10 Decidability and Recursively Enumerable Languages

In this chapter the formal definition of an algorithm is given. The problem of decidability of various class of languages is discussed. The theorem on halting problem of Turing machine is proved.

10.1 THE DEFINITION OF AN ALGORITHM

In Section 4.4, we gave the definition of an algorithm as a procedure (finite sequence of instructions which can be mechanically carried out) that terminates after a finite number of steps for any input. The earliest algorithm one can think of is the Euclidean algorithm, for computing the greatest common divisor of two natural numbers. In 1900, the mathematician David Hilbert, in his famous address at the International congress of mathematicians in Paris, averred that every definite mathematical problem must be susceptible for an exact settlement either in the form of an exact answer or by the proof of the impossibility of its solution. He identified 23 mathematical problems as a challenge for future mathematicians; only ten of the problems have been solved so far.

Hilbert's tenth problem was to devise 'a process according to which it can be determined by a finite number of operations', whether a polynomial over Z has an integral root. (He did not use the word 'algorithm' but he meant the same.) This was not answered until 1970.

The formal definition of algorithm emerged after the works of Alan Turing and Alanzo Church in 1936. The Church–Turing thesis states that any algorithmic procedure that can be carried out by a human or a computer, can also be carried out by a Turing machine. Thus the Turing machine arose as an ideal theoretical model for an algorithm. The Turing machine provided a machinery to mathematicians for attacking the Hilberts' tenth problem. The problem can be restated as follows: does there exist a TM that can accept a

polynomial over n variables if it has an integral root and reject the polynomial if it does not have one.

In 1970, Yuri Matijasevic, after studying the work of Martin Davis, Hilary Putnam and Julia Robinson showed that no such algorithm (Turing machine) exists for testing whether a polynomial over n variables has integral roots. Now it is universally accepted by computer scientists that Turing machine is a mathematical model of an algorithm.

10.2 DECIDABILITY

We are familiar with the recursive definition of a function or a set. We also have the definitions of recursively enumerable sets and recursive sets (refer to Section 4.4). The notion of a recursively enumerable set (or language) and a recursive set (or language) existed even before the dawn of computers.

Now these terms are also defined using Turing machines. When a Turing machine reaches a final state, it 'halts.' We can also say that a Turing machine M halts when M reaches a state q and a current symbol a to be scanned so that $\delta(q, a)$ is undefined. There are TMs that never halt on some inputs in any one of these ways. So we make a distinction between the languages accepted by a TM that halts on all input strings and a TM that never halts on some input strings.

Definition 10.1 A language $L \subseteq \Sigma^*$ is recursively enumerable if there exists a TM M, such that $L = T(M)$.

Definition 10.2 A language $L \subseteq \Sigma^*$ is recursive if there exists some TM M that satisfies the following two conditions.

(i) If $w \in L$ then M accepts w (that is, reaches an accepting state on processing w) and halts.

(ii) If $w \notin L$ then M eventually halts, without reaching an accepting state.

Note: Definition 10.2 formalizes the notion of an 'algorithm'. An algorithm, in the usual sense, is a well-defined sequence of steps that always terminates and produces an answer. The Conditions (i) and (ii) of Definition 10.2 assure us that the TM always halts, accepting w under Condition (i) and not accepting under Condition (ii). So a TM, defining a recursive language (Definition 10.2) always halts eventually just as an algorithm eventually terminates.

A problem with only two answers Yes/No can be considered as a language L. An instance of the problem with the answer 'Yes' can be considered as an element of the corresponding language L; an instance with answer 'No' is considered as an element not in L.

Definition 10.3 A problem with two answers (Yes/No) is decidable if the corresponding language is recursive. In this case, the language L is also called *decidable*.

Definition 10.4 A problem/language is undecidable if it is not decidable.

Note: A decidable problem is called a solvable problem and an undecidable problem an unsolvable problem by some authors.

