

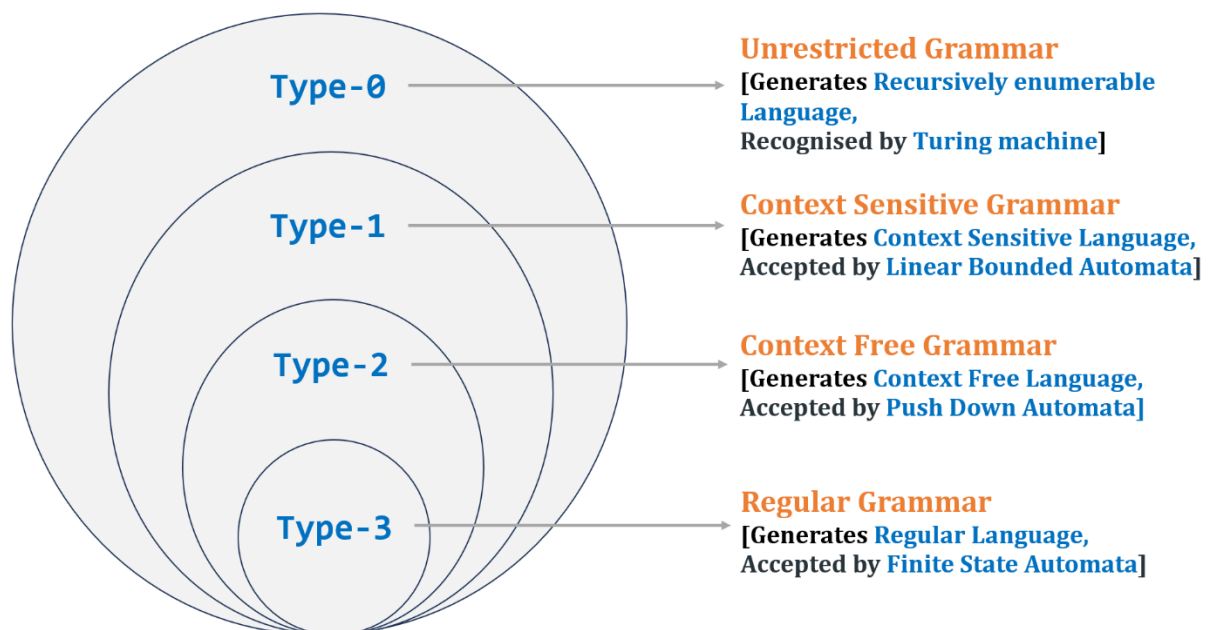


Department of Computer Science and Engineering  
PES University, Bangalore, India

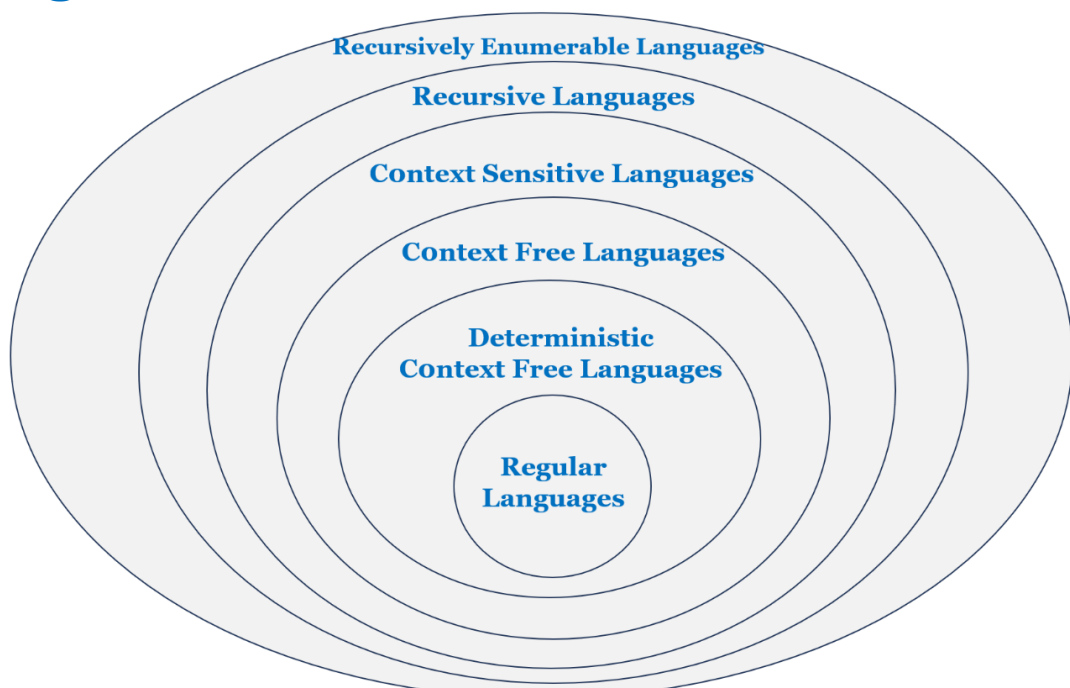
## UE23CS243A: Automata Formal Language and Logic

