Problem 1

Show that the following languages are decidable.

(a) \( L_1 = \{ \langle R \rangle : R \text{ generates at least one string } w \text{ that has 111 as a substring} \} \).
   Here, \( R \) is a regular expression over alphabet \{0, 1\}.

(b) \( L_2 = \{ \langle G, k \rangle : G \text{ generate some string } 1^n \text{ where } n \leq k \} \).
   Here, \( G \) is a context free grammar over alphabet \{1\}.

(c) (Extra credit) \( L_3 = \{ \langle G, k \rangle : G \text{ generate some string } 1^n \text{ where } n \geq k \} \).
   Here, \( G \) is a context free grammar over alphabet \{1\}.

Solution

(a) Let \( L \) be the language of regular expression \((0 + 1)^*111(0 + 1)^*\). We can know \( R \in L_1 \) if only if \( L(R) \cap L \neq \emptyset \). So following TM \( A_1 \) can decide \( L_1 \),

\[ A_1: \text{On input } \langle R \rangle, \text{where } R \text{ is a regular expression.} \]
\[ \quad \text{Convert } R \text{ to DFA } D_1 \text{ and convert } \{0 + 1\}^*111\{0 + 1\}^* \text{ to DFA } D_2. \]
\[ \quad \text{Construct } D_3 \text{ from } D_1 \text{ and } D_2 \text{ so that } L(D_3) = L(D_1) \cap L(D_2) \]
\[ \quad \text{If } L(D_3) = \emptyset, \text{reject, else, accept.} \]

(b) The idea is to parse \( 1^n \) for any \( n \leq k \) by CYK algorithm. Following TM \( A_2 \) can decide \( L_2 \),

\[ A_2: \text{On input } \langle G, k \rangle, \text{where } G \text{ is a context free grammar over alphabet } \{1\}. \]
\[ \quad \text{For all possible strings } x \text{ where } |x| \leq k:} \]
\[ \quad \text{Parse } x \text{ by } G. \]
\[ \quad \quad \text{If } G \text{ rejects, continue.} \]
\[ \quad \quad \text{If } G \text{ accepts, accept.} \]
\[ \quad \text{Reject.} \]

(c) Let \( L \) be the language of regular expression \( 1^k1^* \). We can know \( \langle G, k \rangle \in L_1 \) if only if \( L(G) \cap L \neq \emptyset \). Since the intersection of \( L(G) \) and \( L \) is context free, testing emptiness for their intersection is decidable. So following TM \( A_3 \) can decide \( L_3 \),

\[ A_3: \text{On input } \langle G, k \rangle, \text{where } G \text{ is a context free grammar over alphabet } \{1\}. \]
\[ \quad \text{Convert } G \text{ to PDA } P \text{ and convert } 1^k1^* \text{ to DFA } D. \]
\[ \quad \text{Construct PDA } P' \text{ from } P \text{ and } D \text{ so that } L(P') = L(P) \cap L(D), \]
\[ \quad \text{If } L(P') = \emptyset, \text{reject, else, accept.} \]
Problem 2

For each of these languages, say whether it is decidable. Justify your answer. Here $M, M_1$ and $M_2$ are all Turing Machines.

(a) $L_1 = \{\langle M \rangle : M \text{ accepts } 0 \text{ within } 999 \text{ transitions}\}$.

(b) $L_2 = \{\langle M \rangle : M \text{ doesn’t accept input } 0\}$.

(c) $L_3 = \{\langle M \rangle : M \text{ accepts a finite number of inputs}\}$.

(d) (Extra credit) $L_4 = \{\langle M_1, M_2 \rangle : M_1 \text{ rejects input } M_2 \text{ or } M_2 \text{ accepts input } M_1 \text{ (or both)}\}$.

Solution

(a) $L_1$ is decidable. The following TM $R$ decides $L_1$:

$R$: On input $\langle M \rangle$, where $M$ is a TM.

Simulate $M$ on 0 for 999 transitions.

If $M$ accepts, accept.

If $M$ rejects or hasn’t finished, continue.

Reject.

(b) $L_2$ is undecidable. Suppose it is decidable and there exists a Turing Machine $A$ that can decide this problem. We try to use $M$ to construct the following TM $B$ to decide $\overline{A_{TM}}$.

Recall that $\overline{A_{TM}} = \{\langle M, w \rangle : M \text{ rejects or loops on input } w\}$.

Here is the TM that decides $\overline{A_{TM}}$ using $A$:

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

Construct the following TM $M'$:

$M'$: “On input $z$,

If $z = 0$, simulate $M$ on $w$, and output its answer.

Otherwise, reject.”

Run $A$ on input $M'$ and output its answer.