10.3 DECIDABLE LANGUAGES

In this section we consider the decidability of regular and context-free languages.

First of all, we consider the problem of testing whether a deterministic finite automaton accepts a given input string w.

Definition 10.5

$$A_{DFA} = \{(B, w) \mid B \text{ accepts the input string } w\}$$

Theorem 10.1 A_{DFA} is decidable.

Proof To prove the theorem, we have to construct a TM that always halts and also accepts A_{DFA}. We describe the TM M using high level description (refer to Section 9.5). Note that a DFM B always ends in some state of B after n transitions for an input string of length n.

We define a TM M as follows:

1. Let B be a DFA and w an input string. (B, w) is an input for the Turing machine M.
2. Simulate B and input w in the TM M.
3. If the simulation ends in an accepting state of B, then M accepts w. If it ends in a nonaccepting state of B, then M rejects w.

We can discuss a few implementation details regarding steps 1, 2 and 3 above. The input (B, w) for M is represented by representing the five components $Q, \Sigma, \delta, q_0, f$ by strings of Σ^* and input string $w \in \Sigma^*$. M checks whether (B, w) is a valid input. If not, it rejectes (B, w) and halts. If (B, w) is a valid input, M writes the initial state q_0 and the leftmost input symbol of w. It updates the state using δ and then reads the next symbol in w. This explains step 2.

If the simulation ends in an accepting state w, then M accepts (B, w). Otherwise, M rejects (B, w). This is the description of step 3.

It is evident that M accepts (B, w) if and only if w is accepted by the DFA B. ∎

Definition 10.6

$$A_{CFG} = \{(G, w) \mid \text{the context-free grammar } G \text{ accepts the input string } w\}$$

Theorem 10.2 A_{CFG} is decidable.

Proof We convert a CFG into Chomsky normal form. Then any derivation of w of length k requires $2k - 1$ steps if the grammar is in CNF (refer to Example 6.18). So for checking whether the input string w of length k is

in $L(G)$, it is enough to check derivations in $2k - 1$ steps. We know that there are only finitely many derivations in $2k - 1$ steps. Now we design a TM M that halts as follows.

1. Let G be a CFG in Chomsky normal form and w an input string. (G, w) is an input for M.
2. If $k = 0$, list all the single-step derivations. If $k \neq 0$, list all the derivations with $2k - 1$ steps.
3. If any of the derivations in step 2 generates the given string w, M accepts (G, w). Otherwise M rejects.

The implementation of steps 1–3 is similar to the steps in Theorem 10.1. (G, w) is represented by representing the four components V_N, Σ, P, S of G and input string w. The next step of the derivation is got by the production to be applied.

M accepts (G, w) if and only if w is accepted by the CFG G.

In Theorem 4.3, we proved that a context-sensitive language is recursive. The main idea of the proof of Theorem 4.3 was to construct a sequence $\{W_0, W_1, \ldots, W_k\}$ of subsets of $(V_N \cup \Sigma)^*$, that terminates after a finite number of iterations. The given string $w \in \Sigma^*$ is in $L(G)$ if and only if $w \in W_k$. With this idea in mind we can prove the decidability of the context-sensitive language. ∎

Definition 10.7 $A_{CSG} = \{(G, w) \mid$ the context-sensitive grammar G accepts the input string $w\}$.

Theroem 10.3 A_{CSG} is decidable.

Proof The proof is a modification of the proof of Theorem 10.2. In Theorem 10.2, we considered derivations with $2k - 1$ steps for testing whether an input string of length k was in $L(G)$. In the case of context-sensitive grammar we construct $W_i = \{\alpha \in (V_N \cup \Sigma)^* \mid S \overset{*}{\underset{G}{\Rightarrow}} \alpha$ in i or fewer steps and $|\alpha| \leq n\}$. There exists a natural number k such that $W_k = W_{k+1} = W_{k+2} = \ldots$ (refer to proof of Theorem 4.3).

