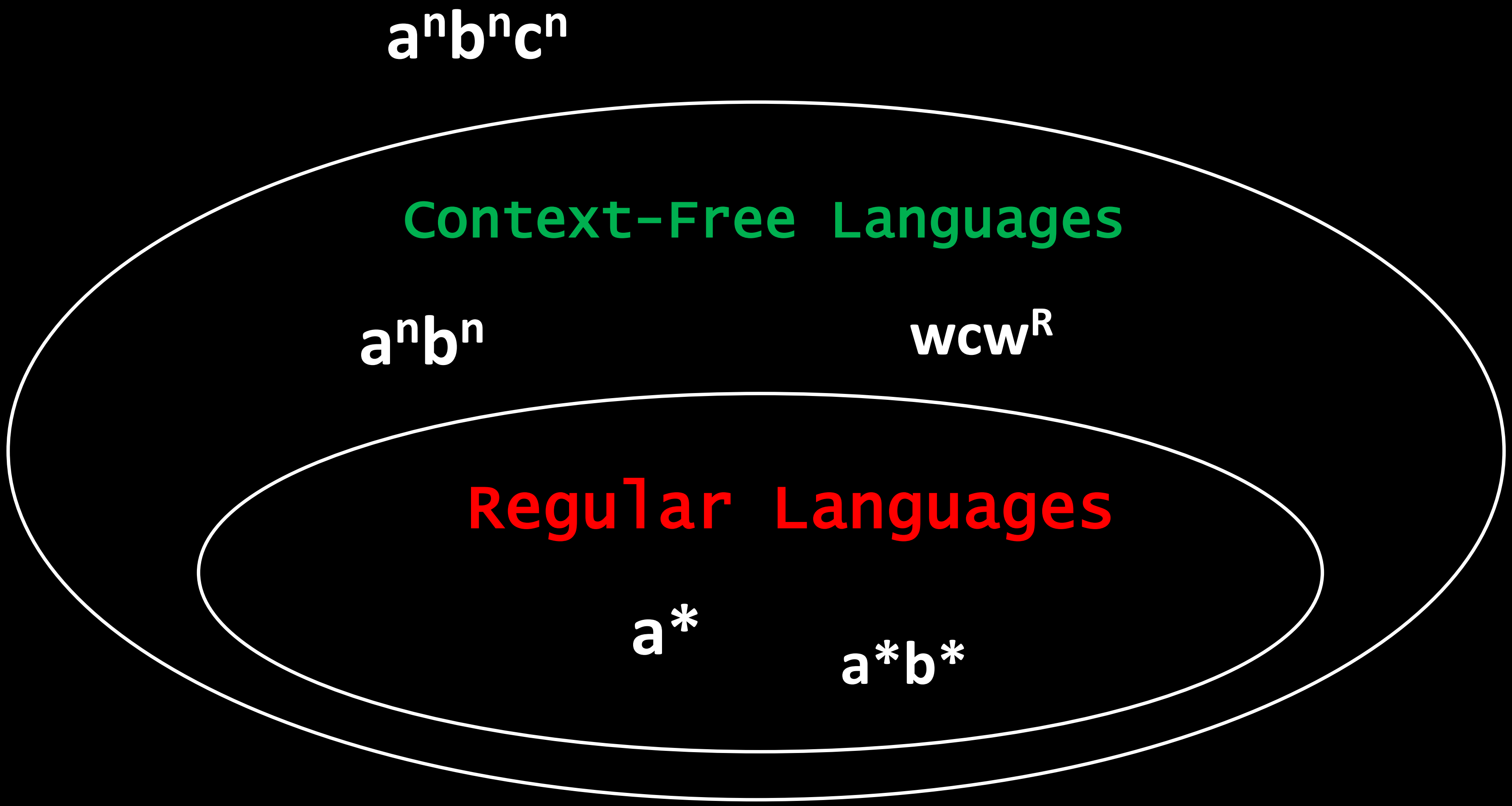
