Abhinav Jain, 2019CS10322
Arpit Chauhan, 2019CS10332
Himanshu Gaurav Singh, 2019CS10358

Problem 1
Show that every infinite Turing-recognizable language has an infinite decidable subset.

Solution:
First we will prove the following lemma :
Lemma : If for an infinite language L there is an enumerator E that prints all strings in a lexicographic order than L is decidable.
Proof : Using E we construct a turing machine M which decides L as follows :
M = "On input x:

1. Simulate the enumerator E.
2. For each string (say s) output by the enumerator, compare s with x.
3. If s is equal to x than accept. If s is lexicographically larger than x than reject and halt. Otherwise generate the next string by simulating E. "

Claim : $L(M) = L(E)$
Proof : First observe that the $L(M)$ is a subset of $L(E)$, as D accepts the input string x only when the same string x is generated by E. Now observe that M will reject the string x only when $x \notin L(E)$. Suppose x is rejected by M. This would have happened when a a string lexicographically larger than x is printed by E. Since E prints strings in lexicographical order we can be sure that x does not belong to E, otherwise E would have already generated x and M would have accepted it. Hence $L(M) = L(E)$.

Claim : On any input string x, M halts.
Proof : Since L is an infinite language there exists a string $y \in L$ such that $y > x$ (lexicographically greater). By the construction of M it must halt by the time E enumerates y. Hence on all input strings x, M halts.

Hence using the above two claims, we can say that M decides L. Hence the lemma is proven.

Now we will use the above lemma to show that every infinite Turing-recognizable language S has an infinite decidable subset. Let E_S be an enumerator which enumerates S. Now we construct an enumerator E'_S which will enumerate an infinite lexicographically ordered sub-sequence of strings enumerated by E_S. Hence from the above lemma it follow that the $L(E'_S)$ is decidable.
E'_S =

1. Simulate the enumerator E_S. Print the first string simulated by E_S (say s_1) and store it i.e let $prev = s_1$.
2. Simulate E until E prints a string (say s_i) such that $s_i > prev$.
3. Print s_i and store s_i in prev. Go to step 2.

Let E_S enumerate the strings in S in the sequence $s_1, s_2, s_3, \ldots$. Observe that E'_S will precisely enumerate the infinite set $S' = \{s_i \mid s_i > s_j \; \forall j < i\}$ in a lexicographic order. Hence from the above lemma we conclude that S' is an infinite decidable subset of S.

Problem 2
Show that single-tape TMs that cannot write on the portion of the tape containing the input string recognize only regular languages.

Solution: Let $M = (Q, \Sigma, \Delta, \delta, q_0, q_{acc}, q_{rej})$ be a single-tape TMs that cannot write on the portion of the tape containing the input string. Consider the run of M on an arbritrary input string x. We define left portion of input tape as the portion containing the input string and right portion of input tape as the portion of tape to the right of input string i.e where the TM can write. Proceeding similar to the proof of equivalence of 2-DFA and regular DFA we define a function $T_x : (Q \cup \{\star\}) \mapsto (Q \cup q_{acc}, q_{rej})$ such that

- $T_x(\star)$ is the state that M is in when it just enters the right portion of tape for the first time. If M never enters the right portion of tape, then define $T_x(\star) = q_{acc}$ if x is accepted by M, otherwise $T_x(\star) = q_{rej}$.
- Consider any $q \in Q$, suppose M starts by moving to the left portion of tape from the state q, then $T_x(q) = p$ when p is the state that M enters the right portion of the input tape for the first. If M never enters the right portion of tape, then define $T_x(q) = q_{acc}$ if x is accepted by M, otherwise $T_x(q) = q_{rej}$.

