Theory of computation **Fall 2015**

Lecture 20: Undecidability

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

20.1 Undecidability

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:

$$\begin{aligned}&\delta(q_i, a) = (q_j, b, L)\\ &= 001^{i+1}01^{2}01^{j+1}01^{3}01^{2}00\end{aligned} \tag{20.1}$$

Similarly, all the transitions can be encoded, hence $\langle M, w\rangle$ is encoded as $00m_i00m_j00\dots00m_k$, where m_i is a transition, like shown in equation 20.1. 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 $\{1, 2, 3, \dots, i, \dots\}$, which is enumeration of Turing machines M_i, and are the countable infinite numbers.

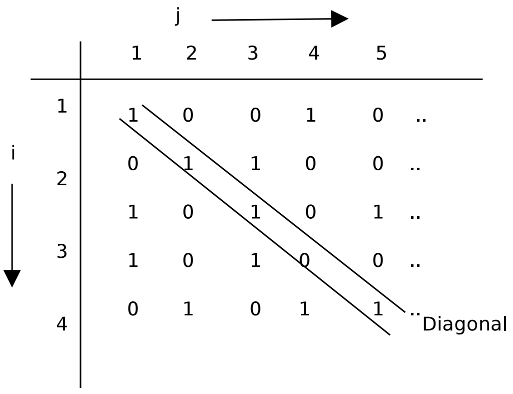

Figure 20.1: Diagonalization.

Consider that language for a TM M_i is $L(M_i)$. For example, in typical case the language is $M_i = \{1, 10, 110, 111, 1001, \dots,\}$. This is subset of lexicographic order binary set: $U = \{1, 10, 11, 100, 101, 110, 111,$

$1000, 1001, \dot{\}}$. In a encoded representation of language M_i, we indicate the presence of string w in U as 1 and absence of that as 0. Hence, the encoded representation of above M_i is $\{1, 1, 0, 0, 0, 1, 1, 0, 1, \dots\}$. 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 $L(M_i)$ (figure 20.1).

For this figure, the diagonal vector is [1, 1, 1, 0, 1 ...]. For this diagonal vector, considering it as a characteristic vector, the corresponding language is $\{1, 10, 11, 101, \dot{\}}$. Now take this diagonal vector and *complement* it, we get it as $\{0, 0, 0, 1, 0, \dots\}$. We note, that it disagrees with every row vector, hence, 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 $L_d \notin \{w_i\}$, where w_i 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 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.

The first problem we consider is "Halting problem" for Turing machines. The halting problem is like a C program, named as "cp" whose input is a file, called "infile", and, output is "Yes" if cp halts when run with infile as input. And, the output is "No" if with input of infile, the cp program does not halt.

Having given above program and input file, we are interested to write another program $HALT$, whose inputs are *cp* and $infile$, and $HALT$ decides whether *cp* halts or not, based on its input the *cp* and $infile$. That is, $HALT$ will print yes, if *cp* prints Yes, an $HALT$ prints No if *cp* prints No. The halting problem states that that: $\langle cp, infil\rangle$ is *undecidable*, as a general case, that is, $HALT$ cannot decide about the outcome of cp with input of infile.

If the above is decidable, then there cannot an infinite loop.

20.2 Barber's Paradox

A barber in a small town declares that he shaves every one in this town who does not shave himself, and that he does not shave any one who shaves himself. We will see that barber's declaration leads to contradiction. This contradiction is heart of the argument to prove that the halting problem is unsolvable.

Let us see this barber's paradox. Let X denote how the barber behaves, $X = 1$ means to shave a villager, and $X = 0$ means not to shave. Similarly, Y denotes how the villager behaves, $Y = 1$ means to shave himself, and $Y = 0$ means not to shave himself.

Clearly, what barber declared is for any villager including himself, so (X, Y) must be $(1, 0)$ or $(0, 1)$.

