4CS4-6: Theory of Computation (Recursive and Recursively Enumerable Langs.)

Lecture 23: April, 20, 2020

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

23.1 Introduction

A language acceptable by Turing machine is called *Recursively Enumerable* (RE), which means that the set of strings in the language accepted by the Turing Machine can be enumerated. Recursively enumerable language is a type of formal language which is also called *partially decidable* or *Turing-recognizable.* It is known as *type-0* language in the Chomsky hierarchy of formal languages. The RE languages are always countably infinite. The class of RE languages has a broad coverage of languages, and they include some languages, which cannot be defined by a mechanical algorithm. TMs will fail to halt on some input not in these languages. If w is a string in RE language then the TM will eventually halt on w.

There exists three equivalent major definitions for the concept of a RE language.

Definition 23.1 RE Language. *A RE formal language is a recursively enumerable subset in the set of all possible words over the alphabet of the language.*

Definition 23.2 RE Language. *A RE language is a formal language for which there exists a Turing machine (or other computable function) which will enumerate all valid strings of the language.*

Note that, if the language is infinite, the enumerating algorithm provided can be chosen so that it avoids repetitions, since we can test whether the n-th string produced is "already" produced for some number less than n. If it is already produced, use the output for input $n + 1$ instead (recursively), but again, test whether it is "new".

Definition 23.3 RE Language. *A RE language is a formal language for which there exists a Turing machine (or other computable function) that will halt and accept when presented with any string in the language as input. But may either halt and reject or loop forever when presented with a string not in the language.*

Contrast this to **recursive** languages, which require that the Turing machine halts in all cases.

All regular, context-free, context-sensitive and recursive languages are recursively enumerable. The RE languages together with their complement *co-RE*, form the basis for the *arithmetical hierarchy.*

However, if M is still running on some input, we can never tell whether M will ultimately accept if it is allowed to run long enough or M will run forever. Therefore, it is appropriate to separate the subclass of RE languages accepted by at least one TM that halts on all inputs. However, halting may or may not be preceded by acceptance.

23.2 Recursive and Recursively Enumerable sets

We wish to study recursively invariant properties of sets of integers, in particular those properties which have to do with *solvability* and *unsolvability*. Most basic is the property of possessing a recursive function.

Definition 23.4 Recursive. *A set is recursive if it possesses a recursive characteristic function. That is to say, set A is recursive if and only if there exists a recursive function f such that for all x, $x \in A \Rightarrow f(x) = 1$, and $k \notin x \Rightarrow f(x) = 0$.* $\square$

Intuitively, a set A is recursive if there exists an effective procedure for deciding, given any x, whether or not $x \in A$.

Following are the recursive sets:

i. The set $\{0, 2, 4, ...\}$ of even numbers;

ii. $\mathbb{N}$ and ϕ;

iii. Any finite set;

iv. Any set with finite complement.

The sets (iii) and (iv) are recursive, since explicit listing of the appropriate finite set can be used to give instructions for the characteristic function.

There are $\mathbb{N}$ recursive sets. By cardinality, nonrecursive sets must exist (since there are total $2^{\mathbb{N}_0}$ subsets of $\mathbb{N}$ in all), and $2^{\mathbb{N}}$ is larger than $\mathbb{N}$.

Definition 23.5 Recursively Enumerable. *A set A is recursively enumerable if either $A = \phi$ or there exits a recursive function f such that $A =$ is range of f.* $\square$

Theorem 23.6 *If A is recursive, then A is recursively enumerable*

Proof: Case I. A is ϕ. Then, A is recursively enumerable by definition.

Case II. A is finite and $A \neq \phi$. Let $A = \{n_0, n_1, ..., n_k\}$.

Now, define f by,

$$f(x) = \begin{cases} n_x, & \text{for } x \leq k, \\ n_k, & \text{otherwise} \end{cases}$$

Case III. A is infinite. Let g be its characteristic function. Define f by

$$F(0) = \mu y.$$

$\blacksquare$

23.3 Recursive and Recursively Enumerable Languages

Definition 23.7 Recursive. *They allow a function to call itself. Or, a recursive language is a recursive subset in the set of all possible words over alphabet* Σ *of that language.* □

A language is *Recursive (R)* if some Turing machine M recognizes it and halts on every input string, $w \in \Sigma^*$. Note that Recognizable is equal to Decidable. Or A language is recursive if there is a membership algorithm for it.

Let L be a *recursive* language and M the Turing Machine that accepts (i.e. recognizes) it. For string w, if $w \in L$, then M halts in final state. If $w \notin L$, then M halts in non-final state, (*halts* always!)