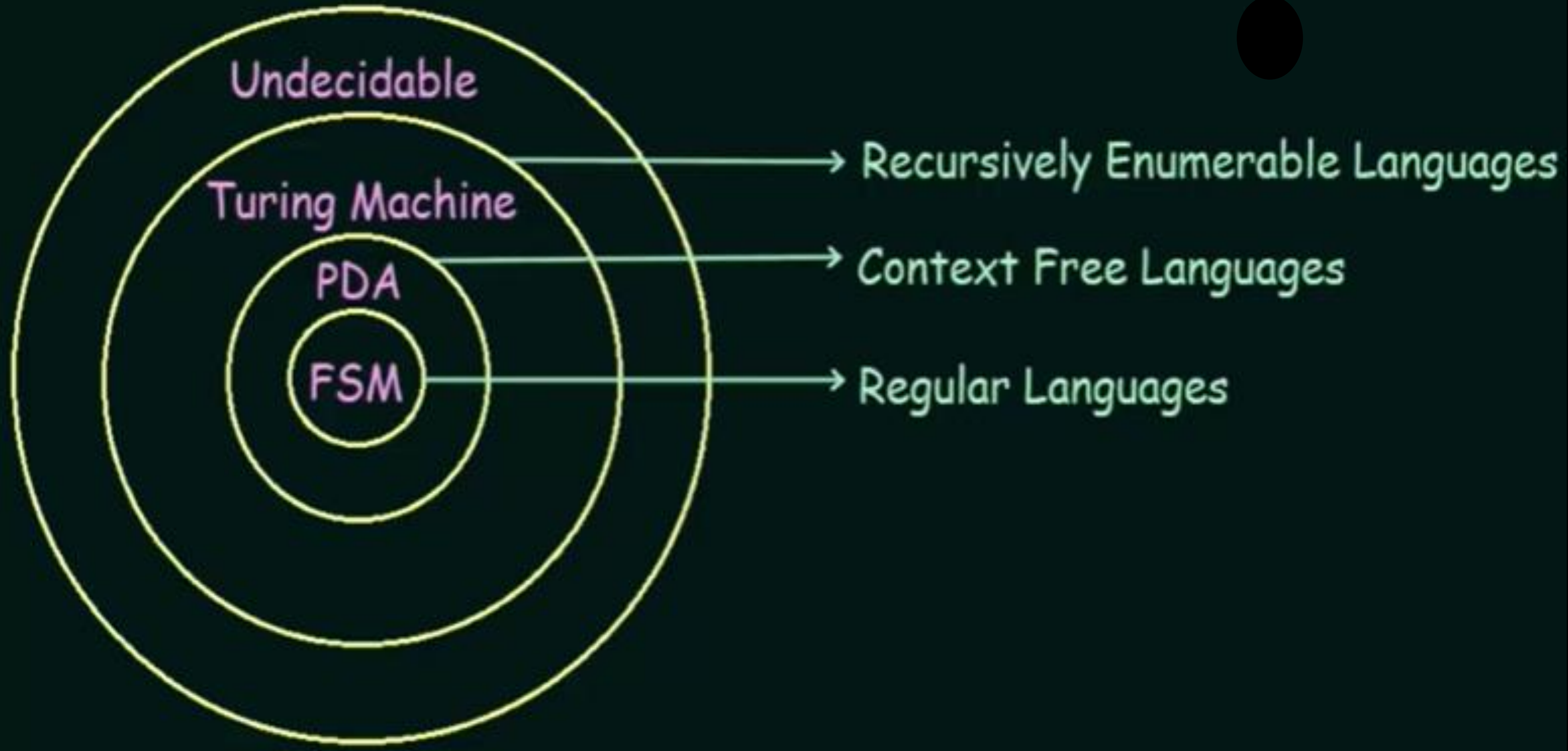

