Theory of Computation

Lecturer:

Safia Abbas*

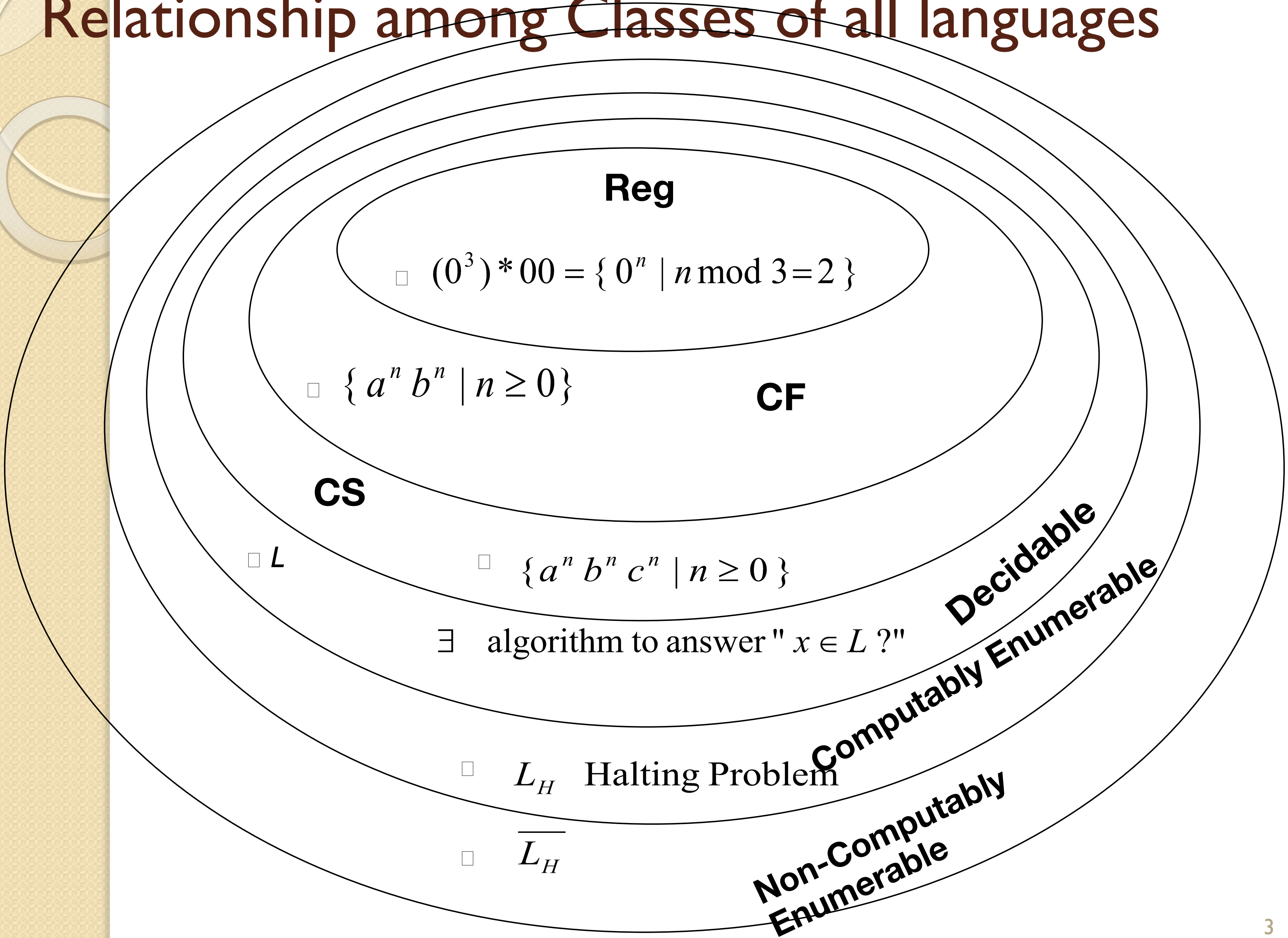
Text Books:
"***An Introduction to the theory of computation***" by Michael Sipser, 2nd Edition, PWS Publishing Company, Inc. 2006; ISBN: 0-534-95097-3

"***Theory of Computation an Introduction***" by James L. Hein, Jones & Bartlett Publishers 1996; ISBN: 0-86720-497-4.

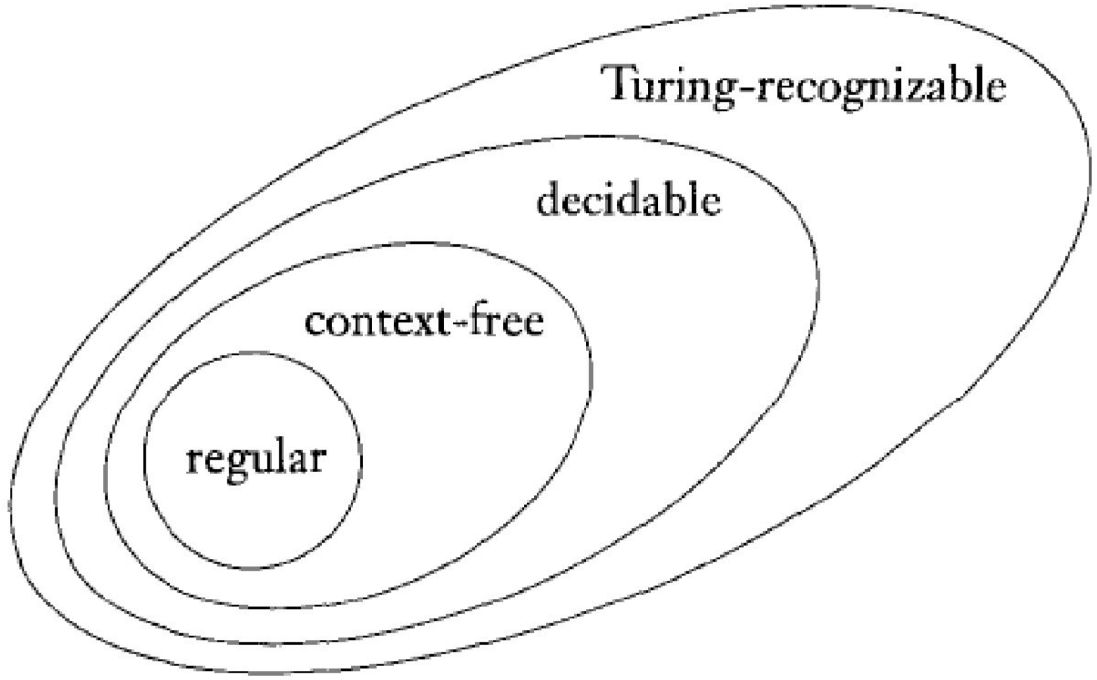
Agenda

- Chomsky hierarchy
- Halting problem
- Complexity theory
  - Class P
    - $0^n 1^n$
    - PATH
  - Class NP
    - HAMPATH
    - CLIQUE
- Non-Robustness of TM

