Ethan Wang
11/23/21
305319001

CS181 HW5

pg 155 1. P: Turing Machine M: Turing Machine

$F: \{0,1\}^* \rightarrow \{0,1\}$

$$F(P,M) = \begin{cases} 1 & x \text{ such that } P \text{ halts on } x, M \text{ does not halt on } x \\ 0 & \text{else} \end{cases}$$

At a high level, this is uncomputable because the function F is checking if P and/or M halt on input x. This is in fact a variation of the halting problem, which we already know is uncomputable.

To prove this, we can reduce from HALTONZERO, which we showed was uncomputable in class

Say we have the function N(x) as follows

```
def N(x):
    if (x == 0): while (1) {}   # loop infinitely
    Return 1
```

This program halts on all inputs that are not 0.

Now say we have an input "M" to HALTONZERO.

R(M) = (M, N), and claim HALTONZERO(M) = F(R(M)).

Case 1: HALTONZERO(M) = 1

- M halts on 0
- N does not halt on 0
- F(R(M)) = F(M, N) = 1

Case 2: HALTONZERO(M) = 0

- M does not halt on 0, or N halts
- F(R(M)) = F(M, N) = 0

This shows a reduction to F using HALTONZERO.

F is therefore uncomputable, because we know that HALTONZERO is uncomputable.

NOT(A) and (B)

0 11

1 00

2. EMPTY : $\{0,1\}^* \rightarrow \{0,1\}$ - returns 1 if DFA accepts nothing

EQUIVALENT : checks if DFAs D and D' are the same

Say we are given two DFAs D and D'

$f(x) = [\underset{①}{\text{NOT}}(D(x)) \underset{②}{\text{AND}} D'(x)] \underset{③}{\text{OR}} [\underset{①}{\text{NOT}}(D'(x)) \underset{②}{\text{AND}} D(x)]$

This function f(x) has a DFA that exists to solve it.

We know this because f(x) is regular, and:

① DFA's are closed under NOT. This is regular

② DFA's are closed under AND. This is regular

③ DFA's are closed under OR. This is regular.

This DFA (called Z) can actually compute f(x).

With this, we can say $R(D, D') = Z$

Also, there does exist some x such that $Z(x) = 1$ i.f.f there exists an x such that $D(x) \neq D'(x)$. This means that EQUIVALENT(D, D') = EMPTY(Z)

Note: If you think about it, EQUIVALENT can be reduced to EMPTY if you just do EQUIVALENT(D, E) where D is a DFA and E is just an empty language.

m = m+n
n = m

3. $L = \{0^m 1 0^n 1 0^{|m-n|} : m, n \geq 0\}$

$0^m 1 0^n 0^{|m-n|} \rightarrow 0^{m+n} | 0^m | 0^n$

Let G be L's corresponding CFG and $G = (V, \Sigma, R, S)$

$V = \{A, B\}$

$\Sigma = \{0, 1\}$

R: $A \rightarrow 1 | B | 0A$

$B \rightarrow 1 | 0B | B0$

$S \rightarrow A$

We can take the conversion in the first line to create a set of rules. If our A leads to a 1, then we want it to use B. If it leads to 0, then we have it reuse A. Everything eventually leads to B.

4. $L = \{x \in \{0,1\}^* : x \text{ has equal number of 1's and 0's}\}$

The idea here is to have the same number of 0's and 1's. So we can expand S to include a 01 pair in all possible orientations before and after the symbol S. S should also map to itself to aid in repetition. Therefore, we have:

$$S \rightarrow SS \mid S01 \mid S10 \mid 01S \mid 10S \mid \varepsilon$$

This can be simplified down to:

$$S \rightarrow SS \mid 0S1 \mid 1S0 \mid \varepsilon$$

This way no matter what gets added, there will be an equal number of 0's and 1's.