It remains to show that $A$ accepts $M'$ iff $B$ doesn’t accept $\langle M, w \rangle$. $A$ accepts $M'$ iff $M'$ doesn’t accept 0. By definition of $M'$, this is equivalent to $M$ doesn’t accept $w$.

(c) $L_3$ is undecidable. We assume it is and there exists a Turing Machine $R$ that can decide this problem. Below we show how $R$ can be used to decide the language $\overline{L_\epsilon}$. Since $L_\epsilon$ is not decidable, we reach a contradiction. Recall that $\overline{L_\epsilon} = \{\langle M \rangle : M \text{ rejects or loops on input } \epsilon\}$.

Here is the TM that decides $\overline{L_\epsilon}$ using $R$:  


On input \( \langle M \rangle \),

Construct the following TM \( M' \):

\( M' \): “On input \( z \)

- Simulate \( M \) on \( \epsilon \) for \( |z| \) steps.
- If simulation has accepted, accept.
- Otherwise (i.e., if simulation has either rejected or not finished), reject.”

Run \( R \) on input \( \langle M' \rangle \) and output its answer.

To prove that this TM decides \( L_\epsilon \), we consider two cases. If \( M \) does not accept \( \epsilon \) (i.e., \( \langle M \rangle \notin L_\epsilon \)), then \( M' \) will reject all its inputs \( z \), so in particular \( M' \) accepts only a finite number of inputs (namely zero). Therefore \( R \) will accept \( M' \), so the above TM accepts \( \langle M \rangle \) as it should.

If \( M \) accepts \( \epsilon \) (i.e., \( \langle M \rangle \notin L_\epsilon \)), then it must do so within \( t \) steps for some \( t \). Then for every \( z \) whose length is at least \( t \), the simulation carried out by \( M' \) will accept, so \( M' \) accept all such \( z \)s. So \( M' \) accepts an infinite number of inputs. Therefore \( R \) will not accept \( M' \), so the above TM does not accept \( \langle M \rangle \).

It follows that this TM accepts \( \langle M \rangle \) if and only if \( \langle M \rangle \notin L_\epsilon \), so it decides the language \( L_\epsilon \).

**Remark 1.** We can also reduce \( A_{TM} \) to this problem by constructing following TM \( M' \): On input \( z \), return \( M \) on \( w \). Then \( M' \) accepts all inputs (and therefore infinite inputs) if and only if \( M \) accepts \( w \).

(d) To show \( L_4 \) is undecidable, we assume it is and reach a contradiction. Suppose \( D \) is a TM that decides \( L_2 \). We can use \( D \) to decide the language \( L_\epsilon \) as follows:

On input \( \langle M \rangle \),

Construct the following TM \( M_1 \):

\( M_1 \): “On input \( x \), reject.”

Construct the following TM \( M_2 \):

\( M_2 \): “On input \( x \), if \( x = \langle M_1 \rangle \), simulate \( M \) on \( \epsilon \), otherwise reject.”

Run \( D \) on input \( \langle M_1, M_2 \rangle \) and output its answer.

Since the above \( M_1 \) always rejects, we have that \( \langle M_1, M_2 \rangle \notin L_2 \) if and only if \( M_2 \) accepts \( \langle M_1 \rangle \). But we constructed \( M_2 \) so that \( M_2 \) accepts \( \langle M_1 \rangle \) if and only if \( M \) accepts \( \epsilon \). Therefore, \( \langle M_1, M_2 \rangle \in L_2 \) if and only if \( M \) accepts \( \epsilon \). So the above TM accepts \( \langle M \rangle \) if \( M \) accepts \( \epsilon \), and rejects otherwise. It follows that the above TM decides the language \( L_\epsilon \), a contradiction.
Problem 3

For each of the following variants of the Post Correspondence Problem (PCP), say if it is decidable or not. Justify your answer by describing a decider, or by reducing from (standard) PCP.

(a) $PCP_{\star} = \{\langle P \rangle : P \text{ is in PCP and every tile in } P \text{ is a } \star\text{-tile.}\}$
A $\star$-tile is a tile where both the top and bottom strings begin with $\star$, for example

\[
\begin{array}{c}
\star ab \star c \\
\star
\end{array}
\quad \text{is a } \star\text{-tile, but}
\begin{array}{c}
\star \\
\star
\end{array}
\quad \text{is not}
\]

(b) $PCP_1 = \{\langle P \rangle : P \text{ is in PCP and every tile in } P \text{ is one-symbol tile.}\}$
A one-symbol tile is a tile where the top and bottom strings have length at most one, so

\[
\begin{array}{ccc}
\varepsilon & b & \varepsilon \\
b & b & b
\end{array}
\]
are all one-symbol tiles.