So $w \in L(G)$ if and only if $w \in W_k$. The construction of W_k is the key idea used in the construction of a TM accepting A_{CSG}. Now we can design a Turing machine M as follows:

1. Let G be a context-sensitive grammar and w an input string of length n. Then (G, w) is an input for TM.
2. Construct $W_0 = \{S\}$. $W_{i+1} = W_i \cup \{\beta \in (V_N \cup \Sigma)^* \mid$ there exists $\alpha_i \in W_i$ such that $\alpha \Rightarrow \beta$ and $|\beta| \leq n\}$. Continue until $W_k = W_{k+1}$ for some k. (This is possible by Theorem 4.3.)
3. If $w \in W_k$, $w \in L(G)$ and M accepts (G, w); otherwise M rejects (G, w). ∎

Note: If $\mathcal{L}_d$ denotes the class of all decidable languages over Σ, then

$$\mathcal{L}_{rl} \subseteq \mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl} \subseteq \mathcal{L}_d$$

10.4 UNDECIDABLE LANGUAGES

In this section we prove the existence of languages that are not recursively enumerable and address the undecidability of recursively enumerable languages.

Theorem 10.4 There exists a language over Σ that is not recursively enumerable.

Proof A language L is recursively enumerable if there exists a TM M such that $L = T(M)$. As Σ is finite, Σ^* is countable (that is, there exists a one-to-one correspondence between Σ^* and N).

As a Turing machine M is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$ and each member of the 7-tuple is a finite set, M can be encoded as a string. So the set I of all TMs is countable.

Let $\mathcal{L}$ be the set of all languages over Σ. Then a member of $\mathcal{L}$ is a subset of Σ^* (Note that Σ^* is infinite even though Σ is finite). We show that $\mathcal{L}$ is uncountable (that is, an infinite set not in one-to correspondence with N).

We prove this by contradiction. If $\mathcal{L}$ were countable then $\mathcal{L}$ can be written as a sequence $\{L_1, L_2, L_3, \ldots\}$. We write Σ^* as a sequence $\{w_1, w_2, w_3, \ldots\}$. So L_i can be represented as an infinite binary sequence $x_{i1}x_{i2}x_{i3}\ldots$ where

$$x_{ij} = \begin{cases} 1 & \text{if } w_j \in L_i \\ 0 & \text{otherwise} \end{cases}$$

Using this representation we write L_i as an infinite binary sequence.

$$\begin{aligned} L_1 &: x_{11}x_{12}x_{13} \ldots x_{1j} \ldots \\ L_2 &: x_{21}x_{22}x_{23} \ldots x_{2j} \ldots \\ &\vdots \qquad\qquad\quad \vdots \\ L_i &: x_{i1}x_{i2}x_{i3} \ldots x_{ij} \ldots \end{aligned}$$

Fig. 10.1 Representation of $\mathcal{L}$.

We define a subset L of Σ^* by the binary sequence $y_1y_2y_3 \ldots$ where $y_i = 1 - x_{ii}$. If $x_{ii} = 0$, $y_i = 1$ and if $x_{ii} = 1$, $y_i = 0$. Thus according to our assumption the subset L of Σ^* represented by the infinite binary sequence $y_1y_2y_3 \ldots$ should be L_k for some natural number k. But $L \neq L_k$, since $w_k \in L$ if and only if $w_k \notin L_k$. This contradicts our assumption that $\mathcal{L}$ is countable. Therefore $\mathcal{L}$ is uncountable. As I is countable, $\mathcal{L}$ should have some members not corresponding to any TM in I. This proves the existence of a language over Σ tnat is not recursively enumerable. ∎

Definition 10.8 $A_{TM} = \{(M, w) \mid \text{The TM } M \text{ accepts } w\}$.

Theorem 10.5 A_{TM} is undecidable.