بالعلم نرتقي

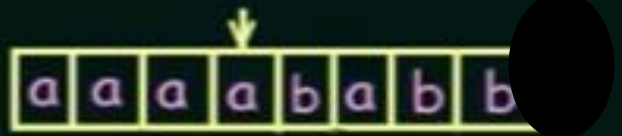
Turing machines

The language hierarchy



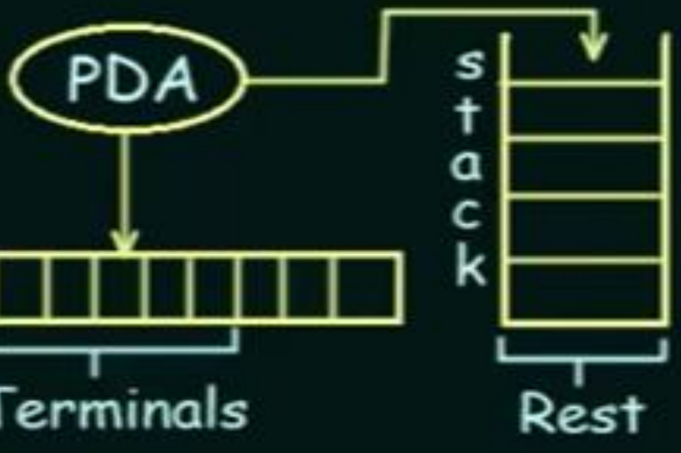

Undecidable
Turing Machine
PDA
FSM
Recursively Enumerable Languages
Context Free Languages
Regular Languages

FSM: The Input String



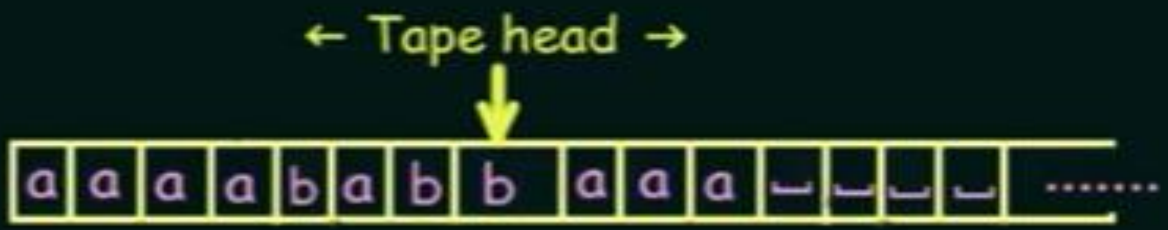
PDA: -> The Input String

-> A Stack



TURING MACHINE:

-> A Tape



Tape Alphabets: $\Sigma = \{0,1,a,b,x,Z_0\}$

The Blank $\sqcup$ is a special symbol. $\sqcup \notin \Sigma$

The blank is a special symbol used the fill the infinite tape

Turing Machine as Problem Solvers

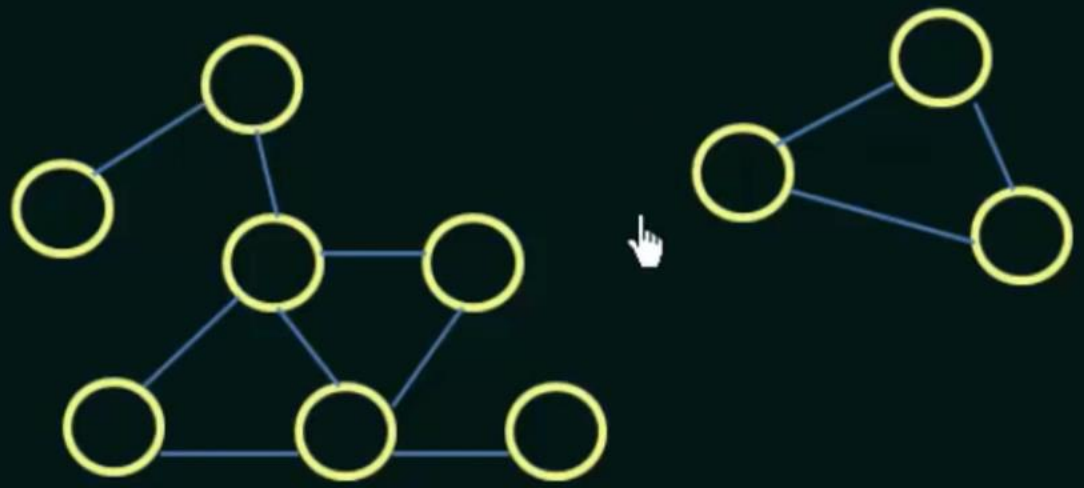
Any arbitrary Problem can be expressed as a language