Prakash C O, Associate Professor, Department of CSE

### Chomsky hierarchy



### All languages over $\Sigma$



## Note:

### 1) Type-0 grammar (or unrestricted Grammar)

- Unrestricted grammars are the most general type of grammar in the Chomsky hierarchy. They are also known as phrase structure grammars.
- A grammar  $G = (V, T, P, S)$  is called unrestricted if all the productions are of the form  $u \rightarrow v$ , where  $u \in (V \cup T)^+$  and  $v \in (V \cup T)^*$ .
- **No restrictions are made on the production rules of an unrestricted grammar**, other than each of their left-hand sides being non-empty.
- **Any language generated by an unrestricted grammar is recursively enumerable.**
- The unrestricted grammars characterize the recursively enumerable languages. This is the same as saying that for every unrestricted grammar  $G$  there exists some Turing machine capable of recognizing  $L(G)$  and vice versa.

### 2) Decidable language

A language  $L$  is called decidable if there exists an algorithm (or, equivalently, a Turing machine) that:

- given an input string,
- returns “yes” or “no” depending on whether this word belongs to this language or not.

### 3) Recursively Enumerable Set (Countable set)

In computability theory, **a set  $S$  is called countable**, computably enumerable (CE), **recursively enumerable (RE)**, semi-decidable, partially decidable, listable, provable or **Turing-recognizable** if:

- **There is a procedure that enumerates the members of set  $S$ .** That means that its output is simply a list of all the members of  $S$ :  $s_1, s_2, s_3, \dots$ . If  $S$  is infinite, this procedure will run forever.
- **If its elements can be put into a one-to-one correspondence with the natural numbers.** By this we mean that the elements of the set can be written in some order, say,  $x_1, x_2, x_3, \dots$ , **so that every element of the set has some finite index.**
- If it has a bijection (one-to-one correspondence i.e., we can index each element) with the natural numbers, and is computably enumerable (CE) if there exists a procedure that enumerates its members. Any non-finite computably enumerable set must be countable since we can construct a bijection from the enumeration.
- **We can generate any element** (irrespective of how large the element is) **of the set  $S$  in finite amount of time.**

Not every set is countable. There are some uncountable sets. But **any set for which an enumeration procedure exists is countable because the enumeration gives the required sequence.** Strictly speaking, an enumeration procedure cannot be called an algorithm since it will not terminate when  $S$  is infinite.

### Examples of Recursively Enumerable (or Countable) sets:

1. **Set of odd/even numbers**
2. **Set of prime numbers**
3.  **$\Sigma^*$  - Set of all strings over the input alphabet  $\Sigma$**

**Answer:** If  $\Sigma = \{a, b\}$ , How should we order  $\Sigma^*$  to show that it is countable?

We cannot use the sequence  $\lambda, a, aa, aaa, aaaa, \dots$

because then b's strings would never appear. This does not imply that the set is uncountable; in this case, there is a clever way of ordering the set to show that it is in fact countable.

Write down strings in the ascending order of string length and among strings of equal length, order them alphabetically.

$\lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, \dots$

Here the string ba occurs in the sixth place, and every element has some position in the sequence.

The set is therefore countable.

#### 4. The set of all Turing machines (encoded in binary)

**Answer:** We know that, we can encode the Transition table of a Turing Machine as Description i.e., a unique binary string of finite length. Encode all Turing machines (i.e., **every Turing machine is a binary string of finite length**).

The total number of possible Turing Machines (i.e., Descriptions encoded in binary) is less than the size of  $\Sigma^*$  for the binary alphabet  $\Sigma = \{0,1\}$ .