On the other hand, the barber shaves "not only as barber", but also as "one of the villager", i.e., himself, which is shaves: to the barber, and also shaves to villager. Therefore, if barber shaves himself, (X, Y) becomes $(1, 1)$. Where as when barber does not shave himself, (X, Y) becomes $(0, 0)$. In either cases, this contacts what barber has declared.

If the barber is not treated as villager, the contradiction does not occur. The contradiction in above results to *self reference* in the declaration.

20.3 Halting Problem for Turing Machine

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" in the above case. For this we feed representation $R(M)$ and w as input to H (see fig. 20.2(a)).

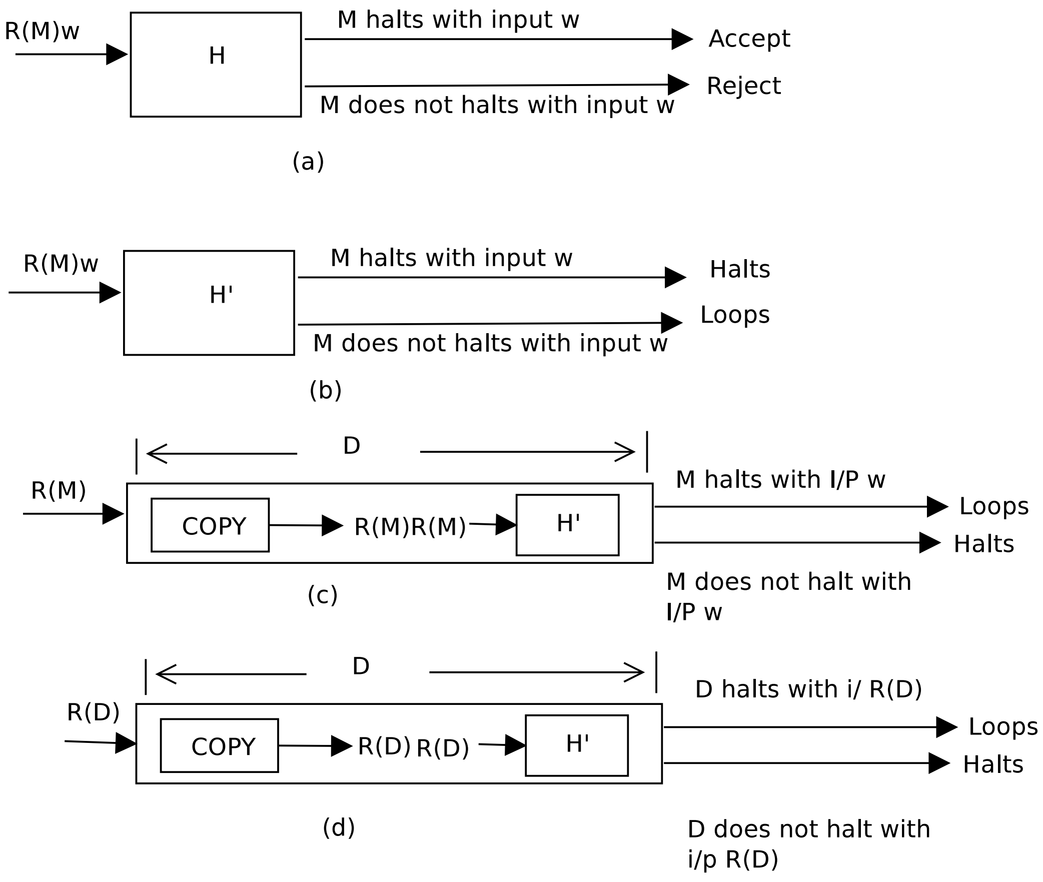

Figure 20.2: Halting Problem.

Next, we do minor manipulations, shown in fig. 20.2(b), where accept is replaced by halts, and reject is replaced by loop. The first is simple, as message content has changed, and when M rejects, the H calls a loop. We refer this modified machine as H'.

In the fig. 20.2(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 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.

