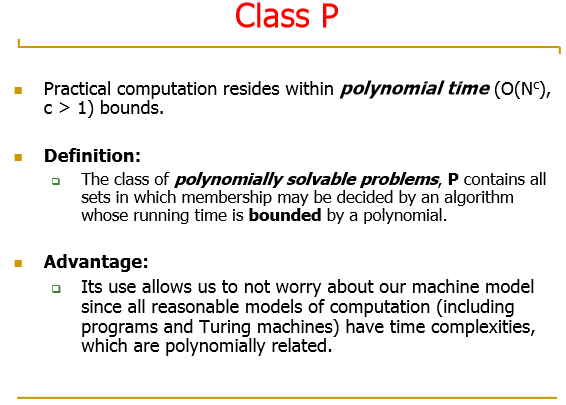
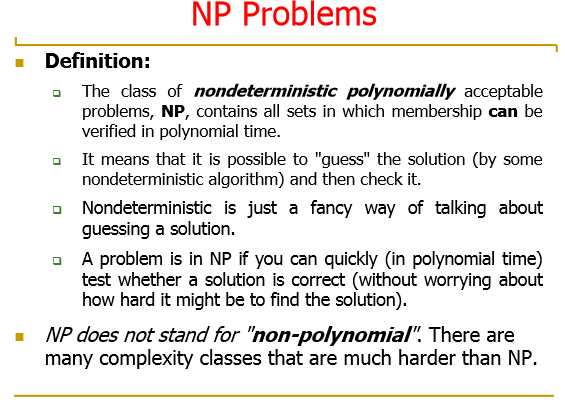
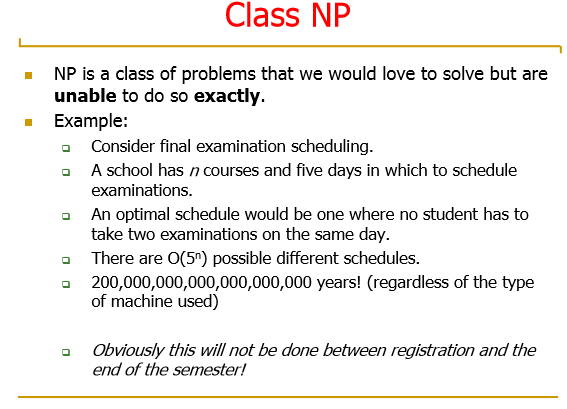
**UNIT-V**

P and NP problem:

The P versus NP problem is to determine whether every language accepted by some nondeterministic algorithm in polynomial time is also accepted by some (deterministic) algorithm in polynomial time. To define the problem precisely it is necessary to give a formal model of a computer.





**P Class-**-- It is a set of ***decision problems*** that are ***solvable*** in ***polynomial time.***..

**NP-Class**--- It is a set of ***decision problems*** that are ***not solvable*** but are ***verifiable*** in ***polynomial time***

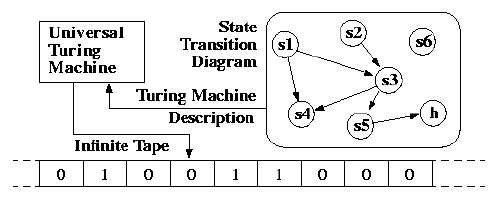
**NP Complete**--- It is one of the ***decision problem*** that belong to ***NP Class*** and all the ***problems in NP Class can be reduced*** to this problem..

**Universal TM:**

A **universal Turing machine** (**UTM**) is a [Turing machine](http://en.wikipedia.org/wiki/Turing_machine) that can simulate an arbitrary Turing machine on arbitrary input. The universal machine essentially achieves this by reading both the description of the machine to be simulated as well as the input thereof from its own tape.

The following example is taken from Turing (1936). For more about this example see the page [Turing machine examples](http://en.wikipedia.org/wiki/Turing_machine_examples).