Write down the encoded binary strings in the ascending order of string length and among strings of equal length, order them alphabetically. Hence every encoded string has some position in the sequence. The set is therefore countable.

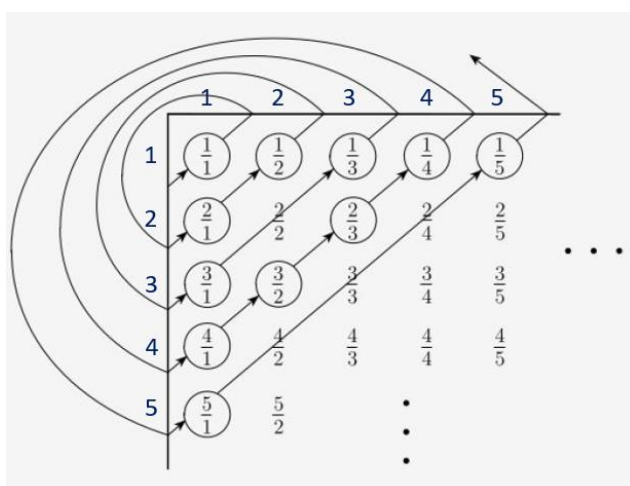
**There are countably infinite numbers of Turing Machines in the world.**

#### 5. Set of positive rational numbers i.e., $S = \{p/q \mid p, q \in \mathbb{N}\}$

**Answer:** How should we order this set to show that it is countable?

We cannot use the sequence  $1/1, 1/2, 1/3, 1/4, \dots$

because then  $2/3$  would never appear. This does not imply that the set is uncountable; in this case, there is a clever way of ordering the set to show that it is in fact countable.



Row headings indicates numerator and Column headings indicates denominator

Look at the scheme depicted in Figure above, and write down the element in the order encountered following the arrows.

This gives us  $1/1, 2/1, 1/2, 3/1, 1/3, \dots$  (enumeration procedure eliminates duplicates such as  $2/2, 3/3, 4/2, 2/4, \dots$ )

Here the element  $1/3$  occurs in the fifth place, and **every element has some position in the sequence**.

The set is therefore countable.

**Note:** The above set of rational numbers is infinite and have the same size (cardinality) as that of  $\mathbb{N}$  as we are able to keep the elements from both the sets in one-to-one correspondence.

A set is countable if either it is finite or it has same size as  $\mathbb{N}$ .

[illegible]

The highlighted digits are  $0.37210\dots$ . Suppose that we add 1 to each of these digits, to get the number

$0.48321\dots$

Now, this number can't be in the table. Why not? Because

- it differs from  $f(1)$  in its first digit;
- it differs from  $f(2)$  in its second digit;
- ...
- it differs from  $f(n)$  in its  $n$ th digit;
- ...

So it can't equal  $f(n)$  for any  $n$  — **that is, it can't appear in the table.**

This looks like a trick, but in fact there are lots of numbers that are not in the table. For example,

- we could subtract 1 from each of the highlighted digits (changing 0's to 9's), getting  $0.26109\dots$  — by the same argument, this number isn't in the table. Or
- we could subtract 3 from the odd-numbered digits and add 4 to the even-numbered digits. Or
- we could even highlight a different set of digits:



$n$	$f(n)$											
1	0	.	3	1	4	1	5	9	2	6	5	3 ...
2	0	.	3	7	3	7	3	7	3	7	3	7 ...
3	0	.	1	4	2	8	5	7	1	4	2	8 ...
4	0	.	7	0	7	1	0	6	7	8	1	1 ...
5	0	.	3	7	5	0	0	0	0	0	0	0 ...
$\vdots$	$\vdots$											

**As long as we highlight at least one digit in each row and at most one digit in each column, we can change each the digits to get another number not in the table.** Here, if we add 1 to all the highlighted digits, we end up with  $0.42981\dots$  — there's a real number that does not equal  $f(n)$  for any positive integer  $n$ .

What is the point of all this? Precisely that the function  $f$  can't possibly be onto — there will always be (infinitely many!) missing values. Therefore, there does not exist a bijection between  $\mathbb{N}$  and real numbers between 0 and 1.

### Solution 2:

Cantor's diagonalization argument goes as follows. If the real numbers were countable, it can be put in 1-1 correspondence with the natural numbers, so we can list them in the order given by those natural numbers.