Relationship among Classes of all languages


Reg
$(0^3)^*00 = \{ 0^n \mid n \bmod 3 = 2 \}$
$\{ a^n b^n \mid n \geq 0 \}$
CF
CS
L
$\{ a^n b^n c^n \mid n \geq 0 \}$
$\exists$ algorithm to answer "$x \in L$?"
Decidable
Computably Enumerable
L_H Halting Problem
$\overline{L_H}$
Non-Computably Enumerable

Relationship among Classes of all languages



Undecidablity and the Halting Problem.

Limitations of Turing machines

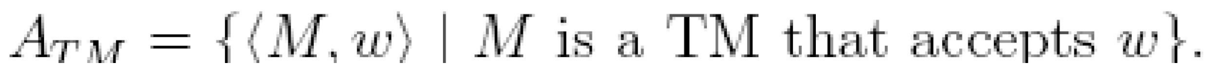
- Many of our computational tasks involve questions or decisions. For example, some problems involving numbers are:
  - *Is this integer a prime?*
  - *Does this equation have a root between 0 and 1?*
  - *Is this integer a perfect square?*
  - *Does this series converge?*
  - *Is this sequence of numbers sorted?*

However, not all tasks contains numbers. For example,

- *Is this program correct?*
- *How long will this program run?*
- *Does this program contain an infinite loop?*
- *Is this program more efficient than that one?*

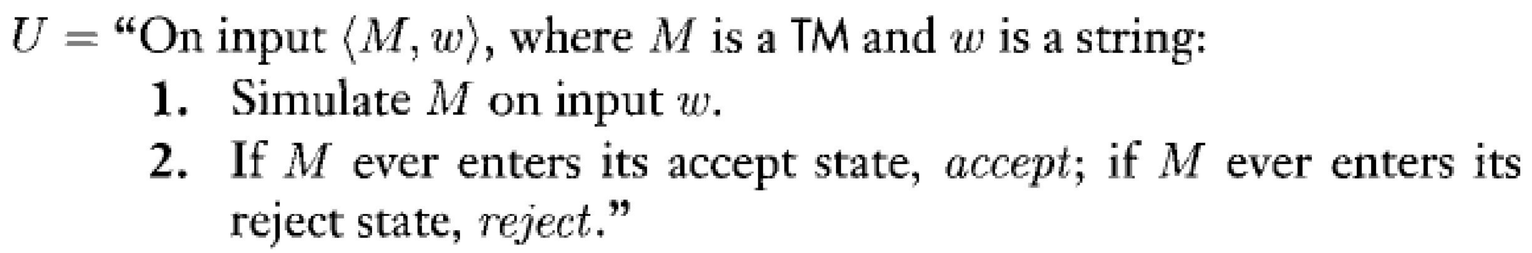
Reformulating the halting problem

Let

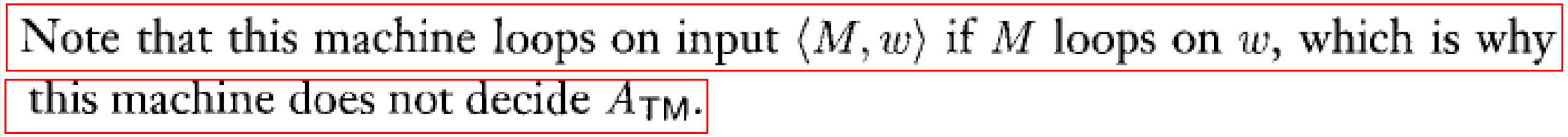


$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}.$

A_{TM} is sometimes called hating problem, A_{TM} is the problem of testing whether a Turing machine accepts a given input, it is an undecidable but recognizable.



```
U = "On input ⟨M, w⟩, where M is a TM and w is a string:
        1. Simulate M on input w.
        2. If M ever enters its accept state, accept; if M ever enters its
           reject state, reject."
```



Note that this machine loops on input $\langle M, w\rangle$ if M loops on w, which is why this machine does not decide A_{TM}.

Reducability

- It is the problem of converting one problem into another.
- The second problem can be solved, so the first problem as well.
- As example, the problem of finding your way around a new city can be reduced to the problem of finding a map to this city.
- The problem of traveling from one city to another can be reduced to the problem of buying the tickets.

Mapping reducibility

- *mapping reducibility,* roughly speaking, being able to reduce problem A to problem *B* .

- *a mapping reducibility:* means that a computable function exists that converts instances of problem A to instances of problem *B.*

- If such a conversion exists, the function called a *reduction,* we can solve A with a solver for *B.*

- The reason is that any instance of A can be solved by first using the reduction to convert it to an instance of *B* and then applying the solver for *B.*

