CMSC 28100-1 / MATH 28100-1
Introduction to Complexity Theory
Fall 2017 – Homework 5
Solution

November 1, 2017

Definition. *A Turing machine* transducer *is a Turing machine that outputs strings, rather than accepting or rejecting. That is, if M is a Turing machine, then M computes the function which takes an input x to the value that is left on the tape when M halts. If M does not halt on input x, then the function computed by M is undefined on x. A function f that is computed by a Turing machine transducer is called* partial computable *(partial, since M may not halt on some inputs, and then f is undefined on those inputs).*

Exercise 1. In class and in the book, you were given the following definition of a recursively enumerable language.

A language L is recursively enumerable if there is a Turing machine M such that M accepts exactly those strings in L. (On strings not in L, the machine M may either reject or not halt.)

Show that each of the following conditions is equivalent to L being recursively enumerable (in the above sense).

(a) There is a partial computable function f such that L is the range of f, that is, we have $L = \{f(x) : x \in \Sigma^*\}$. Such an f is called an *enumerator* of L since when f is run on all strings in parallel, it enumerates/lists the elements of L (possibly with repetitions).

(b) There is a partial computable function f such that L is the domain of f, that is, we have $L = \{x : f(x) \text{ is defined}\}$.

(c) There is a Turing machine M such that $L = \{x : M \text{ halts on input } x\}$.

Solution. Let us first prove that if L is recursively enumerable, then (a) holds.

Indeed, if M is a Turing machine with $L(M) = L$, then an algorithm for the enumerator f is the following.

Algorithm 1.1: Enumerator of $L(M)$.

```
1 Given an input $x \in \Sigma^*$, run $M$ on $x$.
2 if $M$ accepts $x$ then Stop with output $x$.
3 else Loop forever.
```

Clearly the algorithm above computes the partial function defined by

$$f(x) = \begin{cases} x, & \text{if } x \in L(M); \\ \infty, & \text{if } x \notin L(M); \end{cases}$$

where $f(x) = \infty$ denotes that $f(x)$ is undefined. Hence the range of f is $L(M)$.

Let us now prove that (a) implies (b).

Let E be a Turing machine that computes a partial function f whose range is L and consider the following algorithm.

Algorithm 1.2: Function whose domain is L.

```
1 Given an input $x \in \Sigma^*$, do the following.
2 for $n \leftarrow 0, 1, \ldots$ do
3     Let $S_n$ be the set of all words of length at most $n$.
4     for $y \in S_n$ do
5         Run $E$ on input $y$ for $n$ steps.
6         if $E$ stops with output $x$ then
7             Stop with output $\epsilon$.
```

Clearly the algorithm above computes the partial function defined by

$$g(x) = \begin{cases} \epsilon, & \text{if } x \text{ is in the range of } f; \\ \infty, & \text{if } x \text{ is not in the range of } f. \end{cases}$$

Hence the domain of g is equal to the range of f.

Let us now prove that (b) implies (c).

Let D be a Turing machine that computes a partial function g whose domain is L. Then D halts if and only if its input is in L.

Finally, let us prove that (c) implies that L is recursively enumerable. Let H be a Turing machine that halts if and only if its input is in L.

Then the algorithm that runs H in its input and accepts if H halts (H can either accept or reject) corresponds to a Turing machine M such that $L(M) = L$. $\lhd$

Exercise 2. In this exercise, we will show the recursive analogue of item (a) of Exercise 1.

First, fix a computable enumeration $(x_n)_{n\in\mathbb{N}}$ of Σ^* without repetitions, that is, the following hold.

- For every $n \neq m$, we have $x_n \neq x_m$.
- We have $\Sigma^* = \{x_n : n \in \mathbb{N}\}$.
- There is a Turing machine E that halts on every input and if the input is a binary representation of $n \in \mathbb{N}$, the tape contains x_n when E halts, i.e., the machine E computes the function $\langle n \rangle \mapsto x_n$.

This induces an order of the words in Σ^* given by

$$x_n \leqslant x_m \iff n \leqslant m.$$

Show that a language L is recursive if and only if there is an enumerator f for L such that f is *monotone*, that is, for every $x, y \in \Sigma^*$ with $x \leqslant y$, we have $f(x) \leqslant f(y)$, where when $f(x)$ is undefined, we interpret as $f(x) = +\infty$.

Hint: it is useful to separate the case when L is finite.

Solution. Suppose L is recursive, that is, suppose that there exists a Turing machine M that decides L. Consider the following algorithm.

