Module 7

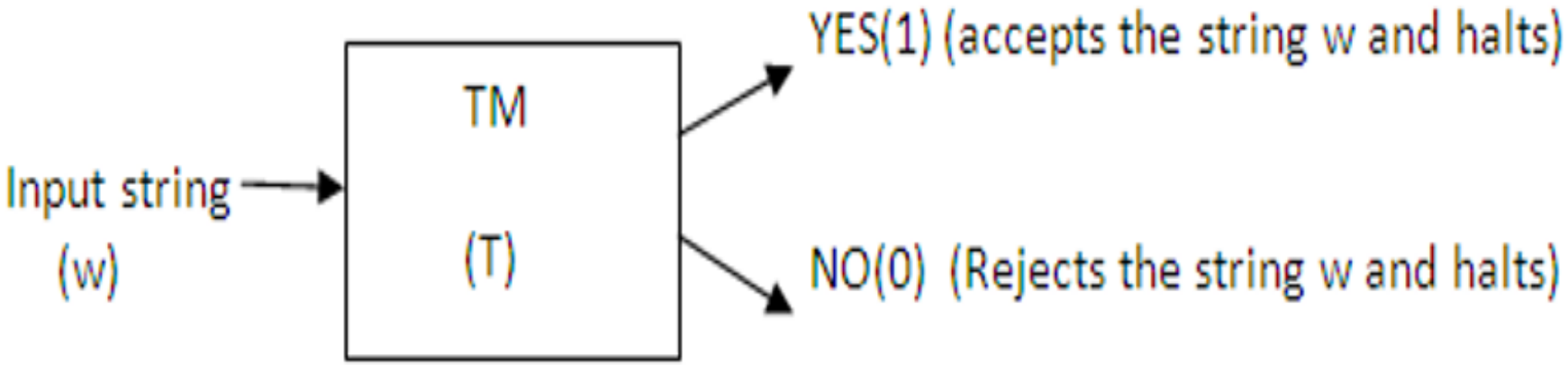
Dr. S. Kiruthika
Assistant Professor – SCOPE
VIT Chennai

Syllabus

Recursive and Recursively Enumerable Languages, Language that is not Recursively Enumerable (RE) – computable functions – Chomsky Hierarchy – Undecidable problems - Post's Correspondence Problem

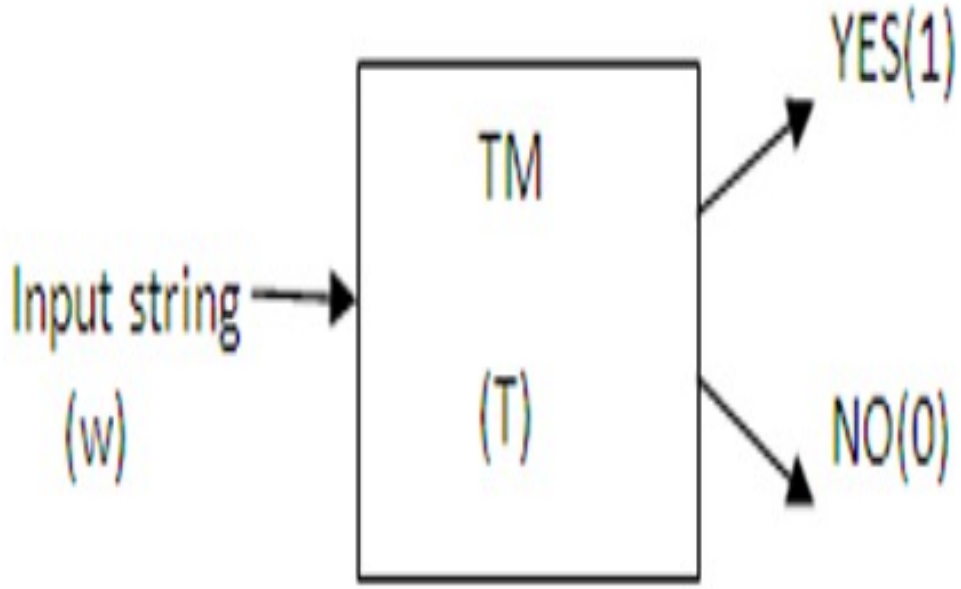
Recursive Language

- A **language** is **recursive** if there exists a Turing machine that halts for every given input string
- When given a finite sequence of symbols as input, accepts it if it belongs to the **language** and rejects it otherwise.
- **Recursive Languages** are also called Turing decidable languages.