$3.14159\dots$   
 $1.41421\dots$   
 $1.73205\dots$   
 $2.23606\dots$   
 $2.71828\dots$   
 $0.14285\dots$   
  
 $3.43625\dots$   
  
 $2.32514\dots$

Now use this list to construct a real number  $X$  that differs from every number in our list in at least one decimal place, by letting  $X$  differ from the  $i^{\text{th}}$  digit in the  $i^{\text{th}}$  decimal place. (A little care must be exercised to ensure that  $X$  does not contain an infinite string of 9's.) This gives a contradiction, because the list

was supposed to contain all real numbers, including X. Therefore, hence a 1-1 correspondence of the reals with the natural numbers must not be possible.

## 2) Set of all languages are not enumerable

(i.e., For any non-empty  $\Sigma$ , there exist languages that are not recursively enumerable)

### Solution:

Any formal language  $L$  is a subset of  $\Sigma^*$ ,  $L \subseteq \Sigma^*$ , we know that  $\Sigma^*$  is enumerable (countably infinite) However, a set of formal languages is not enumerable (uncountably infinite). Since any language is a subset of  $\Sigma^*$ .

Total number of subsets (or languages) that can be formed using the set  $\Sigma^*$  is  $2^{\Sigma^*}$ .

### Formal Proof:

Set of all languages over  $\Sigma$  are enumerable. This means the languages can be numbered as  $L_1, L_2, L_3, \dots$

Let's construct a 2-D matrix with an infinite number of rows.

Rows are enumerated as languages from the power set  $2^{\Sigma^*}$ . We can also see the rows as enumeration of Turing Machines with row  $L_i$  corresponding to the language of the  $i^{\text{th}}$  Turing Machine.

Columns in the matrix are all the strings from  $\Sigma^* = \{s_1, s_2, s_3, \dots\}$

	s1	s2	s3	s4	s5	...
L1	1	0	1	0	1	...
L2	1	1	1	0	1	...
L3	1	0	0	1	1	...
L4	1	1	1	1	1	...
L5	1	0	0	0	0	...
...	...	...	...	...	...	...

Row headings indicates Languages and Column headings indicates strings. Cell value 1 indicates string belongs to the language, and 0 indicates string does not belong to the language.

Consider left to right diagonal, this will represent another language lets say  $L_{\text{diag}}$ .

Thus,  $L_{\text{diag}} = \{s_1, s_2, s_4, \dots\}$

**This language  $L_{\text{diag}}$  must be one of the enumerated languages** (must be same as one of the rows)

Thus, the language is computable and recursively enumerable since we can construct Turing machine for such a language (as per our assumption)

**Complement this language, what we get is a new language  $L_{\text{diag}}^c = \{s_3, s_5, \dots\}$**

This new language is not the same as any of the enumerated elements (rows) because it differs from  $i^{\text{th}}$  diagonal place. Hence it is a new language and not the same as any of the enumerated languages.

**This is contradiction; hence our assumption that all languages are enumerable is wrong.**

### Note:

- The total number of possible Turing Machines (encoded as a unique binary string) is less than size of  $\Sigma^*$  for the binary alphabet. So, we can just enumerate all possible strings of Turing machines and discard invalid encodings.  
There are countably infinite numbers of Turing Machines in the world. But there are uncountably infinite number of formal languages. (we just proved)
- There exists a formal language for which we cannot construct a TM i.e. there exists a language which is not Recursively Enumerable.
- Set of Recursively Enumerable Languages is a proper subset of a set of all formal languages.

- Theorem: Let  $S$  be an infinite countable set. Its powerset  $2^S$  is not countable.

# Turing acceptable languages

## 1. Recursive language (Decidable Language)

A formal language is recursive if there exists a Turing machine that, **when given a finite sequence of symbols as input, always halts and accepts it if it belongs to the language and halts and rejects it otherwise.**

The class of all recursive languages is often called **REC**.

This type of language was not defined in the Chomsky hierarchy of (Chomsky 1959). All recursive languages are also recursively enumerable. All regular, context-free and context-sensitive languages are recursive.

### Definitions

---

There are many equivalent major definitions for the concept of a recursive language:

1. A recursive formal language is a recursive subset in the set of all possible words over the alphabet of the language.
2. A recursive language is a formal language for which there exists a Turing machine that, when presented with any finite input string, halts and accepts if the string is in the language, and halts and rejects otherwise. The Turing machine always halts: it is known as a decider and is said to *decide* the recursive language.
3. **Recursive languages are also called Decidable Languages because a Turing Machine can decide membership in those languages (it can either accept or reject a string).**

By the second definition, any decision problem can be shown to be decidable by exhibiting an algorithm for it that terminates on all inputs. An undecidable problem is a problem that is not decidable.