Algorithm 2.1: Monotone enumerator of $L(M)$.

```
1 Given an input $x \in \Sigma^*$, run $E$ on $n = 0, 1, \ldots$ until $E(n) = x$ (that is, we have $x_n = x$).
2 $k \leftarrow 0$.
3 for $i \leftarrow 0, 1, \ldots$ do
4     Let $x = E(i)$ (i.e., we have $x_i = x$).
5     Run $M$ on $x$.
6     if $M$ accepts $x$ then
7         if $k = n$ then Stop with output $x$.
8         else $k \leftarrow k + 1$
```

Clearly the algorithm above only enumerates words in $L(M)$.

Let

$$I = \begin{cases} \mathbb{N}, & \text{if } L(M) \text{ is infinite;} \\ \{0, 1, \ldots, |L(M)| - 1\}, & \text{if } L(M) \text{ is finite;} \end{cases}$$

and let $(n_i)_{i \in I}$ be such that $n_i < n_j$ if $i < j$ and $L(M) = \{x_{n_i} : i \in I\}$. Note that if f is the partial function of the algorithm above, then we have

$$f(x_i) = x_{n_i}$$

for every $i \in I$. This implies that f is a monotone enumerator for $L(M)$.

Suppose now that f is a monotone enumerator for L and let M be a Turing machine associated with f. If L is finite, then L is clearly recursive, so suppose L is infinite.

Consider the following algorithm.

Algorithm 2.2: Deciding L from monotone enumerator.

```
1 Given an input $x \in \Sigma^*$, run $E$ on $n = 0, 1, \ldots$ until $E(n) = x$ (that is, we have $x_n = x$).
2 $k \leftarrow 0$.
3 for $i \leftarrow 0, 1, \ldots$ do
4     Let $x = E(i)$ (i.e., we have $x_i = x$).
5     Run $M$ on $x$.
6     if $M$ outputs $y$ then
7         if $x = y$ then Accept.
8         else
9             Run $E$ on $m = 0, 1, \ldots$ until $E(m) = y$.
10            if $m > n$ then Reject.
```

First note that since L is infinite, then M must always halt (otherwise, if it does not halt for x_n, then we have $f(x_m) = \infty$ for every $m \geqslant n$, which would mean that L is finite). Also, for every input x, since $f(x_0) < f(x_1) < \cdots$, we know that there exists $m \in \mathbb{N}$ large enough so that $f(x_m) \geqslant x$. Taking the minimum such $m \in \mathbb{N}$, we get that x is enumerated by f if and only if $f(x_m) = x$, which is precisely what is checked by the algorithm above. ◁

Exercise 3. Show that every infinite recursively enumerable language L contains an infinite subset $L' \subseteq L$ such that L' is recursive.

Hint: use Exercises 1 and 2.

Solution. Let M be a Turing machine that computes an enumerator for L and let us construct a monotone enumerator for an infinite language $L' \subseteq L$. Using the notation of Exercise 2, consider the following algorithm.

Algorithm 3.1: Monotone enumerator from M.

```
1  Given an input $x \in \Sigma^*$, run $E$ on $n = 0, 1, \dots$ until $E(n) = x$ (that is, we have $x_n = x$).
2  $\ell \leftarrow -1$.
3  $k \leftarrow 0$.
4  for $s \leftarrow 0, 1, \dots$ do
5      for $i \leftarrow 0$ to $s$ do
6          Run $M$ on $x_i$ for $s$ steps. if $M$ outputs $y$ then
7              Run $E$ on $m = 0, 1, \dots$ until $E(m) = y$.
8              if $m > \ell$ then
9                  $\ell \leftarrow m$.
10                 if $k = n$ then Output $y$.
11
12                 $k \leftarrow k + 1$.
```

Clearly the algorithm above only enumerates words inside L. We claim now that the algorithm above always halts. Indeed, since L is infinite, we know that for i large enough, we must have $M(x_i) > x_\ell$. This means whatever is the value of ℓ, for s large enough, the algorithm will test true in line 8, this will cause either k to be incremented or M to output. Since k can only be incremented at most n times before outputting, it follows that the algorithm always halts.

It remains to prove that the algorithm above is a monotone enumerator f.

Note first that ℓ can only increase its value and it does so every time line 8 tests true.

Note that if $n_1 < n_2$, then the algorithm above runs exactly the same for x_{n_1} and x_{n_2} while $k < n_1$. This means that if $f(x_{n_1}) = x_r$, then when running on input x_{n_2}, right before k is set to $n_1 + 1$, we know that ℓ is set to r. Since the value of ℓ only increases and we know that if $f(x_{n_2}) = x_m$, then we must have $m > \ell \geqslant r$, it follows that $f(x_{n_1}) < f(x_{n_2})$.