Recursively Enumerable Languages

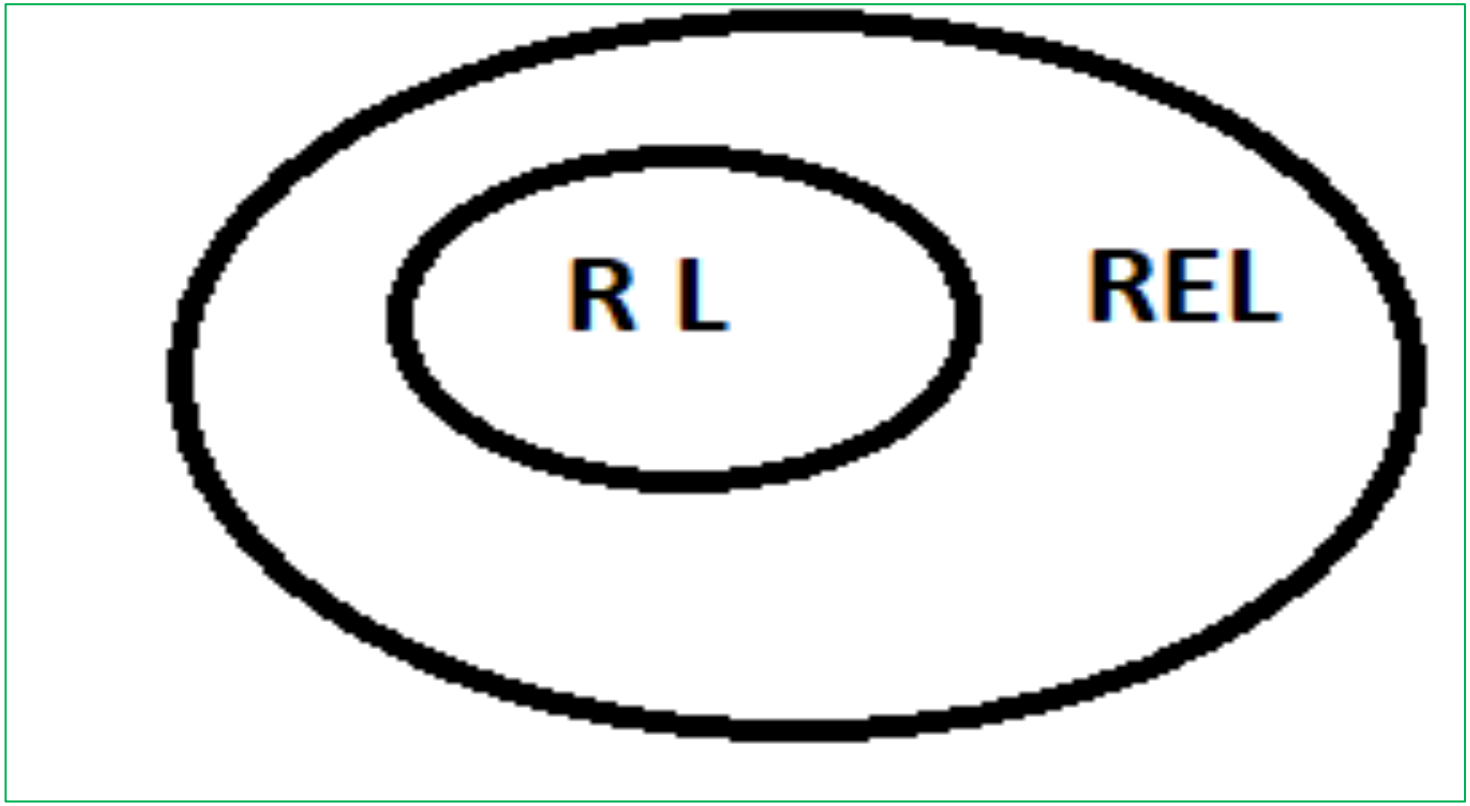
- A **recursively enumerable language** is a formal **language** for which there exists a Turing machine (or other computable function) .



