Exercise 1: Constructing Turing Machines

Construct a Turing Machine for each of the following languages.

(a) \( L_1 = \{a^i b^j a^i b^j | i, j > 0 \} \)

(b) Language \( L_2 \) of all strings over alphabet \( \{a, b\} \) with the same number of \( a \)'s and \( b \)'s.

Remark: It is sufficient to give a detailed description of the Turing Machines. You do not need to give formal definitions.

Sample Solution

The sketch of the Turing Machines:

(a) The computation first makes sure that a string is in the form of having a non-empty substring \( A \) of only \( a \)'s, followed by a non-empty substring \( B \) of only \( b \)'s, a non-empty substring \( C \) of only \( a \)'s, and finally followed by a non-empty substring \( D \) of only \( b \)'s. Then, it checks whether \( A \) and \( C \) have the same size as follows. It replace an \( a \) in \( A \) with \( X \) and then look for an \( a \) in \( C \) to be replaced by \( Y \). If it can find a corresponding \( a \) in \( C \) for each and every \( a \) in \( A \), and having no \( a \)'s left in the input tape, then it confirms the equality of \( A \) and \( C \). It can thus continue the computation by comparing the length of \( B \) and \( D \). If it also confirms their equality, it accepts the input.

(b) The computation begins by finding the first \( a \) in the input and replacing it with an \( X \). Then the tape head is moved to the beginning of the tape. It then looks for a \( b \) in the input tape to replace it with an \( X \). If for each and every \( a \) in the input tape, it can find a corresponding \( b \), and finally no \( a \) or \( b \) left on the input string, it can confirm the equality of the numbers of \( a \)'s and \( b \)'s.

Exercise 2: Semi-Decidable vs. Recursively Enumerable

Very often people in computer science use the terms *semi-decidable* and *recursively enumerable* equivalently. The following exercise shows in which way they actually are equivalent. We first recall the definition of both terms.

A language \( L \) is *semi-decidable* if there is a Turing machine which accepts every \( w \in L \) and does not accept any \( w \notin L \) (this means the TM can either reject \( w \notin L \) or simply not stop for \( w \notin L \)).

A language is *recursively enumerable* if there is a Turing machine which eventually outputs every word \( w \in L \) and never outputs a word \( w \notin L \).

(a) Show that any recursively enumerable language is semi-decidable.

(b) Show that any semi-decidable language is recursively enumerable.
Sample Solution

(a) Let $M_L$ be the TM which enumerates $L$. Construct a TM which, on input $w$, simulates $M_L$. If $M_L$ outputs $w$ the TM accepts $w$, otherwise it might run forever.

(b) Let $M_L$ be a TM which semi-decides $L$. We use a tricky simulation of $M_L$ to construct a TM which recursively enumerates $L$. We order all words lexicographically $w_1, w_2, w_3, \ldots$ and then we simulate $M_L$ as follows:

1) Simulate one step of $M_L$ on $w_1$
2) Simulate one (further) step of $M_L$ on $w_1$ and $w_2$
3) Simulate one (further) step of $M_L$ on $w_1, w_2$ and $w_3$
4) Simulate one (further) step of $M_L$ on $w_1, w_2, w_3$ and $w_4$
5) etc.

Exercise 3: Decidability

1. The special halting problem is defined as

$$H_s = \{\langle M \rangle \mid \langle M \rangle \text{ encodes a TM and } M \text{ halts on } \langle M \rangle \}.$$  

Show that $H_s$ is undecidable.

**Hint:** Assume that $M$ is a TM which decides $H_s$ and then construct a TM which halts iff $M$ does not halt. Use this construction to find a contradiction.

2. Show that $A = \{\langle R, S \rangle \mid R$ and $S$ are regular expressions and $L(R) \subseteq L(S)\}$ is decidable.

3. Show that $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are Turing Machines and } L(M_1) = L(M_2)\}$ is undecidable.

**Hint:** You may use that $E_{TM} = \{\langle M \rangle \mid M \text{ is a Turing Machine and } L(M) = \emptyset\}$ is undecidable.

Sample Solution

(a) Assume that $H_s$ is decidable. Then there is a TM $M$ which decides it. Now let us define a TM $\tilde{M}$ as follows. TM $\tilde{M}$ on input $w$ uses $M$ to test whether $w \in H_s$. If $w \in H_s$ it enters a non terminating loop, otherwise it accepts $w$. We now apply $\tilde{M}$ on input $\langle \tilde{M} \rangle$ and construct a contradiction.

$\langle \tilde{M} \rangle \notin H_s$: Then $M$ rejects $\langle \tilde{M} \rangle$. Thus $\tilde{M}$ accepts $\langle \tilde{M} \rangle$ by the definition of $\tilde{M}$. Thus, $\langle \tilde{M} \rangle \in H_s$, a contradiction.

$\langle \tilde{M} \rangle \in H_s$: Then $M$ accepts $\langle \tilde{M} \rangle$, i.e., $\tilde{M}$ enters a non terminating loop on $\langle \tilde{M} \rangle$ and does not halt on $\langle \tilde{M} \rangle$ which means that $\langle \tilde{M} \rangle \notin H_s$, a contradiction.

(b) Let $T$ be the Turing Machine deciding the language $\{\langle D \rangle \mid D$ is a DFA with $L(D) = \emptyset\}$ (known from the lecture). We have $L(R) \subseteq L(S) \iff L(R) \setminus L(S) = \emptyset$. Thus we construct a decider for $A$ in the following way:

On input $\langle R, S \rangle$ where $R, S$ are regular expression:

- Convert $R$ and $S$ into equivalent DFAs (like in the lecture)
• Construct a DFA $D$ for the regular language $L(R) \setminus L(S) = L(R) \cap \overline{L(S)}$
• Run $T$ on input $\langle D \rangle$. Accept iff $T$ accepts.

(c) Assume we had a TM $R$ that decides $EQ_{TM}$. We construct a decider for $E_{TM}$:

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

• Construct a TM $B$ that rejects all inputs.
• Run $R$ on $\langle M, B \rangle$. Accept iff $R$ accepts.