
    string input = getInput();
    if (willAccept(me, input)) {
        reject();
    } else {
        accept();
    }
}
```

Theorem. *This program must loop infinitely on all inputs.*

Proof. By contradiction. Assume for the sake of contradiction that there exists some string w that causes the program to terminate. If the program terminates, it must accept or reject w.

In the case that the program accepts the input w, it satisfies the conditional `if (willAccept (me, input))`, which causes the program to `reject()`. This is a contradiction, because the method `willAccept()` returns true when the program in fact rejects the input.

Alternatively, in the case that the program rejects the input w, it does not satisfy the conditional and is punted to the `else` case, and the program will `accept()`. This is a contradiction, because the method `willAccept()` returns false, whereas the program accepts the input.

Both cases lead to a contradiction, so our assumption must be false. There must not exist a string w that causes the program to terminate. Thus, this means that the program loops infinitely on all inputs. □

Problem 8

i

```
function bool inL1uL2(string w) {
     inL1(w) OR inL2(w);
}
```

From this new method we can see that L1 ∪ L2 is also decidable. When either L1(w) or L2(w) is accepted, inL1uL2 returns true. If both L1(w) and L2(w) are rejected then inL1uL2 returns false. This covers all possibilities and always returns either an accepting or rejecting state, so the method is therefore decidable.

ii

```
function bool Concat (w) {
     for all L1 {
          for all L2 {
              for ( i = 0; i++; i < w.length) {
                  if (L1(w.split AHHH!!  )  {
                        return true;     }
```

iii

```
function bool SymDiff (w) {
    for all L1, L2 {
          if (inL1(w) OR inL2(w)) {
               if( !( inL1(w) AND inL2(w)) {
                    return true;
               }
          else return false;
          }
     else return false;
```

```
}
```

This method returns true whenever inL1 or inL2 is accepts, but not when inL1 and inL2 both accept. In this case and also when both reject, the method rejects. This covers all possibilities and always returns either an accepting or rejecting state, so the method is therefore decidable.