Reducability

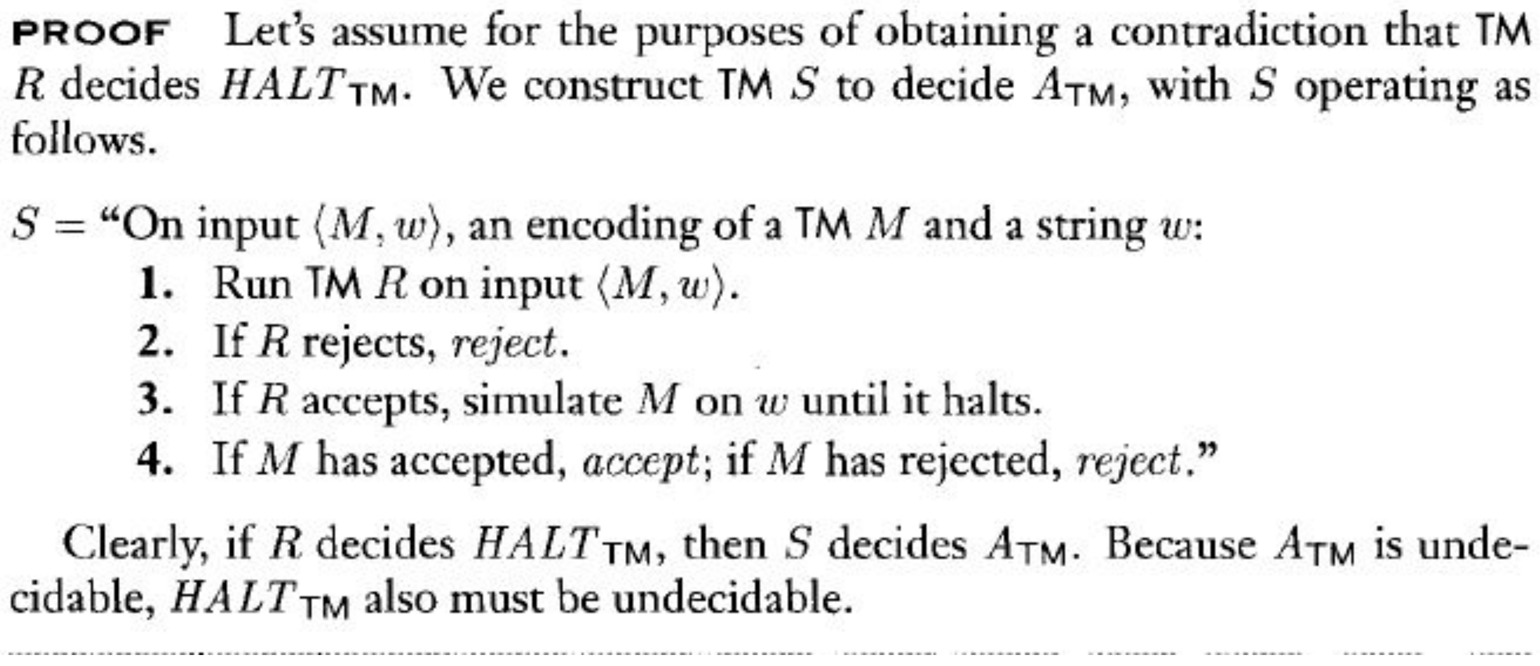
- The HALT_{TM} : the problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input.
- We use the undecidability of A_{TM} to prove the undecidability of $HALT_{TM}$ by reducing A_{TM} to $HALT_{TM}$. Let

Reducability

$HALT_{TM}$ is undecidable.

$HALT_{TM} = \{\langle M, w\rangle | \; M \text{ is a TM and } M \text{ halts on input } w\}$

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



PROOF Let's assume for the purposes of obtaining a contradiction that TM R decides $HALT_{TM}$. We construct TM S to decide A_{TM}, with S operating as follows.

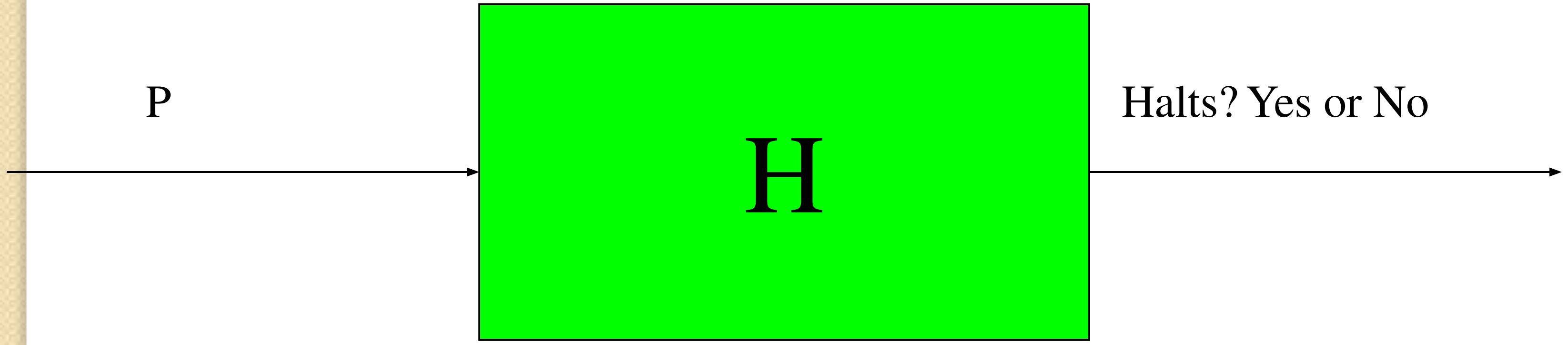
```
S = "On input ⟨M, w⟩, an encoding of a TM M and a string w:
    1. Run TM R on input ⟨M, w⟩.
    2. If R rejects, reject.
    3. If R accepts, simulate M on w until it halts.
    4. If M has accepted, accept; if M has rejected, reject."
```

Clearly, if R decides $HALT_{TM}$, then S decides A_{TM}. Because A_{TM} is undecidable, $HALT_{TM}$ also must be undecidable.

Halting Problem

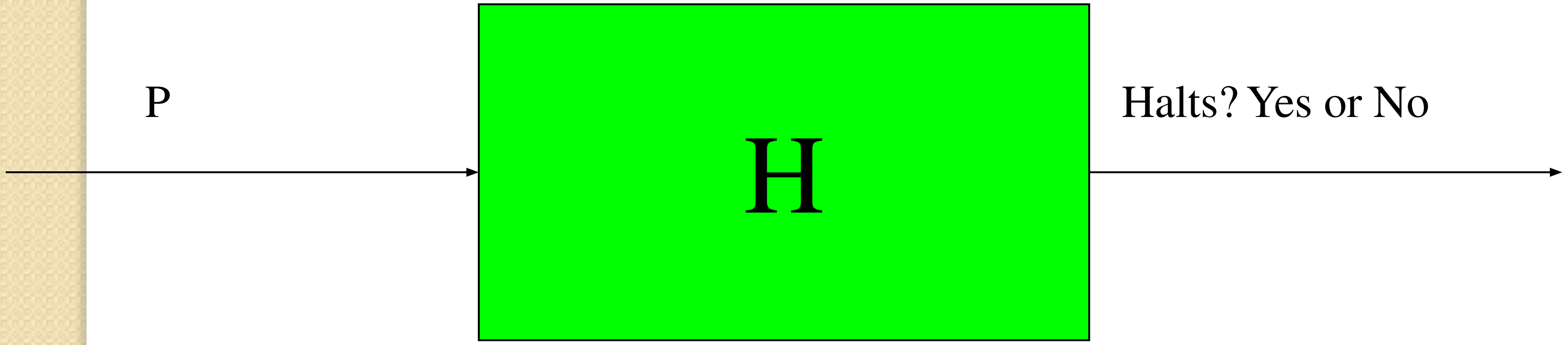
In other words, using recursion and paradox concepts

Program H will always halt and give the correct answer no matter what program has been given as input.



