CMSC 28100 Introduction to Complexity Theory
Problem Set 3

Bayard Walsh

January 2024

1

L is decidable.

Algorithm:

$f(n)$:

NOTE: If infinite twin primes skip next line. If finite, let T_{max} be the largest number where T_{max} is prime and $T_{max} + 2$ is prime.

if $n > T_{max}$
    return **FALSE**

for i in $(n, n+1, n+2, \cdots)$
    for k in $(2, 3, \cdots, i-1)$
        if $i \,\%\, k = 0$ or $(i+2) \,\%\, k = 0$:
            BREAK, go to next i in outer for loop
return **TRUE**

Proof:

Assume there are infinite twin primes.

if i and $i+2$ both don't have reminder 0 when divided by every $(2, 3, \cdots, i-1)$, then they are prime by definition, because $i, i+2$ are not a product of two smaller natural numbers greater than 1. Therefore we have found a set of twin primes, so we correctly return **TRUE**. Therefore if the algorithm halts it will be true. Next, we know that this algorithm will halt because there are infinite twin primes by assumption, so therefore there must be some set of twin primes, say $i', i'+2$, where that $n \leq i'$. Based on the linear search in the algorithm, we will find these primes eventually, meaning our algorithm will halt with the

correct answer.

Next, assume there are a finite set of twin primes.

Therefore, there must be some largest number in the set of twin primes, say T_{max}. Therefore our algorithm checks if $n > T_{max}$, and if so it returns **FALSE**, because it is impossible to find a twin primes $p, p+2$ where $n < p$ if $n > T_{max}$. If not, we continue with our linear search algorithm mentioned before, and if $n < T_{max}$ our algorithm will eventually find some $i', i'+2$ and return **TRUE**, which will also be correct. Therefore in both cases our algorithm will eventually return **TRUE** or **FALSE**, meaning the algorithm will not loop and will eventually return the right answer.

As the set of twin primes is either finite or infinite, and our algorithm correctly decides **TRUE** or **FALSE** in both cases, we have that L is decidable.

2

L is undecidable. Assume for contradiction that some Turing machine, say M_L solves L, and show that by reduction it will solve $\overline{HALT}$ problem, which is undecidable, causing a contradiction.

Reduction: $f(\langle M, w \rangle))$

let M_1 accept all inputs
let M_2 ignore its input and instead simulate M on w:
    if M_2 halts on w:
        accept w regardless of what it is
        RETURN $L(M_1) \cap L(M_2)$
    if M_2 doesn't halt on w:
        reject w regardless of what it is
        RETURN $L(M_1) \cap L(M_2)$

Run M_L on $\langle M_1, M_2 \rangle$, if accept then accept, if reject then reject

Proof:

f is computable

M_1 Turing Machine is trivially computable as it ignores all inputs. For M_2 we simulate on M which is also computable as M is given, and any arbitrary Turing Machine must be computable. Accepting or rejecting any input is also trivially computable, and calculating the intersection of two languages is also computable. Therefore f is computable.

YES maps to YES

We have that $L(M_1)$ is infinite as it accepts all inputs, and if M_2 doesn't halt on w, then we reject all w, meaning that $L(M_2)$ is finite, as $L(M_2)$ is $\emptyset$. Therefore $L(M_1) \cap L(M_2)$ is finite if M_2 doesn't halt on w. As M_2 simulates M on w and checks for halting, we have that not halting for the $\overline{HALT}$ problem is equivalent to determining a finite set for $L(M_1) \cap L(M_2)$, so we have YES maps to YES for the reduction.

NO maps to NO

If M_2 halts on w, then we accept all w, meaning that $L(M_2)$ is infinite and accepts all Σ^*. As $L(M_1)$ is also infinite, then $L(M_1) \cap L(M_2)$ is also infinite if M_2 halts on w. As M_2 simulates M on w and checks for halting, we have that not halting for $\overline{HALT}$ is equivalent to determining an infinite set for $L(M_1) \cap L(M_2)$, so we have NO maps to NO for the reduction.

As we have a reduction from the $\overline{HALT}$ problem that is computable, YES maps to YES, and NO maps to NO, we have that the L is undecidable by contradiction.

3

L is undecidable. Assume for contradiction that some Turing Machine M_L decides L for some arbitrary W. A reduction from the Halting problem will show this as a contradiction.

Reduction: $f(\langle M, W \rangle))$

$M' = M$ however use α instead of $\sqcup$ when M replaces a non-blank symbol with the blank symbol $\sqcup$. Treat α as $\sqcup$ otherwise in M'.

Let M' simulate on W:

if M' halt on W:
step to the end of the string at the output of M' on W
replace first $\sqcup$ on end string with α.
replace α with $\sqcup$.
RETURN

if M' doesn't halt on W:
RETURN

Run $M_L\langle M', W\rangle$ if accept then accept, if reject then reject

Proof:

f is computable

We have M' which is M using α instead of $\sqcup$ in M writing behavior. Let α be any symbol such that $\alpha \notin M$ alphabet, and we have M' have the same alphabet as M but add α. For M', we will treat α as $\sqcup$ in every way that M would use $\sqcup$. M' modification from a valid Turing Machine is simple and computable. The other parts of f involve stepping to the end of the string, replacing a $\sqcup$ on blank string with α, and replacing α with $\sqcup$, which are all also computable. Therefore f is computable.

