COL352 PS5

Harshil Vagadia, Aniket Gupta, Shrey Patel

31^{st} January, 2022

Contents

1 Question 1

Show that every infinite Turing-recognizable language has an infinite decidable subset.

Solution:
Let L be the given language. It is given that L is infinite. Also, since it is Turing-recognizable, $\exists$ a Turing Machine M that recognizes it and therefore an enumerator E which enumerates the strings of L. Now, assume that E prints the following strings $\{s_1, s_2, ...\}$ in order. Then we first prove the following claim:

Claim 1.1: $\forall$ i $\exists$ j $>$ i such that the string s_j is lexicographically greater than the string s_i.
Proof of Claim 1.1:
We prove this by contradiction. Suppose the given claim is false, i.e. for any i, $\forall$ j $>$ i, s_j cannot be lexicographically greater than s_i. Since s_i is a finite length string, there are finitely many strings which are lexicographically smaller than s_i. Given some i, consider the sets S = $\{s_1, s_2, ...s_i\}$ and S' = $\{s | s$ is lexicographically smaller than or equal to $s_i\}$ both of which are finite. Then, $\forall$ j $>$ i, $s_j \in$ S'. So, L $\subseteq \{s_1, s_2, ..s_i, s_{i+1}, ...\} \subseteq$ S $\cup$ S'. Since S $\cup$ S' is finite, the set of strings enumerated by E should also be finite. This means that L is also finite. We have a contradiction since we have been given that L is infinite.

Now, we construct a language L' = $\{w_1, w_2, ...\}$ inductively from L as follows:

- $w_1 = s_1$
- $\forall$ k ≥ 1, if $w_k = s_i$ for some i, then choose the smallest j $>$ i such that s_j is lexicographically greater than s_i. (We can always do so as proved in claim 1.1). Then $w_{k+1} = s_j$.

Clearly, L' $\subseteq$ L since we have constructed L' entirely using the strings of L. Now, we need to prove that L' is infinite. This proof follows directly from claim 1.1. Note from the second point

of the construction, $\forall$ k $\geq$ 1, we can always find some string w_{k+1} such that it is lexicographically greater than w_k, which means there is no upper bound on k. And $|L'|$ = k which means that there is no upper bound on the size of L'. So, L' is infinite.

Now, we prove that L' is decidable. For that, we first derive an enumerator E' that enumerates the language L'. We construct E' from E using the following steps. Note that E prints the enumeration $\{s_1, s_2, ...\}$.

1. At the beginning, E' prints s_1 and sets a variable $w = s_1$.
2. Now, if the next string printed by E is w', then there are two cases:
   - **Case 1:** If w' is lexicographically greater than w, then E' prints w' and set w = w'.
   - **Case 2:** Else, ignore w'.

   Move onto the next string of E and repeat this step.

Note that since we have already established that a Turing Machine simulation is equivalent to the enumeration of an enumerator, this construction works, since it is equivalent to constructing it from the the Turing Machine M which recognizes L.

Claim 1.2 E' enumerates the strings of L'
Proof of claim 1.2
We prove a stronger claim here. Let the sequence of strings emitted by E' be $\{u_1, u_2, ...\}$ and L' = $\{w_1, w_2, ...\}$. Then we prove that $u_i = w_i$ $\forall$ i $\geq$ 1. Proving this means that E' exactly enumerates L'. We prove this by induction on i.

- **Base Case:** i = 1. Then from point 1 of constructions of L' and E' from E, $w_1 = u_1 = s_1$. So, base case holds.
- **Induction Step:** Assume from the induction hypothesis that $w_i = u_i$. Then since E' has freshly printed u_i, the variable w = $u_i = w_i$. Then from the construction of L', w_{i+1} is the first string after w_i which is lexicographically greater than w_i. In other words, no string w' between w_i and w_{i+1} in the enumeration sequence of E is greater then w_i. Now, consider the construction of E' from E. After printing u_i, E' doesn't print anything (i.e. ignores) any string in the enumeration sequence of E after u_i which is smaller then u_i. In other words, the next string it prints i.e. u_{i+1} is the first string after u_i which is strictly greater than u_i. Thus, from above descriptions, $w_{i+1} = u_{i+1}$.