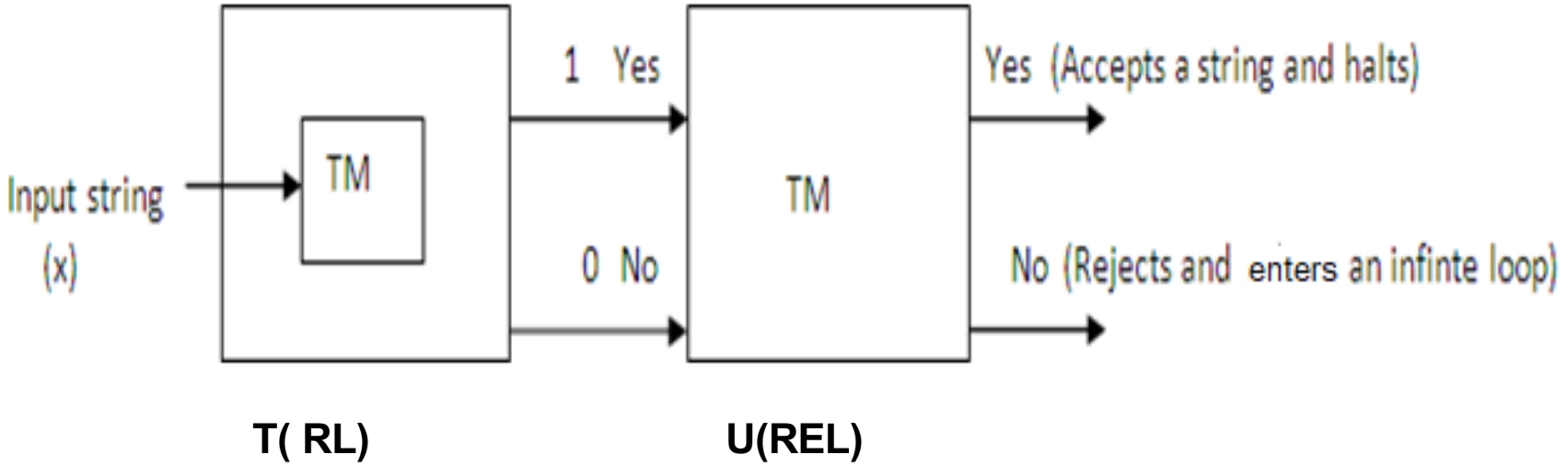
Accepts the string w and halts

Reject and halt or goes into an infinite loop

Relation between RL and REL

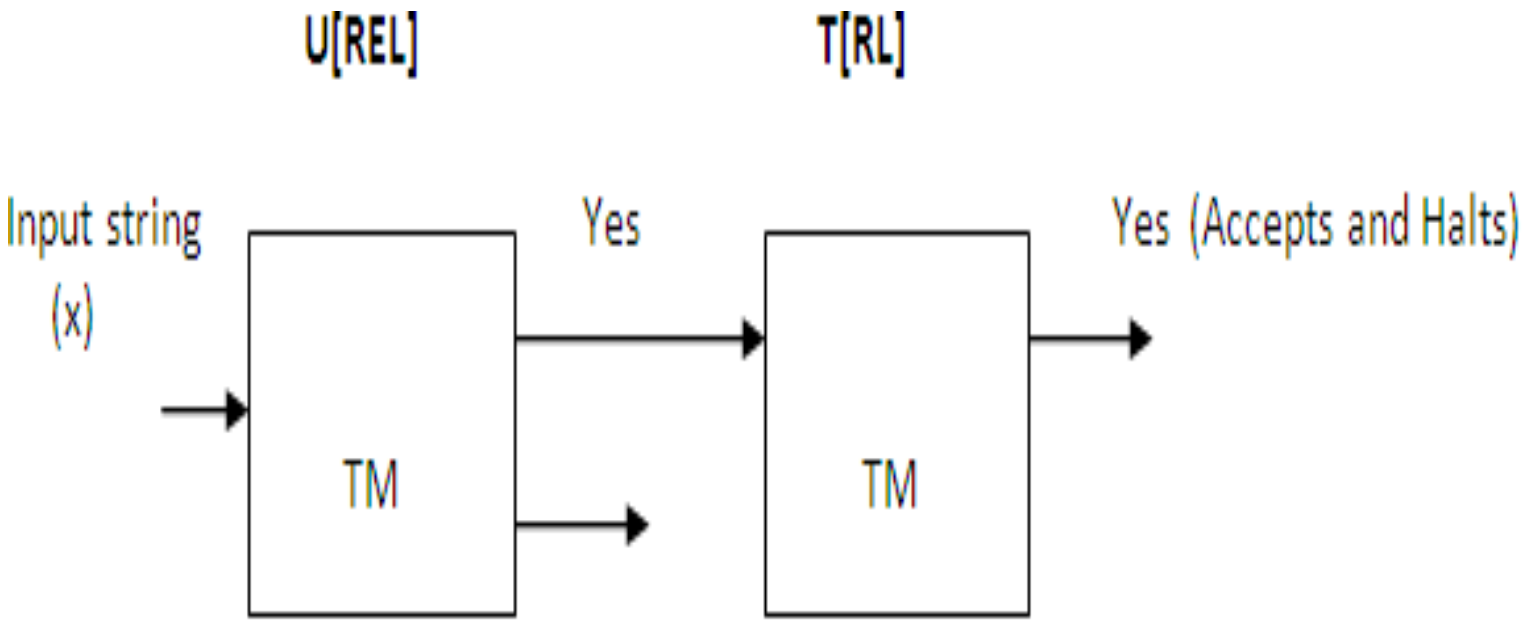


Every recursive language is Recursively enumerable.

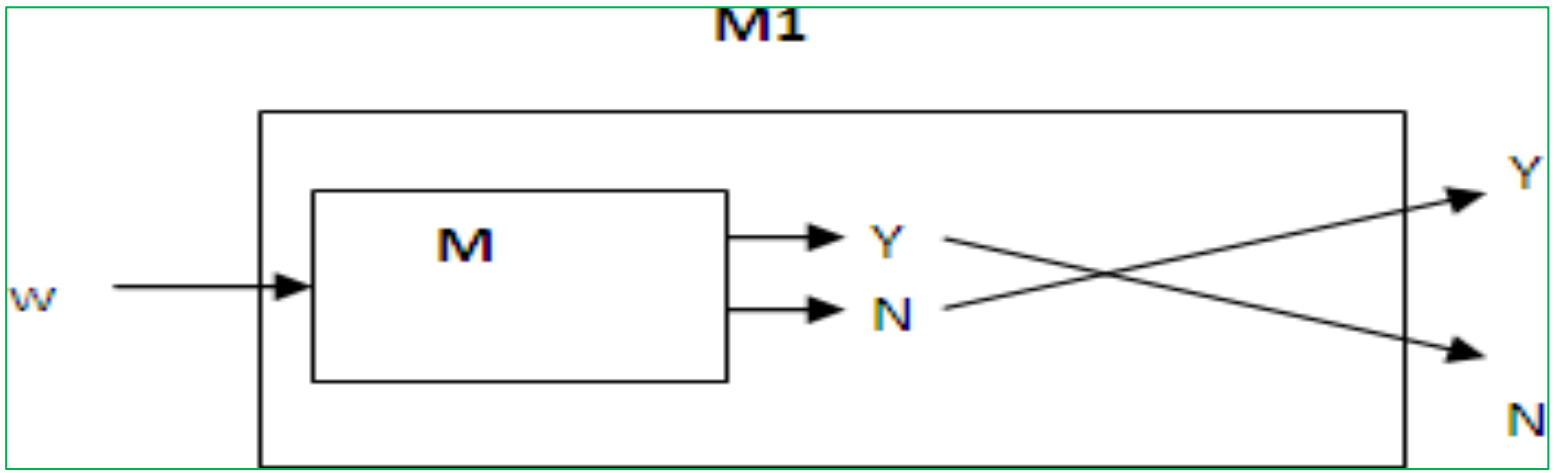


- If T is TM recognizing L(RL) then we can get a TM that accepts the language L by modifying T so that when the output is 0 it may halt or into an infinite loop.

Every REL is not recursive



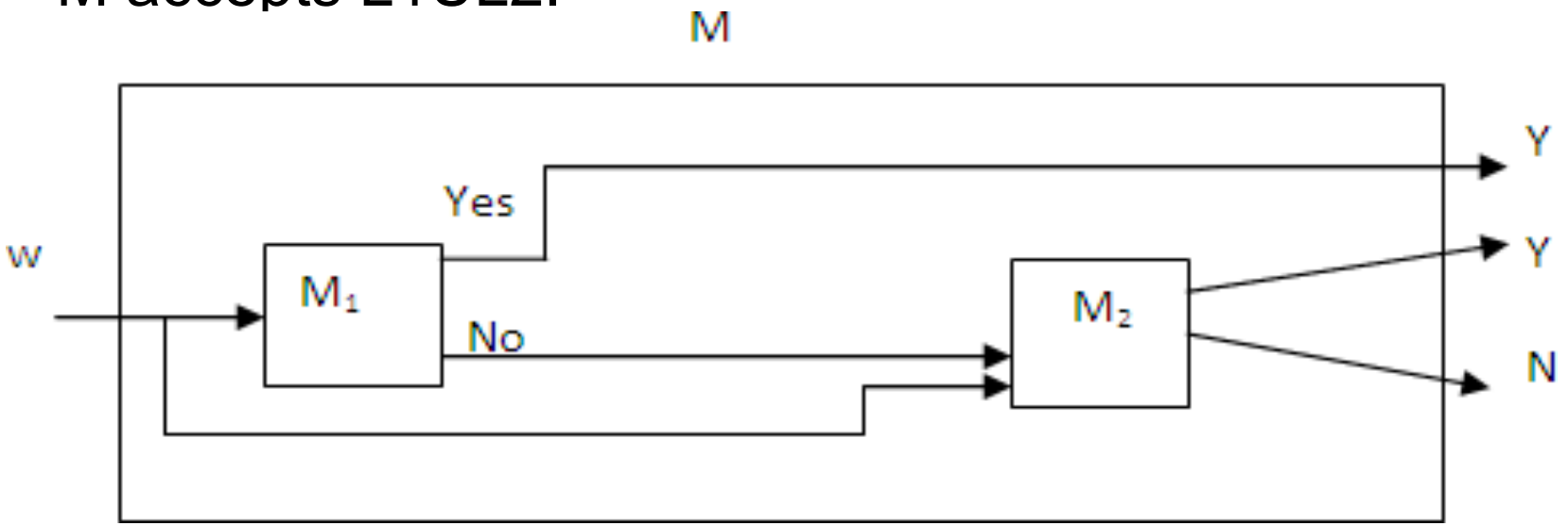
The complement of a RL is recursive



- Let L be a Recursive Language and M be the TM that halts on all inputs and accepts L.
- Construct M1 for L′.
- M accepts and halts for Yes, then M′ rejects and halts.
- M rejects and halts for N, then M′ accepts and halts.

