Theory of Formal Languages (Decidability & Computability)

Lecture 10: Feb. 8, 2019

Prof. K.R. Chowdhary *: Professor of CS*

10.1 Introduction

A simple example of decision function is an arbitrary polynomial $P(x_1, x_2, \ldots, x_n)$, with say non-negative integral coefficients, which are not identically zero. If the x's are assigned arbitrary positive integral values, for example, in the Arabic notation, the algorithms comprising for addition and multiplication operations in that notation enable us to calculate the corresponding positive integral value of the polynomial. That is, $P(x_1, x_2, \ldots, x_n)$ is an *effectively calculable* function of positive integers. The importance of the technical concept – recursive function derives from the overwhelming evidence that it is coextensive with the intuitive concept effectively calculable function.

We know that a, set of positive integers is said to be recursively enumerable (RE) if there is a recursive function $f(x)$ of one positive integral variable whose values, for positive integral values of x, constitute the given set. The sequence $f(1), f(2), f(3), \ldots$ is then said to be a *recursive enumeration* of the set. The corresponding intuitive concept is that of an effectively enumerable set of positive integers. To prepare us for this intuitive approach, consider the following examples of recursively enumerable sets of positive integers.

1. $1^2, 2^2, 3^2, \ldots,$
2. $1, 2, 2^{1+2}, 2^{1+2+2^{1+2}}, \ldots$
   (a) $1^2, 2^2, 3^2, \ldots$
   (b) $1^3, 2^3, 3^3, \ldots$
   (c) $1^4, 2^4, 3^4, \ldots$

In the first example, the set is produced by a recursive enumeration using the recursive function x^2. In the second example, the set is generated in a linear sequence, each new element being effectively obtained from the elements previously generated, in this case by raising 2 to the power the sum of the preceding elements. The set is effectively enumerable, since given n, the nth element of the sequence can be found by regenerating the sequence through its first n elements.

In the third example with three series, we rather imagine the positive integers 1, 2, 3, ... generated in their natural order, and, as each positive integer n is generated, a corresponding process set up which generates $n^2, n^3, n^4, \ldots,$ all these to be in the set. Actually, the standard method for proving that an enumerable set of enumerable sets is enumerable yields an effective enumeration of the set.

Several more examples can be given to convey the concept of a generated set, in the present instance of positive integers. Suffice it to say that each element of the set is at some time written down, and earmarked as belonging to the set, as a result of predetermined effective processes. It is understood that once an element is placed in the set, it stays there. Some times this is referred to as a generalization, which may be restated *every generated set of positive integers is recursively enumerable.* For comparison purposes this may be resolved into the two statements: every generated set is effectively enumerable, every effectively enumerable set of positive integers is recursively enumerable. The first of these statements is applicable to generated sets of arbitrary symbolic expressions; their converses are immediately seen to be true. We shall find the above concept and generalization very useful in our intuitive development.

But while we shall frequently say, explicitly or implicitly, "set so and so of positive integers is a generated, and hence recursively enumerable set," that is merely to mean "the set has intuitively been shown to be a generated set; it can indeed be proved to be recursively enumerable." Likewise, for other identifications of informal concepts with corresponding mathematically defined formal concepts.

10.2 Decision Problems

In the previous chapters we designed machines to detect patterns in strings, and recognize the languages based on these patterns. Many higher level problems can be posed based on these pattern Recognitions of strings and string manipulations. For example, We may be interested in following questions:

"Is a given natural number perfect square?"

"Is a given natural number prime?"

"Does a given graph has cycle?"

"Does the computation of TM halt before 25th transition?"

Each of these general questions describe a *decision problem.* A decision problem P is a set of *related questions* p_i, each of which has Yes/No answer. For example, to determine, if a natural number is a perfect square, requires answering following questions, and we carry on the process until we reach to that natural number.

p_0: Is 0 a perfect square?

p_1: Is 1 a perfect square?

p_2: Is 2 a perfect square?

...

Each of the p_i is an instance of the problem P. The solution of a decision problem P is an algorithm that determines the answer of every question "$p_i \in P$"? A *decision problem* is *decidable*, if it has a solution.

Definition 10.1 Decision Problem. *By the decision problem of a given set of positive integers we mean the problem of* effectively *determining for an arbitrarily given positive integer whether it is, or is not, in the set.* □

While, in a certain sense, the theory of recursively enumerable sets of positive integers is potentially as wide as the theory of general recursive functions. The decision problems for such sets constitute a very special class of decision problems. Since decision problem's solution requires an algorithm, an intuitive notion of algorithm is essential. Following are the properties of such algorithms.

- *Complete*: Correct answer should be given for every problem instance,
- *Mechanistic*: Consists of finite sequence of instructions, each can be carried out without requirement of insight, ingenuity, or guesswork, and
- *Deterministic*: With identical input, the identical computation is carried out.

Definition 10.2 Effective procedure. *A procedure having the properties of complete, Mechanistic, and deterministic, is called* effective procedure.