So, $u_i = w_i$ $\forall$ i $\geq$ 1, which means that E' exactly recognizes L'.
From above proof, E' print order is exactly same as L' string order. So, E' prints the strings of L' in a lexicographical order. We now prove our final claim:

Claim 1.3: If an enumerator enumerates a language in lexicographical order, then the language is decidable.
Proof of Claim 1.3
Given an enumerator E for L which prints in lexicographical order, we need to construct a Turing Machine M which decides L. We consider the case when L is infinite since if L is finite, then it is trivially decidable (since we can just hard-code the Turing Machine to manually accept the finite amount of strings). We construct such a Turing Machine M as follows:

1. On some input w, M uses E to start enumerating all the strings in lexicographical order.
2. After every string s is printed by E, it is compared with w. Then one of the following cases occur
   (a) s < w. In this case, ignore s and go back to step 2
   (b) s = w. Accept w.
   (c) s > w. Reject w.

From point 2c, it directly follows that M accept w $\iff$ w $\in$ E and so w $\in$ L. Now, all we need to prove is that M halts on all strings not in L. For any string w, $\exists$ some string w' in E such that w' > w or else L will be finite as discussed in claim 1.1. So, w will be rejected. Thus, M halts on all inputs. So, M decides L.

Thus, using claim 1.3 and the fact that E' prints in lexicographical order, L' is decidable. Hence Proved.

2 Question 2

Show that single-tape TMs that cannot write on the portion of the tape containing the input string recognize only regular languages.

Solution:
The proof of this question is similar to the proof of the statement that "2DFAs recognize the class of regular languages". However, in this case, instead of trying to partition the input string, we will partition the tape of the Turing Machine into the input and non-input portion.

Let the given Turing Machine M(Q, Σ, Γ, δ, q_0, q_a, q_r) where q_a is the accept state while q_r is the reject state. By convention, q_a, $q_r \notin$ Q. Now, since we need to prove that the class of languages accepted by this machine is regular, we will try to define a Myhill-Nerode relation. For that, first define a function $T_x : Q \cup \{\bowtie\} \to Q \cup \{q_a, q_r, \perp\}$ for a given input string x. The function is defined as:

- $T_x(q) = q'$, q' $\in$ Q. This means that if the machine enters the input portion of the tape from the non-input portion (from the right) into state q, then it leaves the input portion and enters the non-input portion in state q'.
- $T_x(q) = q_a$. This means that if the machine enters the input portion of the tape from the non-input portion into state q, then it never leaves the input portion and gets accepted somewhere in between.
- $T_x(q) = q_r$. This means that if the machine enters the input portion of the tape from the non-input portion into state q, then it never leaves the input portion and gets rejected somewhere in between.
- $T_x(q) = \perp$. This means that if the machine enters the input portion of the tape from the non-input portion into state q, then it never leaves the input portion and keeps looping inside the input part.
- $T_x(\bowtie) = q'$, q' $\in$ Q. This means that the first time the machine emerges into the non-input part, it will land in the state q'.
- $T_x(\bowtie) = q_a$. This means that on reading string x, the machine never leaves the input portion and x is accepted by the language.
- $T_x(\bowtie) = q_r$. This means that on reading string x, the machine never leaves the input portion and x is rejected by the language.
- $T_x(\bowtie) = \perp$. This means that on reading string x, the machine never leaves the input portion and keeps looping inside the input portion without ever getting outside the input part.

All the above functions are well defined since the machine never writes on the input portion of the tape which means that throughout the simulation of the machine on any input, the contents of the input part will remain unchanged and therefore the above functions are valid for the entire duration of the simulation. This is precisely the part where we exploit the given property of the Turing Machine M.

