

Problem Set 5 Solutions

Problem 1

Consider the language

$$L = \{\langle M \rangle \mid M \text{ is a DFA and all strings accepted by } M \text{ are palindromes}\}$$

(Remember, w is a palindrome if w reads the same forward and backward.) Prove that L is a decidable language. [Hint: first determine if the set of all strings that are *not* palindromes is context free.]

Solution:

In order to prove that L is a decidable language, we must come up with a Turing machine that decides L . Let P be such a Turing machine. Upon receiving an input of the form $\langle M \rangle$ (which is supposed to be an encoding of a DFA), the Turing machine P needs to determine that M is a DFA and all strings accepted by M are palindromes. How can P do this? Simulating M on some string w won't do the job because the language requires P to check if *all* strings accepted by M are palindromes, not just some. Directly simulating the DFA M on all possible input strings w will not work since the number of all possible input strings is infinite.

Hence, we need to take a different approach that cleverly uses some of the known facts about some other languages. From the hint, consider the language $L_2 = \{w \mid w \text{ is not palindrome}\}$. Let $L_1 = \mathcal{L}(M)$, where M is the DFA that P gets as the input. Consider the intersection of L_1 and L_2 . If all strings that belong to L_1 are palindromes, then what can we say about the intersection of L_1 and L_2 ? That will be the empty set \emptyset because L_2 contains all strings that are not palindromes and all strings in L_1 are palindromes. What if L_1 contains some strings that are not palindromes? Then $L_1 \cap L_2$ will contain the strings that are not palindromes that belong to L_1 . So $L_1 \cap L_2 \neq \emptyset$ in this case. Now if we have some Turing machine (TM) that decides whether $L_1 \cap L_2$ is \emptyset or not, then we can use the TM to decide that L_1 contains only palindromes (i.e. all strings that are accepted by the DFA M are palindromes or not). We can find this if we know whether L_2 is context free or not. If L_2 is context free, then there exists a context free grammar G that generates it and we can convert G into a PDA N and use the fact that the intersection of a context free language and a regular language is context free to build a PDA N' that recognizes $L_1 \cap L_2$, where $L_1 = \mathcal{L}(M)$ and $L_2 = \mathcal{L}(N)$. Once we have N' we can convert it to a context free grammar G' and use a Turing machine E that

decides $E_{CFG} = \{\langle G \rangle \mid G \text{ is a context free grammar and } \mathcal{L}(G) = \emptyset\}$. Note that if E accepts, then $L_1 \cap L_2 = \emptyset$, which means all strings accepted by M are palindromes (i.e. $\langle M \rangle \in L$). Hence, P should accept. Otherwise, P rejects.

Let us see if L_2 is context free. It turns out that there is a context free grammar $G = (\{S, T\}, \{a, b\}, R, S)$ that generates L_2 as follows (here, we use the alphabet $\Sigma = \{a, b\}$ for simplicity):

$$\begin{aligned} S &\rightarrow aSa \mid bSb \mid aTb \mid bTa \\ T &\rightarrow aT \mid bT \mid \epsilon \end{aligned}$$

In summary, given a Turing machine E that decides the language E_{CFG} , we can build a Turing machine P that decides L using E as follows:
 $P =$ "On input $\langle M \rangle$:

1. Check if $\langle M \rangle$ is a correct encoding of a DFA and if not, reject.
2. Build a context free grammar G that generates the set of all strings that are not palindromes (i.e. $\mathcal{L}(G) = \{w \mid w \text{ is not palindrome}\}$) as described above.
3. Convert G into a PDA N .
4. Construct a PDA N' such that $\mathcal{L}(N') = \mathcal{L}(N) \cap \mathcal{L}(M)$.
5. Convert the PDA N' into the equivalent grammar G' .
6. Run the Turing machine E on input $\langle G' \rangle$.
7. If E accepts, accept; if E rejects reject."