-Any instance of the problem is encoded into a string

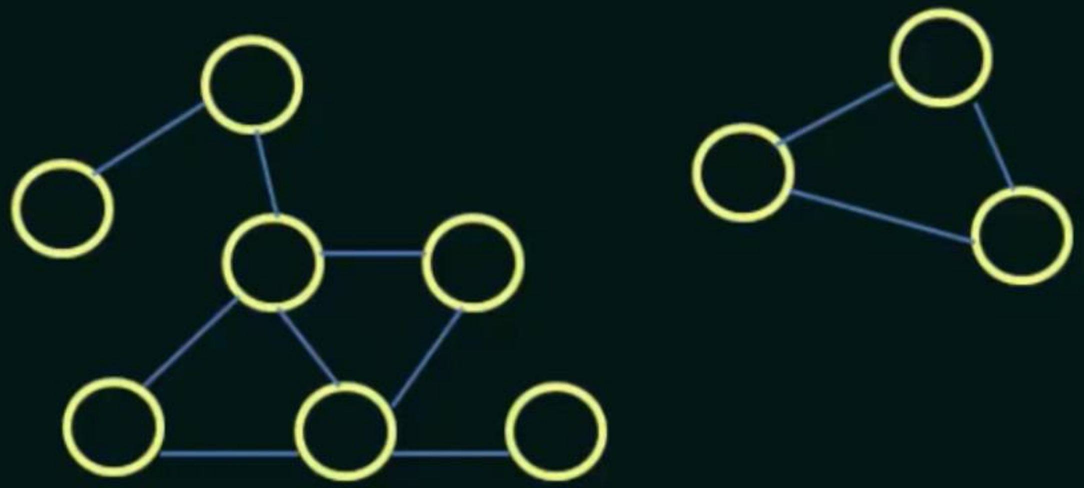
The string is in the language $\Rightarrow$ The answer is YES

The string is not in the language $\Rightarrow$ The answer is NO

Example: Is this undirected graph connected?



Example: Is this undirected graph connected?



We must encode the problem into a language.

A = { ‹G› | G is a connected graph }

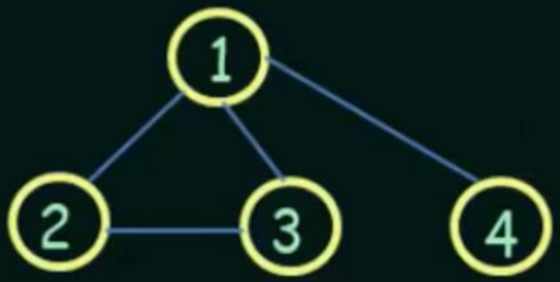
We would like to find a TM to decide this language:

ACCEPT = "YES", This is a connected graph

REJECT = "NO", This is not a connected graph / or this is not a valid Representation of a graph.

LOOP = This problem is decidable. Our TM will always halt

Representation of Graph:



$$\langle G \rangle = \underbrace{(1,2,3,4)}_{\text{List of nodes}} \quad \underbrace{(\,(1,2),\,(2,3),\,(1,3),\,(1,4)\,)}_{\text{Edges}}$$

$$\Sigma = \{\ (,\ ),\ ,,\ 1,\ 2,\ 3,\ 4,\ \ldots\ldots,0\ \}$$

(	1	,	2	,	3	,	4	,	....	)	)	␣	.....

High Level Algorithm:

```
Select a Node and Mark it
REPEAT
    For each node N
        If N is unmarked and there is an edge from N to an already marked
        node
    Then
        Mark Node N
    End

Until no more nodes can be marked
    For each Node N
        If N is unmarked
        Then REJECT
    End
ACCEPT
```

Implementation Level Algorithm:

- Check that input describes a valid graph
- Check Node List
    - Scan "(" followed by digits ...
    - Check that all nodes are different i.e. no repeats
    - Check edge lists ...

        etc.