Claim : Consider any two strings x and y such that $T_x = T_y$. Then for all strings $z \in \Sigma^*$ we have (M accepts xz) $\Leftrightarrow$ (M accepts yz).
Proof : Observe that table T_x (T_y) and z completely determine the sequence of states that the machine M is in as it passes the boundary between x and z (between y and z) in either direction. Now if the machine M accepts xz then at some point it must enter the state q_{acc}. Since the sequence of states along the boundary is same and the actions taken by the machine while scanning z are exactly same, hence machine M must enter the state q_{acc} for yz as well. (and vice versa).

Now we will use the Myhill-Nerode theorem to show that $L(M)$ is regular. Observe that
$T_x = T_y \implies \forall z$ (M *accepts* $xz \Leftrightarrow$ M *accepts* yz) $\Leftrightarrow \forall z$ ($xz \in L(M) \Leftrightarrow$ $yz \in L(M)$).
Since we have $x \equiv_{L(M)} y \Leftrightarrow \forall z$ ($xz \in L(M) \Leftrightarrow$ $yz \in L(M)$). Hence for any two strings x and y for which $T_x = T_y$ then they both are in same equivalence class under the relation $\equiv_{L(M)}$. Observe that the number of distinct functions T is bounded by $(|Q| + 1)^{(|Q|+1)}$, hence the relation $\equiv_{L(M)}$ is of finite index. Therefor by Myhill-Nerode theorem $L(M)$ is regular language.

Problem 3
Let C be a language. Prove that C is Turing-recognizable iff a decidable language D exists such that

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

Solution:
Suppose that C is turing recognisable. Then there exist a turing machine M_C which recognises C. For a string $x \in C$, let $steps_M(x)$ denote the number of steps taken by M_C to accept x, then define D as the language consisting of encoding of x and $steps_{MC}(x)$. Formally $D = \{< x, 1^z > \mid x \in C \text{ and } z \geq steps_{MC}(x)\}$. Now we will construct a turing machine M_D which decides D.
M_D = "On input $< x, y >$:

1. Take input. Reject if input is not a valid encoding of form $< x, 1^z >$.
2. Simulates M_C for at most z steps.
3. It accepts $< x, y >$ if and only if M_C accepts x (in at most z steps)."

Observe that for any $x \in C$ there will be a $y(= 1^z)$ such that M_C recognises x within $|y|(= z)$ steps. If no such y exists that M_C doesn't recognise x. Hence $x \in C$ iff there a y such that $< x, y >$ belongs to D.

Now suppose that such a decidable language D exists and let M_D be the turing machine which decides it. Using M_D we construct a turing machine M_C which recognises C.
M_C = "On input x:

1. Simulate M_D on $< x, y >$ for all possible strings y of form 1^z.
2. If M_D accepts any such encoding $< x, y >$, stop and accept x."

Observe that for each $x \in C$, a y will exist such that $< x, y >$ is in D. Therefor M_C recognises C as it take only a finite number of steps to find such y.

Problem 4
Say that a variable A in CFL G is usable if it appears in some derivation of some string w ∈ G. Given a CFG G and a variable A, consider the problem of testing whether A is usable. Formulate this problem as a language and show that it is decidable.

Solution:
We can consider encoding of the $CFG\ G$ and non-terminal A to formulate the problem as a language. The formulation will be as follows:

$$ENC_{A,G} = \{< G, A >: S \xrightarrow{*} \alpha A \gamma, \alpha, A, \gamma \textit{ reduce to some string } G =< N, T, P, S >.$$

We will prove that $ENC_{A,G}$ is decidable.
We first mark all non-terminals that generate some string via derivations. This can be done using the turing machine T_1 with the following high level description.

1. With input as $< G, A >$. Mark all the terminals.
2. Till in some iteration, no variable is marked:
   - For any production rule $A \to B_1 B_2 .. B_k$, mark A if $B_1 B_2 .. B_k = \epsilon$ or all of $B_1, B_2, .. B_k$ are marked.
3. Reject if A is not marked. Accept otherwise.

