4CS4-6: Theory of Computation (Undecidability)

Lecture 24: April, 16, 2019

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

24.1 Unsolvable problem of elementary number theory

There is a class of problems in elementary number theory which can be stated as: it is required to find an effectively calculable function $f(x_1, x_2, ..., x_n) = 2$, where arguments are positive variables, is necessary and sufficient condition for the truth of certain proposition of elementary number theory having $x_1, x_2, ..., x_n$ as free variable.

An example of such as problem is to find a means of determining of any given positive integer n, whether or not there exists positive integers x, y, z, such that $x^n + y^n = z^n$. This problem may be interpreted as, it is required to find an effectively calculable function f, such that $f(n)$ is equal to 2 if an only if there exists positive integers x, y, z, such that $x^n + y^n = z^n$. Clearly, the condition that the function f be effectively calculable is an essential part of the problem, since without it the problem becomes trivial.

24.2 Unsolvability

There are infinitely more problems than algorithms, and there are in fact infinitely more problems than machines. So building a machine that is more powerful than computer, would not help a bit. There would still be an infinite number of problems it would not be able to solve. In the next we list the steps to define such a problem.

Defining the problem The next problem is to show a one-to-one function from the set of problems on to the set of real numbers $[0, 1]$. We do this by concatenating the output digits of all possible inputs (one to infinity) of the problem of primality test mentioned above. The problem can be stated as,

Input: Natural Number
Output: 1 if input is prime and 0 otherwise.

For input 1, output is 0, for input 2 the output is 1, for input 3, output is 1, and so on. The real that corresponds to this problem is 0.0110101000101..., which is infinite in length. We have given it only for the first 13 integers. We may consider that for infinitely large number

of problems, every problem will have some kind of representation. Like this, every problem has a unique representation.

The next step is to show how we compare cardinality of infinite sets. Obviously, the problem, represented by a sequence like this real number, are *uncountably infinite*. Whereas, the number of algorithms (step 3 above), are *countably infinite*. Using *Cantor's diagonalization* method (chapter 1), we conclude that there are more real numbers between 0 and 1, than (the number of algorithms).

After the described steps, we conclude that there are infinite numbers of problems that cannot be solved by a computer.

24.3 Unsolvability of Diagonal Language

In the following discussions, our objective will be those problems which cannot be solved by computer. In other words, we will be able to show that no Turing Machine exists to solve such problems.

Consider that M is a TM with alphabet $\Sigma = \{a, b\}$ and $w \in \Sigma^*$, and M accepts w. We had discussed that a TM M can be encoded as $en(M)$, and its input w as $en(w)$, accordingly, for the alphabet Σ, the encoding is:

$$\begin{aligned} &\delta(q_i, a) = (q_j, b, L) \\ &= 001^{i+1}01^201^{j+1}01^301^200 \end{aligned} \tag{24.1}$$

Similarly, all the transitions can be encoded. Hence $\langle M, w\rangle$ is encoded as $00m_i\ 00m_j00\ldots 00m_k$, where m_i is a transition, like shown in equation 24.1. We note that all the transitions of a TM, separated by 00 is a big integer number. If this process is followed, we can encode all the possible TMs. If a string is not a well-formed representation of any TM, we take this as encoding of a TM with null moves, hence every binary string, whether long or short, can be taken as representation of some TM. Therefore, all Turing machines are represented by integers $\{1, 2, 3, \ldots, i, \ldots\}$, which is enumeration of Turing machines M_i, and are the countable infinite numbers.

Consider that language for a TM M_i is $L(M_i)$. For example, in typical case the language is $M_i = \{1, 10, 110, 111, 1001, \ldots, \}$. This is subset of lexicographic order binary set: $M_u = \{1, 10, 11, 100, 101, 110, 111, 1000, 1001, \dot{\}}$. In a encoded representation of language M_i, we indicate the presence of string w in M_u as 1 and absence of that as 0. Hence, the encoded representation of above M_i is $\{1, 1, 0, 0, 0, 1, 1, 0, 1, \ldots\}$. Hence, if we construct a matrix of $M_i \times w_j$, then $(i, j) = 1$ if w_j is accepted by M_i, else 0. So, ith row is a characteristic vector for language $L(M_i)$ (Fig. 24.1).

We think of the diagonal vector for the above figure, which is $[1, 1, 1, 0, 1...]$, and take this as a characteristic vector, the corresponding language is $\{1, 10, 11, 101, \dot{\}}$, which is obvious by the definition of language we defined in the beginning of this discussion. As next step take the *complement* of this diagonal vector, which is $\{0, 0, 0, 1, 0, \ldots\}$.