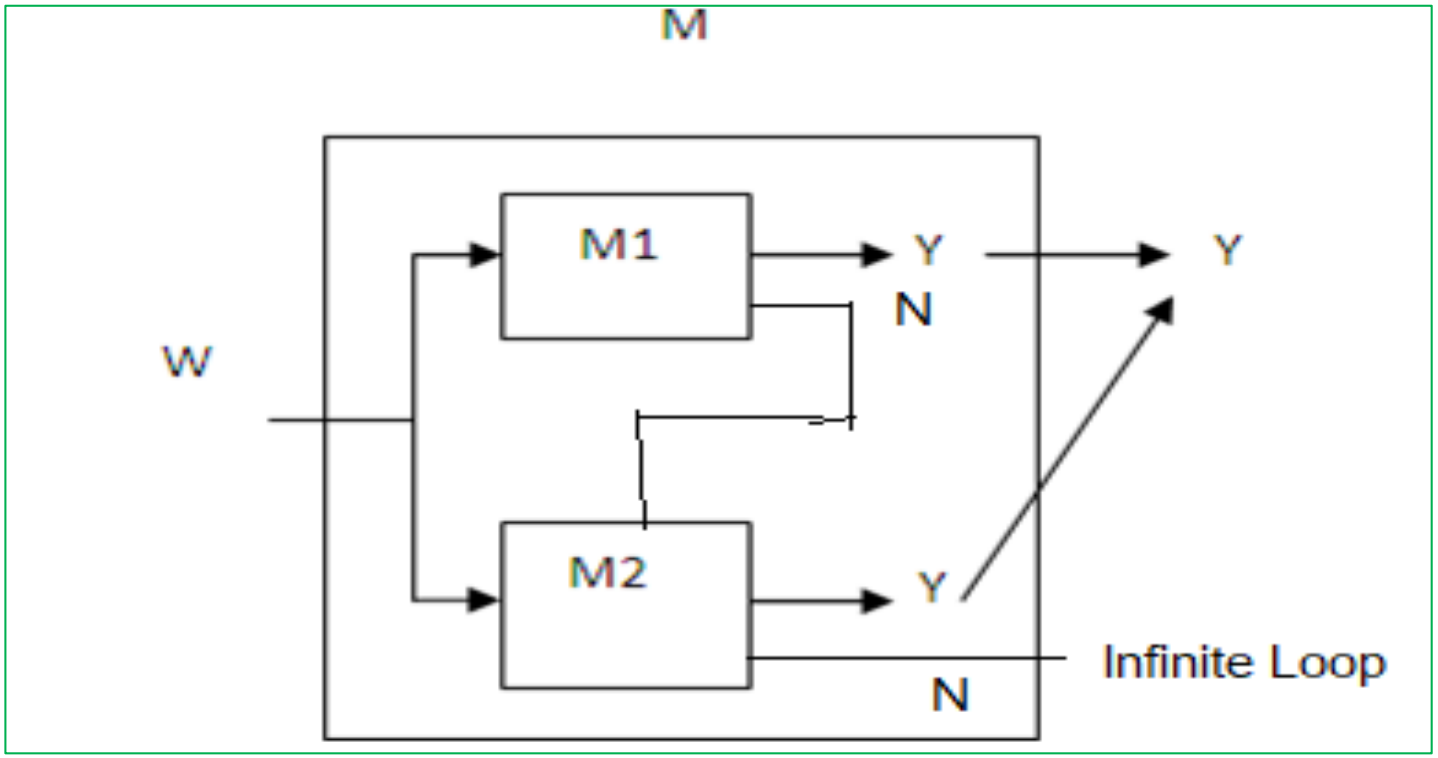
Union of two recursive languages is recursive

- Let L1 and L2 be the Recursive languages accepted by TMs M1 and M2.
- **Construct M:**
  - It first simulates M1, If Yes than M accepts.
  - If M1 rejects, then M simulates M2 and accepts if and only if M2 accepts.
  - M accepts L1UL2.



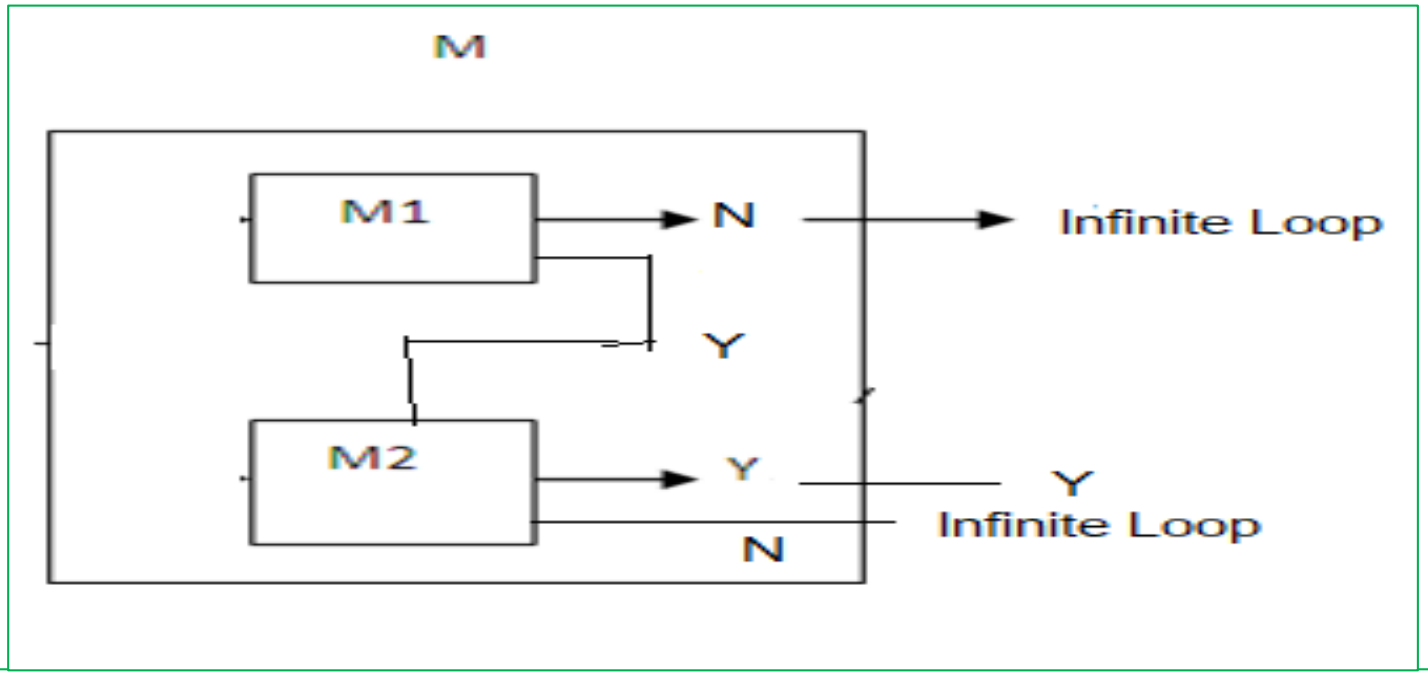
Union of two Recursively Enumerable Languages is REL