Observe that T_1 halts on all inputs, since the number of iteration of step 2 is bounded by the number of non-terminals in G. In this way, we can mark all variables in G that generate some string. In the next step, we check if there is some derivation $S \to X_1 X_2 .. A .. X_k$ such that all X_i are either terminals or non-terminals that generate some string.
The whole process can be carried out using a turing machine T with the following high-level description.

- Take input $< G, A >$. Run T_1 on the input to get the marked non-terminals and reject if T_1 rejects.
- Mark(differently than T_1) A. While there is no change in the set of marked symbols
  - If $X \to X_1 X_2 ... X_k$, mark X if all the symbols on the right are either marked by T_1 or are terminals and at-least one marked(by T) non-terminal is present on the right hand side.
- Accept is S is marked, else reject.

Observe that the T halts on all inputs since T_1 halts and the number of iterations of step 2 is bounded by the number of non-terminals in G. Also, S is marked iff there exist $S \xrightarrow{*} \alpha A \gamma$ and α, A, γ reduce to some string. Thus, there exists a turing machine that decides $ENC_{G,A}$. Hence, it is decidable.

Problem 5
Consider the problem of determining whether a Turing machine M on an input w ever attempts to move its head left when its head is on the left-most tape cell. Formulate this problem as a language and show that it is undecidable.

Solution: Let

$L_{TM} = \{\langle M, w\rangle \mid On\ input\ w,\ M\ ever\ attempts\ to\ move\ its\ head\ left\ when\ its\ on\ the\ left{-}most\ tape\ cell\}$

The proof is by contradiction. We assume that L_{TM} is decidable and use this assumption to show that A_{TM} is decidable, which will be a contradiction(Theorem 4.11, Sipser). Let R be the TM that decides L_{TM}
Consider the following Turing machine,
S = "On input $\langle M, w\rangle$, where M is a TM and w is a string:

1. Construct the following TM M_2.
   M_2 = "On input x:
   (a) Shift the input x one cell to the right and put a special symbol $'\#'$ at the left-most position of the tape.
   (b) Now, M_2 simulates M on the input x. If M_2 ever moves to $'\#'$ before M reaching an accept state, we force it to go back to the previous tape symbol by adding an extra right move. Hence, if M attempts a left move from the left-most input symbol before accepting the string, M_2 is prevented from moving to the special symbol at the left-most tape cell.
   (c) If M accepts string x, we move M_2 to the left-most tape cell and then attempt to move further to the left."
2. Run R on input $\langle M_2, w\rangle$
3. If R accepts, accept; if R rejects, reject."

Claim: TM S decides A_{TM}.
Proof: Observe that the only way in which TM M_2 can reach the special symbol $'\#'$ and attempt to move left of the left-most tape cell is when the TM M accepts the string w. Before the string w is accepted by TM M, M_2 is never allowed to stay at the left-most tape symbol($'\#'$), and hence, it can never attempt to move its head left from the left-most tape cell. Thus if TM R accepts the input $\langle M_2, w\rangle$, TM M accepts the input string w, otherwise rejects. Hence, TM S decides A_{TM}.

Thus, we have reached a contradiction that A_{TM} is decidable and hence our assumption of L_{TM} being decidable is false. Therefore, L_{TM} is undecidable.

Problem 6
Consider the problem of determining whether a Turing machine M on an input w ever attempts to move its head left at any point during its computation on w. Formulate this problem as a language and show that it is decidable.

Solution: Let

$$L_{TM} = \{\langle M, w\rangle \mid \textit{On input } w,\ \textit{M attempts to move its head left at any point}\}$$

We know that a language is decidable if and only if some Turing machine decides it. Consider the following Turing Machine,

S = "On input $\langle M, w\rangle$, where M is a TM and w is a string:

1. Let Q be the set of states of Turing Machine M. Simulate M on w for $|Q| + |w| + 1$ steps.
2. If M attempts to move its head left at any time during these finite number of steps, accept $\langle M, w\rangle$, otherwise reject."

