Introduction to Algorithms **Solution Set 10**
CS 4820, Spring 2018

(1) *(10 points) Recall that an instance of the halting problem is a string of the form $x;y$ and the goal is to decide if the Turing machine M_x encoded by the string x halts on input y. If M_x halts on y, then $x;y$ is a* YES *instance. Otherwise, $x;y$ is a* NO *instance. Let us say that a Turing machine M* fails to solve *the halting problem for instance $x;y$ if it never terminates or if it produces the wrong answer, i.e., it rejects in case that $x;y$ is a* YES *instance or it accepts in case that $x;y$ is a* NO *instance.*

Because the halting problem is undecidable, we know that for every Turing machine M there exists an instance $x;y$ of the halting problem such that M fails to solve $x;y$. Describe an algorithm to find such an instance. The input to your algorithm is a description of a Turing machine M. For every such input, your algorithm should run for a finite number of steps and output a halting problem instance $x;y$ such that M fails to solve $x;y$.

Solution. Given the descripton of M, modify it to the description of a machine M' that operates as follows: on any input x, it writes $x;x$ on the tape and then simulates M on input $x;x$, modifying the halting behavior as follows. When M is about to enter the "yes" state, M' instead enters a special "looping" state in which it runs forever (i.e., the looping state transitions to itself without moving left or right, no matter what symbol it is reading). When M is about to enter the "no" or "halt" state, M' enters the "yes" state. Let x' denote the description of M'. An instance of the halting problem that M fails to solve is $x';x'$.

To prove correctness, we must show that M fails to solve the halting problem instance $x';x'$. If M never terminates on input $x';x'$, then by definition it fails to solve $x';x'$. There are two more cases to consider.

1. Suppose M accepts $x';x'$. Then by the definition of M', when presented with input x', it transforms it to $x';x'$, runs M on this input, and enters the looping state when M is about to output "yes". Therefore, M' doesn't halt on input x'. Since x' is the description of M', this means that $x';x'$ is a NO instance of the halting problem, so M should not accept it.

2. Finally, suppose M rejects $x';x'$. Then by the definition of M', when presented with input x', it transforms it to $x';x'$, runs M on this input, and enters the "yes" state when M is about to output "no" or "halt". Therefore, M' accepts x'. Since x' is the description of M', this means that $x';x'$ is a YES instance of the halting problem, so M should accept it.

(2 *(10 points) Call a Turing machine M* termination-safe *if $M(y) \neq \nearrow$ for all $y \in \Sigma^*$. In other words, a termination-safe machine is one which is guaranteed to terminate on every input string.*

Let $T \subset \Sigma^*$ denote the set of all strings x such that x is the description of a termination-safe Turing machine. Prove that T is not recursively enumerable.

Solution. To prove that T is not recursively enumerable we reduce the complement of the halting problem, $\bar{H}$, to T. In other words, given a hypothetical Turing machine M_T such that M_T accepts x if and only if $x \in T$, we show how to design a Turing machine $M_{\bar{H}}$ that accepts $x;y$ if and only if $x;y \in \bar{H}$, which contradicts the fact that $\bar{H}$ is not r.e. and thereby disproves the existence of M_T.

For any string w, let $M'(w)$ denote a Turing machine that does two things in parallel: it counts the characters in its input string while simultaneously running a universal Turing machine simulation on input w. If the number of characters in the input string exceeds the number of steps in the universal

Turing machine simulation, then $M'(w)$ enters an infinite loop and never terminates. Otherwise, it terminates. Pseudocode for $M'(w)$ is as follows; in the pseudocode we implement the character-counting process by using a special character '#' to separate the portion of the tape containing the input string from the portion on which we are running the universal Turing machine simulation. As long as the simulation is running, we overwrite the rightmost remaining character of the input string with '#' until every character has been overwritten.

```
Write a separator character (say, '#') to the right of the input string.
Write the string w to the right of the separator character.
while the leftmost character on the tape is not '#' do
    Find the leftmost '#' on the tape.
    Move one step to the left and write the symbol '#'.
    Move to the rightmost '#' on the tape.
    Run one round of universal Turing machine simulation on the string to the right of this '#'.
    if universal Turing machine halts then
9:      Enter an infinite loop.
    end if
end while
Transition to the "halt" state.
```