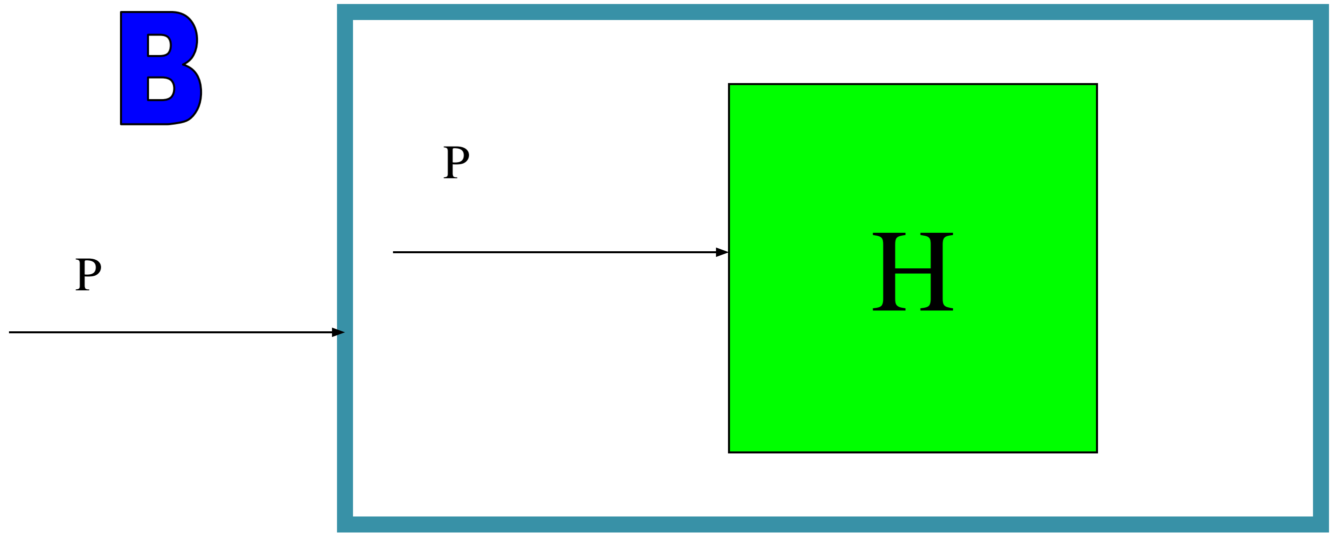
Assume Halting Problem OK

- Let H be a program (or sub-program) that determines whether a program will halt.



Let’s Build Another Program

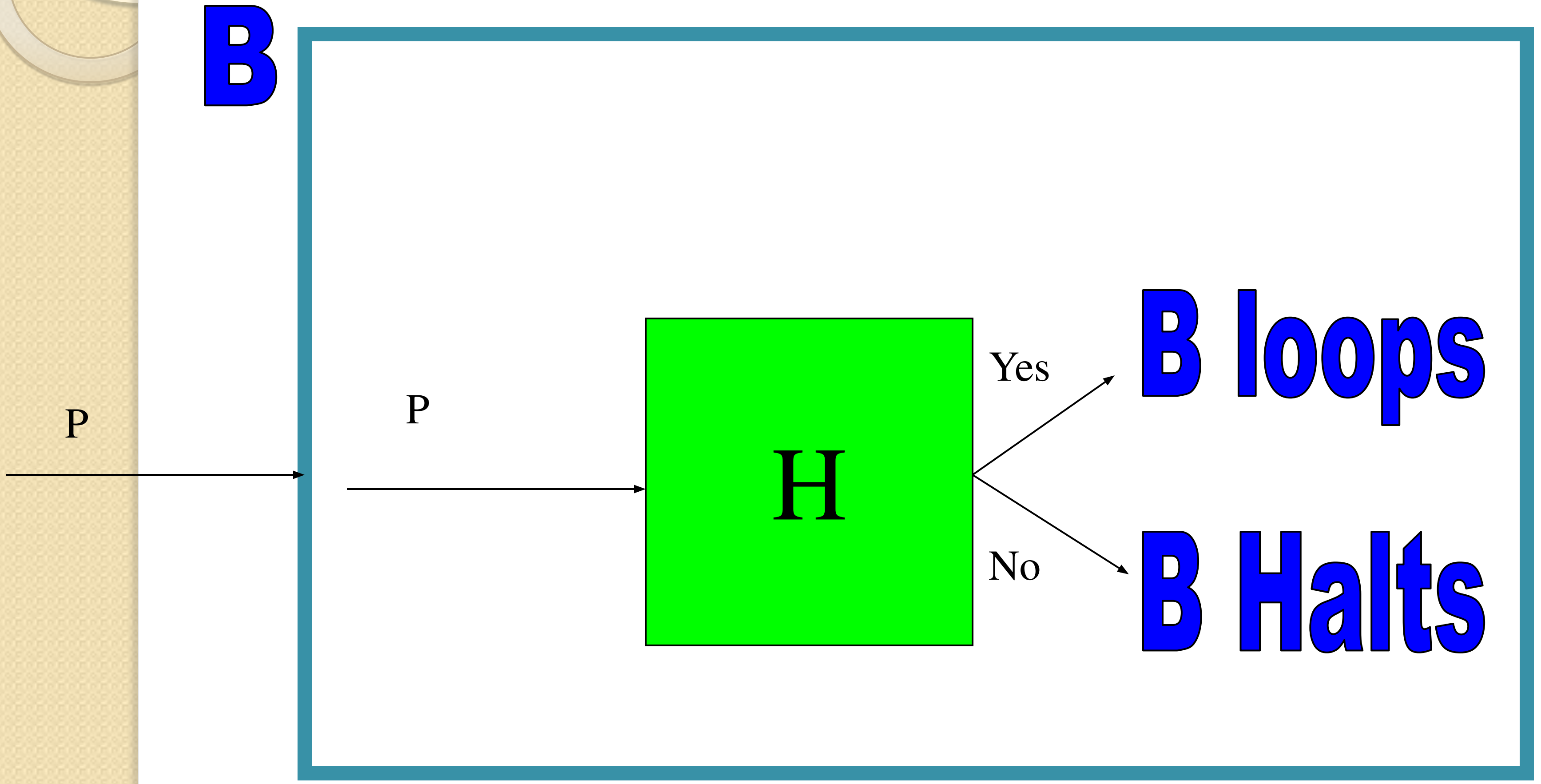
- Let B be a program that uses H.
- For any given program, B will call H and pass it the given program.