Now, using the function T_x, define the relation $\sim$ on the strings of Σ^* as follows:

$$\text{x} \sim \text{y} \iff T_x = T_y$$

We now prove that $\sim$ is a Myhill-Nerode relation for L by proving its three properties separately:

- **Finite Index:** We need to prove that $\sim$ has finite number of equivalence classes. The total number of distinct functions T_x is $(|Q|+3)^{(|Q|+1)}$, which will also be equal to the maximum number of equivalence classes of $\sim$. Since this quantity is finite, number of equivalence classes will also be finite.

- **Refinement:** We need to prove that:

$$\text{x} \sim \text{y} \implies (\text{x} \in \text{L} \iff \text{y} \in \text{L})$$

  Assume that x $\sim$ y. Then, $T_x(q) = T_y(q)$ $\forall$ q $\in Q \cup \{\bowtie\}$. If we prove that x $\in$ L $\implies$ y $\in$ L, then we are done since the other side is symmetrical. Since x $\in$ L, $\exists$ q $\in Q \cup \{\bowtie\}$ such that $T_x(q) = q_a$. Then since $T_x = T_y$, $T_y(q) = q_a$. So, y $\in$ L. As mentioned the reverse direction is symmetrical. Hence Proved.

- **Right Congruence:** We need to prove that:

$$\text{x} \sim \text{y} \implies (\forall \text{z}, \text{xz} \sim \text{yz})$$

  Assume that x $\sim$ y. Then, $T_x(q) = T_y(q)$ $\forall$ q $\in Q \cup \{\bowtie\}$. Consider the strings xz and yz for some string z. We need to prove that $T_{xz} = T_{yz}$. We do this proof by cases on the state q on which the function T is to applied.

  - **Case 1:** When q $\in$ Q. Let's say that the machine enters the string z from the non-input portion in state q. If the machine never leaves the string z then the machine would emerge out in the same state in both the strings xz and yz. So, $T_{xz}(q) = T_{yz}(q)$.

    On the other hand, if the machine emerges out of z and into x in case of xz and into y in case of yz. Then since the machine passes through the same string z up till now, it must also enter x and y in same state, say q'. Then since $T_x(q') = T_y(q')$, the machine will once again emerge into z in the same state.

    The logic can be extended further using the above two cases repeatedly. In every case, the state of the machine when scanning xz and yz will be same when the machine is currently in the z part. And the machine leaves the input part from z only. So, if the machine M emerges out into the non-input part again, it will do so in same states in case of xz and yz. So, $T_{xz}(q) = T_{yz}(q)$ $\forall$ $q \in Q$.

  - **Case 2:** When q is $\bowtie$. This case is invoked when the machine never leaves the part xz or yz. If the machine never leaves x or y, then the string z will never be read, in which case, $T_{xz}(\bowtie) = T_x(\bowtie) = T_y(\bowtie) = T_{yz}(\bowtie)$.

    But, if the machine leaves x in xz, then it should also leave y in yz. In that case, they will enter z for the first time in the same state, q' = $T_x(\bowtie) = T_y(\bowtie)$. From then, the machine simulation will be the same regardless of whether it is simulating xz or yz, since it is reading the same string z (and starting from the same state) in both cases. Because of this, if the machine accepts/rejects while being in z, then it will do the same in xz as well as yz. But, if it again enters the part x in xz and part y in yz, then also since x $\sim$ y, the final accept/reject/looping state will be same. So, $T_{xz}(\bowtie) = T_{yz}(\bowtie)$.

  So, from all cases, $T_{xz}(q) = T_{yz}(q)$. Thus, $\forall$z, xz $\sim$ yz.

Thus, $\sim$ follows all three properties. So, $\sim$ is a Myhill-Nerode relation for the language L (recognized by the given machine M) on Σ^*. So, L is a regular languauge. Hence Proved.

3 Question 3

For a language C, we show that language C is Turing-recognizable iff a decidable language D exists such that

$$C = \{x | \exists x(< x, y > \in D)\}$$

by showing the implications in both the directions.