Non-recursive should not be taken as simpler version of computation, i.e., e.g., obtaining factorial value without recursion method. We have the relation:

$$\begin{aligned}&\text{Regular languages} \subseteq CF \text{ languages} \subseteq CS \text{ languages}\\ &\qquad\subseteq R \text{ languages} \subseteq RE \text{ languages.}\end{aligned} \tag{23.1}$$

Definition 23.8 Recursively Enumerable. *A language is* Recursively Enumerable *(RE) if some Turing machine accepts it.* □

A Turing Machine M with alphabet Σ, *accepts* language L if,

$$L = \{w \mid w \in \Sigma^* \text{ and } M \text{ halts with input } w\}. \tag{23.2}$$

Let L be a RE language and M the Turing Machine that accepts it. Therefore, for $w \in L$, the TM M halts in final state, and for $w \notin L$, M halts in non-final state or *loops for ever.*

Relation between Recursive and RE Languages Every Recursive (R) language is Recursively Enumerable (RE). Therefore, if M is Turing Machine recognizing language L, which is R, then M can be easily modified so its accepts language that is RE. The languages which are *non-RE* cannot be recognized by TM. These are diagonal (L_d) languages of the diagonal of x, y, where x_i is language string w_i, and y_i is TM M_i.

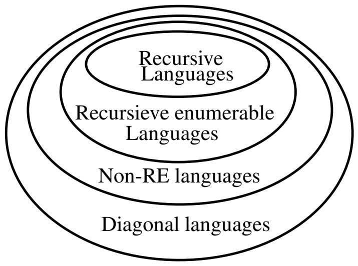

Figure 23.1: Relation between R and RE.

Theorem 23.9 *If a language L is recursive then there exists an enumeration procedure for it.*

Proof: If $\Sigma = \{a, b\}$, then M' can enumerate strings: $a, b, aa, ab, ba, bb, aaa, \ldots$.

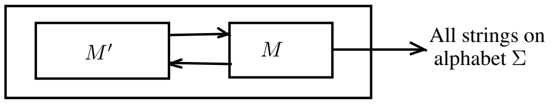

Figure 23.2: Enumeration of language.

The Enumeration procedure is as follows. Let M' generates string w (see Fig. 23.2). Next, the TM M checks, if $w \in L$ is "yes," then it outputs w else ignore w. Let $L = \{a, ab, bb, aaa, \ldots\}$. The M' output is $\{a, b, aa, ab, ba, bb, aaa, \}$, $L(M) = \{a, ab, bb, aaa, \ldots\}$, and enumerated output is $a, ab, bb, aaa, \ldots$ ■

The terms Turing Recognizable and RE are used interchangingly, as they ultimately mean the same. A language L is Turing recognizable if it is the language of some TM M (i.e., $L = L(M)$). A language is RE if there exists a TM M that can enumerate string in L, on its tape or say output tape, such that $x \in L$ is generated to appear in a finite amount of time.

23.4 Class of Languages

Following are the classes of languages:

- If a language is Recursive, then it is decidable, and the corresponding their Turing Machine always halts.
- Recursive Enumerable means semi-decidable, but not recursive, which implies that their Turing Machines always halt if they accept, otherwise halts in non-accepting state or loops.
- Non-recursively enumerable (*non-RE*) means there are no Turing Machine for them to recognize. See Fig. 23.1 for relations between languages.

The Recursive languages are closed under *complementation*, the following theorem proves this.

Theorem 23.10 *If L is recursive then $\overline{L}$ is also recursive.*

Proof: Following are the steps:

1. The accepting states of M are made non-accepting states of M' with no transitions, i.e., here M' will halt without accepting.
2. If s is new accepting state in M', then there is no transition from this state.

3. If L is recursive, then $L = L(M)$ for some TM M that always halts. Transform M into M' so that M' accepts when M does not and vice-versa. So M' always halts and accepts $\overline{L}$. Hence $\overline{L}$ is recursive.

The Fig. 23.3 explains the process.

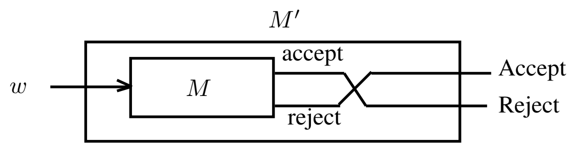

Figure 23.3: Co-Recursive is Recursive.

■

Theorem 23.11 *If L and $\overline{L}$ are* RE*, then L is Recursive.*

Proof: Let $L = L(M_1)$ and $\overline{L} = L(M_2)$. Construct a TM M that simulates M_1 and M_2 in parallel, using two tapes and two heads. If input to M is in L, then M_1 accepts it and halts, hence M accepts it and halts. Otherwise, if input to M is not in L, hence it is in $\overline{L}$, therefore, M_2 accepts and halts, hence M *halts* without accepting. Hence, M halts on every input and $L(M) = L$. So L is recursive (see Fig. 23.4).