Proof We can prove that A_{TM} is recursively enumerable. Construct a TM U as follows:

(M, w) is an input to U. Simulate M on w. If M enters an accepting state, U accepts (M, w). Hence A_{TM} is recursively enumerable. We prove that A_{TM} is undecidable by contradiction. We assume that A_{TM} is decidable by a TM H that eventually halts on all inputs. Then

$$H(M, w) = \begin{cases} \text{accept} & \text{if } M \text{ accepts } w \\ \text{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

We construct a new TM D with H as subroutine. D calls H to determine what M does when it receives the input $\langle M \rangle$, the encoded description of M as a string. Based on the received information on $(M, \langle M \rangle)$, D rejects M if M accepts $\langle M \rangle$ and accepts M if M rejects $\langle M \rangle$. D is described as follows:

1. $\langle M \rangle$ is an input to D, where $\langle M \rangle$ is the encoded string representing M.
2. D calls H to run on $(M, \langle M \rangle)$
3. D rejects $\langle M \rangle$ if H accepts $(M, \langle M \rangle)$ and accepts $\langle M \rangle$ if H rejects $(M, \langle M \rangle)$.

Now step 3 can be described as follows:

$$D(\langle M \rangle) = \begin{cases} \text{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \text{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

Let us look at the action of D on the input $\langle D \rangle$. According to the construction of D,

$$D(\langle D \rangle) = \begin{cases} \text{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \text{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

This means D accepts $\langle D \rangle$ if D does not accept $\langle D \rangle$, which is a contradiction. Hence ATM is undecidable. ∎

The Turing machine U used in the proof of Theorem 10.5 is called the *universal Turing machine*. U is called universal since it is simulating any other Turing machine.

10.5 HALTING PROBLEM OF TURING MACHINE

In this section we introduce the reduction technique. This technique is used to prove the undecidability of halting problem of Turing machine.

We say that problem A is reducible to problem B if a solution to problem B can be used to solve problem A.

For example, if A is the problem of finding some root of $x^4 - 3x^2 + 2 = 0$ and B is the problem of finding some root of $x^2 - 2 = 0$, then A is reducible to B. As $x^2 - 2$ is a factor of $x^4 - 3x^2 + 2$, a root of $x^2 - 2 = 0$ is also a root of $x^4 - 3x^2 + 2 = 0$.

Note: If A is reducible to B and B is decidable then A is decidable. If A is reducible to B and A is undecidable, then B is undecidable.

Theorem 10.6 $HALT_{TM} = \{(M, w) \mid$ The Turing machine M halts on input $w\}$ is undecidable.

Proof We assume that $HALT_{TM}$ is decidable, and get a contradiction. Let M_1 be the TM such that $T(M_1) = HALT_{TM}$ and let M_1 halt eventually on all (M, w). We construct a TM M_2 as follows:

1. For M_2, (M, w) is an input.
2. The TM M_1 acts on (M, w).
3. If M_1 rejects (M, w) then M_2 rejects (M, w).
4. If M_1 accepts (M, w), simulate the TM M on the input string w until M halts.
5. If M has accepted w, M_2 accepts (M, w); otherwise M_2 rejects (M, w).

When M_1 accepts (M, w) (in step 4), the Turing machine M halts on w. In this case either an accepting state q or a state q' such that $\delta(q', a)$ is undefined till some symbol a in w is reached. In the first case (the first alternative of step 5) M_2 accepts (M, w). In the second case (the second alternative of step 5) M_2 rejects (M, w).

It follows from the definition of M_2 that M_2 halts eventually.

Also, $$T(M_2) = \{(M, w) \mid \text{The Turing machine accepts } w\}$$
$$= A_{TM}$$

This is a contradiction since A_{TM} is undecidable. ∎

10.6 THE POST CORRESPONDENCE PROBLEM

The Post Correspondence Problem (PCP) was first introduced by Emil Post in 1946. Later, the problem was found to have many applications in the theory of formal languages. The problem over an alphabet Σ belongs to a class of yes/no problems and is stated as follows: Consider the two lists $x = (x_1 \ldots x_n)$, $y = (y_1 \ldots y_n)$ of nonempty strings over an alphabet $\Sigma = \{0, 1\}$. The PCP is to determine whether or not there exist $i_1, \ldots, i_m$, where $1 \leq i_j \leq n$, such that

$$x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$$

Note: The indices i_j's need not be distinct and m may be greater than n. Also, if there exists a solution to PCP, there exist infinitely many solutions.