Problem 2

In class we have seen that the languages A_{DFA} , A_{REX} , E_{DFA} , E_{REX} , ALL_{DFA} , ALL_{REX} , EQ_{DFA} , EQ_{REX} , A_{CFG} , A_{PDA} , E_{CFG} , E_{PDA} are decidable, but the languages EQ_{CFG} , ALL_{CFG} , EQ_{PDA} , ALL_{PDA} are not. For each of the following languages say whether the language is decidable or not, and prove your answer by reduction to or from any of the above languages.

1. The set of all strings $\langle G, R \rangle$ where G is a context free grammar, R is a regular expression and G and R are equivalent, i.e., they generate the same language.
2. The set of all strings $\langle G, n \rangle$ such that G is a context free grammar, n a positive integer, and all strings generated by G have length n .

3. The set of all strings $\langle N_1, N_2 \rangle$ such that N_1, N_2 are nondeterministic finite state automata, and $L(N_1) \subseteq L(N_2)$.

Solution:

Part 1. Let L_1 denote the language described in part 1. That is, let $L_1 = \{\langle G, R \rangle \mid G \text{ is a context free grammar, } R \text{ is a regular expression and } \mathcal{L}(G) = \mathcal{L}(R)\}$ for convenience.

L_1 is *not* decidable. We will prove it by a reduction from the undecidable language $ALL_{CFG} = \{\langle G \rangle \mid G \text{ is a context free grammar and } \mathcal{L}(G) = \Sigma^*\}$. Assume for contradiction that L_1 is decidable. Then, there exists a Turing machine that decides L_1 . Let N denote such a Turing machine. Then we can construct a TM T that decides ALL_{CFG} using N as follows:

$T =$ "On input $\langle G \rangle$:

1. Check if $\langle G \rangle$ is a correct encoding of a context free grammar and if not, reject.
2. Construct a regular expression R such that $\mathcal{L}(R)$ is the set of all strings. We can let $R = \Sigma^*$ if we let Σ denote the shorthand for the regular expression that is a union of all elements in the alphabet (e.g. $(a \cup b)$, where the alphabet is $\{a, b\}$).
3. Run the Turing machine N on input $\langle G, R \rangle$.
4. If N accepts, accept; if N rejects reject."

It is easy to see that the TM T described above decides the language ALL_{CFG} correctly if N decides L_1 correctly. However, we know that ALL_{CFG} is not decidable. Hence, having a decider T for ALL_{CFG} is a contradiction, and the contradiction has resulted from the assumption that L_1 is decidable. Hence, we can conclude that L_1 is not decidable.

Remark 1 At first glance, the language may look decidable, and some people may show that the language L_1 is decidable by using the following statement: $\mathcal{L}(G)$ and $\mathcal{L}(R)$ are equivalent if $\mathcal{L}(G) \cap \overline{\mathcal{L}(R)} = \emptyset$. However, this is NOT true because it is missing one important case. The statement is true if the relationship between the languages are subset relationship instead of equivalence; that is, $\mathcal{L}(G) \subseteq \mathcal{L}(R)$ if $\mathcal{L}(G) \cap \overline{\mathcal{L}(R)} = \emptyset$. For equivalence, the other subset relationship should be tested also, meaning you need to test if $\mathcal{L}(R) \subseteq \mathcal{L}(G)$ as well as $\mathcal{L}(G) \subseteq \mathcal{L}(R)$. By testing just one case, you would not know whether the languages are equivalent because a language, say B , containing all of the elements of the other language, say A , may contain more elements than the other when A belongs to B (i.e. B is a strictly larger set than A or A is a proper subset of B if $A \subset B$). Hence, in order to

test whether two languages are equivalent, you need to test both cases as follows: $\mathcal{L}(G) = \mathcal{L}(R)$ if and only if $\mathcal{L}(G) \subseteq \mathcal{L}(R)$ and $\mathcal{L}(R) \subseteq \mathcal{L}(G)$, which is equivalent to the statement $\mathcal{L}(G) \cap \overline{\mathcal{L}(R)} = \emptyset$ and $\mathcal{L}(R) \cap \overline{\mathcal{L}(G)} = \emptyset$. Notice that the second case (i.e. $\mathcal{L}(R) \cap \overline{\mathcal{L}(G)} = \emptyset$) cannot be tested because context free languages are not closed under the complement operation.