### Summary.

If  $L$  is a recursive language, then

- If  $w \in L$  then a TM halts in a final state and accept the string.
- If  $w \notin L$  then TM halts in a non-final state and reject the string.

### Note:

- A language  $L \subseteq \Sigma^*$  is called recursively enumerable if it is accepted by some Turing machine  $M$ .
- A language  $L \subseteq \Sigma^*$  is called recursive if it is accepted by some Turing machine  $M$  which halts on all inputs.
- If a type of problem or, equivalently, a formal language has a decider Turing Machine, then it is said to be computable or decidable. In other words, all recursive languages are computable or decidable.
- If there was no halting problem, we could convert every acceptor machine to a decider, thereby making every recursively enumerable language recursive.

### Recursive Languages are also Recursively Enumerable

Proof: If  $L$  is a recursive then there is TM which decides a member in language then

- $M$  accepts  $x$  if  $x$  is in language  $L$ .
- $M$  rejects on  $x$  if  $x$  is not in language  $L$ .

According to the definition,  $M$  can recognize the strings in language that are accepted on those strings.



## Closure properties

---

Recursive languages are closed under the following operations. That is, if  $L$  and  $P$  are two recursive languages, then the following languages are recursive as well:

- the Kleene star  $L^*$  of  $L$
- the concatenation  $L.P$  of  $L$  and  $P$
- the union  $L \cup P$
- the intersection  $L \cap P$
- The complement of  $L$
- The set difference  $L - P$

The last property follows from the fact that the set difference can be expressed in terms of intersection and complement.

## Examples of

---

As noted above, every **context-sensitive language** is recursive. Thus, a simple example of a recursive language is the set  $L = \{abc, aabbcc, aaabbbccc, \dots\}$ ; more formally, the set

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

is context-sensitive and therefore recursive.

$L = \{a^n b^n c^n \mid n \geq 1\}$  is recursive because we can construct a Turing machine which will move to final state if the string is of the form  $a^n b^n c^n$  else move to non-final state. So, the TM will always halt in this case.

A **context-sensitive grammar** for  $L = \{a^n b^n c^n \mid n \geq 1\}$  is

$$S \rightarrow aBSc \mid abc$$

$$Ba \rightarrow aB$$

$$Bb \rightarrow bb$$

Try to derive  $aaabbbccc$ ?

$$[S \Rightarrow aBSc \Rightarrow aBaBScc \Rightarrow aBaBabccc \Rightarrow aaBBabccc \Rightarrow aaBaBbccc \Rightarrow aaBabbccc \Rightarrow aaaBbbccc \Rightarrow aaabbbccc]$$

## Note:

- **Context-Sensitive Grammar:**
  - CSG Production rules constraints:  $\alpha A \beta \rightarrow \alpha \gamma \beta$  (where  $\alpha, \beta, \gamma$  = string of terminals and/or non-terminals,  $A$  = Non-terminal)
- **A linear bounded automata (LBA) is a restricted form of Turing machine.**

An LBA satisfies the following three conditions:

- Its input alphabet includes two special symbols, serving as left and right endmarkers.
- Its transitions may not print other symbols over the endmarkers.
- Its transitions may neither move to the left of the left endmarker nor to the right of the right endmarker.

## 2. Recursively Enumerable language (Type-0 Language)

(also called partially decidable, semi decidable, Turing-acceptable or Turing-recognizable)

A language  $L$  is recursively enumerable if there exists a Turing Machine  $M$  such that  $L = L(M)$ .

The class of all recursively enumerable languages is called **RE**.



## Definitions

---

There are many equivalent definitions of a recursively enumerable language:

1. A recursively enumerable language is a recursively enumerable subset in the set of all possible words over the alphabet of the language, i.e., if there exists a Turing machine which will enumerate all valid strings of the language.
2. A recursively enumerable 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 string produced for number  $n$  is "already" produced for a number which is less than  $n$ . If it already is produced, use the output for input  $n+1$  instead (recursively), but again, test whether it is "new".
3. A recursively enumerable 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.
4. A language is called **Recursively Enumerable** if there is a Turing Machine that accepts on any input within the language. **Recursively Enumerable Languages are also called *Recognizable* because a Turing Machine can recognize a string in the language (accept it). It might not be able to decide if a string is not in the language since the machine might loop for that input.**

## Summary.

If  $L$  is a Recursively Enumerable language (or partially decidable), then

- If  $w \in L$  then a TM halts in a final state and accept the string,
- If  $w \notin L$  then TM
  - halts in a non-final state and reject the string or
  - never halt, i.e., it may go into an infinite loop.