A standard TM is an effective algorithm if it is, Mechanistic, deterministic, and complete. However, it is complete only if, it halts on every input. Thus, TM can be used to solve decision problems. To solve a problem using TM, we need to transform the problem instances into strings, called, representation of the decision problem. A problem instance is answered by a TM, if TM accepts and halts, else rejects and halts.

The *Church-Turing* thesis for decision problem states that TM can be designed to solve the decision problem if it can be solvable by an *effective procedure*.

Following table shows the representation of a problem: "acceptance of unary number strings of even numbers." It shows the general representation of problem P, in the form of its instances $p_1, ..., p_l$, with inputs to TM as $w_1, ..., w_l$, and for each instance the TM gives output Yes/No, when it completes the processing (see Table 10.1).

Table 10.1: General format for problem instances

Problem instance	TM Input	Answer after the Turing machine computes
p_1	w_1	Yes/No
p_2	w_2	Yes/No
...	...	...
p_i	w_i	Yes/No

We can have following instances of the problem: "Whether a natural number is even?" (see Table 10.2).

Table 10.2: Problem instances with solutions

Problem instance	TM Input	Is number even?
0	1	Yes
1	11	No
2	111	Yes
3	1111	No
...	...	...

In general, 1^i, such that i is odd, is an even number. The TMs for acceptance of odd numbers, can be constructed for unary and binary numbers, as shown in figure 10.1(a) and (b). In case of input as a unary number, when TM reaches to blank symbol at the end of input, it will halt at state q_2 if the number is even. In case of binary number as input, which should be terminated by 0 for input to be an even number, it halts in q_2 (see Figure 10.1(b)).

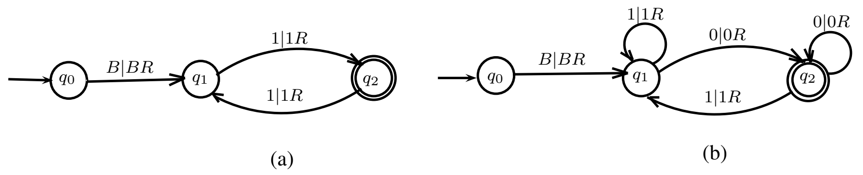

Figure 10.1: (a)TM recognizes even unary numbers, (b) TM recognizes even binary numbers.

10.3 Decision Problems and Recursive Languages

Since, *completeness* (see page 10-3) requires that the corresponding TM should terminate on every input string, hence, the language recognized by this is *Recursive* (R). In other words, every TM solution to a decision problem defines a recursive language. Hence, the decision problem is membership problem: "Is $w \in L$?"

The duality between solvable decision problems and recursive languages can be exploited to broaden the techniques available for establishing the decidability of a decision problem. For example, the decision problem of determining whether any given number is perfect square, is *decidable.*

Based on the above conclusions, now we construct the following definition for Turing Recognizable and decidable languages.

Definition 10.3 Turing recognizable. *A language L is Turing recognizable (also called, Recursively Enumerable (RE)) if there exists a Turing machine M such that $L(M) = L$. It is possible for a TM to never reach a halting configuration, and on some given input w, it might instead loop for ever.* □

Accordingly, in case of L as Turing recognizable, there are three possible outcomes for a TM for any given input: 1. *accept*, 2. *reject*, and 3. *loop*.

Definition 10.4 Turing Decidable. *A language L is Turing-decidable or simply* decidable*, if there is a Turing machine M that decides L. Such TMs halts on every input $w \in L$.* □

Definition 10.5 Semi-decidable. *Let $M = (Q, \Sigma, \Gamma, \delta, s, H)$ be a Turing machine and let $L \subseteq \Sigma^*$ be a language. In this case, M* semi-decides *L if for any string $w \in \Sigma^*$ the following holds: $w \in L$ if and only if M halts on input w. In addition, L is recursively enumerable if and only if there is a Turing machine M that semi-decides L.* □

If M is supplied with input $w \in L$, then M is required to halt eventually. However, it is not important as what halting configuration it reaches, it is all right as long as it reaches to some halting configuration. However, if $w \in \Sigma^* - L$, then M will never enter a halting state. This is because, if any configuration that is not halting, it yields some other configuration, and the machine will continue to run indefinitely.

Theorem 10.6 *If a language L is recursive then its complement is also recursive.*

Proof: Let the language L is decided by the Turing machine $M = (Q, \Sigma, \Gamma, \delta, s, \{y, n\})$. Let M' be another Turing machine, such that,

$$M' = (Q, \Sigma, \Gamma, \delta', s, \{y, n\}).$$

M' decides L in a different way (the difference is in respect of transition function δ') given as follows.

$$\delta'(q, a) = \begin{cases} n, \text{ if } \delta(q, a) = y, \\ y, \text{ if } \delta(q, a) = n, \\ \delta(q, a), \text{ otherwise }. \end{cases} \tag{10.1}$$

It is clear that $M(w)$ yields y iff $M'(w)$ yields n. Thus, M decides the language $\overline{L}$, and, this proves the theorem. ■