EXAMPLE 10.1

Does the PCP with two lists $x = (b, bab^3, ba)$ and $y = (b^3, ba, a)$ have a solution?

Solution

We have to determine whether or not there exists a sequence of substrings of x such that the string formed by this sequence and the string formed by the sequence of corresponding substrings of y are identical. The required sequence is given by $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$, i.e. (2, 1, 1,3), and $m = 4$. The corresponding strings are

bab^3	b	b	ba	=	ba	b^3	b^3	a
x_2	x_1	x_1	x_3		y_2	y_1	y_1	y_3

Thus the PCP has a solution.

EXAMPLE 10.2

Prove that PCP with two lists $x = (01, 1, 1)$, $y = (01^2, 10, 1^1)$ has no solution.

Solution

For each substring $x_i \in x$ and $y_i \in y$, we have $|x_i| < |y_i|$ for all i. Hence the string generated by a sequence of substrings of x is shorter than the string generated by the sequence of corresponding substrings of y. Therefore, the PCP has no solution.

Note: If the first substring used in PCP is always x_1 and y_1, then the PCP is known as the *Modified Post Correspondence Problem.*

EXAMPLE 10.3

Explain how a Post Correspondence Problem can be treated as a game of dominoes.

Solution

The PCP may be thought of as a game of dominoes in the following way: Let each domino contain some x_i in the upper-half, and the corresponding substring of y in the lower-half. A typical domino is shown as

x_i	upper-half
y_i	lower-half

The PCP is equivalent to placing the dominoes one after another as a sequence (of course repetitions are allowed). To win the game, the same string should appear in the upper-half and in the lower-half. So winning the game is equivalent to a solution of the PCP.

We state the following theorem by Emil Post without proof.

Theorem 10.7 The PCP over Σ for $|\Sigma| \geq 2$ is unsolvable.

It is possible to reduce the PCP to many classes of two outputs (yes/no) problems in formal language theory. The following results can be proved by the reduction technique applied to PCP.

1. If L_1 and L_2 are any two context-free languages (type 2) over an alphabet Σ and $|\Sigma| \geq 2$, there is no algorithm to determine whether or not
   (a) $L_1 \cap L_2 = \emptyset$,
   (b) $L_1 \cap L_2$ is a context-free language,
   (c) $L_1 \subseteq L_2$, and
   (d) $L_1 = L_2$.
2. If G is a context-sensitive grammar (type 1), there is no algorithm to determine whether or not
   (a) $L(G) = \emptyset$,
   (b) $L(G)$ is infinite, and
   (c) $x_0 \in L(G)$ for a fixed string x_0.
3. If G is a type 0 grammar, there is no algorithm to determine whether or not any string $x \in \Sigma^*$ is in $L(G)$.

10.7 SUPPLEMENTARY EXAMPLES

EXAMPLE 10.4

If L is a recursive language over Σ, show that $\bar{L}$ ($\bar{L}$ is defined as $\Sigma^* - L$) is also recursive.

Solution

As L is recursive, there is a Turing machine M that halts and $T(M) = L$. We have to construct a TM M_1, such that $T(M_1) = \bar{L}$ and M_1 eventually halts.

M_1 is obtained by modifying M as follows:

1. Accepting states of M are made nonaccepting states of M_1.
2. Let M_1 have a new state q_f. After reaching q_f, M_1 does not move in further transitions.
3. If q is a nonaccepting state of M and $\delta(q, x)$ is not defined, add a transition from q to q_f for M_1.

