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

October 31, 2017

Problem 1 (3 points)**.** Give precise definitions of the diagonal language and the universal language.

Solution. The diagonal language is

$$D = \{\langle M\rangle : \langle M\rangle \notin L(M)\},$$

that is, it is the set of encodings of Turing machines M such that M does not accept its own encoding $\langle M\rangle$ (i.e., it either rejects or loops forever).

The universal language is

$$L_u = \{\langle M, w\rangle : w \in L(M)\},$$

that is, it is the set of (encoded) pairs (M, w) such that M is a Turing machine that accepts the input w. ◁

Problem 2 (3 points)**.** Give examples of languages A, B and C such that $A \subseteq B \subseteq C$, A and C are undecidable, and B is decidable.

Solution. Let

$$\begin{aligned} A &= \{\langle M, w\rangle : w \in L(M) \text{ and } |w| \text{ is even}\};\\ B &= \{\langle M, w\rangle : |w| \text{ is even}\};\\ C &= \{\langle M, w\rangle : w \in L(M) \text{ or } |w| \text{ is even}\}; \end{aligned}$$

where $|w|$ denotes the length of w.

Clearly we have $A \subseteq B \subseteq C$. Furthermore, it is easy to see that B is decidable by using an algorithm to compute the length of a word.

Now A is the intersection of B with the universal language L_u. It is equally as hard to decide A as it is to decide L_u since for a Turing machine M, we can consider a Turing machine M' that starts by erasing the symbol under the head, moving right, then simulating M; for such M' we have $\langle M, w\rangle \in L_u$ if and only if $\langle M', 0w\rangle \in L_u$. But then we get

$$\langle M, w\rangle \in L_u \iff \langle M, w\rangle \in A \vee \langle M', 0w\rangle \in A.$$

On the other hand, the language C is the union of B with the universal language L_u. With the same construction of M' as above, we have

$$\langle M, w\rangle \in L_u \iff \langle M, w\rangle \in C \wedge \langle M', 0w\rangle \in C.$$

Therefore A and C are both undecidable. ◁

Problem 3 (7 points)**.** Give one example of each (a) a language that is r.e. but not recursive, (b) a language that is r.e. but its complement is non-r.e., (c) a language that is non r.e. and its complement is also non-r.e.

Solution. For items (a) and (b), we have seen in class that the universal language $L_u = \{\langle M, w\rangle : w \in L(M)\}$ is r.e. but not recursive. Since languages that are both r.e. and co-r.e. are recursive, it follows that L_u is also not co-r.e. (i.e., its complement is not r.e.).

For item (c), consider the language

$$L = \{\langle M_1, w_1, M_2, w_2\rangle : w_1 \in L(M_1) \text{ and } w_2 \notin L(M_2)\}.$$

Let M_a be a Turing machine that accepts all inputs and let M_r be a Turing machine that rejects all inputs. Let also ϵ be the empty string.

We claim that L is not r.e. Indeed, if this was the case, since we have

$$\langle M_a, \epsilon, M_2, w_2\rangle \in L \iff \langle M_2, w_2\rangle \notin L_u,$$

this would imply that $\overline{L_u}$ is r.e., i.e., that L_u is co-r.e., a contradiction.

We claim now that L is not co-r.e. Indeed, if this was the case, since we have

$$\langle M_1, w_1, M_r, \epsilon\rangle \in L \iff \langle M_1, w_1\rangle \in L_u,$$

this would imply that L_u is co-r.e., a contradiction. ◁

Problem 4. Given languages L and R, let $L * R = \{xy : x \in L, y \in R\}$, where xy denotes the concatenation of x and y.

(a) (5 points) Suppose L and R are r.e. Is $L * R$ then r.e.? If so, prove this. If not, disprove it.

(b) (5 points) Suppose L and R are recursive. Is $L * R$ then recursive? If so, prove this. If not, disprove it.

Solution. For item (a), let us prove that $L * R$ is r.e. Let M_L and M_R be Turing machines such that $L(M_L) = L$ and $L(M_R) = R$ and consider the following algorithm.

Algorithm 4.1: Algorithm for recognizing $L * R$

```
1 On input x, let n = |x| be the length of x and write x = x_1 x_2 ··· x_n for x_i ∈ Σ.
2 for s ← 0, 1, ... do
3     for i ← 0 to n do
4         Run M_L on x_1 ··· x_i (if i = 0, this is the empty string) for s steps.
5         if M_L accepts then
6             Run M_R on x_{i+1} ··· x_n (if i = n, this is the empty string) for s steps.
7             if M_R accepts then Accept.
```

We claim that the algorithm above accepts exactly $L * R$ (it may not halt though).

Indeed, if $x = x_1x_2\cdots x_n \in L*R$, then there exists $i \in \{0, 1, \ldots, n\}$ such that $y = x_1\cdots x_i \in L$ and $z = x_{i+1}\cdots x_n \in R$, which implies that $y \in L(M_L)$ and $z \in L(M_R)$. If s is large enough so as the computations of M_L on input y and of M_R on input z both take less than s steps, then the algorithm above will see this and accept. Since the algorithm never explicitly rejects, it follows that the algorithm accepts x.