Turing used seven symbols { A, C, D, R, L, N, ; } to encode each 5-tuple; as described in the article [Turing machine](http://en.wikipedia.org/wiki/Turing_machine), his 5-tuples are only of types N1, N2, and N3. The number of each "m-configuration" (instruction, state) is represented by "D" followed by a unary string of A's, i.e. "q3" = DAAA. In a similar manner he encodes the symbols blank as "D", the symbol "0" is "DC", the symbol "1" as DCC, etc. The symbols "R", "L", and "N" remain as is.



A Turing Machine is the mathematical tool equivalent to a digital computer. It was suggested by the mathematician Turing in the 30s, and has been since then the most widely used model of computation in computability and complexity theory.

The model consists of an input output relation that the machine computes. The input is given in binary form on the machine's tape, and the output consists of the contents of the tape when the machine halts.

What determines how the contents of the tape change is a finite state machine (or FSM, also called a finite automaton) inside the Turing Machine. The FSM is determined by the number of states it has, and the transitions between them.

At every step, the current state and the character read on the tape determine the next state the FSM will be in, the character that the machine will output on the tape (possibly the one read, leaving the contents unchanged), and which direction the head moves in, left or right.

The problem with Turing Machines is that a different one must be constructed for every new computation to be performed, for every input output relation.

This is why we introduce the notion of a universal turing machine (UTM), which along with the input on the tape, takes in the description of a machine M. The UTM can go on then to simulate M on the rest of the contents of the input tape. A universal turing machine can thus simulate any other machine.

**Context-sensitive language**

A **context-sensitive grammar** (**CSG**) is a [formal grammar](http://en.wikipedia.org/wiki/Formal_grammar) in which the left-hand sides and right-hand sides of any [production rules](http://en.wikipedia.org/wiki/Production_%28computer_science%29) may be surrounded by a context of [terminal](http://en.wikipedia.org/wiki/Terminal_symbol) and [nonterminal symbols](http://en.wikipedia.org/wiki/Nonterminal_symbol). Context-sensitive grammars are more general than [context-free grammars](http://en.wikipedia.org/wiki/Context-free_grammar) but still orderly enough to be [parsed](http://en.wikipedia.org/wiki/Parsing) by a [linear bounded automaton](http://en.wikipedia.org/wiki/Linear_bounded_automaton).

[Noam Chomsky](http://en.wikipedia.org/wiki/Noam_Chomsky) introduced context-sensitive grammars in the 1950s as a way to describe the syntax of [natural language](http://en.wikipedia.org/wiki/Natural_language) where it is indeed often the case that a word may or may not be appropriate in a certain place depending upon the context. A [formal language](http://en.wikipedia.org/wiki/Formal_language) that can be described by a context-sensitive grammar is called a [context-sensitive languag](http://en.wikipedia.org/wiki/Context-sensitive_language)

A [formal grammar](http://en.wikipedia.org/wiki/Formal_grammar) *G* = (*N*, Σ, *P*, *S*), where *N* is a set of nonterminal symbols, Σ is a set of terminal symbols, *P* is a set of production rules, and *S* is the start symbol, is context-sensitive if all rules in *P* are of the form

α*A*β → αγβ

where *A* ∈ *N* (i.e., *A* is a single [nonterminal](http://en.wikipedia.org/wiki/Nonterminal)), α,β ∈ (*N* U Σ)\* (i.e., α and β are strings of nonterminals and [terminals](http://en.wikipedia.org/wiki/Terminal_symbol)) and γ ∈ (*N* U Σ)+ (i.e., γ is a nonempty string of nonterminals and terminals).

Some definitions also add that for any production rule of the form u → v of a context-sensitive grammar, it shall be true that |u|≤|v|. Here |u| and |v| denote the length of the strings respectively.

In addition, a rule of the form

S → λ

where λ represents the [empty string](http://en.wikipedia.org/wiki/Empty_string) and S does not appear on the right-hand side of any rule is permitted. The addition of the empty string allows the statement that the context sensitive languages are a proper superset of the context free languages, rather than having to make the weaker statement that all context free grammars with no →λ productions are also context sensitive grammars.