What does P do?

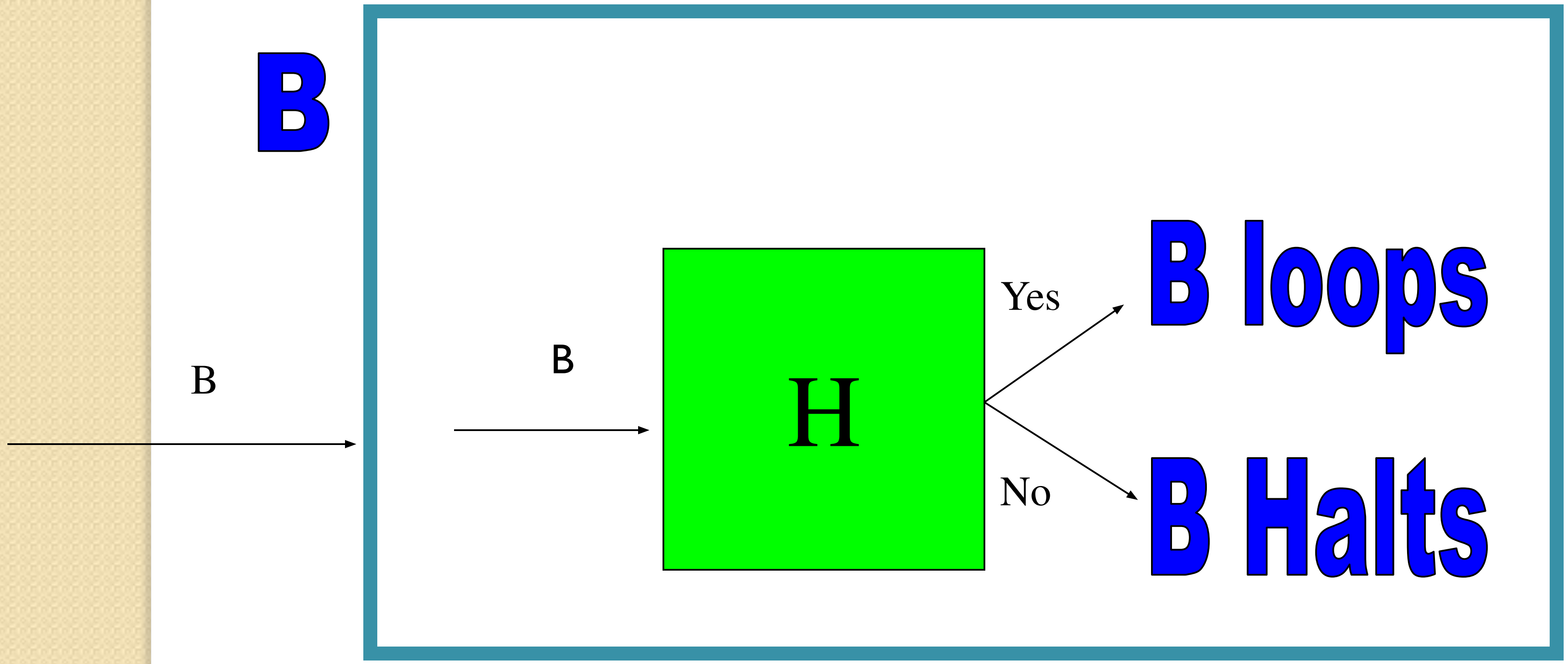
- Now, B acts as follows:
  - B takes a program as input and feeds the program to H as input.
  - If H answer yes, then B will enter an infinite loop and run forever.
  - If H answer no, then B will stop.

What does B do?

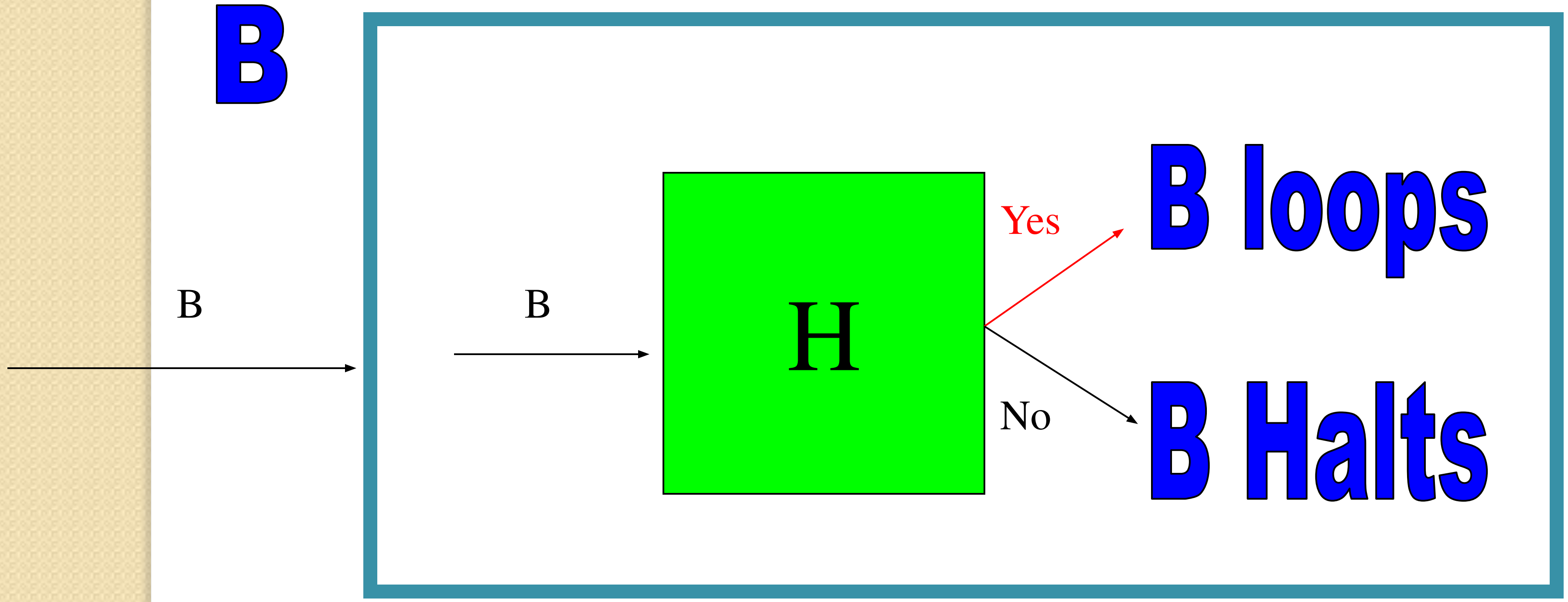


What Can We Do With P?