$\Rightarrow$
If C is Turing recognizable, then there is a decidable language D that satisfies the above condition.
Proof: Let T be a Turing machine that recognizes the given language C. Then we will construct another Turing machine T' that decides a language D such that $x \in C$ iff $< x, y > \in D$.
The Turing machine T' will operate using the machine T as follows:

On input $< x, y >$:

1. T' first checks if y is of form $c_0\#c_1\#c_2...\#c_n$. If y is not of this form, it rejects the input.
2. If y satisfies point 1 above, then check if c_0 is a start configuration for string x in Turing machine T. If it is not so, reject the input.
3. For i$\leftarrow$0 to n-1:
   (a) Check if configuration c_i can be obtained from configuration c_{i-1} on the string x using the transition function of Turing machine T. This can be checked by considering all the next possible configurations from configuration c_{i-1} and comparing them with c_i.
   (b) If the above point is not satisfied, reject the input.
4. Check if c_n is an accepting configuration for string x in Turing machine T. If it is not so, reject the input.
5. If all the above steps are satisfied, accept the input $< x, y >$.

Turing machine T' defined above is decidable as it either accepts or rejects any input. Note that for any $x \in C$, there will be sequence of configurations that T will go through to accept x. Let that sequence of configurations be $c_0, c_1, c_2, ..., c_n$. Combine all these configurations into a string as $c_0\#c_1\#c_2...\#c_n$. Now, T' will accept the input $< x, c_0\#c_1\#c_2...\#c_n >$.
Similarly, if T' accept any input $< x, y >$, then y will be of form $c_0\#c_1\#c_2...\#c_n$ and $c_0, c_1, c_2, ..., c_n$ will be a sequence of configurations that T will go through to accept the input x (due to the definition of T').

$\Leftarrow$
If there is a decidable language D that accepts strings of form $< x, y >$, then there is a recognizable language C that satisfies

$$C = \{x | \exists y (< x, y > \in D)\}$$

Proof: Assume T' is the Turing machine that decides language D. Since D is decidable language, T' will either accept or reject any input $< x, y >$. Now, to show that C is recognizable, we define a Turing machine T that will recognize C. The idea is that for any input string x, T will try to simulate T' on all strings of form $< x, y >$ by enumerating $y \in \Sigma^*$ in lexicographical order.
We can define T as follows:
On any input x,

1. For $y \in \Sigma^*$ in lexicographical order:
   (a) Simulate T' on input $< x, y >$, if T' accepts, then accept.

Now, since D is decidable, if for input x, there is a $< x, y > \in D$, then y will eventually occur in the lexicographical order of all the strings and the input x will be accepted by T. Thus, C is recognizable by Turing machine T.

4 Question 4

Language formulation for this problem:

$$L = \{< G, A > \mid \text{G is description of a CFG, A is a usable variable in G}\}$$

To show that this language is decidable, we construct a Turing machine T that decides L. The idea is to check if the variable A is reachable from the start variable S i.e., $S \rightarrow^* \gamma A \lambda$ such that γ, A and λ all derive some terminal string.

The description of the Turing machine T is as follows:

On input $< G, Z >$:

1. Check that G is a correct description for a CFG and A is a variable in the grammar G. If it is not so, reject the input.

2. Now, we find the set **Terminable** that contain all the variables A that derive some terminal string $z \in \Sigma^*$ i.e., $A \rightarrow^* z$. Initialize the set **Terminable** with all the terminal variables in G.
   While the set **Terminable** keeps changing:
   For rule R in G:
   If all the variables in the right side of R are in set **Terminable**:
   Include the variable in the left side of R in the set **Terminable**.
   Finally, it will contain all the variables A that derive some terminal string $z \in \Sigma^*$ i.e., $A \rightarrow^* z$.

3. Now, we will find the variables that are usable. We start with empty set **Usable** = {}. Include S, start variable in **Usable** if S is in **Terminable**.
   While **Usable** keeps changing:
   For rule R in G:
   If all the variables in the right side of R are in set **Terminable** and variable on the left side is already in set **Usable**:
   Include all the variable in the right side of R in the set **Usable**.