As M halts, M_1 also halts. (If M reaches an accepting state on w, then M_1 does not accept w and halts and conversely.)

Also M_1 accepts w if and only if M does not accept w. So I is recursive.

EXAMPLE 10.5

If L and $\overline{L}$ are both recursively enumerable, show that L and $\overline{L}$ are recursive.

Solution

Let M_1 and M_2 be two TMs such that $L = T(M_1)$ and $\overline{L} = T(M_2)$. We construct a new two-tape TM M that simulates M_1 on one tape and M_2 on the other.

If the input string w of M is in L, then M_1 accepts w and we declare that M accepts w. If $w \in \overline{L}$, then M_2 accepts w and we declare that M halts without accepting. Thus in both cases, M eventually halts. By the construction of M it is clear that $T(M) = T(M_1) = L$. Hence L is recursive. We can show that $\overline{L}$ is recursive, either by applying Example 10.4 or by interchanging the roles of M_1 and M_2 in defining acceptance by M.

EXAMPLE 10.6

Show that $\overline{A}_{TM}$ is not recursively enumerable.

Solution

We have already seen that A_{TM} is recursively enumerable (by Theorem 10.5). If $\overline{A}_{TM}$ were also recursively enumerable, then A_{TM} is recursive (by Example 10.5). This is a contradiction since A_{TM} is not recursive by Theorem 10.5. Hence $\overline{A}_{TM}$ is not recursively enumerable.

EXAMPLE 10.7

Show that the union of two recursively enumerable languages is recursively enumerable and the union of two recursive languages is recursive.

Solution

Let L_1 and L_2 be two recursive languages and M_1, M_2 be the corresponding TMs that halt. We design a TM M as a two-tape TM as follows:

1. w is an input string to M.
2. M copies w on its second tape.
3. M simulates M_1 on the first tape. If w is accepted by M_1, then M accepts w.
4. M simulates M_2 on the second tape. If w is accepted by M_2, then M accepts w.

M always halts for any input w.

Thus $L_1 \cup L_2 = T(M)$ and hence $L_1 \cup L_2$ is recursive.

If L_1 and L_2 are recursively enumerable, then the same conclusion gives a proof for $L_1 \cup L_2$ to be recursively enumerable. As M_1 and M_2 need not halt, M need not halt.

SELF-TEST

1. What is the difference between a recursive language and a recursively enumerable language?
2. The DFA M is given by

$$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

where δ is defined by the transition Table 10.1.

TABLE 10.1 Transition Table for Self-Test 2

State	0	1
$\rightarrow(q_0)$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Answer the following:
(a) Is $(M, 001101)$ in A_{DFA}?
(b) Is $(M, 01010101)$ in A_{DFA}?
(c) Does $M \in A_{DFA}$?
(d) Find w such that $(M, w) \notin A_{DFA}$.

3. What do you mean by saying that the halting problem of TM is undecidable?
4. Describe A_{DFA}, A_{CFG}, A_{CSG}, A_{TM}, and $HALT_{TM}$.
5. Give one language from each of $\mathcal{L}_{rl}$, $\mathcal{L}_{cfl}$, $\mathcal{L}_{csl}$.
6. Give a language
   (a) which is in $\mathcal{L}_{csl}$ but not in $\mathcal{L}_{rl}$
   (b) which is in $\mathcal{L}_{cfl}$ but not in $\mathcal{L}_{csl}$
   (c) which is in $\mathcal{L}_{cfl}$ but not in $\mathcal{L}_{rl}$.