The name *context-sensitive* is explained by the α and β that form the context of *A* and determine whether *A* can be replaced with γ or not. This is different from a [context-free grammar](http://en.wikipedia.org/wiki/Context-free_grammar) where the context of a nonterminal is not taken into consideration. (Indeed, every production of a context free grammar is of the form V → w where V is a *single* nonterminal symbol, and w is a string of terminals and/or nonterminals (w can be empty)).

If the possibility of adding the empty string to a language is added to the strings recognized by the noncontracting grammars (which can never include the empty string) then the languages in these two definitions are identical.

Example:

* This grammar generates the canonical non-[context-free language](http://en.wikipedia.org/wiki/Context-free_language) Description:  \{ a^n b^n c^n \mid n \ge 1 \} :

1. Description: S \rightarrow aSBC
2. Description: S \rightarrow aBC   
3. Description: CB \rightarrow HB   
4. Description: HB \rightarrow HC   
5. Description: HC \rightarrow BC   
6. Description: aB \rightarrow ab   
7. Description: bB \rightarrow bb   
8. Description: bC \rightarrow bc   
9. Description: cC \rightarrow cc   

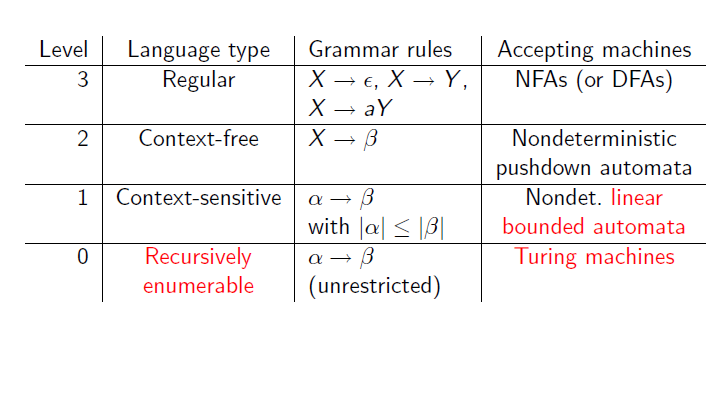
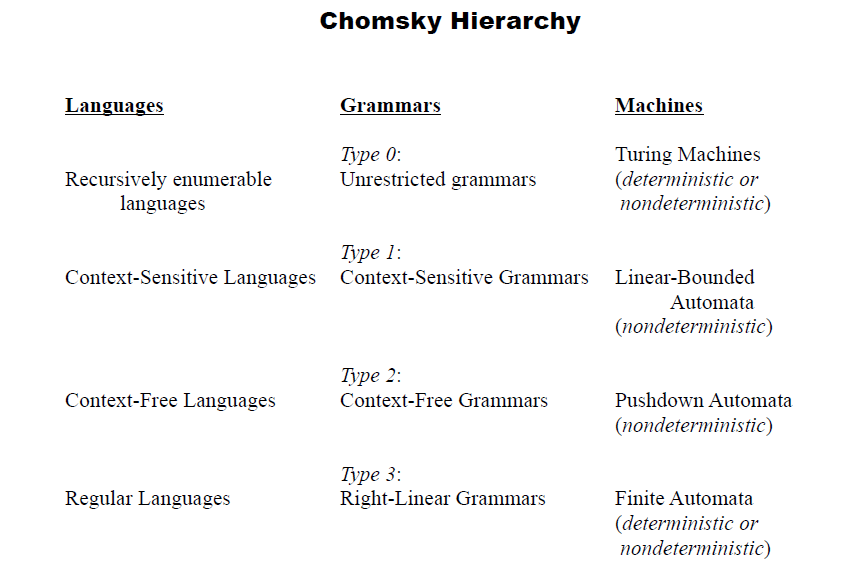
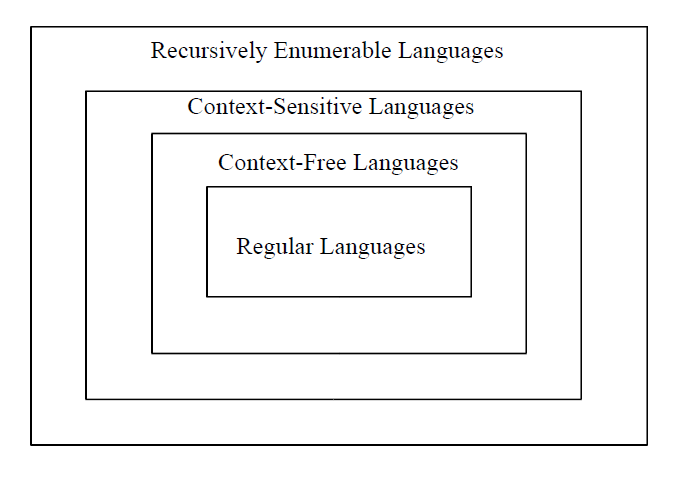
Linear bounded automata:

