CS 121 Section 7: Solutions

Problem 1: Let $TIMEDHALT : \{0,1\}^* \to \{0,1\}$ be the function that on input (a string representing) a triple (M, x, t), $TIMEDHALT(M, x, t) = 1$ iff the Turing machine M, on input x, halts within at most t steps (where a step is defined as one sequence of reading a symbol from the tape, updating the state, and writing a new symbol and (potentially) moving the head.)

Prove that $TIMEDHALT$ is computable.

Solution 1: To show that this function is computable, we'll write a program to compute it. By the "have your cake and eat it too" principle, we can use Python-esque pseudocode to do so.

Think of the *simulateOneStep* program as similar to what's described in 7.2 of Barak's book. It takes the description of a Turing machine, the current tape, the current state, and the current index within the tape, and outputs the new state, new index in the tape, updated tape, and last move made (L, R, H, S) based on the transition function of M.

```
def computesTIMEDHALT(M, x, t):
    i = 0
    state = 0
    tape_index = 0
    tape = x
    while (i < t):
        state, tape_index, tape, last_move = simulateOneStep(M, tape, state, tape_index)
        if (last_move is HALT):
            return 1
        i = i + 1
    return 0
```

Correctness: we claim that a correct Python-esque program *simulateOneStep* exists due to the equivalence of Turing machines and such programming languages. Our program therefore simulates one step of Turing machine M on input x, t times. If M has halted after any one of these t steps, we return 1, as $TIMEDHALT$ is supposed to do. If we have not halted after t steps we return 0.

Problem 2: Let $IS\text{-}TM\text{-}ONE$: $\{0,1\}^* \to \{0,1\}$ be the function that takes as input a string representation of a Turing machine M and outputs 1 iff $M(x) = 1$ for every $x \in \{0,1\}^*$. Prove or disprove: $IS\text{-}TM\text{-}ONE$ is computable.

Solution 2: This function is uncomputable.

Assume toward a contradiction that $IS\text{-}TM\text{-}ONE$ is computable.

We will now reduce $HALTONZERO$ to $IS\text{-}TM\text{-}ONE$.

Reduction: Given a TM M, take the TM N that runs M on 0 and then outputs 1 and halts. $HALTONZERO(M) = IS\text{-}TM\text{-}ONE(N)$.

Can also think of the reduction as follows:

```
def computesHOZ(M):
    N = description of TM that runs M(0) and then outputs 1 and halts
    return computesISTMONE(N)
```

Proof of correctness for our reduction requires two cases:

1. $HALTONZERO(M) = 1$. Since M halts on 0, N will always halt and output 1. Thus $IS\text{-}TM\text{-}ONE(N) = 1$.

2. $HALTONZERO(M) = 0$. Since M doesn't halt on zero, N will not halt. Since N does not halt, it does not output 1, so $IS\text{-}TM\text{-}ONE(N) = 0$.

We've therefore correctly reduced $HALTONZERO$ to $IS\text{-}TM\text{-}ONE$. By our original assumption, $IS\text{-}TM\text{-}ONE$ is computable, so $HALTONZERO$ must be too. This is a contradiction, since we proved in class and lecture that $HALTONZERO$ is uncomputable. So our assumption is wrong and $IS\text{-}TM\text{-}ONE$ is uncomputable.

Problem 3: Prove that the following function is uncomputable:

$$COMPUTES - PARITY(P) = \begin{cases} 1 & \text{P computes the parity function} \\ 0 & \text{otherwise} \end{cases}$$

Solution 3:

Assume toward a contradiction that $COMPUTES\text{-}PARITY$ is computable.

We will now reduce $HALTONZERO$ to $COMPUTES\text{-}PARITY$.

Reduction: Given a TM M, construct TM N that runs M on 0, then counts the number of 1s in the input, and returns 1 if that number of 1s is odd. If the number of 1s is even, N returns 0. $HALTONZERO(M) = COMPUTES\text{-}PARITY(N)$.

Proof of correctness for our reduction requires two cases:

1. $HALTONZERO(M) = 1$. Since M halts on 0, N will also halt; it will return 1 if there are an odd number of 1s in the input and 0 otherwise, by construction. This is exactly what computing parity is (returns 1 iff there's an odd number of 1s in input.) Thus $COMPUTES\text{-}PARITY(N) = 1$

2. $HALTONZERO(M) = 0$. Since M doesn't halt on zero, N will not halt (since it tries to run M on 0). Since N does not halt, it does not ever output anything, so it definitely does not compute the parity function. So $COMPUTES\text{-}PARITY(N) = 1$.

We've therefore correctly reduced $HALTONZERO$ to $COMPUTES\text{-}PARITY(N)$. By our original assumption, $COMPUTES\text{-}PARITY(N)$ is computable, so $HALTONZERO$ must be too. This is a contradiction, since we proved in class and lecture that $HALTONZERO$ is uncomputable. So our assumption is wrong and $COMPUTES\text{-}PARITY(N)$ is uncomputable.