Part 2. Let L_2 denote the language described in part 2. That is, $L_2 = \{\langle G, n \rangle \mid G \text{ is a context free grammar, } n \text{ a positive integer, and all strings generated by } G \text{ have length } n\}$.

L_2 is decidable, and we can use a similar technique used in the Problem 1 (which asked to prove that L is decidable) to prove that L_2 is decidable. As is done in Problem 1, in order to show that L_2 is decidable, we need to build a Turing machine that decides L_2 . Let T be such a Turing machine. The task of T is as follows: given $\langle G, n \rangle$ as input, T needs to decide whether all strings generated by the context free grammar G have the length n . One obvious approach might be to generate all possible strings of length n (since n is a fixed positive integer, this can be done in a finite amount of time) and check if each string of length n is generated by G using the decider Turing machine A that decides $A_{CFG} = \{\langle G, w \rangle \mid G \text{ is a CFG that generates string } w\}$. However, this does not work because even if A answers decides on each input $\langle G, w \rangle$ where w is a string of length n , it does not tell if *all* strings generated by G have length n . Some strings that G generates might have length greater or less than n , and running the TM A that decides A_{CFG} on input w of a certain fixed length n does not tell this. To be sure about this, we may need to run the TM A on all possible strings w and we know already that it is not feasible (the TM T won't halt, and therefore it won't be a decider). Hence, we need to take a different approach that is similar to the approach taken in the solution to Problem 1. It begins by observing that the language that is the set of all strings that have length n (a given fixed constant) is regular, and therefore there exists a DFA that recognizes them. In other words, the language can be describes as follows: Σ^n , where n is a fixed constant and Σ is a shorthand for the regular expression that describes the union of all alphabets. Let us denote the language by L_n . Since L_n is regular and regular languages are closed under complement, we can build a DFA that recognizes $\overline{L_n}$. Why are we talking about $\overline{L_n}$? Because we want to use it with $\mathcal{L}(G)$, which we denote L_G . We want to know whether $L_G \subseteq L_n$, which is true if $L_G - L_n = \emptyset$ (where “ $-$ ” is the set difference operation). We know that $A - B = A \cap \overline{B}$, where A, B are sets and “ $-$ ” is the set difference operation. Hence, $L_G - L_n = L_G \cap \overline{L_n}$. If $L_G \cap \overline{L_n} = \emptyset$ then the input string $\langle G, n \rangle$ belongs to L_2 ; otherwise, it doesn't. Also, recall that the intersection of a CFL and a regular language is a CFL. Hence, we can test if $L_G \cap \overline{L_n} = \emptyset$ using a Turing machine E that decides E_{CFG} in the same way used in Problem 1.

We now describe the algorithm for the Turing machine T that decides L_2 in more detail. Given a Turing machine E that decides the language E_{CFG} , we can build a Turing machine T that decides L_2 using E as follows:

$T =$ "On input $\langle G, n \rangle$:

1. Check if $\langle G, n \rangle$ is a correct encoding of a CFG and a positive integer, respectively, and if not, reject.
2. Build a DFA M that recognizes the language $\overline{L_n}$ (i.e. the set of all strings that do not have length n) as described above.
3. Convert G into a PDA N .
4. Construct a PDA N' such that $\mathcal{L}(N') = \mathcal{L}(N) \cap \mathcal{L}(M)$, (which is equivalent to $L_G \cap \overline{L_n}$ just described above).
5. Convert the PDA N' into the equivalent grammar G' .
6. Run the Turing machine E on input $\langle G' \rangle$.
7. If E accepts, accept; if E rejects reject."

Note that in the proof shown above, we have reduced the problem of deciding L_2 into the problem of deciding E_{CFG} . Since we know how to decide E_{CFG} , we can decide L_2 using above shown reduction.

Part 3. Let L_3 denote the language described in part 3. That is, $L_3 = \{\langle N_1, N_2 \rangle \mid N_1, N_2 \text{ are NFA, and } L(N_1) \subseteq L(N_2)\}$.