EXERCISES

10.1 Describe the Euclid's algorithm for finding the greatest common divisor of two natural numbers.

10.2 Show that $A_{NDFA} = \{(B, w) \mid B \text{ is an } N_{DFA} \text{ and } B \text{ accepts } w\}$ is decidable.

10.3 Show that $E_{DFA} = \{M \mid M \text{ is a } D_{FA} \text{ and } T(M) = \emptyset\}$ is decidable.

10.4 Show that $EQ_{DFA} = \{(A, B) \mid A \text{ and } B \text{ are DFAs and } T(A) = T(B)\}$ is decidable

10.5 Show that E_{CFG} is decidable (E_{CFG} is defined in a way similar to that of E_{DFA}).

10.6 Give an example of a language that is not recursive but recursively enumerable.

10.7 Do there exist languages that are not recursively enumerable?

10.8 Let L be a language over Σ. Show that only one of the following are possible for L and $\overline{L}$.
(a) Both L and $\overline{L}$ are recursive.
(b) Neither L nor $\overline{L}$ is recursive.
(c) L is recursively enumerable but $\overline{L}$ is not.
(d) $\overline{L}$ is recursively enumerable but L is not.

10.9 What is the difference between A_{TM} and $HALT_{TM}$?

10.10 Show that the set of all real numbers between 0 and 1 is uncountable. (A set S is uncountable if S is infinite and there is no one-to-one correspondence between S and the set of all natural numbers.)

10.11 Why should one study undecidability?

10.12 Prove that the recursiveness problem of type 0 grammar is unsolvable.

10.13 Prove that there exists a Turing machine M for which the halting problem is unsolvable.

10.14 Show that there exists a Turing machine M over $\{0, 1\}$ and a state q_m such that there is no algorithm to determine whether or not M will enter the state q_m when it begins with a given ID.

10.15 Prove that the problem of determining whether or not a TM over $\{0,1\}$ will ever print the symbol 1, with a given tape configuration, is unsolvable.

10.16 (a) Show that $\{x \mid x \text{ is a set and } x \notin x\}$ is not a set. (Note that this seems to be well-defined. This is one version of Russell's paradox.)
(b) A village barber shaves those who do not shave themselves but no others. Can he achieve his goal? For example, who is to shave the barber? (This is a popular version of Russell's paradox.)

Hints: (a) Let $S = \{x \mid x \text{ be a set and } x \notin x\}$. If S were a set, then $S \in S$ or $S \notin S$. If $S \notin S$ by the 'definition' of S, then $S \in S$. On the other hand, if $S \in S$ by the 'definition' of S, then $S \notin S$. Thus we can neither assert that $S \notin S$ nor $S \in S$. (This is Russell's paradox.) Therefore, S is not a set.

(b) Let $S = \{x \mid x \text{ be a person and } x \text{ does not shave himself}\}$. Let b denote the barber. Examine whether $b \in S$. (The argument is similar to that given for (a).) It will be instructive to read the proof of HP of Turing machines and this example, in order to grasp the similarity.

10.17 Comment on the following: "We have developed an algorithm so complicated that no Turing machine can be constructed to execute the algorithm no matter how much (tape) space and time is allowed."

10.18 Prove that PCP is solvable if $|\Sigma| = 1$.

10.19 Let $x = (x_1 \ldots x_n)$ and $y = (y_1 \ldots y_n)$ be two lists of nonempty strings over Σ and $|\Sigma| \geq 2$. (i) Is PCP solvable for $n = 1$? (ii) Is PCP solvable for $n = 2$?

10.20 Prove that the PCP with $\{(01, 011), (1, 10), (1, 11)\}$ has no solution. (Here, $x_1 = 01$, $x_2 = 1$, $x_3 = 1$, $y_1 = 011$, $y_2 = 10$, $y_3 = 11$.)

10.21 Show that the PCP with $S = \{(0, 10), (1^20, 0^3), (0^21, 10)\}$ has no solution. [*Hint:* No pair has common nonempty initial substring.]

10.22 Does the PCP with $x = (b^3, ab^2)$ and $y = (b^3, bab^3)$ have a solution?

10.23 Find at least three solutions to PCP defined by the dominoes:

1
111

10
0

10111
10

10.24 (a) Can you simulate a Turing machine on a general-purpose computer? Explain.

(b) Can you simulate a general-purpose computer on a Turing machine? Explain.