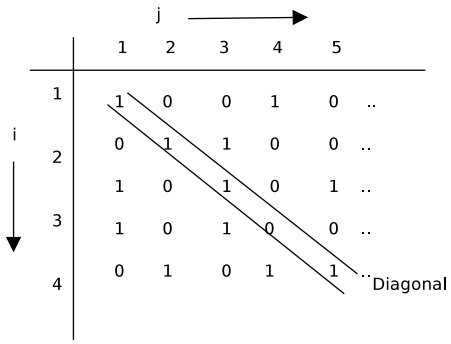

Figure 24.1: Diagonalization

We note, that the the complement disagrees with every row vector, because the first element of the complement disagrees with the first element of first row, and similarly it disagrees with all the rows we have listed. Hence, it represents a language which is not accepted by any TM. The complement of the diagonal vector cannot be characteristic vector of any TM, because the TMs are enumerated. Let the language of this complemented characteristic vector be L_d. This language is undecidable, as there does not exists any TM which can recognize it.

Let us assume that L_d is some language. Therefore, it should appear as some string w_j. But, we note that $L_d \notin \{w_k\}$, where w_k is some language. This is a contradiction. So, there does not exist any TM which recognizes the L_d, hence L_d is undecidable.

The Church-Turing Thesis (CTT) also has consequence for undecidability. If a problem cannot be solved by any TM, then it cannot be solved by any algorithm. A decision language having no algorithmic solution, is called *undecidable*.

There are countable TMs (i.e., Algorithms), but the number of problems are uncountable. It follows that there are languages whose membership problems are undecidable.

24.4 Undecidability of Halting Problem

In general there exists no Turing machine that eventually halts and decides YES/NO for a given problem, then the problem is called as *undecidable*. The problem called the *halting problem*, seeks the answer to the question, whether any Turing machine will eventually halt or run forever. We will show in the remaining discussion in this section that halting problem is undecidable.

At the first instance the halting problem seems decidable. A Turing machine M with input w yielding YES, when w is accepted, and NO, otherwise. This machine with input w is simulated on the Universal Turing machine M_u, so M_u should yield YES and halt when simulated M accepts w. But, when M with input w reject, then the Universal Turing machine M_u runs forever, and hence cannot output NO ever. So we cannot solve the halting problem with the universal Turing machine running M.

Theorem 24.1 *Show that Halting problem is undecidable.*

Proof: Let us assume that there is a TM M that accepts input w with $w \in L(M)$, and rejects $w \notin L(M)$. To model the halting problem, let us assume that there is a TM H that decides this. That is, when H is simulated to run M with input w, so that when M prints "accept", H will also print "accept", and when M print "reject", the H will also print "reject". For this we feed representation $R(M)$ and w as input to H (see Fig. 24.2(a)).

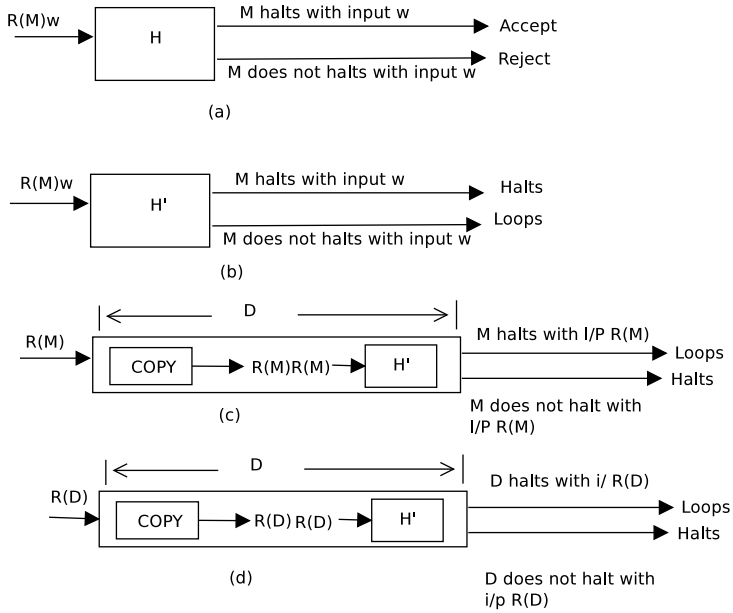

Figure 24.2: Halting Problem

Next, we do minor modification in H, as shown in Fig. 24.2(b), where accept is replaced by halts, and reject is replaced by loop. The modified H is shown as H'. The first is simple, as only the message content has changed. For the reject case, when M rejects w, the H' calls a loop.

In the Fig. 24.2(c), we only input the representation $R(M)$, and by a "copy" TM, another similar string is obtained, so in place of w we have input to H' as $R(M)$. In addition, the outputs: loop and halt are exchanged. Let this modified machine is D (for Diagonalization), as we did complement of the diagonal language L_d (see page no. **??**). And therefore, the complement is not arbitrary.

Since $R(M)$ was input to TM M simulated by H' earlier, we can also feed $R(D)$ as input to D machine, just like $R(M)$. With this final modification, we have shown the machine in Fig. 24.2(d).

Now we analyze the output of D, and appropriately restate as follows:

1. *If D halts with input $R(D)$ then D loops,*
2. *If D does not halt with input $R(D)$ then D halts.*

In other words, we note that "D loop if D halts with input $R(D)$", and "D halts with input $R(D)$ if, D does not halt with input $R(D)$". This is a self contradicting statement. Therefore, the halting problem is "undecidable", i.e., when a machines receives its own program (encoding) as input, we cannot decide whether it will halt or not. ■