- Let L1 and L2 be the Recursive Enumerable languages accepted by TMs M1 and M2.
- M simultaneously simulates M1 and M2.
- If either accepts, then M accepts .i.e. M accepts L1UL2.

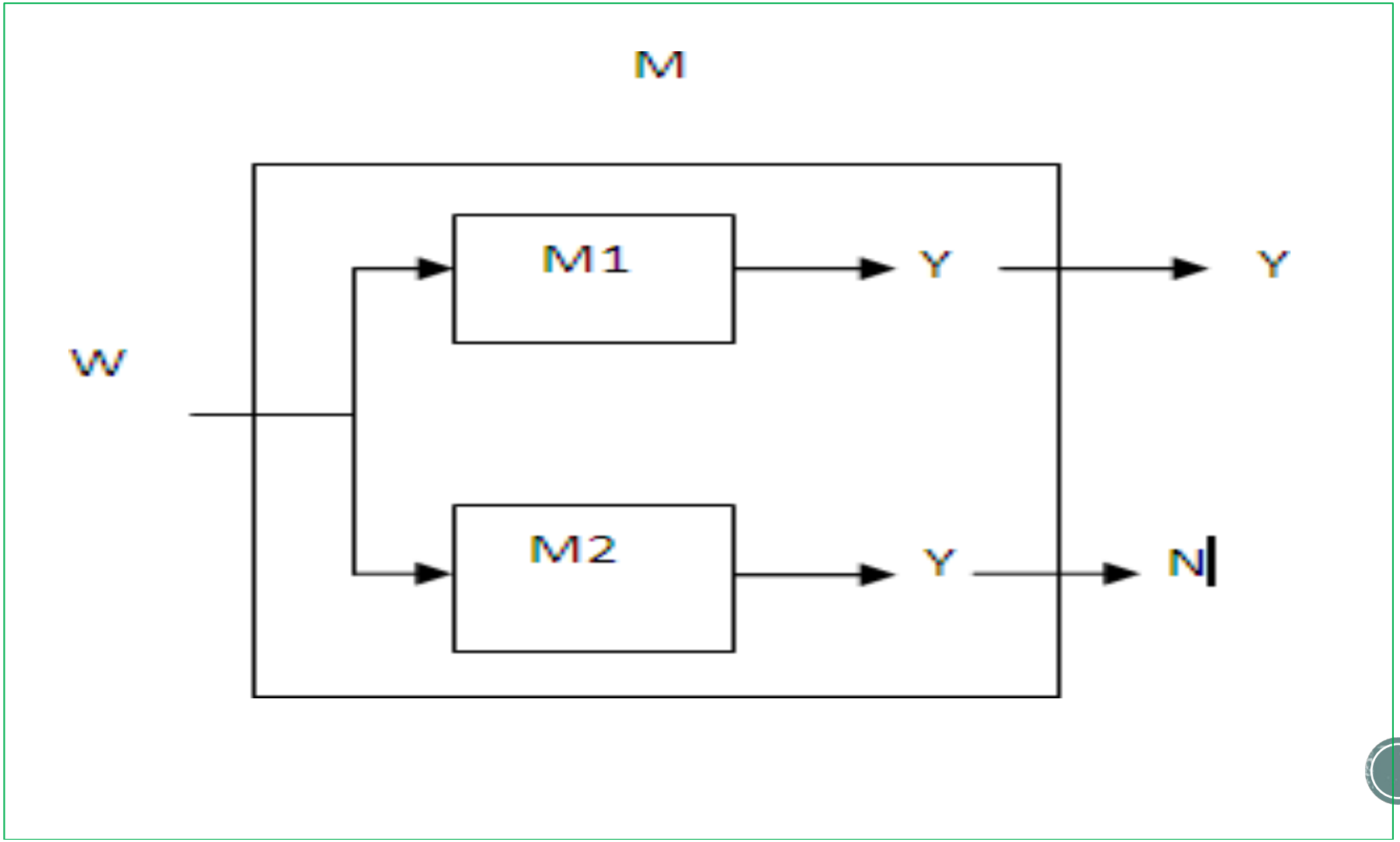


Intersection of two recursively enumerable languages is REL

- Let L1 and L2 be the Recursive Enumerable languages accepted by TMs M1 and M2.
- M halts if both M1 and M2 halts.
- M will never halt if either M1 or M2 enter into infinite loop.
- (i.e.)M accepts L1∩L2.



If a Language L and its complement L1 are both RE, then L U L1 is recursive.