- Mark First Node
    - Place a dot under the first node in the node list
    - Scan the node list to find a node that is not marked

        etc.

Decidability and Undecidability

Recursive Language:

- A language 'L' is said to be recursive if there exists a Turing machine which will accept all the strings in 'L' and reject all the strings not in 'L'.
- The Turing machine will halt every time and give an answer (accepted or rejected) for each and every string input.

Recursively Enumerable Language:

- A language 'L' is said to be a recursively enumerable language if there exists a Turing machine which will accept (and therefore halt) for all the input strings which are in 'L'.
- But may or may not halt for all input strings which are not in 'L'.

Decidable Language:

A language 'L' is decidable if it is a recursive language. All decidable languages are recursive languages and vice-versa.

Partially Decidable Language:

A language 'L' is partially decidable if 'L' is a recursively enumerable language.

Undecidable Language:

- A language is undecidable if it is not decidable.
- An undecidable language may sometimes be partially decidable but not decidable.
- If a language is not even partially decidable, then there exists no Turing machine for that language

Recursive Language	TM will always Halt
Recursively Enumerable Language	TM will halt sometimes & may not halt sometimes
Decidable Language	Recursive Language
Partially Decidable Language	Recursively Enumerable Language
UNDECIDABLE	No TM for that language

The Universal Turing Machine

The Language

A_{TM} = { <M,w> | M is a Turing Machine and M accepts w}

is Turing Recognizable

Given the description of a TM and some input, can we determine whether the machine accepts it?

- Just Simulate/ Run the TM on the input

M Accepts w: Our Algorithm will Halt & Accept

M Rejects w: Our Algorithm will Halt & Reject.

M Loops on w: Our Algorithm will not Halt.

The Halting Problem

Given a Program, WILL IT HALT ?

Given a Turing Machine, will it halt when run on some particular given input string?

Given some program written in some language (Java/C/ etc.) will it ever get into an infinite loop or will it always terminate?

Answer:

- In General we can't always know.
- The best we can do is run the program and see whether it halts.
- For many programs we can see that it will always halt or sometimes loop

BUT FOR PROGRAMS IN GENERAL THE QUESTION IS UNDECIDABLE.

The Universal Turing Machine

Input: M = the description of some TM
w = an input string for M

Action: - Simulate M
- Behave just like M would (may accept, reject or loop)

The UTM is a recognizer (but not a decider) for

$A_{TM} = \{ \langle M,w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$

The Halting Problem

Given a Program, WILL IT HALT ?

Given a Turing Machine, will it halt when run on some particular given input string?

Given some program written in some language (Java/C/ etc.) will it ever get into an infinite loop or will it always terminate?

Binary strings - end - 0

11010110 = 0 ✓

= 1 ✗

Accepts all valid Java codes

binary

valid ?

invalid

✓

Eg. Compilers

Accepts all valid Java codes and never goes into infinite loop.

The Halting Problem

Given a Program, WILL IT HALT ?

Given a Turing Machine, will it halt when run on some particular given input string?

Given some program written in some language (Java/C/ etc.) will it ever get into an infinite loop or will it always terminate?

Answer:

- In General we can't always know.
- The best we can do is run the program and see whether it halts.
- For many programs we can see that it will always halt or sometimes loop

BUT FOR PROGRAMS IN GENERAL THE QUESTION IS UNDECIDABLE.

Undecidability of the Halting Problem

Given a Program, WILL IT HALT ?

Can we design a machine which if given a program can find out or decide if that program will always halt or not halt on a particular input?

Let us assume that we can:

H (P, I)
- Halt
- Not Halt

This allows us to write another Program:

```
C ( X )
  if { H (X, X) == Halt }
      Loop Forever;
  else
      Return;
```

Let us assume that we can:

H (P, I)

- Halt
- Not Halt

This allows us to write another Program:

```
C ( X )
  if { H (X, X) == Halt }
      Loop Forever;
  else
      Return;
```

<u>If we run 'C' on itself:</u>

C (C)

H (C,C) == Halt → Not Halt

H (C,C) == Not Halt → Halt

END