**CS 208 : Automata Theory and Logic** DATE

# Lecture - 36

Topic: Recursive and Recursively Enumerable Languages

*Scribed by*: K.S.Shriram Kumar (22B1280)
*Checked and compiled by*:

## 1 Recap : Halting Problem

### 1.1 Encoding of Turing machines

As we saw in the last few lectures, every turing can be encoded as a binary string of '1's and '0's and thus as a natural number. While every natural number's binary encoding need not be a valid turing machine, such strings can be thought of as turing machines that do not take any steps regardless of input string. Thus every integer corresponds to a turing machine and vice versa[1].

### 1.2 Formulating the Problem

Let $M_i$ denote a turing machine which is encoded as the integer $i$ in binary form. Similarly let $w_j$ be the integer $j$ encoded in binary form as a string of '0's and '1's. Thus our fundamental question about turing machines (ie) "will a string 'w' halt on a turing machine 'M'" can be reformulated as whether '$w_j$' halts on '$M_i$' where 'i' and 'j' represent the integers associated with encoding of '1's and '0's for the word and the turing machine respectively. This can represented as just a pair of integers $(i, j)$ whose meanings are told above.

### 1.3 Diagonal Language

Imagine a 2D matrix of infinite dimensions. Each row of this matrix corresponds to $M_i$ and each column corresponds to $w_i$ as denoted above. The entries of the matrix corresponds to the formulation as told above (ie)

$$(i, j)^{\text{th}} \text{ entry set to } 1 \Rightarrow \text{will a string '}w_j\text{' halt on a turing machine '}M_i\text{'}$$

An entry is set to 1 iff the statement turns out to be true and it is set to 0 otherwise. Let's take the diagonal entries of the matrix which represent pairs of type $(i, i)$. We get a sequence of '1's and '0's corresponding to each $(i, i)$ and this $i$. Let's complement this sequence, whenever we see a '1' we replace it with a '0' and vice versa. The language thus obtained is called the "Diagonal Language". To be precise the language here is all the $(i, i)$ pairs that have their corresponds entry in the matrix we described set to '0'. Since both the elements of the pair are the natural number the elements of this language can be represented as a single integer $i$. Thus,

$$L_d = \{w_i \mid i \in N,\ w_i \text{ doesn't halts on } M_i\}$$

[1]note that this is not neccesarily a bijectiion

$$
\begin{array}{c|ccccccc|}
 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & \\
w_1 & 1 & 0 & 1 & 1 & 0 & 1 & \cdots \\
w_2 & 0 & 1 & 0 & 0 & 1 & 0 & \cdots \\
w_3 & 1 & 0 & 1 & 1 & 0 & 1 & \cdots \\
w_4 & 0 & 1 & 0 & 0 & 1 & 0 & \cdots \\
w_5 & 1 & 0 & 1 & 0 & 1 & 1 & \cdots \\
w_6 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

Figure 1: Matrix to represent $i, j$ pairs

### 1.4 Acceptance of diagonal language

Does there exist a turing machine $M_k$ that accepts by halting, the language $L_d$. Assume such a during machine exists. As per our assumption it can be encoded as the integer $k$.

Does this turing machine accept the word $w_k$ or does it not? That is on taking the word $w_i$ as an input does our turing machine halt?

There are two possibilites. The word either halts or it doesn‘t.

- **Doesn't Halt:** If the turing machine does not accept this word it implies that $w_k$ doesn't halt on the turing machine $M_k$. According to the definition of $L_d$ it belongs to the language and thus by the definition of $M_k$, $w_k$ halts on $M_k$.
- **Does Halt:** If the turing machine accepts this word it implies that $w_k$ halts on the turing machine $M_k$. According to the definition of $L_d$ it doesn't belong to the language and thus by the definition of $M_k$, $w_k$ doesn't halts on $M_k$.

In both cases the string is accepted and not accepted by $M_k$ which is clearly not possible. We have a contradiction and thus our assumption that such a turing machine exists is invalid. Since we have an example of a Language which can never be accepted by a turing machine, we can state that not all languages are neccesarily accepted by a turing machine.

## 2 Recursively Enumerable Language

A lanuguage $L$ is called a Recursively Enumerable or (RE) Language iff there exists a turing machine that halts on taking words from the language as an input. That is a language is Recursively Enumerable iff $\exists M_i$ such that $L = H(M_i)$ where $H(M_i)$ is the halting Language[2] of $M_i$.

Obviously all context free languages and regular languages are also recursively enumerable since they can be represented by turing machines. For regular languages the turing machine halts if the language is accepted else it keeps reading blanks indefinitely. For context free languages the turing machine can mimic its corresponding Pushdown Automata with the tape being used for the stack.

[2]words which when given as an input to $M_i$ make it halt in finite number of steps

### 2.1 Is $\overline{L_d}$ recursively Enumerable?

As we discussed above $L_d$ is not recursively enumerable since we cannot have a turing machine which can accept $L_d$ by halting. What about the complement of this set?

$$\overline{L_d} = \{w_i \,|\, i \in N,\, w_i \text{ halts on } M_i\}$$

To show that such a language is recursively enumerable we need to construct a turing machine whose halting language is the same as $\overline{L_d}$. This can be achieved if on reading input $w_i$ we simulate a run of the turing machine $M_i$. This works because when a word is present in $\overline{L_d}$ by definition we know that the turing machine $M_i$ accepts it by halting. Is such a turing machine possible?. It is possible however the proof is involved and one can refer to Section 9.2.3 in this book.