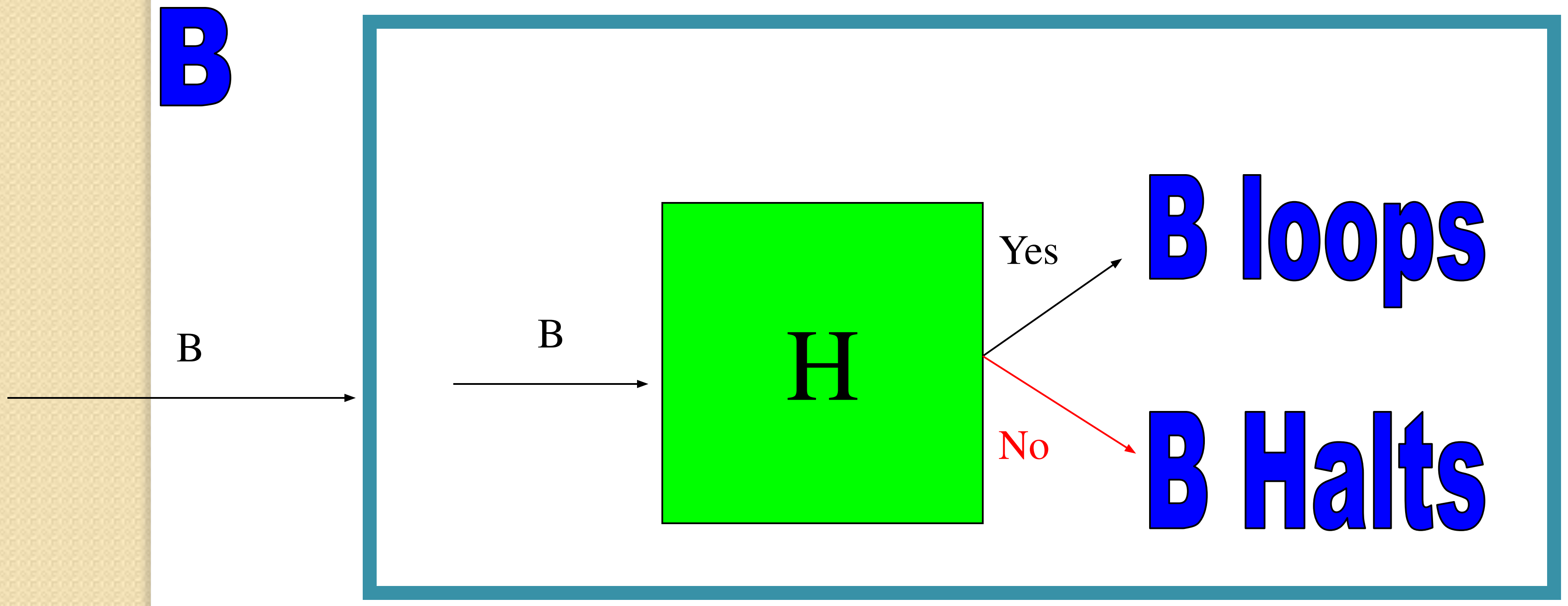
- Let’s give B a copy of itself as its input.



What if B Halts?



What if B Loops Forever?



The Paradox

- If B is a program that halts when given itself as its input,

  then, when given itself as input,
  B will go into an infinite loop.

- If B is a program that loops indefinitely when given itself as its input,

  then , when given itself as input,

  B will halt immediately.

So, Proof by Contradiction:

- We assumed that the Halting Problem COULD be computed.
- We developed another program that used the Halting Problem function.
- We found ourselves caught in a paradox (the contradiction).
- We proved that the Halting Problem is not computable.

COMPLEXITY THEORY

Studies what can and can't be computed under limited resources such as time, space, etc.

- **We measure time complexity by counting the elementary steps required for a machine to halt**

MEASURING TIME COMPLEXITY

Consider the language $A = \{ 0^k 1^k \mid k \geq 0 \}$

2k 1. Scan across the tape and reject if the string is not of the form $0^m 1^n$

$2k^2$ 2. Repeat the following if both 0s and 1s remain on the tape:
    Scan across the tape, crossing off a single 0 and a single 1

2k 3. If 0s remain after all 1s have been crossed off, or vice-versa, reject. Otherwise accept.

$O(K^2)$ is the order of A.