## Closure properties

---

Recursively enumerable languages (REL) are closed under the following operations. That is, if  $L$  and  $P$  are two recursively enumerable languages, then the following languages are recursively enumerable as well:

- the Kleene star  $L^*$  of  $L$
- the concatenation  $L.P$  of  $L$  and  $P$
- the union  $L \cup P$
- the intersection  $L \cap P$ .

Recursively enumerable languages are not closed under set difference or complementation. The set difference  $L - P$  is recursively enumerable if  $P$  is recursive. If  $L$  is recursively enumerable, then the complement of  $L$  is recursively enumerable if and only if  $L$  is also recursive.

## Note:

- An undecidable language maybe a partially decidable language or something else but not decidable.
- Undecidable languages are not recursive languages, they may be subsets of Turing recognizable languages: i.e., such undecidable languages may be recursively enumerable.

- If a language is not even partially decidable, then there exists no Turing machine for that language.
- All regular, context-free, context-sensitive and recursive languages are recursively enumerable.

## Examples of Recursively Enumerable Language

---

Some recursively enumerable languages that are not recursive (or undecidable) include:

### 1. Language $HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \}$

(Recursively Enumerable/An Undecidable/Non-computable Problem)

**Answer:** (in brief)

- We know that, we can encode the Transition table of a Turing Machine as Description i.e., a unique binary string of finite length. Encode all Turing machines (i.e., every Turing machine is a binary string of finite length).
- The total number of possible Turing Machines (i.e., Descriptions encoded in binary) is less than the size of  $\Sigma^*$  for the binary alphabet  $\Sigma = \{0,1\}$ .
- There are countably infinite numbers of Turing Machines in the world.
- Write down the encoded binary strings in the ascending order of string length and among strings of equal length, order them alphabetically. Hence every encoded string has some position in the sequence. The set of Turing machines is therefore countable (or recursively enumerable).
- **Given a description of a Turing machine and its input i.e.,  $\langle M \rangle, w$ , there is no general algorithm (or  $U_{TM}$ ) that decide correctly whether Turing machine will halt or run indefinitely forever, for all possible Turing Machine and input pairs.**
- **Hence, the halting problem over Turing machines is undecidable (i.e., unsolvable also).**

### 2. $L_{PCP} = \{ \langle P \rangle \mid P \text{ is an instance of the PCP with a solution/match} \}$

(Recursively Enumerable/An Undecidable/Non-computable Problem)

#### What is Post Correspondence Problem(PCP)?

The Post correspondence problem is **an undecidable decision problem** that was introduced by Emil Post in 1946. Because it is simpler than the halting problem and the *Entscheidungsproblem* it is often used in proofs of undecidability.

#### Definition of the problem

---

Let  $\Sigma$  be an alphabet with at least two symbols. The input of the problem consists of 2-finite lists  $x_1, x_2, \dots, x_n$  and  $y_1, y_2, \dots, y_n$  of n-words over  $\Sigma$ .

**A solution to this problem is a sequence of indices, such that the concatenation of the words from the respective list in that sequence yields the same resultant string.**

The decision problem then is to decide whether such a solution exists or not, for the given two lists above. We will prove here that this problem is unsolvable by any algorithm.

#### Note:

- Every word in the list should appear at least once in the **resultant string**.
- A word can appear any number of times in the **resultant string**.

**Example 1:** Consider the following two lists

	List-1	List-2
Word-1	a	baa
Word-2	ab	aa
Word-3	bba	bb

A solution to this problem would be the sequence (3, 2, 3, 1), because

	List1				
	Word-3	Word-2	Word-3	Word-1	
<b>Result1 =</b>	<b>bba</b>	<b>ab</b>	<b>bba</b>	<b>a</b>	<b>= bbaabbbbaa</b>
<b>Result2 =</b>	<b>bb</b>	<b>aa</b>	<b>bb</b>	<b>baa</b>	<b>= bbaabbbbaa</b>
	Word-3	Word-2	Word-3	Word-1	
	List2				

Furthermore, since (3, 2, 3, 1) is a solution, so are all of its "repetitions", such as (3, 2, 3, 1, 3, 2, 3, 1), etc., that is, when a solution exists, there are infinitely many solutions of this repetitive kind.