Definition. *A linear bounded automaton (lba) is a multi-track Turing machine which has only one tape, and this tape is exactly the same length as the input.*

Linear bounded automata satisfy the following three conditions:

1. Its input alphabet includes two special symbols, serving as left and right endmarkers.
2. Its transitions may not print other symbols over the endmarkers.
3. Its transitions may neither move to the left of the left endmarker or to the right of the right endmarker.

Chomsky hierarchy:

*  Type-0 grammars ([unrestricted grammars](http://en.wikipedia.org/wiki/Unrestricted_grammar)) include all formal grammars. They generate exactly all languages that can be recognized by a [Turing machine](http://en.wikipedia.org/wiki/Turing_machine). These languages are also known as the [recursively enumerable languages](http://en.wikipedia.org/wiki/Recursively_enumerable_language). Note that this is different from the [recursive languages](http://en.wikipedia.org/wiki/Recursive_language) which can be *decided* by an [always-halting Turing machine](http://en.wikipedia.org/wiki/Machine_that_always_halts).
* Type-1 grammars ([context-sensitive grammars](http://en.wikipedia.org/wiki/Context-sensitive_grammar)) generate the [context-sensitive languages](http://en.wikipedia.org/wiki/Context-sensitive_language). These grammars have rules of the form Description: \alpha A\beta \rightarrow \alpha\gamma\betawith Description: Aa nonterminal and Description: \alpha, Description: \betaand Description: \gammastrings of terminals and nonterminals. The strings Description: \alphaand Description: \betamay be empty, but Description: \gammamust be nonempty. The rule Description: S \rightarrow \epsilonis allowed if Description: Sdoes not appear on the right side of any rule. The languages described by these grammars are exactly all languages that can be recognized by a [linear bounded automaton](http://en.wikipedia.org/wiki/Linear_bounded_automaton) (a nondeterministic Turing machine whose tape is bounded by a constant times the length of the input.)
* Type-2 grammars ([context-free grammars](http://en.wikipedia.org/wiki/Context-free_grammar)) generate the [context-free languages](http://en.wikipedia.org/wiki/Context-free_language). These are defined by rules of the form Description: A \rightarrow \gammawith Description: Aa nonterminal and Description: \gammaa string of terminals and nonterminals. These languages are exactly all languages that can be recognized by a non-deterministic [pushdown automaton](http://en.wikipedia.org/wiki/Pushdown_automaton). Context-free languages are the theoretical basis for the syntax of most [programming languages](http://en.wikipedia.org/wiki/Programming_language).
* Type-3 grammars ([regular grammars](http://en.wikipedia.org/wiki/Regular_grammar)) generate the [regular languages](http://en.wikipedia.org/wiki/Regular_language). Such a grammar restricts its rules to a single nonterminal on the left-hand side and a right-hand side consisting of a single terminal, possibly followed (or preceded, but not both in the same grammar) by a single nonterminal. The rule Description: S \rightarrow \epsilonis also allowed here if Description: Sdoes not appear on the right side of any rule. These languages are exactly all languages that can be decided by a [finite state automaton](http://en.wikipedia.org/wiki/Finite_state_automaton). Additionally, this family of formal languages can be obtained by [regular expressions](http://en.wikipedia.org/wiki/Regular_expressions). Regular languages are commonly used to define search patterns and the lexical structure of programming languages.