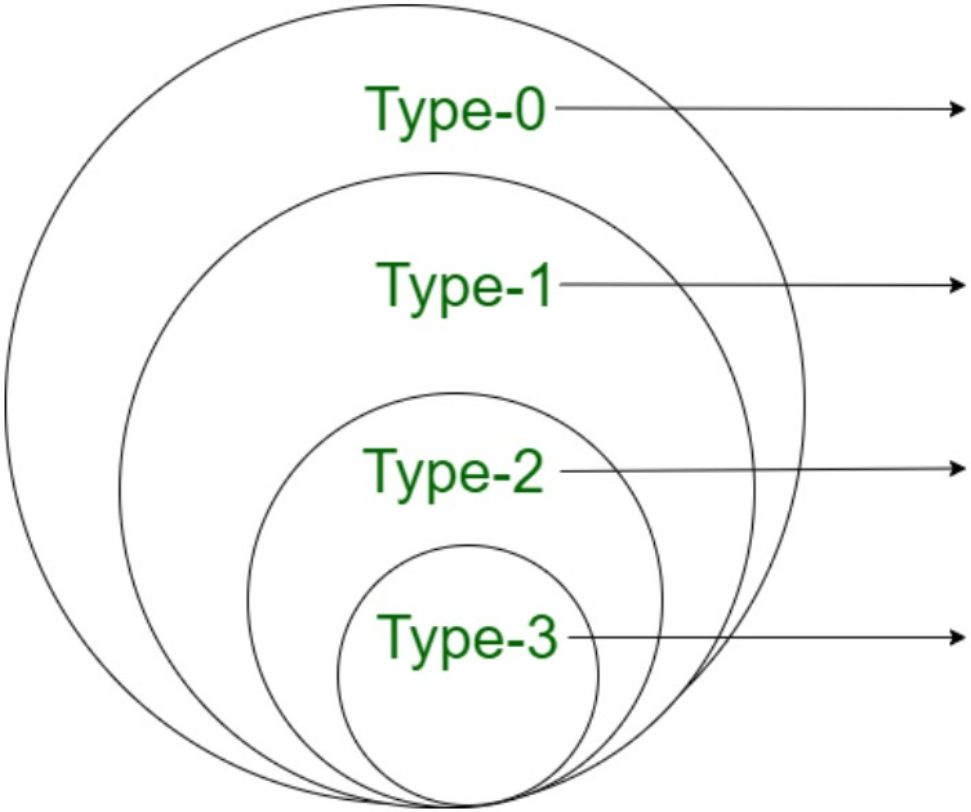
If a Language L and its complement L′ are both RE, then L U L′ is recursive.

Let M1 and M2 be the TMs accepting L1 and L2 respectively.

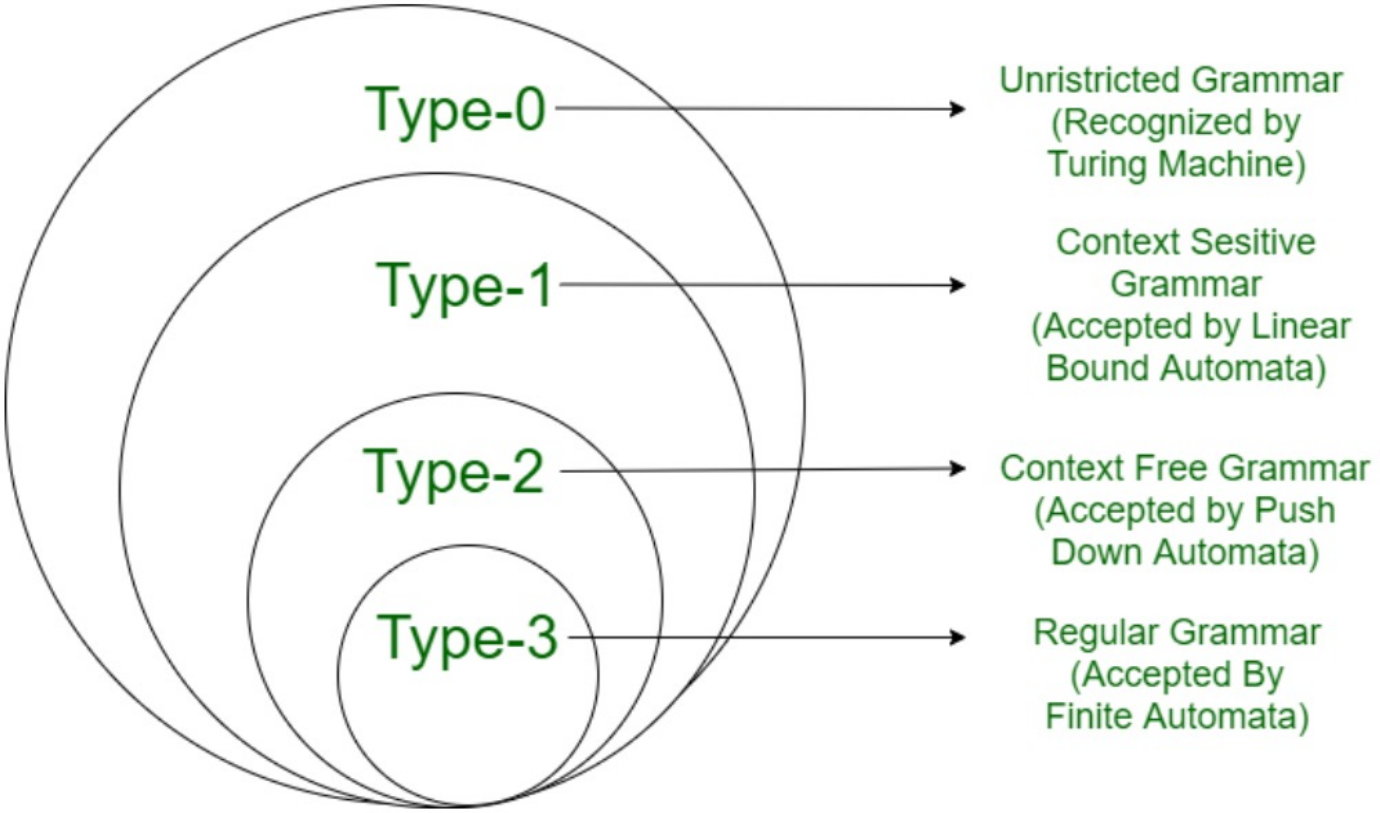
- M simultaneously simulates M1 and M2.
- M1 accepts L and M2 accepts L′

- If input to M is in L then M1 accepts and hence M accepts and halts. If input to M is not in L then M2 accepts and halts and hence M halts. Hence M halts on every input and so it is recursive.