Therefore f is monotone. ◁

Exercise 4. Consider the following two languages.

$$\text{Inf} = \{\langle M \rangle : L(M) \text{ is infinite}\};$$
$$\text{Tot} = \{\langle M \rangle : M \text{ halts on all inputs}\}.$$

(a) Give a reduction from Inf to Tot.

(b) Give a reduction from Tot to Inf.

(c) Using the generalized/second Rice's Theorem (a.k.a. Rice–Shapiro Theorem), show that neither Inf nor its complement $\overline{\text{Inf}}$ are recursively enumerable.

(d) Using items (a) and (c), show that neither Tot nor its complement $\overline{\text{Tot}}$ are recursively enumerable.

(e) Using the Rice–Shapiro Theorem directly (i.e., without using the previous items), show that neither Tot nor its complement $\overline{\text{Tot}}$ are recursively enumerable.

Solution. For item (a), consider the following algorithm that uses a Tot-oracle.

Algorithm 4.1: Algorithm for Inf, using Tot-oracle.

```
1  On input $\langle M \rangle$, do the following.
2  Consider the Turing machine $M'$ associated with the following algorithm:
3  **Algorithm** $M$’**:**
4      On input $x$, let $n = |x|$ be the length of $x$.
5      **for** $t \leftarrow 0, 1, \ldots$ **do**
6          Let $S_t$ be the set of all words of length greater or equal to $n$ but smaller or equal
           to $n + t$.
7          **for** $y \in S_t$ **do**
8              Run $M$ on $y$ for $t$ steps.
9              **if** $M$ *accepts* **then** Accept.
10 **if** $\langle M' \rangle \in$ Tot **then**
11     Accept.
12 **else**
13     Reject.
```

Note that the Tot-oracle algorithm above always halts.

Consider an input $\langle M \rangle$ in Inf. Then we claim that the machine M' built by the algorithm on input $\langle M \rangle$ is in Tot.

Indeed, since $L(M)$ is infinite, whatever is the input x of M', we know that there exists $y \in L(M)$ such that $|y| \geqslant |x|$, which means that for t large enough, we have both $y \in S_t$ and M accepts y in at most t steps, so M' must accept x. Therefore M' accepts all words, that is, we have $L(M') = \Sigma^*$, which implies that M' halts in all inputs, hence $\langle M' \rangle \in$ Tot.

But then the algorithm above accepts $\langle M \rangle$.

Consider now an input $\langle M \rangle$ that is not in Inf. Then we claim that the machine M' built by the algorithm on input $\langle M \rangle$ is not in Tot.

Indeed, since $L(M)$ is finite, we know that M' cannot accept any input x of length greater than the largest length of a word in $L(M)$, since M' only tests words that are of length greater or equal to $|x|$. But M' never explicitly rejects any word, so it must loop forever in these large length words, hence $\langle M' \rangle \notin$ Tot.

But then the algorithm above rejects $\langle M \rangle$.

Therefore, the algorithm above decides Inf with the help of a Tot-oracle.

For item (b), consider the following algorithm that uses an Inf-oracle.

Algorithm 4.2: Algorithm for Tot, using Inf-oracle.

```
1  On input $\langle M \rangle$, do the following.
2  Consider the Turing machine $M'$ associated with the following algorithm:
3  **Algorithm** $M$’**:**
4      On input $x$, let $n = |x|$ be the length of $x$.
5      Let $S_n$ be the set of all words of length at most $n$.
6      Run $M$ on all words in $S_n$.
7      If $M$ halts on all these words, accept.
8  **if** $\langle M' \rangle \in$ Inf **then** Accept.
9  **else** Reject.
```

Note that the Inf-oracle algorithm above always halts.

Consider an input $\langle M \rangle$ in Tot. Then we claim that the machine M' built by the algorithm on input $\langle M\rangle$ is in Inf.

Indeed, since M always halts, it follows that M' will always accept any input, hence $L(M') = \Sigma^*$, which is infinite.

But then the algorithm above accepts $\langle M\rangle$.

Consider now an input $\langle M\rangle$ that is not in Tot. Then we claim that the machine M' built by the algorithm on input $\langle M\rangle$ is not in Inf.

Indeed, since M does not always halt, there exists a word x such that M does not halt on x, which implies that M' will never halt in any word of length at least $|x|$ (since it will run M on x). Therefore $L(M')$ can only contain words of length less than $|x|$, i.e., it must be a finite language, hence $\langle M'\rangle \notin \mathrm{Inf}$.

But then the algorithm above rejects $\langle M\rangle$.