L_3 is decidable, and it can be proven using a similar technique used in Part 2. For the proof, as usual, we need to construct a Turing machine that decides L_3 using a Turing machine that decides some language. Since from part 2, we showed that a language is a subset of another language (i.e. $\mathcal{L}(G) \subseteq \mathcal{L}(L_n)$) using a decider for E_{CFG} , we can apply the idea here also because it tests subset relationship of two languages. In other words, we can reduce L_3 into some other language that is known to be decidable. One possible candidate may be E_{CFG} . However, we need to realize that the decider for E_{CFG} decides something about context free grammars, whereas in L_3 , we are dealing with NFA. Of course, we can always construct a CFG that generates the same strings that are accepted by an NFA because all regular languages are also context free languages. However, we can do even more simpler thing in this case. Instead of using E_{CFG} , we can use E_{DFA} or even E_{NFA} directly. Recall that $E_{DFA} = \{\langle M \rangle \mid M \text{ is a DFA and } \mathcal{L}(M) = \emptyset\}$. Since E_{DFA} is one of the languages mentioned in the problem statement, from or to which the language can be reduced (whereas E_{NFA} is not), we will use a TM that decides E_{DFA} to build a TM that decides L_3 (i.e. we reduce L_3 to E_{DFA}). Recall the following relationship from part 2: $A \subseteq B$ if and only if $A \cap \overline{B} = \emptyset$.

We now describe the reduction algorithm—a Turing machine that decides L_3 using a TM that decides E_{DFA} —in more detail. Let S denote a TM that decides E_{DFA} . Given S , we can construct the TM T that decides L_3 as follows:

$T =$ "On input $\langle N_1, N_2 \rangle$:

1. Check if $\langle N_1, N_2 \rangle$ is a correct encoding of two NFA, and if not, reject.
2. Convert the NFA N_1 and N_2 into corresponding DFA M_1 and M_2 , respectively. Note that this is possible because *NFA* and *DFA* are equivalent. (We learned how to convert a NFA into a DFA before.)
3. Construct a DFA M such that $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \overline{\mathcal{L}(M_2)}$. Note that this is possible because regular languages are closed under intersection and complement operations.
4. Run the Turing machine S on input $\langle M \rangle$.
5. If S accepts, accept; if S rejects reject.”

Note again that in order to construct a TM that decides the language L_3 , we used a TM that decides E_{DFA} . Hence, we reduced L_3 to E_{DFA} .

Problem 3

In class we proved that the language A_{TM} is Turing-recognizable, but not Turing-decidable.

1. Prove that E_{TM} is also undecidable (You can either give a direct proof by diagonalization, or prove your answer by reduction from A_{TM})
2. Is E_{TM} Turing-recognizable? Is it co-Turing-recognizable? Briefly justify your answers.
3. Prove that A_{TM} is not map-reducible to E_{TM} .

Solution:

Part 1. This problem is solved on the book. See pages 173 and 174.

Part 2. E_{TM} is not Turing-recognizable but it is co-Turing-recognizable. We prove these facts in the two following claims.

Claim 2 E_{TM} is co-Turing-recognizable.

Proof: Recall that $E_{TM} = \{ \langle M \rangle : M \text{ is a Turing machine and } \mathcal{L}(M) = \emptyset \}$ and then, $\overline{E_{TM}} = \{ \langle M \rangle : (M \text{ is not a TM}) \text{ or } (M \text{ is a TM but } \mathcal{L}(M) \neq \emptyset) \}$
 The following Turing machine S is a recognizer for $\overline{E_{TM}}$.

$S(\langle M \rangle)$:

Check if $\langle M \rangle$ encodes a proper Turing machine. If it does not, accept.

Set $i = 0$

Repeat forever

For all $x \in \Sigma^*$ of length at most i , taking them in lexicographical order

Run $M(x)$ for at most i steps ($M(x)$ may stop in less than i steps).

If M accepts, accept.

Set $i \leftarrow i + 1$.