**Example 2:** Consider the following two lists

	List-1	List-2
Word-1	1	111
Word-2	10111	10
Word-3	10	0

**Explanation:**

- **Step-1:** We will start with a word in which both list-1 and list-2 are starting with same number, so we can start with either word-1 or word-2. Let's go with **word-2**, hence

Result1 = 10111

Result2 = 10

- **Step-2:** We need 1s in result2 to match 1s in result1 so we will go with **word-1**, hence

Result1 = 10111.1 = 101111

Result2 = 10.111 = 10111

- **Step-3:** There is extra 1 in result1 to match this 1 so we will go with **word-1**, hence

Result1 = 101111.1 = 1011111

Result2 = 10111.111 = 1011111

- **Step-4:** Now there is extra 1 in result2 to match it we will add **word-3**, hence

Result1 = 1011111.10 = 101111110

Result2 = 10111111.0 = 101111110

- As you can see, strings are same. The final solution sequence is **2 1 1 3**

**Example 3: Consider the following two lists**

	List-1	List-2
Word-1	100	1
Word-2	0	100
Word-3	1	00

We can try unlimited combinations for the above problem but none of the combination will lead us to solution, thus this problem does not have solution.

**Example 4: Consider the following two lists**

	List-1	List-2
Word-1	abb	bba
Word-2	aa	aaa
Word-3	aaa	aa

**Solution:** 2 1 3

**Example 5: Consider the following two lists**

	List-1	List-2
Word-1	ab	a
Word-2	bab	ba
Word-3	bbaaa	bab

**No solution**

**Example 6: Consider the following two lists**

	List-1	List-2
Word-1	10	101
Word-2	011	11
Word-3	101	011

**No solution**

**Example 7: Consider the following two lists**

	List-1	List-2
Word-1	abab	ababaaa
Word-2	aaabbb	bb
Word-3	aab	baab
Word-4	ba	baa
Word-5	ab	ba
Word-6	aa	a

There is a solution for the string 123456:

**Note:** It turns out that only some instances of PCP have solutions and others have no solution (**we cannot solve them, we may enter into an infinite loop while solving**)

**We can express the PCP as a language as:**

$$L_{PCP} = \{ \langle P \rangle \mid P \text{ is an instance of the PCP with a solution/match} \}$$

For a given instance, the instance having a solution (also called a match) is the same as saying that the instance belongs to the language  $L_{PCP}$  above. Similarly, the instance not having a solution/match means it doesn't belong to  $L_{PCP}$ .

### **Undecidability of $L_{PCP}$ :**

As theorem says that  $L_{PCP}$  is undecidable. That is, there is no particular algorithm that determines whether any PCP has solution or not and hence, it is undecidable (i.e., unsolvable also).

#### **Proof of $L_{PCP}$ Undecidability:** (in brief)

The most common proof for the undecidability of  $L_{PCP}$  describes an instance of PCP that can simulate the computation of an arbitrary Turing machine on a particular input. A match will occur if and only if the input would be accepted by the Turing machine. Because deciding if a Turing machine will accept an input is a basic undecidable problem, PCP cannot be decidable either.

## **3. Non-Recursively Enumerable language (Non computable)**

### **Examples:**

- 1. For any non-empty  $\Sigma$ , there exist languages that are not recursively enumerable (proved above)**

**Note:** There exists a recursively enumerable language whose complement is not recursively enumerable.

### **Note:**

- If a language  $L$  and its complement  $L^c$  are both recursively enumerable, then both languages are recursive.
- If  $L$  is recursive, then  $L^c$  is also recursive, and consequently both are recursively enumerable.

### **References:**

1. [https://en.wikipedia.org/wiki/Recursively\\_enumerable\\_language](https://en.wikipedia.org/wiki/Recursively_enumerable_language)
2. <https://jlmartin.ku.edu/courses>
3. <https://www.geeksforgeeks.org/post-correspondence-problem/>
4. Undecidable(PCP,UTM,Reduction).pdf from Prof. Preet Kanwal, PESU