Claim: TM S decides L_{TM}.
Proof: If M never attempts to move its head left at any point during its computation on w, then it will also not have any left move in the first $|Q| + |w| + 1$ steps and thus, will be rightly rejected by S.
In case M does move its head left at some point during the computation of w, let R be the shortest sequence of states till the first left move. Since R is shortest, at any step the Machine M either reads an input character(|w| in number) and changes state(possibly), or reads a blank character and changes state(|Q| in number). Hence, after reading all the |w| characters and then going to all |Q| states on blank spaces (in the worst case), the machine must make a left move on the next step because R is the shortest sequence of states and thus does not involve any redundant cycles. Hence, if M does have to make a left move while computing on w, it will do so in the first $|Q| + |w| + 1$ steps and get accepted by S, otherwise if it were to loop forever while moving right, S rejects it because it does not find a left move in the first $|Q| + |w| + 1$ states. Infact we can say that if the machine does not make a left move in the first $|Q| + |w| + 1$ steps, then it must be in some cycle and will continue to move right forever. Hence, TM S decides language L_{TM}.

Since the above claim proves that TM S decides L_{TM}, we can say that Language L_{TM} is decidable.

Problem 7
Let $AMB_{CFG} = \{\langle G\rangle | G \text{ is an ambiguous } CFG\}$. *Show that* AMB_{CFG} *is undecidable via a reduction from PCP.*

Solution: We will create a CFG G using the structure of the PCP problem such that G will be ambiguous iff a correspondence matching exists in PCP .
Consider the dominoes in PCP be $D_1, D_2, ..D_k$ where $D_i = \alpha_i|\beta_i$. We will construct G such that the corresponding equal string generated in PCP will have two leftmost derivations in G. Consider $G =< \{S, S_1, S_2\} \cup \{D_1, D_2, ..D_k\}, \Sigma, P, S >$ where Σ is the alphabet of $\{\alpha_i, \beta_i\}, i \in \{1, 2, ..k\}$ along with symbols $\{1', 2', ..k'\}$(such that these symbols are not used by αs and βs.) corresponding to each domino in PCP. The production rules P are as follows:

$$S \to S_1|S_2$$
$$S_1 \to i'S_1\alpha_i|i'\alpha_i \ \forall \ i \in \{1, 2..k\}$$
$$S_2 \to i'S_2\beta_i|i'\beta_i \ \forall \ i \in \{1, 2..k\}$$

Lemma 1. *A matching exists in* $PCP \iff$ *G is ambiguous.*

Proof. Observe that if there exists a matching in PCP corresponding to the sequence of dominoes say $D_{i_1}, D_{i_2}, ..D_{i_z}$. Then, the string $s = i'_1i'_2..i'_z\alpha_{i_z}\alpha_{i_{z-1}}\alpha_{i_{z_2}}..\alpha_{i_1}$. has two different leftmost derivations as follows:

$$S \to S_1 \to i'_1S_1\alpha_{i_1} \to i'_1i'_2S_1\alpha_{i_1}\alpha_2... \to i'_1i'_2..i'_z\alpha_{i_z}\alpha_{i_{z-1}}\alpha_{i_{z_2}}..\alpha_{i_1}$$
$$S \to S_2 \to i'_1S_2\beta_{i_1} \to i'_1i'_2S_1\beta_{i_1}\beta_{i_2}... \to i'_1i'_2..i'_z\beta_{i_z}\beta_{i_{z-1}}\beta_{i_{z-2}}..\beta_{i_1}$$

Obsrve that the final strings generated are same in both the derivations because $\alpha_{i_z}\alpha_{i_{z-1}}\alpha_{i_{z_2}}..\alpha_{i_1} = \beta_{i_{z-1}}\beta_{i_{z_2}}..\beta_{i_1}$ since $D_{i_1}, D_{i_2}, ..D_{i_z}$ provides a solution to PCP.