Recap - Chomsky Hierarchy



Recap


Type-0
Type-1
Type-2
Type-3
Unristricted Grammar (Recognized by Turing Machine)
Context Sesitive Grammar (Accepted by Linear Bound Automata)
Context Free Grammar (Accepted by Push Down Automata)
Regular Grammar (Accepted By Finite Automata)

Undecidability

- A decision problem is a problem that requires a yes or no answer.
- A problem that cannot be decided by algorithmic means even after giving an unlimited resource and infinite amount of times, is termed as Undecidable.
- A decision problem that admits no algorithmic solution is said to be undecidable.

Undecidability

- For an undecidable language, there is no Turing Machine which accepts the language and makes a decision for every input string **w** (TM can make decision for some input string though).
- Undecidable languages are not recursive languages, but sometimes, they may be recursively enumerable languages.

Halting Problem

Halt **Never halt** **May Halt may not**

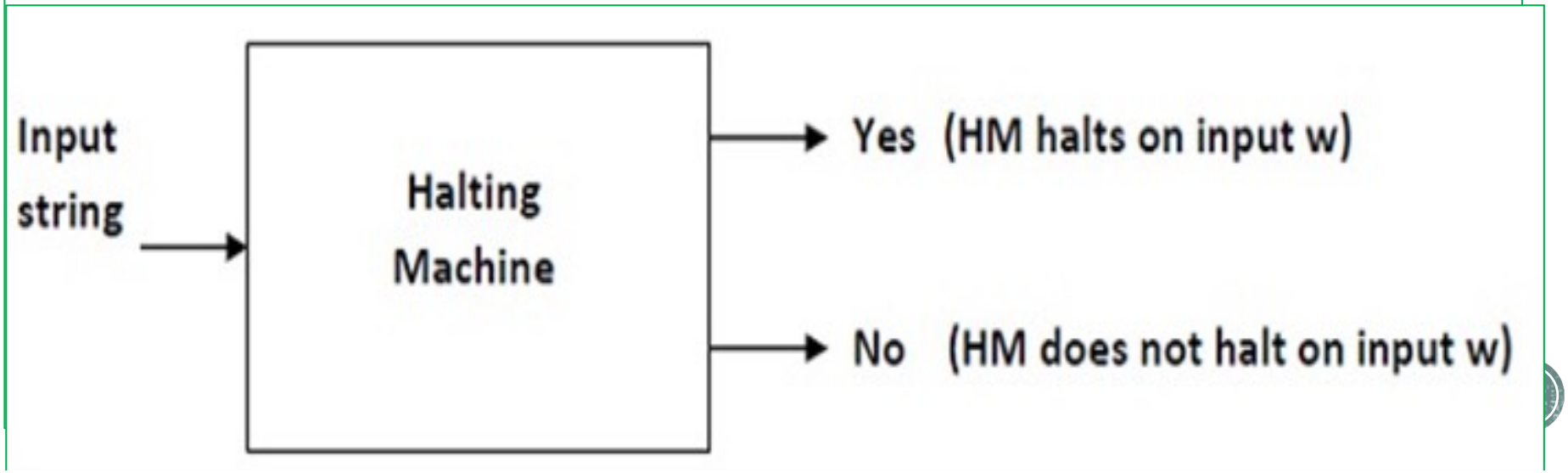
```
Program1()
{
  Print "Hello"
}
```

```
Program2()
{
while(true)
    {
Print "Hello"
}
}
```

```
Program3(b)
{
while(b)
    {
      Print "Hello"
    }
}
```

Halting Problem

- Given a program/algorithm will ever halt or not?
- Halting means that the program on certain input will accept it and halt or reject it and halt and it would never go into an infinite loop.
- Basically halting means terminating.
- So can we have an algorithm that will tell that the given program will halt or not.



- In terms of Turing machine, will it terminate when run on some machine with some particular given input string.
  - The answer is NO. we cannot design a generalized algorithm which can appropriately say that given a program will ever halt or not?
- Halting Problem is an undecidable problem because we cannot have an algorithm which will tell us whether a given program will halt or not in a generalized way i.e by having specific program/algorithm.

Solution:

- Let us assume that we can design that kind of machine called as HM(P, I) where HM is the machine/program, P is the program and I is the input.

- On taking input the both arguments the machine HM will tell that the program P either halts or not.

- If we can design such a program this allows us to write another program we call this program CM(X) where X is any program(taken as argument) and according to the definition of the program CM(X) shown in the figure.

Halting Problem is undecidable

Proof by Contradiction :

- Design a machine which if given a program can find out if that program will always halt or not halt on a particular input?

Solution:

```
HM ( P,I)
 {  Halt

or

May not Halt
 }
```

```
CM ( X)
 {
   if(HM(X,X)==Halt)
         loop forever;
   else
         return;

 }
```

```
HM ( P,I)
{  Halt

or

May not Halt
}
```

```
CM ( X)
{
   if ( H ( X,X)  ==HALT)
         loop forever;

    else
         return;

}
```

if we run CM on itself:

CM(CM,CM)

HM(CM,CM) ==HALT

```
{       Loop forever;
// it means it will never halt;
}
```

H(C,C)==NOT HALT

```
{       Halt;
// it will never halt because of the
above non-halting condition;
}
```

Halting Problem is undecidable

- In the program CM(X) we call the function HM(X), which we have already defined and to HM() we pass the arguments (X, X), according to the definition of HM() it can take two arguments i.e one is program and another is the input.
- Now in the second program we pass X as a program and X as input to the function HM().
- We know that the program HM() gives two output either “Halt” or “Not Halt”.
- But in case second program, when HM(X, X) will halt loop body tells to go in loop and when it doesn’t halt that means loop, it is asked to return.

Halting Problem is undecidable

- It is impossible for outer function to halt if its code (inner body) is in loop and also it is impossible for outer non halting function to halt even after its inner code is halting.
- So the both condition is non halting for CM machine/program even we had assumed in the beginning that it would halt.
- So this is the contradiction and we can say that our assumption was wrong and this problem, i.e., halting problem is undecidable.

The Post ‘ S Correspondence Problem (PcP)

- An instance of Post’s correspondence problem (PCP) is a set {(α1, β1), (α2, β2), . . . , (αn, βn)} of pairs, where n ≥ 1 and the αi’s and βi’s are all nonnull strings over an alphabet .