Clearly, machine S will accept input $\langle M \rangle$ only if (1) it does not encode a valid Turing machine or (2) M ever accepts a string. The first case is easy: we just need to check if $\langle M \rangle$ is a proper encoding of any Turing machine (note this is just a *syntactic* check). For the second case (in order to see if M ever accepts a string) we cannot just let M run freely on input x because it may loop forever. So, in order to “try all possible strings” on M , we run machine M on each string of length i for at most i steps¹. Thus, if M ever accepts a string $w \in \Sigma^*$ (say $|w| = t$ and M accepts w in k steps), machine S will run M on w first when $i = t$ for just t steps; then for each $i > t$ for i steps, so when $i = k$ machine M will accept w , and therefore S will accept too. ■

Claim 3 E_{TM} is not Turing-recognizable.

Proof: By contradiction. Assume E_{TM} is Turing-recognizable. Then, since E_{TM} is co-Turing-recognizable, we have that E_{TM} is decidable. Since we know (Part 1.) E_{TM} is undecidable, we have got a contradiction. Then, it must be that our assumption is not true and E_{TM} is not Turing-recognizable. ■

Part 3.

Claim 4 A_{TM} is not map-reducible to E_{TM} , that is, $A_{\text{TM}} \leq_m E_{\text{TM}}$ is not possible.

Before proving the claim, we need two useful lemmas first (use them in your proofs if you need them!). The first one shows that mapping-reducibility is preserved under complement.

Lemma 5 Let $A, B \subset \Sigma^*$ be languages. If $A \leq_m B$ then $\overline{A} \leq_m \overline{B}$.

Proof: The definition of mapping-reducibility states that there exists a computable function f such that

if $x \in A \Rightarrow f(x) \in B$, and

if $x \notin A \Rightarrow f(x) \notin B$, and

¹This trick is similar to what we did to create an *enumerator* for a Turing-recognizable language.

We can write the condition $x \in A$ as $x \notin \bar{A}$ and $x \notin A$ as $x \in \bar{A}$. The same with condition $f(x) \in B$. Applying those “changes” and swapping the two lines we get

$$\begin{aligned} \text{if } x \in \bar{A} &\Rightarrow f(x) \in \bar{B}, \text{ and} \\ \text{if } x \notin \bar{A} &\Rightarrow f(x) \notin \bar{B}, \text{ and} \end{aligned}$$

which means that the same computable function f works to get $\bar{A} \leq_m \bar{B}$. ■

The second lemma says that the property of being co-Turing-recognizable “propagates” from left to right (as the Turing-recognizability does).

Lemma 6 Let $A, B \subset \Sigma^*$ be languages and suppose $A \leq_m B$. Then if A is not co-Turing-recognizable then B is not co-Turing-recognizable.

Proof: Assume $A \leq_m B$. By previous lemma we know that $\bar{A} \leq_m \bar{B}$ too. Now, if A is not co-Turing-recognizable, it means \bar{A} is not Turing-recognizable (by definition of the “co-” classes). By Corollary 5.23 (page 193) from the book, we know that if \bar{A} is not Turing-recognizable and $\bar{A} \leq_m \bar{B}$, then also \bar{B} is not Turing-recognizable. Saying that \bar{B} is not Turing-recognizable it is the same as saying that B is not co-Turing-recognizable. ■

Now we can easily prove the claim.

Proof: (of Claim 3) By contradiction. Assume $A_{\text{TM}} \leq_m E_{\text{TM}}$. Then, since A_{TM} is not co-Turing-recognizable, we have that E_{TM} is not co-Turing-recognizable. But, by Part 2. we know that E_{TM} is co-Turing-recognizable, so we have a contradiction. Then, it must be that our assumption is not true and there exists no such a mapping. ■

Remark 7 An alternative proof of Claim 3 would be the following: assume that $A_{\text{TM}} \leq_m E_{\text{TM}}$, and since E_{TM} is co-Turing-Recognizable (by Part 2., Claim 1) then A_{TM} is co-Turing-recognizable (well, we’d need a lemma for this fact, namely that “if $A \leq_m B$ and B is co-Turing-recognizable, then A is co-Turing-recognizable”, but it easily follows from Lemma 4 and Theorem 5.22, page 193 in the book). Then we’d have a contradiction because we know that A_{TM} is not co-Turing-recognizable (Corollary 4.17, page 168).
