Section 6 Solutions

Problem 1: Consider the function $f : \{0,1\}^* \to \{0,1\}$ such that $f(x) = 1$ if and only if $|x| = n$ is even and $x_0 = x_{n/2}$. In other words, the first bit of x is equal to the first bit of the second half of x. Construct a Turing Machine that computes f.

Solution 1:

Consider the Turing Machine that first tests the input length through states $EVEN$ and ODD, then if the string length is even goes back to the beginning through state $GO\ BACK$, then uses the Turing Machine defined in class which zags back and forth to determine the middle of the string (states $3, 4, 5, 6$), then goes back to the beginning to test the first bit of the string, then tests to see if the first bit is equal to the other bits. For example, $GO\ BACK_0$ is the state that you enter once the middle bit is determined to be 0, and then we'll enter state $TEST_0$ which outputs 0 and halts if the input bit is 1 and 0 otherwise.

States/Inputs	▷	**0**	**1**	⌀	**#_0**	**#_1**
0 (EVEN)	invalid	(1, 0, R)	(1, 1, R)	(2, ⌀, L)	invalid	invalid
1 (ODD)	invalid	(0, 0, R)	(0, 1, R)	(11, ⌀, L)	invalid	invalid
2 (GO BACK)	(3, ▷, R)	(2, 0, L)	(2, 1, L)	invalid	invalid	invalid
3 (START)	invalid	(4, #_0, R)	(4, #_1, R)	(-, ⌀, H)	(7, 0, L)	(8, 1, L)
4 (ZAG FORWARDS)	invalid	(4, 0, R)	(4, 1, R)	(5, ⌀, L)	(5, 0, L)	(5, 1, L)
5 (REPLACE END)	invalid	(6, #_0, L)	(6, #_1, L)	invalid	invalid	invalid
6 (ZAG BACKWARDS)	invalid	(6, 0, L)	(6, 1, L)	invalid	(3, 0, R)	(3, 1, R)
7 (GO BACK 0)	(9, ▷, R)	(7, 0, L)	(7, 1, L)	invalid	(7, 0, L)	(7, 1, L)
8 (GO BACK 1)	(10, ▷, R)	(8, 0, L)	(8, 1, L)	invalid	(8, 0, L)	(8, 1, L)
9 (Test 0)	invalid	(-, 1, H)	(-, 0, H)	invalid	(-, 1, H)	(-, 0, H)
10 (Test 1)	invalid	(-, 0, H)	(-, 1, H)	invalid	(-, 0, H)	(-, 1, H)
11 (CLEAR_0)	(12, ▷, R)	(11, 0, L)	(11, 1, L)	(11, ⌀, L)	(11, #_0, L)	(11, #_1, L)
12 (OUTPUT_0)	invalid	(-, 0, H)	(-, 0, H)	(-, 0, H)	(-, 0, H)	(-, 0, H)

Problem 2: Suppose that $F : \{0,1\}^* \to \{0,1\}$ is a computable function. For each one of the definitions below of a function G mapping $\{0,1\}^*$ to $\{0,1\}$, prove that G is computable.

Note: When you need to show the existence of a Turing Machine (or a NAND-TM program) to establish that a function is computable, you don't have to fully specify the transition table, etc. You can use the equivalences we proved with other models and our "eat your cake and have it too" paradigm to describe the Turing Machine at a high level, as long as your provide enough details so the operation of the algorithm is clear. However, it is crucial that you *prove* that the machine you described actually computes the desired function.

Problem 2.1: For every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, $G(x_0 \cdots x_{n-1}) = F(x_{n-1} \cdots x_0)$.

Solution 2.1:

We will use the "eat your cake and have it too" paradigm. If G is computable in a language like Python, it is computable with a TM. Let PF be the program that computes F. We will construct a program PG that computes G.

```
PG(x):
    x_rev = ""
    for i in range(len(x)-1,-1,-1): # Syntax for iterate backwards
        x_rev.append(x[i])
    return PF(x_rev)
```

The first three line of the code reverses the input, so $x_{rev} = x_{n-1} \cdots x_0$. Since PF computes F, this program computes $F(x_{rev})$, which is equal to $G(x)$.

Problem 2.2: For every $x \in \{0,1\}^*$, $G(x) = 1$ iff there exist a list $u_0, \ldots, u_{t-1}$ of non empty strings such that $F(u_i) = 1$ for every $i \in [t]$ and $x = u_0 u_1 \cdots u_{t-1}$.

Solution 2.2:

Again, use the cake paradigm. Let PF be the program that computes F. We now construct a recursive program that computes G.

```
PG(x):
    if x == "":
        return 0
    if F(x) == 1:
        return 1
    for k in range(0,len(x)):
        if PF(x[0:k]) == 1 and PG(x[k+1:len(x)-1]) == 1:
            return 1
    return 0
```

We argue by induction on the length of x that PG computes G. Base case: $|x| = 0$, then $G(x) = 0$ and our program returns 0.

Inductive step: Suppose that PG computes G for all inputs of length at most m. Let x be an input where $|x| = m + 1$.

If $G(x) = 1$, there exist a a partition $x = u_0 u_1 \cdots u_{t-1}$ where $F(u_i) = 1$ for all i. If $u_0 = x$, then we get $F(x) = 1$ and the program returns 1. Otherwise, notice that $x' = u_1 \cdots u_{t-1}$ has length at most m and satisfies $G(x) = 1$. We will encounter the partition u_0, x in the loop and return 1.

If $G(x) = 0$, then $F(x) = 0$ and for any partition $x = u_0 x'$, either $F(u_0) = 0$ or $G(x') = 0$. Therefore, PG never returns 1 in the loop so it returns 0, as desired.