   Note: The above two step can be done on the tape of the Turing machine. For every variable A in CFG G, we have both A and A' in the tape variables. Here, A is original variable in the CFG and A' shows that it has been included in the set **Terminable** or **Usable** as required. Initially, we can use some part of the tape to maintain the set that contains all the original variables of the grammar. Now, replace X with X' on the tape to include it in the set. Keep traversing the portion of the tape that contains the set and keep changing variables as long as the set keeps changing.

4. Check if Z is present in the set **Usable**. If it is present, accept the input. Otherwise, reject.

Note, that the turing machine T defined above will always accept or reject any input $< G, A >$. This is because, steps 2 and 3 will always terminate after finite number of iterations since the number of variables in G is finite. Also, the step 4 will check if the variable A is present in the set and accept or reject the input accordingly.

Claim: The Turing machine T defined above decides language L where,

$$L = \{< G, A > \mid \text{G is description of a CFG, A is a usable variable in G}\}$$

Proof: Note that finding the sets **Terminable** and **Usable** takes only finite amount of steps since the number of variables and rules in any CFG is finite. Also, checking if a variable is present in the set **Usable** will also take finite number of steps. Thus, the machine T will never go in infinite loop. It accepts $< G, A >$ iff $S \rightarrow^* \gamma A \lambda$ such that γ, A and λ all derive some terminal string.

5 Question 5

Language formulation for this problem:
$L = \{< M, w > |$TM M tries to move its head left when its head is on the left-most tape cell atleast once on input w$\}$

We will show that this language is undecidable by contradiction. Let us assume that L is decidable. Then, there exist a TM T that decides L. We will now construct a Turing machine A that decides A_{TM} ($A_{TM} = \{< M, w > |$ M is a TM and M accepts w$\}$).

Description of TM A is as follows:
A = " On input $< M, w >$:

1. Construct the following TM N:
   N = " On input x:
   (a) Append # before string x to get #x.
   (b) When head pointer is at # and the current configuration is not accepting, then move the pointer towards right.
   (c) When the current state is an accepting state, then keep moving the head pointer towards left until you reach # and then again move left trying to move left from the leftmost cell on the tape.
   (d) At all other positions of the head pointer, use transition function of M to simulate M's actions on string x. "
2. Use TM T on input $< N, w >$. If T accepts, then accept. Otherwise, reject."

Claim: TM A as defined above can decide A_{TM} if T decides L.
Proof: TM A accepts $< M, w >$ iff T accepts $< N, w >$ where N is a Turing machine as defined in point 1 above. Note that TM N just append # before string w and simulates M when head pointer is not at #, otherwise moves left from # only when it is in an accepting state. Thus, TM T accepts $< N, w >$ iff M accepts w. Therefore, TM A accepts $< M, w >$ iff M accepts w. Otherwise, it rejects.
Thus, TM A as defined above can decide A_{TM} if T decides L.

But, we know that A_{TM} is undecidable. Thus, our assumption that language L, as defined above, is decidable cannot be true. Therefore, we proved by contradiction that the given problem is undecidable.

6 Question 6

We can show that if a Turing machine M moves right for more than $|Q| + |w|$ steps on a string w, then it will never move left (Q are the states of M). Hence we only need to check any attempted left moves in the first $|Q| + |w| + 1$ steps. Using this result, a decidable Turing machine can be constructed as follows: $H =$ "On input $< M, w >$:

1. Get $|Q|$ from the encoding of M.
2. Simulate M on w for $|Q| + |w| + 1$ steps.
3. If M ever attempted to move left, accept. Otherwise, reject".

H is decidable as it terminates on all input strings and accepts all the $< M, w >$ that attempts to move left.

Claim: If a Turing machine moves right for more than $|Q| + |w|$ steps on a string w, then it will never attempt to move left.
Proof: If the machine moves right for $|w|$ steps, it will see the empty tape alphabet $\sqcup$. If it moves right for further $|Q| + 1$ steps, it is guaranteed to read $\sqcup$ for all the steps. Further, by pigeonhole principle, some state is bound to repeat in the $|Q| + 1$ steps. The sequence between the two repeated states keeps on repeating and the machine moves right on each of them on reading $\sqcup$. Hence the machine will never attempt to move left. ■

This completes the proof.