### 2.2 Universal language

To prove that $\overline{L_d}$ is a recursively enumerable language we ended up constructing a 'Universal turing machine' which takes in a string of the form $w_i\#w_j$ and can simulate a run of the string $w_j$ on the turing machine $M_i$. We can also define a Universal language denoted by $L_u$ as the strings accepted by this Universal turing machine.

$$L_u = \{w_i\#w_j \,|\, w_j \in H(M_i)\}$$

A turing machine to accept $\overline{L_d}$ could just be one that takes in $w_i\#w_j$ and apart from running the Universal turing machine also runs a check to see if $i = j$.

### 2.3 Complement of Universal Language

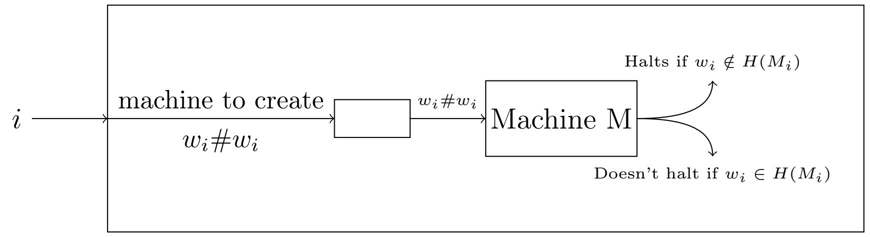
The Universal language is recursively Enumerable since the Universal turing machine accepts it. What about its complement? Note that the notion of complement is not the usual definition here. The complement of $L_u$ denoted by $\overline{L_u}$ would be

$$\overline{L_u} = \{w \,|\, w \notin \{w_i\#w_j | i, j \in N\} \text{ or } w_j \notin H(M_i)\}$$

However we are only interested in a version of $\overline{L_u}$ restricted to elements of the type $w_i\#w_j$. Thus the set becomes.

$$\overline{L_u} = \{w_i\#w_j \,|\, w_j \notin H(M_i)\}$$

Now lets see if $\overline{L_u}$ is Recursively enumerable. Assume the existence of such a turing machine say $M$. Such a turing machine can be used to accept the language $L_d$. For a given input $i$ construct $w_i\#w_i$ and run it on M.



$i$
machine to create $w_i\#w_i$
$w_i\#w_i$
Machine M
Halts if $w_i \notin H(M_i)$
Doesn't halt if $w_i \in H(M_i)$

Figure 2: Machine to accept $L_d$ given M

## 3 Recursive Languages

Take a recursively enumerable language and its corresponding turing machine. When we run an arbitrary string w on the machine, if it belongs to the language it the machine eventually. However if it doesn't belong in the language it runs indefinitely. We do not have any information on how long the machine will run till it halts to accept a language. Thus it is impossible to comment if a string is not in the halting language. Say you run the turing machine for i steps it could be that the machine halts after i + 1 steps. Thus any number of finite steps cannot guarantee a string is not in the language.

It is thus better to have a turing machine which halts regardless of if or not the input is in the corresponding language. Rather acceptance is returned as an yes or no output by the turing machine itself. For a turing machine M, $L(M)$ refers to all the strings which when inputted to the machine make the machine halt in a finite number of steps and output yes on the tape.

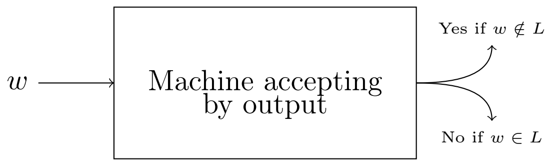


Figure 3: Machine to accept L by outputting yes

A turing machine which accepts by final state can easily be made into the above described machine since after reaching final state we can just transition to a set of states which can output yes or no.[3] A language is said to be **recursive** if it has such a turing machine to represent it.

Formally a language is said to be **recursive** iff $\exists M$ such that $H(M) = \Sigma^*$ and $L(M) = L$.

### 3.1 Complementation of Recursive languages

As we saw clearly Recursively enumerable languages are not closed under complementation. A clear example is $L_d$ and $\overline{L_d}$. What about recursive languages?

Its easy to see that they are closed under complementation. Assume a recursive language $L_k$ exists and has a turing $M_k$ such that $L(M_k) = L_k$. What could happen if a word from $\overline{L_k}$ is fed into the machine. We would get a no output. So to get a machine such that $L(M_{k'}) = \overline{L_k}$ we just need to input our string into $M_k$ and then complement the output. Since this can be easily done with a turing machine we can conclude that $\overline{L_k}$ is recursive.

### 3.2 Recursive $\rightarrow$ Recursively Enumerable

It is easy to see that if a language is recursive then it is also recursively enumerable since a turing machine which accepts by outputting yes can just be converted into one to accept the same language via halting. Whenever we get a no output we can make the turing machine go into an infinite loop and terminate for yes inputs. But the otherway is not so obvious.

As we saw recursive languages are closed under complementation. Thus $\overline{L_d}$ cannot be recursive. If it was recursive then its complement (ie) $L_d$ will also be recursive. This means that $\overline{L_d}$ is also recursively enumerable but we have proved that it is not. We thus have a contradiction and $L_d$ is not recursive. However it was recursively enumerable.

[3]It may run in infinite loops for unaccepted strings

Thus being a recursive language is a stronger condition than a language being recursively enumerable.

## 4 Decidable languages

A language is sais to be **decidable** if it is recursive, else it is said to be undecidable. We also have semidecidable languages which are recursively enumerable but not recursive such as $\overline{L_d}$

Why do we care about decidable languages? As said above we care about languages such that we can determine if an arbitrary word is a member of it in finite time. This is because our computers cannot run for infinite time for us to get results. The existence of undecidable languages tells us that regardless of our model we will be unable to solve some problems involving such languages. This is an inherent problem in computation which cannot be fixed.