# Finite Automata with Output

There are two finite state machine models :

**Mealy model** – in which outputs occur during transitions.

**Moore model** – outputs are produced upon arrival at a new state.

## Mealy Model of FSM

**Mealy model** – transition assigned output

Mt = <Q, S, R, f, g, qI>

Q = finite set of states // the machine’s memory

S = input alphabet // set of stimuli

R = output alphabet // set of responses

qI = the machine’s initial state

f : state transition function (or next state function)

f : Q \* S 🡪 Q

g : output function

g : Q \* S 🡪 R

**example**

Design a FSM (Mealy model) which takes in binary inputs and produces a ‘1’ as output whenever the parity of the input string ( *so far* ) is even.

S = R = {0, 1}

When designing such models, we should ask ourselves *“What is the state set of the machine?”.*

*The state set Q corresponds to what we need to remember about input strings*. We note that the number of possible input strings corresponds to |S\*| which is *countably infinite.*

We observe, however, that a string may have only one of two possible parities.

**even parity** – if n1(w) is even.

**odd parity** – if n1(w) is odd.

*And this is all that our machine must remember about a string scanned so far.*

Hence |Q| = 2 where Q = {E, σ} with qI = E indicating the string has *even parity* and if Mt *is in state σ*, then the string has *odd parity*.

* And finally, of course, we must specify *the output function g* for this Mealy machine.
* According to this machine’s specifications, it is supposed to produce an output of ‘1’ whenever the parity of the input string so far is even. Hence, *all arcs leading into state E should be labeled with a ‘1’ output.*

**Parity Checker (Mealy machine)**

**state diagram**

E

σ

0/1

0/0

1/1

1/0

Observe our notation that **g(σ, 1) = 1** is indicated by the arc from state σ to state E with a ‘1’ after a slash

The output of our machine is 0 when the current string ( *so far* ) has odd parity.

|  |  |  |  |
| --- | --- | --- | --- |
| state table | present state | input = 0  next state, output | input = 1  next state, output |
| for this  parity machine | E | E, 1 | σ, 0 |
|  | σ | σ, 0 | E, 1 |

Observe for the input 10100011 our machine produces the output sequence 00111101

1/0

σ

σ

E

E

E

E

E

σ

E

0/0

1/1

0/1

0/1

0/1

1/0

1/1

the corresponding *admissible state sequence*

**a second example**

Construct a **Mealy model** of an FSM that behaves as a two-unit delay. i.e.

r(t) = {s(t - 2), t > 2

{ 0 , otherwise

*A sample input/output session is given below :*

**time** 1 2 3 4 5 6 7 8 9

**stimulus** 0 0 0 1 1 0 1 0 0

**response** 0 0 0 0 0 1 1 0 1

Observe that r(1) = r(2) = 0

r(6) = 1 which equals s(4) and so on

We know that S = R = {0, 1}.

**But what is the state set Q ???**

**Moore model of FSM**

**Moore model of FSM** – the output function assigns an output symbol to each state.

Ms = <Q, S, R, f, h, qI>

Q = finite set of internal states

S = finite input alphabet

R = finite output alphabet

f : state transition function

f : Q \* S 🡪 Q

h : output function

h : Q → R

qI = Є Q is the initial state

**example**

Design a **Moore machine** that will analyze input sequences in the binary alphabet S = {0, 1}. Let w = s(1) s(2) … s(t) be an input string

N0(w) = number of 0’s in w

N1(w) = number of 1’s in w

then we have that |w| = N0(w) + N1(w) = t.

The last output of Ms should equal : r(t) = [N1(w) – N0(w)] mod 4.

So naturally, *the output alphabet R = {0, 1, 2, 3}*

A sample stimulus/response is given below :

**stimulus** 1 1 0 1 1 1 0 0

**response** 0 1 2 1 2 3 0 3 2

* Observe that the length of the output sequence is one longer than the input sequence. Why is this so?
* Btw : This will always be the case.