On the other hand, if the algorithm accepts $x = x_1\cdots x_n$, then there must exist $i \in \{0, 1, \ldots, n\}$ such that $x_1\cdots x_i \in L(M_L)$ and $x_{i+1}\cdots x_n \in L(M_R)$, so we have $x \in L * R$.

For item (b), let us prove that $L * R$ is recursive. Let M_L and M_R be Turing machines that decide L and R respectively and consider the following algorithm.

Algorithm 4.2: Algorithm for deciding $L * R$

```
On input x, let n = |x| be the length of x and write x = x_1 x_2 ··· x_n for x_i ∈ Σ.
for i ← 0 to n do
    Run M_L on x_1 ··· x_i (if i = 0, this is the empty string).
    if M_L accepts then
        Run M_R on x_{i+1} ··· x_n (if i = n, this is the empty string).
        if M_R accepts then Accept.
Reject.
```

Note that since M_L and M_R always halt, the algorithm above always halts.

Indeed, if $x = x_1x_2 \cdots x_n \in L*R$, then there exists $i \in \{0, 1, \ldots, n\}$ such that $y = x_1 \cdots x_i \in L$ and $z = x_{i+1} \cdots x_n \in R$, which implies that $y \in L(M_L)$ and $z \in L(M_R)$. This implies that the algorithm above accepts x on or before iteration i.

On the other hand, if the algorithm accepts $x = x_1 \cdots x_n$, then there must exist $i \in \{0, 1, \ldots, n\}$ such that $x_1 \cdots x_i \in L(M_L)$ and $x_{i+1} \cdots x_n \in L(M_R)$, so we have $x \in L * R$. ◁

Problem 5 (7 points)**.** Let M_i denote the i-th Turing machine. Show that the language

$$L = \{(i, j) : \text{there is some input on which both } M_i \text{ and } M_j \text{ halt}\}$$

is undecidable.

Solution. We will reduce L to the universal language $L_u = \{\langle M, w\rangle : w \in L(M)\}$.

Suppose toward a contradiction that L is decidable, that is, suppose that there exists a Turing machine M_L that decides L. Consider then the following algorithm.

Algorithm 5.1: Deciding L_u from L.

```
On input ⟨M, w⟩, compute i such that M_i is the Turing machine with the following
    algorithm: "On input x, run M on w. If M accepts, accept; otherwise loop forever.".
Run M_L on input (i, i).
if M_L accepts then Accept.
else Reject.
```

Note that since M_L always halts, the algorithm above always halts.

Note also that if i is the index computed by the algorithm above for the input $\langle M, w\rangle$, then we have

$$w \in L(M) \Longrightarrow L(M_i) = \Sigma^*; \qquad w \notin L(M) \Longrightarrow M_i \text{ never halts on any input.}$$

Hence $\langle M, w\rangle \in L_u$ if and only if $(i, i) \in L$, which implies that the algorithm above decides L_u, a contradiction. ◁

Problem 6 (10 points)**.** Prove Rice's Theorem that every nontrivial property of r.e. languages is undecidable.

Solution. Rice's Theorem is the following.

Theorem. *Let P be a subset of all r.e. languages that is not trivial (that is, there is at least one r.e. language in P and at least one r.e. language not in P) and let*

$$L_P = \{\langle M\rangle : L(M) \in P\}.$$

Then L_p is undecidable.

Proof. Note that if $\overline{P}$ is the complement of P with respect to the set of all r.e. languages, then $L_{\overline{P}} = \overline{L_P} = \{\langle M \rangle : L(M) \notin P\}$.

This means that by possibly replacing P with $\overline{P}$, we may suppose without loss of generality that $\varnothing \notin P$ (i.e., the empty language is not in P).

Since P is non-trivial, we know that there exists $L \in P$, and since L is r.e., we know that there exists a Turing machine M_L such that $L(M_L) = L$.

Suppose toward a contradiction that L_P is decidable, that is, there exists a Turing machine M_P that decides L_P. Consider then the following algorithm.

Algorithm 6.1: Deciding L_u from L_P.

1 On input $\langle M, w \rangle$, let M' be the Turing machine associated with the algorithm: "On input x, run M on w. If M accepts, run M_L on x and return its result. If M rejects, then reject.".
2 Run M_P on M'.
3 **if** M_P *accepts* **then** Accept.
4 **else** Reject.

Note that since M_P always halts, the algorithm above always halts.

Note also that if M' is the Turing machine computed by the algorithm above for the input $\langle M, w \rangle$, then we have

$$L(M') = \begin{cases} L(M_L), & \text{if } w \in L(M); \\ \varnothing, & \text{if } w \notin L(M); \end{cases}$$
$$= \begin{cases} L, & \text{if } w \in L(M); \\ \varnothing, & \text{if } w \notin L(M). \end{cases}$$

Hence $\langle M, w \rangle \in L_u$ if and only if $\langle M' \rangle \in L_P$ (since $\varnothing \notin P$ and $L \in P$), so the algorithm above decides L_u, a contradiction. ■

◁