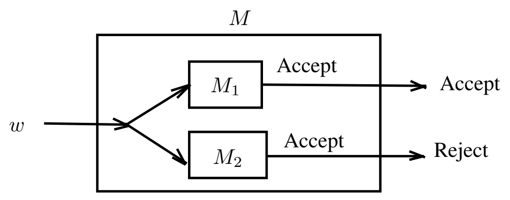

Figure 23.4: L and $\overline{L}$ are *RE*, then L is Recursive.

■

Closure Properties of Recursive Languages Set Recursive languages are closed under following set operations,

- Union,
- Concatenation,
- Intersection,
- Kleene star,
- Complement, and
- Set difference $(L_1 - L_2)$.

Theorem 23.12 *A language L is recursively enumerable iff there exists an enumeration procedure for it.*

Proof: If there is an enumeration procedure, then we can enumerate all the strings, and compare each with given string w each time till it is generated. If the language is RE, then we can follow an enumeration procedure to systematically generate all the strings (see Fig. 23.5).

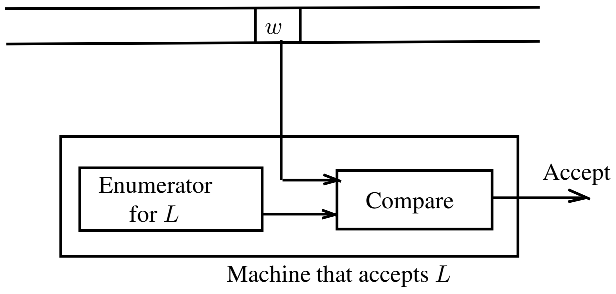

Figure 23.5: Enumeration machine for language L.

The algorithm for enumeration of RE language is as follows:

```
while (1) {
generate();
compare();
if same exit();
}
```

■

Intersection of *RE* and *R* Languages Given a *Recursive* and a *RE* languages: Their Union is *RE*, Intersection is *RE*, Concatenation is *RE*, and Kleenes closure is *RE*. If L_1 is *Recursive* and L_2 is *RE*, then $L_2 - L_1$ is *RE* and $L_1 - L_2$ is not *RE*.

Theorem 23.13 *The intersection R and* RE *languages is RE.*

Proof:

The proof is as follows.

1. Let L_1 and L_2 be languages recognized by Turing machines M_1 and M_2, respectively.
2. Let a new TM $M_\cap$ is for the intersection $L_1 \cap L_2$. $M_\cap$ simply executes M_1 and M_2 one after the other on the same input w: It first simulates M_1 on w. If M_1 halts by accepting it, $M\cap$ clears the tape, copies the input word w on the tape and starts simulating M_2. If M_2 also accepts w then $M_\cap$ accepts.

3. Clearly, $M_\cap$ recognizes $L_1 \cap L_2$, and if M_1 and M_2 halt on all inputs then also $M_\cap$ halts on all inputs.

■

Theorem 23.14 *The union of two Recursive languages is recursive.*

Proof:

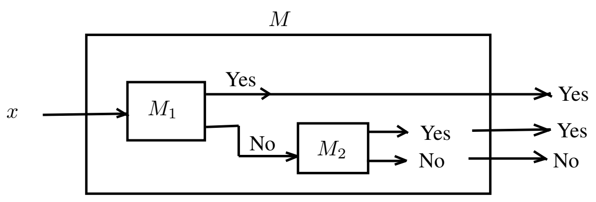

Figure 23.6: Union operation of Recursive Languages.

The TM corresponding to this must halt always. Let L_1 and L_2 be sets accepted by M_1 and M_2, respectively. Then $L_1 \cup Ł_2$ is accepted by TM M, where $x = w_1 \cup w_2$, for $w_1 \in L_1$ and $w_2 \in L_2$ (see Fig. 23.6). ■

Theorem 23.15 *The union of two* RE *languages is* RE.

Proof: Let L_1 and L_2 be sets accepted by M_1 and M_2, respectively. Then $L_1 \cup Ł_2$ is accepted by TM M, where $x = w_1 \cup w_2$, for $w_1 \in L_1$ and $w_2 \in L_2$.

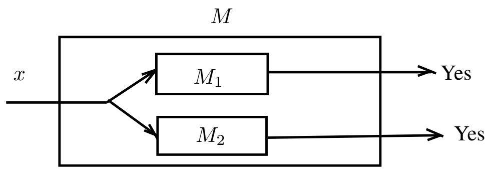

Figure 23.7: Union of two *RE* languages.

To determine if M_1 or M_2 accepts x we run both M_1 and M_2 simultaneously, using a two-tape TM M. The M simulates M_1 on the first tape and M_2 on the second tape. If either one enters the final state, the input is accepted (see Fig. 23.7). ■