COMPLEXITY THEORY

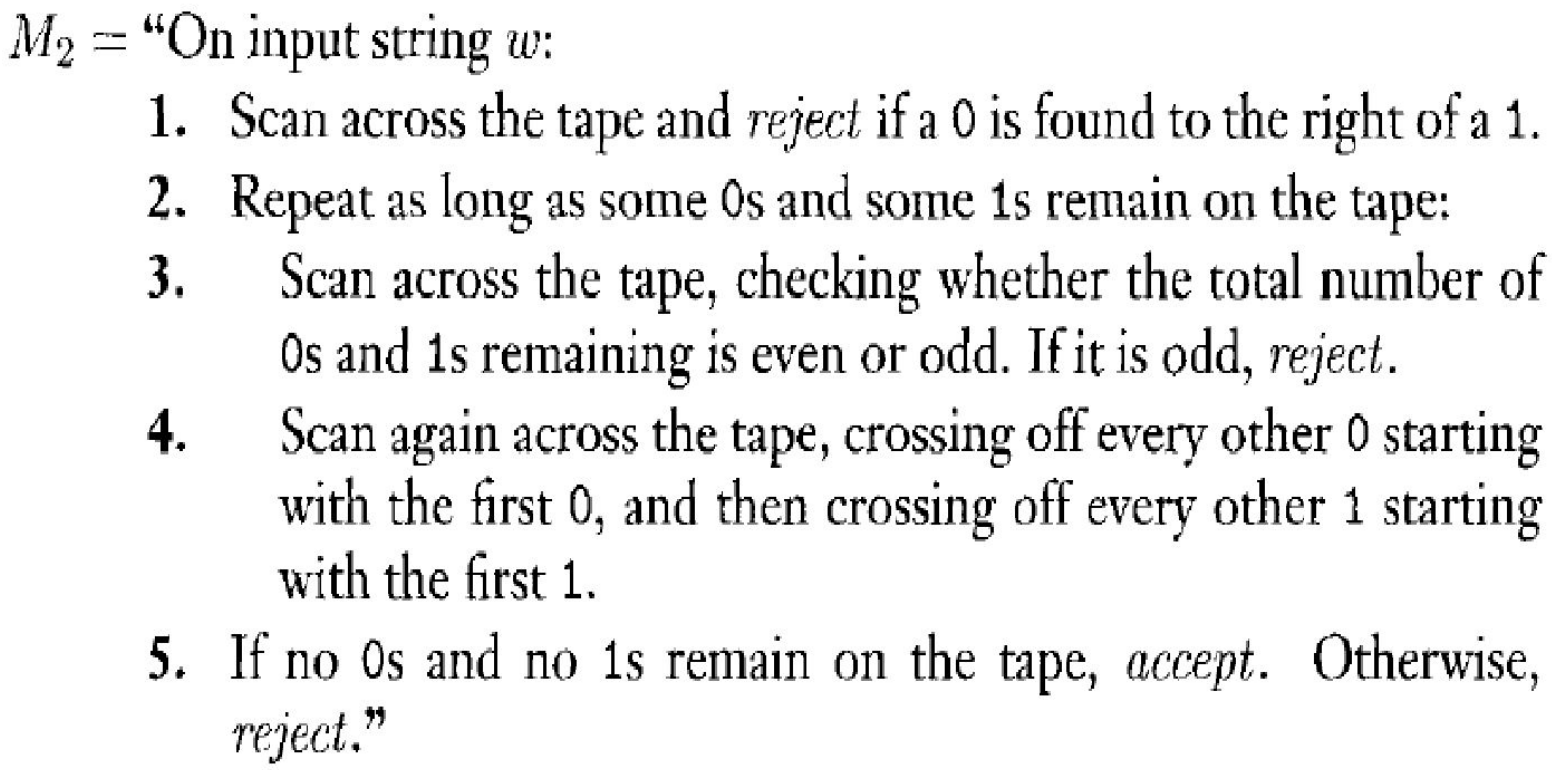
- The number of steps that an algorithm uses on a particular input may depend on several parameters.
- For instance, if the input is a graph, then the number of steps may depend on the number of nodes, the number of edges, *etc…*
- For simplicity, we compute the running time purely as a function of the length of the input string and don’t consider any other parameters.

Definition: TIME(t(n)) is the set of all languages that are decidable in O(t(n)) time by a Turing Machine.

$\{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n^2)$

We can prove that a (single-tape) TM can't decide *A* faster than O(n log n).

$$A = \{ 0^k 1^k \mid k \geq 0 \}$$



```
M_2 = "On input string w:
  1. Scan across the tape and reject if a 0 is found to the right of a 1.
  2. Repeat as long as some 0s and some 1s remain on the tape:
  3.     Scan across the tape, checking whether the total number of
         0s and 1s remaining is even or odd. If it is odd, reject.
  4.     Scan again across the tape, crossing off every other 0 starting
         with the first 0, and then crossing off every other 1 starting
         with the first 1.
  5. If no 0s and no 1s remain on the tape, accept. Otherwise,
     reject."
```

A = { 0^k1^k | k ≥ 0 } ∈ TIME(n log n)

Cross off every other 0 and every other 1. If the # of 0s and 1s left on the tape is odd, reject.

```
0000000000000111111111111 1
x0x0x0x0x0x0xx1x1x1x1x1x1x

xxx0xxx0xxx0xxxx1xxx1xxx1x

xxxxxxx0xxxxxxxxxxxx1xxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxx
```

Can $A = \{ 0^k 1^k \mid k \geq 0 \}$ be decided in time $O(n)$ with a two-tape TM?

- Scan all 0s and copy them to the second tape.
- Scan all 1s, crossing off a 0 from the second tape for each 1.

Different models of computation yield different running times for the same language!

Complexity Relationships Among Models

Does the computational model affects the time complexity of languages?????

Theorem:
Let t(n) be a function such that $t(n) \geq n$.
Then every t(n)-time multi-tape TM has an equivalent $O(t^2(n))$ single-tape TM.

Definition: P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine.
In other words,

$$P = \bigcup_{k \in N} \mathrm{TIME}(n^k)$$

The class P

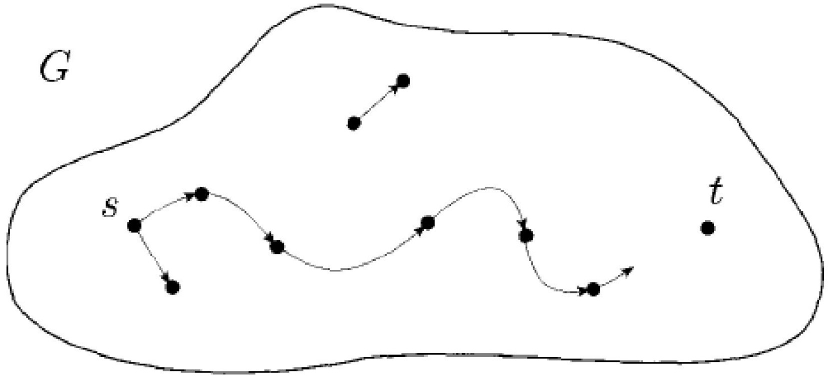
- P is the class of languages that are decidable in polynomial time on a deterministic, single tape TM
- Problems in class P
  - PATH: { <G,s,t> | G is a directed graph, find a directed path from s to t }
  - RELPRIME: {<x, y> | x and y are relatively prime}
    - Euclidean algorithm
- Every context-free language is in P □ $O(n^3)$
  - See page 262

The PATH problem O(n) ∈ Class P

$PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s to $t\}$.