The corresponding Moore machine :

B, 1

A, 0

C, 2

D, 3

1

0

1

0

1

0

0

1

**state diagram**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 |  |
| A | D | B | 0 |
| B | A | C | 1 |
| C | B | D | 2 |
| D | C | A | 3 |

**state table**

This machine is referred to as an *up-down counter.*

For the previous input sequence : 11011100 the *state sequence is :*

(A, 0)

(B, 1)

(C, 2)

(B, 1)

(C, 2)

(D, 3)

(A, 0)

(D, 3)

(C, 2)

1

1

0

1

1

1

0

0

**second example**

Design a **Moore machine** that functions as a *pattern recognizer* for “1011”. Your machine should output a ‘1’ whenever this pattern matches the last four inputs, and there has been no overlap, otherwise output a ‘0’.

Hence S = R = {0, 1}.

* Here is a sample input/output sequence for this machine :

**t** = 1 2 3 4 5 6 7 8 9 10 11 12

**S** = 0 1 0 1 1 0 1 1 0 1 1 0

**R** = 0 0 0 0 1 0 0 0 0 0 0 1 0

* We observe that **r(5) = 1** because s(2) s(3) s(4) s(5) = 1011

however **r(8) = 0** because there has been overlap

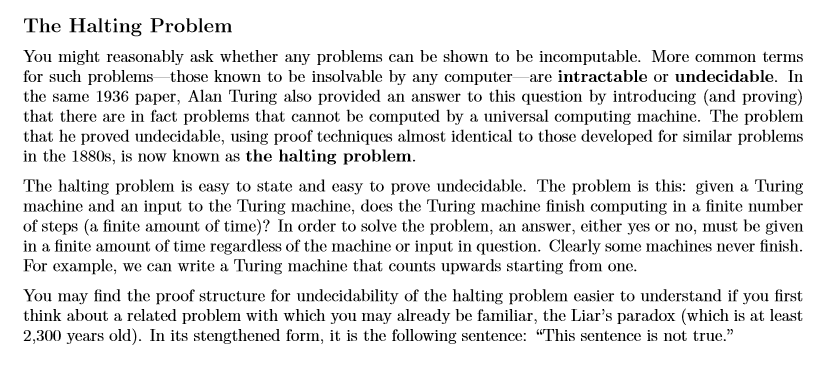
**r(11) = 1** since s(8) s(9) s(10) s(11) = 1011

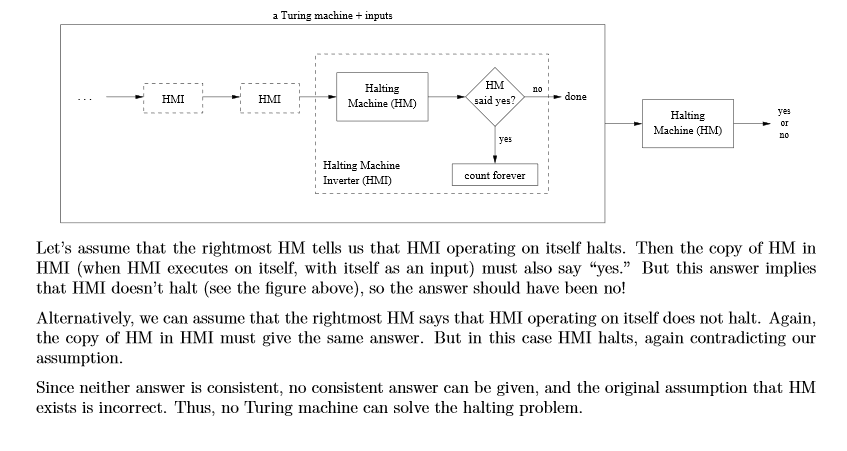
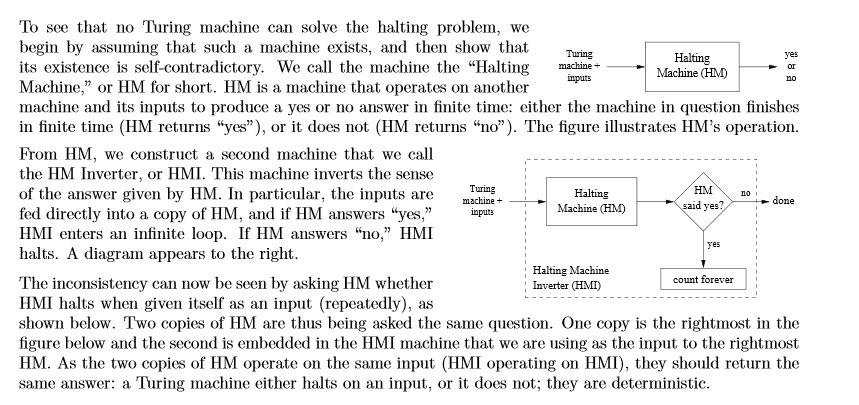
**Decidability of problems**