Solution

(a) **Undecidable.** We reduce from $PCP$ to $PCP_{\star}$. Given an instance $T$ of $PCP$, we form the following instance $T'$ of $PCP_{\star}$: We replace each symbol $x$ in $T$ by the pair $\star x$ in $T'$. Now it should be clear that $T$ has a match if and only if $T'$ has one. To go from a match in $T$ to a match in $T'$, we replace each symbol $a$ of $T$ by $\star a$. To go the other way, we replace each representing sequence with the corresponding symbol.

Therefore, if $PCP_{\star}$ were decidable, we could also decide $PCP$. Since $PCP$ is undecidable, $PCP_{\star}$ must also be undecidable.

(b) **Decidable.** Let’s do an example first. Suppose you are given the following $PCP_1$ instance:

\[
P_1 = \{\varepsilon, a, b, c, d, e\}
\]

Look at the following graph. This graph has a node for every symbol and an edge from the top symbol to the bottom symbol at every tile that doesn’t have $\varepsilon$. The nodes with tiles that have $\varepsilon$ on them are special: The bottom symbols on such tiles are marked by a square and the bottom symbols are marked by a diamond.
Notice that this graph has a path from a square to the diamond, namely the path $a \rightarrow b \rightarrow c \rightarrow e$. This path gives rise to the following match:

```
 e a b c e e
```

Now suppose instead you are looking at another instance:

$$P_2 = \{ \varepsilon, \varepsilon, a b, b c, c e, d b, d e, d \}$$

The corresponding graph is now

```
  b
 /\ 
/   
 a--c
   \
   
    d
```

This tile does not have a match: The tiles that can continue a match with $d$ at the top and those that can continue a match with $a$ at the bottom are disjoint. This can be seen by looking at the graph: There is no path from a square node to a diamond node.

In general, we can decide if a PCP instance have a match by looking for such a path. So the following Turing Machine decides $PCP_1$. We use $t/b$ to denote a tile with $t$ on top and $b$ at the bottom.

On input $\langle P \rangle$, where $P$ is a collection of one-symbol PCP tiles,
- If $P$ contains the tile $\varepsilon/\varepsilon$, accept.
- Construct the following directed graph $G$:
  - The nodes of $G$ are the symbols that appear on the tiles of $P$ (except $\varepsilon$).
  - For every tile $t/b$ in $P$ where $t \neq \varepsilon$ and $b \neq \varepsilon$, add an edge from $t$ to $b$ in $G$.
  - For every tile $\varepsilon/b$ in $P$, mark node $b$ by a square.
  - For every tile $t/\varepsilon$ in $P$, mark node $t$ by a diamond.
  - If there is a path from a square to a diamond in $G$, accept.
- Otherwise, reject.

**Problem 4**

You just got hired by Doodle, the hottest software company of the 2010s. Your new boss has several project proposals for you. However, you suspect that some of her proposals may be a bit unrealistic. But you won’t turn down a proposal just because you don’t like it, or else you might get fired pretty soon. You have to give a reason why you think it is not going to work.

For each of these software projects, say if you think the project is feasible or not. If you think it is feasible, say how you would approach it. If you think it is infeasible, explain why.
(a) Doodle just came up with a new programming language called Plum. However most of their old programs, and there are many of them, are written in Java. Write an application which converts all their Java programs into Plum.

(b) Doodle solicits applications from developers that it then sells in its Doodle Store. But some of these developers are very clumsy and their applications tend to crash. It would be nice to have a tool that detects these crashing applications so Doodle can take them out of their store. Write a program called crash detector, which looks at the code of an application (written in Plum) and figures out if the application will ever crash.

(c) Some programs take a very long time to run. It would be nice to know roughly how long a program is going to run for ahead of time, so if it takes a long time you can go out and get lunch. Write an application called timer, that looks at a computer program and gives an estimate of its running time.

Solution

(a) It is feasible. By the Church-Turing Thesis, all programming languages are equivalent in power to one another (and to a Turing Machine). Therefore, we should be able to take write a tool that takes programs written in one language and converts them to another language. In more detail, this program will take every step of a Java program and convert it to a sequence of equivalent steps in Plum.

(b) This task is infeasible. If we want to figure out if an application will ever crash by looking at its code, we actually want to construct a TM \( \text{HALT}_{TM} \) that determines whether the program will halt or not. We know \( \text{HALT}_{TM} \) is undecidable, so this task is impossible.

(c) This task is also infeasible. If we could determine the running time of every program, then we could be able to distinguish those programs that loop from those programs that halt. This would allow us to decide \( \text{HALT}_{TM} \). We know \( \text{HALT}_{TM} \) is undecidable, so this task is impossible.