To prove the converse, observe that any string generated by S will be of the form $i_zi_{z-1}i_{z-2}..i_1\alpha_1\alpha_2..\alpha_z$(if the first derivation is $S \to S1$) or of the form $i_zi_{z-1}i_{z-2}..i_1\beta_1\beta_2..z$(if the first derivation is $S \to S_2$).

To prove the converse, assume that G is ambiguous, this implies that some string s has two different leftmost derivations. Consider any two distinct leftmost derivations in G that are same at the first step($S \to S_1$ or $S \to S_2$), then the strings generated by them must be different. To prove this, assume otherwise. Suppose that the derivations are different first time at the $i(\neq 1)$-th step, then the $(i-1)$-th character in the generated strings will be different(x' in first, y' in other where the step $..S_1.. \to ...x'S_1\alpha_{x'}$ and $..S_1.. \to ...y'S_1\beta_{y'}(x \neq y)$).
Thus, the derivations of s must differ in the first step(one $S \to S_1$ and other $S \to S_2$). Consider the i-th step of any two such derivations. Observe that the production rule used for S_1 in the first derivation must correspond to the rule used for S_2 in the other derivation because the i-th character has to be the same(the superscripted integer denoting the particular domino). Thus, the leftmost derivations must be of the form

$$S \to S_1 \to i_zS_1\alpha_{i_z}... \to i_zi_{z-1}..i_2i_1\alpha_1\alpha_2..\alpha_z$$
$$S \to S_2 \to i_zS_2\beta_{i_z}... \to i_zi_{z-1}..i_2i_1\beta_1\beta_2..\beta_z$$

Since the string generated by both the derivations are equal, $\alpha_1\alpha_2..\alpha_z = \beta_1\beta_2..\beta_z$. This implies $D_1, D_2, ..D_z$ are a solution to the Post's Correspondence Problem. □

From the above claim, given a turing machine R to decide whether a CFG is ambiguous, we will construct a turing machine S to decide PCP as follows:

1. Given a description of PCP, $< D_1, D_2, ..D_z >$.
2. Construct G as defined in the construction above. Observe that the steps in the construction are all computable in finite steps.
3. Give the description $< G >$ of G as an input to R.

4. Accept if R accepts, reject if R rejects.

The above constructed turing machine S decides PCP. However, PCP is undecidable. Hence, AMB_{CFG} is undecidable.

Problem 8
In the Silly Post Correspondence Problem (SPCP), the top string in each pair has the same length as the bottom string. Show that the SPCP is decidable.

Solution: We claim that there exists a solution to $SPCP$ iff atleast one domino has exactly the same string in the upper and lower half.

Lemma 2. *$SPCP$ for the dominoes $D_1, D_2, .., D_k$ has a solution iff for some $D_j(= \alpha_j|\beta_j)$ $\alpha_j = \beta_j$*

Proof. Suppose there exists a solution to $SPCP$, say $D_{i_1}, D_{i_2}, , D_{i_z}$. Consider the first domino D_{i_1} in the sequence. Since the lengths of the upper and bottom string in it is same, both the strings must be same for the final string to be same.
To prove the converse, suppose there exist some D_j such that $\alpha_j = \beta_j$. The domino D_j alone forms the solution sequence in this case. □

From the above claim, we can conclude that computability of $SPCP$ is equivalent to checking whether some pair of strings in a list of pairs is same. This is certainly computable. We can design a turing machine T with the following high level description:

1. Take input. Reject if input is not a valid encoding of some $< D_1, D_2, ..D_k >$ for SPCP.
2. For each D_i, check if the corresponding α_i, β_i are same. If yes for some i, then accept.
3. If α_i, β_i are different for all i, then reject.

Using the claim proved above, we can conclude that the turing machine as described above accepts exactly the instances of SPCP for each a solution exists and rejects all others. Also, observe that T halts since all the steps in the description get completed in finite steps. Hence, $SPCP$ is decidable.