Since $R(M)$ was input to TM M simulated by H' earlier, we can also feed $R(D)$ as input to D machine. With this final manipulation, we have shown the machine in fig. 20.2(d).

Now let us analyze the output of D, we note that:

If D halts with input $R(D)$ then D loops,

If D does not halt with input $R(D)$ then D halts.

In other words, we note that "D halts with input $R(D)$ if and only 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.

The contradiction in the above proof uses self reference and Diagonalization. To obtain the standard relation table for a Diagonalization argument, we consider every string $v \in \{0,1\}^*$ to represent TM. If v does not have the form $R(M)$, the one state TM with no transition is assigned to it v.

Thus, Turing machines can be listed as M_0, M_1, …, corresponding to strings ε, 0, 1, 00, 10, …. Now consider the task that lists the table along the horizontal and vertical lines. The (i,j)th entry is:

$$\begin{cases} i, & \text{if } M_i \text{ halts when run with } R(M_j) \\ 0, & \text{if } M_i \text{ does not halts when run with } R(M_j) \end{cases} \tag{20.2}$$

Corollary 20.1 *A language $L_H = \{R(M)w|R(M)$ is representation of TM M and M halts over input $w\}$ is not Recursively Enumerable. Since it is not RE, it is also not R (recursive).*

The Diagonal of the table represents the answers to the self reference question: "Does M_i halt when run on itself?" The machine D was constructed to produce contradiction in response to this question.

In general if there exists a TM that eventually halts and decides Yes/No for a given problem, the problem is called *decidable*. The problem called "halting problem" to tell whether TM will eventually halt or run for ever is such a problem, i.e., undecidable. In the above we have concluded that halting problem is undecidable.

At first instance, the halting problem seems decidable. It seems reasonable that we can let universal TM U solve the hating problem by running U for an arbitrary TM M together with input w and producing "Yes" when U halts in the course of simulation. In fact, output Yes is correct as long as M with input w eventually halts. But if the machine M with input w runs forever, the Universal Turing machine U will also run forever, and hence can never output "No" forever. So we cannot solve the halting problem by simply running the Universal Turing machine. But, this is not the proof. The proof is shown in systematic transformation of the Universal TM, as shown above.

Theorem 20.2 *The language L_H=$\{R(M)w \mid R(M)$ is representation of M, with input $w \in \{0,1\}^*$ is not recursive but recursively enumerable (RE)$\}$.*

Proof: *The proof is left as an exercises (see fig. 20.3).*

Consider that P is a program, and x is a string over input alphabet, which we fix as $\{0,1\}$. Following are some fundamental problems because they are inherently important or because they have played historical role in development of computation theory.

1. Universal Simulation: Given a program P and input x to P, determine the output (if any) that P would produce on input x.

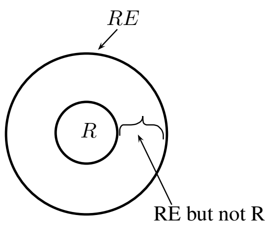

Figure 20.3: RE and R languages.

2. Halting Problem: Given P and x, output 1 (for yes) and if P would halt - what input x, and output null, if P would not halt.

3. Type-0 grammar membership: Given a type-0 grammar G and a string x, determine whether x can be derived from start symbol of G.

Theorem 20.3 *The diagonal language L_d is NOT recursively enumerable.*

Proof: Suppose that for the sake of contradiction that L_d is RE. Then there is a TM M' that accepts L_d. Now what does M' do on input $w(= R(M'))$? If M' accepts w, i.e., $R(M')$, then $R(M') \notin L_d$. But this contradicts $L(M') = L_d$. Hence, the TM M', such that $L(M') = L_d$ cannot exist. So, L_d is not RE. □

The definition of L_d is motivated by Russell's paradox, reading "$\notin$" as "does not accept". Where as in Russell's paradox we had to conclude that S is not a set, here we conclude that L_d is not a Turing-acceptable set.