**1.Halting problem**

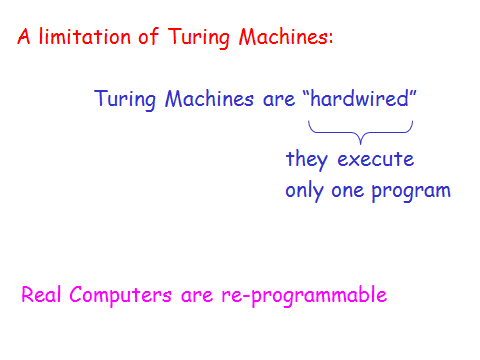
**2.Post correspondence problem**

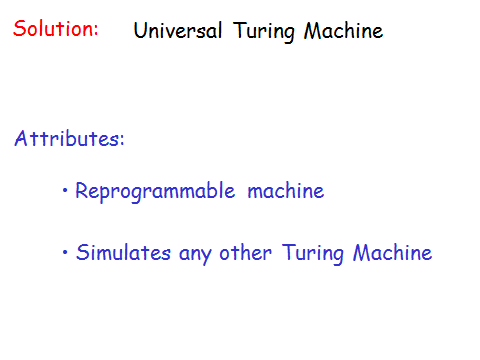
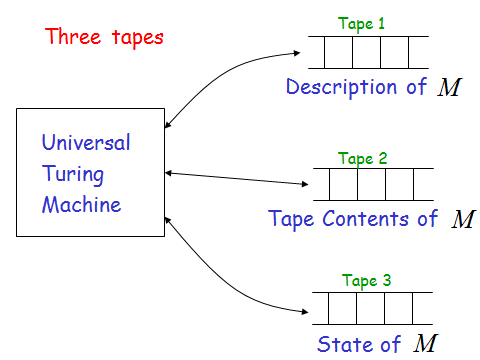
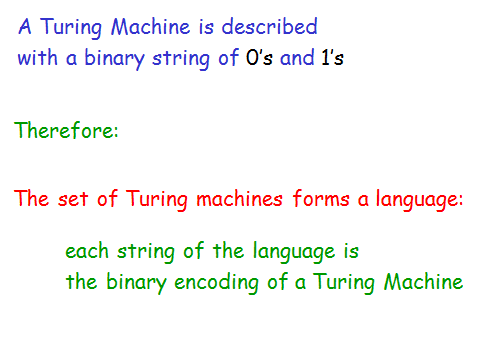
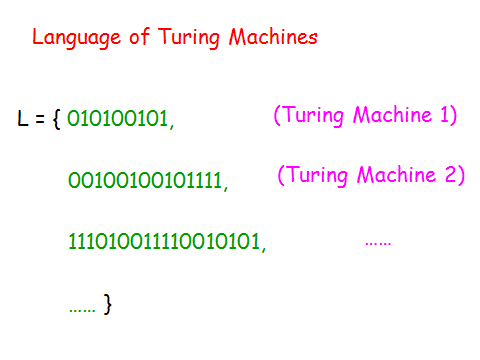
**Halting problem:**





**Universal Turing Machine**

****

**** ****  

Post’s correspondence problem.