Having defined this family of Turing machines $M'(w)$, we can now specify the machine $M_{\bar{H}}$. On input $x; y$, it constructs the description of the Turing machine $M'(x; y)$, passes this description as an input to M_T, and does whatever M_T does thereafter (i.e., running forever or halting in one of the three termination states).

By design, the machine $M_{\bar{H}}$ accepts a string $x; y$ if and only if the machine $M'(x; y)$ is termination-safe. To prove the correctness of the reduction we must prove two things: first, if $x; y \in \bar{H}$ then $M'(x; y)$ is termination-safe, and second, if $x; y \in H$ then $M'(x; y)$ is not termination-safe.

If $x; y \in \bar{H}$ it means that a universal Turing machine simulation on input $x; y$ will run forever. This means that no matter what input string is given to $M'(x; y)$, it will eventually overwrite every character of its input string with '#', exit the while loop, and halt. Hence, $M'(x; y)$ is termination-safe. Conversely, if $x; y \in H$ then a universal Turing machine simulation on input $x; y$ will run for some finite number of steps, $t(x; y)$. If $M'(x; y)$ is given an input string whose length is strictly greater than $t(x; y)$, then it will enter an infinite loop. Hence, $M'(x; y)$ is not termination-safe.

Discussion. This was the toughest question on the problem set. To solve it, one must first understand what it's asking. The problem can be rephrased — informally, but fairly accurately — as follows: *how could we use the fact that one Turing machine is guaranteed to terminate on all inputs as evidence that another Turing machine does not terminate on one specific input?* It's counterintuitive that one machine running forever on particular input could be linked with a different machine being guaranteed to terminate on all inputs, but the reduction presented above shows how to link these two notions: by using the non-terminating machine as a "timer" whose non-termination furnishes the conditions that allow the other machine to finish arbitrary inputs, no matter what size they are.

(3) *(10 points) As in question (2), let $T \subset \Sigma^*$ denote the set of all strings x such that x is the description of a termination-safe Turing machine. Prove that $\overline{T}$, the complement of T in Σ^*, is not recursively enumerable.*

Solution. Again we reduce from the co-halting problem, $\bar{H}$. Given an instance $x; y$ let $M''(x; y)$ be the machine that ignores its input, overwrites it with the string $x; y$, and simulates a universal Turing machine running on $x; y$. Given a Turing machine $M_{\overline{T}}$ that accepts strings if and only if they belong to $\overline{T}$, we use $M_{\overline{T}}$ to design a machine $M_{\bar{H}}$ that accepts strings if and only if they belong to $\bar{H}$. The machine $M_{\bar{H}}$ takes an input $x; y$, writes the description of the Turing machine $M''(x; y)$, passes this description as an input to $M_{\overline{T}}$, and does whatever $M_{\overline{T}}$ does thereafter (i.e., running forever or halting in one of the three termination states).

To prove correctness, we make the following two observations. First, if $x; y \in \bar{H}$ then the universal Turing machine doesn't halt when processing $x; y$, so $M''(x; y)$ never halts, no matter input it is given. In particular, $M''(x; y)$ is not termination-safe, so its description belongs to $\overline{T}$. Second, if $x; y \in H$ then the universal Turing machine halts when processing $x; y$, so $M''(x; y)$ always halts, no matter input it is given. In particular, $M''(x; y)$ is termination-safe, so its description does not belong to $\overline{T}$. We have shown that $x; y \in \bar{H}$ if and only if the description of $M''(x; y) \in \overline{T}$, hence the machine $M_{\bar{H}}$ described above has $L(M_{\bar{H}}) = \bar{H}$ as claimed.