α1		α2		α3		α4	αn
β1		β2		β3	.....	β4	βn

- The decision problem is this: Given an instance of this type, does there exist a positive integer k and a sequence of integers i1, i2, ...,ik with each ij satisfying 1 ≤ ij ≤ n, satisfying

$$\alpha_{i1}\alpha_{i2} \ldots \alpha_{ik} = \beta_{i1}\beta_{i2} \ldots \beta_{ik}$$

B
BA

A
AB

CA
A

ABC
C

Example 1: PCP

Consider the following domino's

B	A	CA	ABC
BA	AB	A	C

Solution :

B	**A**	**ABC**							
BA	AB	C							

BAABC

BAABC

Order: 1,2,4

Example 2: PCP

Consider the following domino's

	A	B
1.	1	111
2.	10111	10
3.	10	0

Solution :

1	**1**	**1**	**…..**						
111	111	111	… …						

111

11111111

Order: 1,

Example 3: PCP

Consider the following domino's

	A	B
1.	10	101
2.	011	11
3.	101	011

Solution :

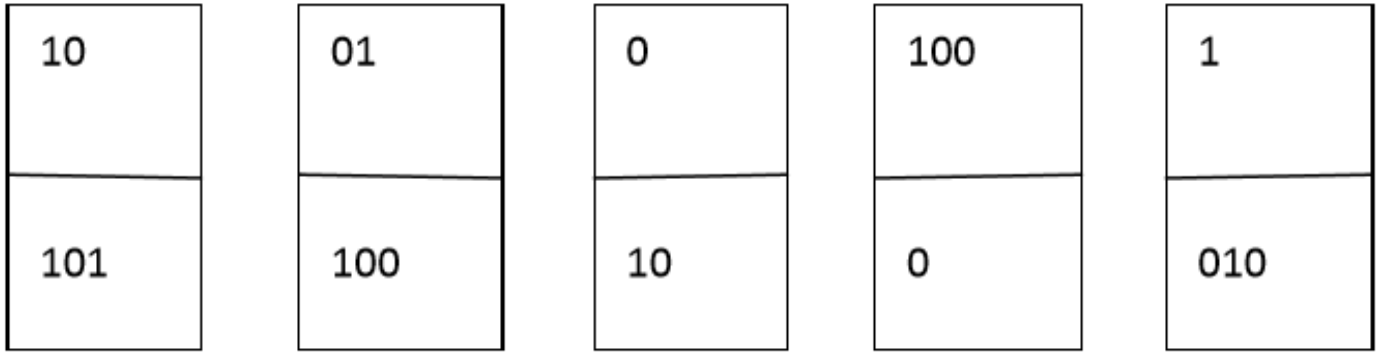
10	**10**	**10**	**…**						
101	101	101	….						

10101101101

101011011011

Order: 1,3

- Example 4:



- S

1

10	1	01	0	100	100	0	100
101	010	100	10	0	0	10	0

Undecidability of PCP

We have already seen that A_{TM} is an undecidable language. In order to prove that PCP is undecidable, it suffices to show that we can reduce A_{TM} to PCP. Thus, we will show that if we can decide PCP, then we can decide A_{TM}. Given any Turing machine M and input w, we construct an instance $P_{M,w}$ such that $P_{M,w}$ has a solution index sequence (or match) iff M accepts w. That is, $P_{M,w}$ has a match iff there is an accepting CH of M on w. So, if we can determine whether the instance has a match, we would be able to determine whether M accepts w. For this purpose, we choose the dominos in $P_{M,w}$ so that making a match forces a simulation of M to occur. To force simulation of M, we make 2 modifications to Turing Machine M and one change to our PCP problem:

1. M on input w never attempts to move the tape head beyond the left end of input tape.
2. If input w is the empty string ϵ, then we use the string $\_$ in place of w.
3. We modify the PCP problem to require that the first index in the solution index sequence is 1, i.e. the match starts with the first domino.

We call this problem the Modified Post Correspondence Problem (MPCP).

$$MPCP = \{< D > | D \textit{ is an instance of the PCP with a match that starts with first domino}\} \quad (4)$$

We will now prove the undecidability of PCP by first proving that MPCP is undecidable and then proving that MPCP can be reduced to PCP.

Proof: Let TM R decide the PCP and construct S deciding A_{TM}. S constructs an instance of the PCP $P_{M,w}$(=P) that has a match iff $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ accepts $w = w_1 w_2 ... w_n$. We will firstly show the construction of an instance P' of the MPCP by S. We do it through the following parts:

1. **First domino of the match**: Put $[\frac{\#}{\#q_0w_1w_2...w_n\#}]$ into P' as the first domino $[\frac{a_1}{b_1}]$ (this will be, by definition, the first domino of the match). The bottom string corresponds to the first configuration in the configuration history of M while M works on $w = w_1w_2...w_n$

2. **A transition of the TM M for the tape head moving right:** For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$, if $\delta(q, a) = (r, b, R)$, put $[\frac{qa}{br}]$ into P'. This handles head motions to the right.

3. **A transition of the TM M for the tape head moving left:** For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$, if $\delta(q, a) = (r, b, L)$, put $[\frac{cqa}{rcb}]$ into P'. This handles head motions to the left.

4. **Taking care of the contents of the tape where the tape head isn't there:** For every $a \in \Gamma$, put $[\frac{a}{a}]$ into P'. This handles the tape cells not adjacent to the head.

5. Note that only the first domino's numerator has a # while there are two #'s from the first domino itself. So, we need to add one more domino for copying the symbol. In addition to that, we need to add a blank symbol at the end of the configuration to simulate the infinitely many blanks to the right. So, put $[\frac{\#}{\#}]$ and $[\frac{\#}{\_\#}]$ into P'.

6. For every $a \in \Gamma$, put $[\frac{aq_{accept}}{q_{accept}}]$ and $[\frac{q_{accept}a}{q_{accept}}]$ into P'. These dominos have the effect of adding "pseudo-steps" of the TM after it has halted, where the head "eats" adjacent symbols until none are left.

7. Finally, we add the domino $[\frac{q_{accept}\#\#}{\#}]$ into P' and complete the match.

Proof and example in Board

THANK YOU