Therefore, the algorithm above decides Tot with the help of an Inf-oracle.

Let us now solve item (c).

Consider the set P_∞ of all recursively enumerable languages that are infinite. Note that

$$\mathrm{Inf} = \{\langle M\rangle : L(M) \in P_\infty\}.$$

However, note that for every $L \in P_\infty$ and every $L' \subseteq L$ finite, we have $L' \notin P_\infty$ (since L' is finite). So by Rice–Shapiro Theorem, we get that Inf is not recursively enumerable.

Let then P_f be the set of all recursively enumerable languages that are finite. Note that

$$\overline{\mathrm{Inf}} = \{\langle M\rangle : L(M) \in P_f\}.$$

Take any $L \in P_f$ and note that $\Sigma^* \supseteq L$, but $\Sigma^* \notin P_f$ (as it is infinite). Hence by Rice–Shapiro Theorem, we get that $\overline{\mathrm{Inf}}$ is not recursively enumerable.

We now proceed to item (d).

Suppose toward a contradiction that Tot is recursively enumerable, then we can replace the Tot-oracle of item (a) with a Turing machine M such that $L(M) = \mathrm{Tot}$ and the resulting algorithm will accept every input in Inf and either reject or loop forever in every input not in Inf. This implies that Inf is recursively enumerable, which is a contradiction with item (c).

Suppose now toward a contradiction that $\overline{\mathrm{Tot}}$ is recursively enumerable and let T be a Turing machine such that $L(T) = \overline{\mathrm{Tot}}$, then we can replace the last two lines of the algorithm in item (a) with "Run T on $\langle M'\rangle$, if M' accepts, accept; otherwise reject.", and the resulting algorithm will accept every input in $\overline{\mathrm{Inf}}$ and either reject or loop forever in every input not in $\overline{\mathrm{Inf}}$. This implies that $\overline{\mathrm{Inf}}$ is recursively enumerable, which is a contradiction with item (c).

Finally, let us solve item (e).

Let $P = \{\Sigma^*\}$ be the set of languages that contains only the language of all words. Note that Σ^* does not have any finite sublanguage that is also in P, so by the Rice–Shapiro Theorem, we know that $L_P = \{\langle M\rangle : L(M) \in P\} = \{\langle M\rangle : L(M) = \Sigma^*\}$ is not recursively enumerable.

Suppose toward a contradiction that Tot is recursively enumerable and let M_T be a Turing machine such that $L(M_T) = \text{Tot}$. Consider the following algorithm.

Algorithm 4.3: Recognizing L_P from Tot.

```
1  On input $\langle M \rangle$, consider the Turing machine $M'$ associated with the following algorithm:
2  **Algorithm** $M'$:
3      On input $x$, run $M$ on $x$.
4      if $M$ accepts then
5          Accept.
6      else
7          Loop forever.
8  Run $M_T$ on $\langle M' \rangle$. if $M_T$ accepts then
9      Accept.
10 else
11     Reject.
```

Note that the machine M' constructed by the algorithm above on input $\langle M \rangle$ satisfies $L(M) = L(M')$ and M' never explicitly rejects an input. This means that $\langle M' \rangle \in \text{Tot}$ if and only if $L(M') = \Sigma^*$. This implies that M_T accepts $\langle M' \rangle$ if and only if $L(M) = \Sigma^*$, so the algorithm above accepts exactly the words in L_P, a contradiction.

Therefore Tot is recursively enumerable.

Let now $P' = \{L \subseteq \Sigma^* : L \text{ is recursively enumerable and } L \neq \Sigma^*\}$ be the set of all recursively languages that do not contain all the words. Clearly Σ^* is not in P and for any language L in P (e.g., for $\varnothing \in P$), we have $L \subsetneq \Sigma^*$. By the Rice–Shapiro Theorem, we know that $L_{P'} = \{\langle M \rangle : L(M) \in P'\} = \{\langle M \rangle : L(M) \neq \Sigma^*\}$ is not recursively enumerable.

Suppose toward a contradiction that $\overline{\text{Tot}}$ is recursively enumerable and let M_T be a Turing machine such that $L(M_T) = \overline{\text{Tot}}$. Consider Algorithm 4.3 again but with this definition of M_T. Again, we know that the M' constructed by the algorithm is such that $L(M) = L(M')$ and furthermore, we know that $\langle M' \rangle \in \text{Tot}$ if and only if $L(M') = \Sigma^*$. This implies that M_T accepts $\langle M' \rangle$ if and only if $L(M) \neq \Sigma^*$, so Algorithm 4.3 accepts exactly the words in $L_{P'}$, a contradiction. ◁