7 Question 7

Assume by contradiction, that AMB_{CFG} is decidable. Then we can define a decidable for PCP problem in the following way:

Given a PCP $P = \{[\frac{t_1}{b_1}], [\frac{t_2}{b_2}], ..., [\frac{t_k}{b_k}]\}$ define a CFG G_P as follows:

1. $S \to T|B$
2. $T \to t_1Ta_1|t_2Ta_2|...|t_kTa_k|t_1a_1|t_2a_2|...|t_ka_k$
3. $B \to b_1Ta_1|b_2Ta_2|...|b_kTa_k|b_1a_1|b_2a_2|...|b_ka_k$

Here all a_is are distinct characters.
Claim: P has a solution iff G_P is ambiguous.
Proof: $\Rightarrow$ If P has a solution then G_P is ambiguous. Let $i_1i_2...i_m$ be the sequence of indices such that $t_{i_1}...t_{i_m} = b_{i_1}..b_{i_m}$. Then the string $S = t_{i_1}...t_{i_m}a_{i_m}..a_{i_1} = b_{i_1}...b_{i_m}a_{i_m}..a_{i_1}$ has two leftmost derivation:

1. $S \to T \to^* t_{i_1}...t_{i_k}Ta_{i_k}..a_{i_1} \to^* t_{i_1}...t_{i_m}a_{i_m}..a_{i_1}$
2. $S \to B \to^* b_{i_1}...b_{i_k}Ba_{i_k}..a_{i_1} \to^* b_{i_1}...b_{i_m}a_{i_m}..a_{i_1}$

$\Leftarrow$ If G_P is ambiguous, then P has a solution. Each derivation of G_P generates string of the form $t_{i_1}...t_{i_m}a_{i_m}..a_{i_1}$ or $b_{i_1}...b_{i_m}a_{i_m}..a_{i_1}$. For different orderings $i_1..i_m \neq j_1..j_m$, the strings $t_{i_1}...t_{i_m}a_{i_m}..a_{i_1} \neq t_{j_1}...t_{j_m}a_{j_m}..a_{j_1}$ as the string of last m characters will be different (All a_is are distinct characters). Similar reasoning holds for strings generated through B. The only way two derived strings can be similar is if there exists a ordering $i_1..i_m$ such that $t_{i_1}...t_{i_m}a_{i_m}..a_{i_1} = b_{i_1}...b_{i_m}a_{i_m}..a_{i_1}$. But this means that $t_{i_1}...t_{i_m} = b_{i_1}..b_{i_m}$, which means P has a solution. ■

Using this claim, we can construct a decidable Turing machine for PCP problem. Given a decidable TM H for AMB_{CFG}, construct TM $A =$ "On input $< P >$:

1. Generate grammar G_P.
2. Run H on G_P. If it accepts then accept, otherwise reject".

The existence of such A is a contradiction to the fact that PCP is undecidable. Hence AMB_{CFG} cannot be decidable.

Disclaimer: The grammar G_P was inspired from the hint given in the same question in the Sipser book.

8 Question 8

The following claim can be used to construct a decidable TM for SPCP.
Claim: A SPCP problem P has a solution iff there exists a block of the form $[\frac{a}{b}]$ where $a = b$.
Proof: $\Rightarrow$ If there exists a block of the form $[\frac{a}{b}]$ where $a = b$, then the string $a = b$ is trivially a solution. $\Leftarrow$ Let there be a solution to P. Assume by contradiction there is no block $[\frac{a}{b}]$ where $a = b$. Let $i_1...i_m$ be a ordering such that $a_{i_1}...a_{i_m} = b_{i_1}...b_{i_m}$. Since $a_{i_1} \neq b_{i_1}$, the first $|a_{i_1}| = |b_{i_1}|$ are different in the given string. Hence the string cannot be equal, which gives the required contradiction. ■

Based on the above claim, we can construct a decidable TM A as follows: "On input P:

1. If there exists a block of form $[\frac{a}{b}]$ where $a = b$ in P, then accept. Otherwise reject."