YES maps to YES

If M' halt on W, then we step to the end of the string at the output of M' on W, replace the first $\sqcup$ on the end string with α, and then replace α with $\sqcup$. This sequence leaves the output of M' the same as the output of M, however it means that if M' halts on W then f will always replace a non-blank symbol with the blank symbol before terminating. Therefore YES maps to YES for this case.

NO maps to NO

We create M' to handle the case where M could not halt and also replace a non-blank symbol with the blank symbol at some point. Note that M' never replaces a non-blank symbol with the blank symbol, because by nature of its definition, it will use α whenever it would write $\sqcup$ in M, so while computing M', a blank symbol is never placed. Therefore while we simulate M' on W, a non-blank symbol is never replaced with the blank symbol if and only if M' doesn't halt. If M' doesn't halt on W, then f never replaces a non-blank symbol with the blank symbol, meaning we have NO maps to NO for our reduction.

As we have a reduction from the halting problem that is Computable, YES maps to YES, and NO maps to NO, we have that the L is undecidable.

4

L is undecidable. For contradiction, we will assume that L is decidable by some Turing Machine M_L. Let us have M be some arbitrary Turing Machine and w be some arbitrary input. We have the following reduction from the halting problem, which is undecidable and proves that L is undecidable.

Reduction:
Given $\langle M, w\rangle$, run $f(\langle M, w\rangle)$, defined as:

Let $\Lambda = \{\$, \#, \sqcup, q_0, q_1 \cdots q_k, q_{accept}, q_{reject}\} \cup \{L(M)\}$ *where* $q_0, q_1 \cdots q_k$ *are all the states in* M *and* $L(M)$ *is the alphabet of* M *with respect to* w.

Let R be a transition set that simulates the computation configurations of M on w.
NOTE: *R is explained more below*

Let $u = \$q0w\#$ and $v = \epsilon$

Run $M_L\langle(L = \Lambda, R, u, v)\rangle$ if accept then accept, if reject then reject

Proof:

f is computable

Transition function R:
For each potential configuration, $c_0, c_1, \cdots c_n$ in the Turing Machine M, create a tuple $\in R$ where the first part of the tuple represents the current state and the current tape at an arbitrary configuration, and the second part of the tuple represents the next state and next tape symbol according to the M transition function for the following configuration. Therefore for every $c_i \in c_0, c_1, \cdots c_n$, we have (c_i, c_{i+1}) as a tuple in R, stored long form as $(\$qw\#, \$q'w'\#)$, where q, w is the state and tape position at c_i and q', w' is the state and tape position at c_{i+1}, and $\$, \#$ function as start and end of tape indicators. Additionally, we handle ϵ by adding the special translations $(\$q_{accept}w'\#, \epsilon)$ and $(\$q_{reject}w''\#, \epsilon)$ so that all configurations that are at the accept or reject step in M step to ϵ, and no other search and replace translations step to ϵ. Therefore, if a w is decided by M, our R will be able to reach accept or reject at some configuration and take one more step at that configuration to ϵ. We can determine these transitions from the Turing Machine M, so we can compute R.

The other aspects of the reduction are easy to compute, as we set u, v manually to start string q_0 and ϵ. There are also finitely many states, so our language $\{q_0, q_1, \cdots q_k\}$ is also computable, and there is a finite size to the $L(M)$ alphabet so that is also computable. Therefore f is computable.

YES maps to YES

If M halts on w in the halting problem, then we know M either accepts or rejects w, and therefore we know that $\Lambda, R, u, v \in L$. This is because we have set up our R to simulate the transition function of M, and u, v set as the

start (q_0) and end (ϵ after one step from either q_{accept} or q_{reject}) states of the Turing Machine, meaning that there must be some sequence of configurations $c_0, c_1, \cdots, c_k$ such that $\$q0w\# \Rightarrow \epsilon$ if M halts on w. Note that we can consider this halting behavior without formally simulating M on w because of the translation of M transition function to R. Therefore halting implies $\Lambda, R, u, v \in L$, and YES maps to YES for this case.

NO maps to NO

If M doesn't halt on w, then $\Lambda, R, u, v \notin L$. This is because $u \Rightarrow v$ is only true if at some point during the Turing Machine M simulation on w starting at q_0, M steps to either q_{accept} or q_{reject}, meaning that in our search and replace system we would eventually step from $\$q0w\#$ to ϵ in R. As M doesn't halt on w it will never step to q_{accept} or q_{reject}, meaning that it will never step to the only states that step to ϵ, and therefore never step to ϵ in R. Again, note that we can consider M halting on w without formally simulating M on w because of the translation of the M transition function to R. Therefore if M doesn't halt on w, $\Lambda, R, u, v \notin L$, meaning we have NO maps to NO for our reduction.

As we have a reduction from the halting problem that is Computable, YES maps to YES, and NO maps to NO, we have that the L is undecidable.