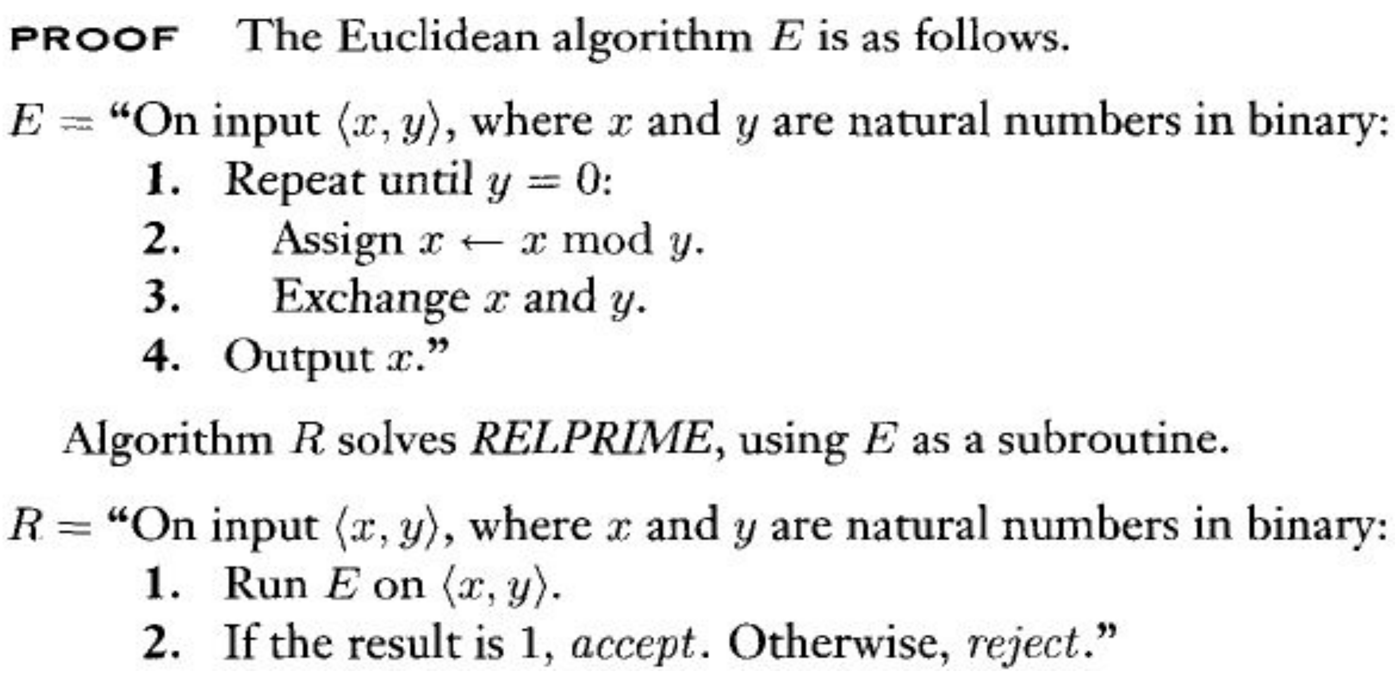
PROOF A polynomial time algorithm M for $PATH$ operates as follows.

```
M = "On input ⟨G, s, t⟩ where G is a directed graph with nodes s and t:
     1. Place a mark on node s.
     2. Repeat the following until no additional nodes are marked:
     3.     Scan all the edges of G. If an edge (a, b) is found going from
            a marked node a to an unmarked node b, mark node b.
     4. If t is marked, accept. Otherwise, reject."
```



Relative Prime "RELPRIME" problem O(n) ∈ P

- RELPRIME: {<x, y> | x and y are relatively prime}



PROOF The Euclidean algorithm E is as follows.

```
E = "On input ⟨$x, y$⟩, where $x$ and $y$ are natural numbers in binary:
    1. Repeat until $y = 0$:
    2.     Assign $x \leftarrow x \bmod y$.
    3.     Exchange $x$ and $y$.
    4. Output $x$."
```

Algorithm R solves *RELPRIME*, using E as a subroutine.

```
R = "On input ⟨$x, y$⟩, where $x$ and $y$ are natural numbers in binary:
    1. Run $E$ on ⟨$x, y$⟩.
    2. If the result is 1, accept. Otherwise, reject."
```

- Example: 10 and 21 are REPRIME, however 10 and 22 are not.
- Note that 10 and 21 are both not prime numbers.

Nondeterministic single-tape TM

Theorem:

Let *t(n)* be a function, where *t(n)* >= *n.* Then every *t(n)* time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

Definition: NTIME(t(n)) is the set of languages decided by a O(t(n))-time non-deterministic Turing machine.

TIME(t(n)) ⊆ NTIME(t(n))

Non-deterministic polynomial time

- *Deterministic Polynomial Time*: The TM takes at most O(n^c) steps to decide a string of length n.
- *Non-deterministic Polynomial Time*: The TM takes at most O(n^c) steps on each computation path to decide a string of length n

The Class P and the Class NP

- P = { L | L is decidable by a deterministic single tape Turing Machine in polynomial time }
- NP = { L | L is decidable by a non-deterministic single tape Turing Machine in polynomial time }
- They are sets of languages

The class NP

- NP is the class of languages that are decidable in polynomial time on a nondeterministic single tape TM.

  - Problems in class NP
    - HAMPATH: { <G, s, t> | G is a directed graph, with Hamilton path from s to t } (a path that passes through every vertex of a graph exactly once)
    - The problem of COMPOSITES={x| x = pq, for integers p,q>1}
    - The CLIQUE = { <G, k> | G is a graph with a clique of size k }

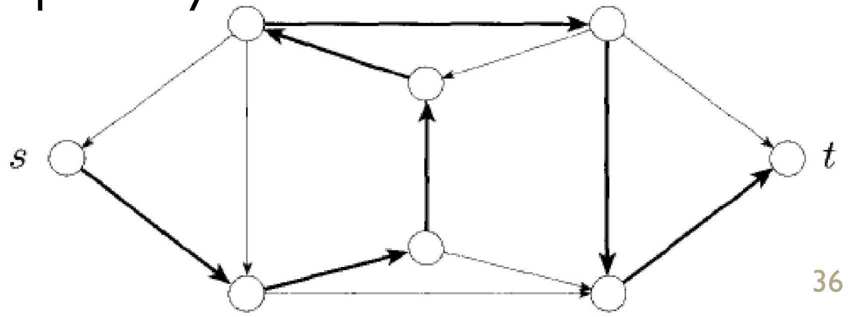
  - These problems are decidable on a deterministic single tape TM in exponential time

The class NP

- A *Hamiltonian path* in a directed graph G is a directed path that goes through each node exactly once. We consider the problem of testing whether a directed graph contains a Hamiltonian path connecting two specified nodes, as shown in the following figure.
- HAMPATH: { <G, s, t>|*G* is a directed graph with a Hamiltonian path from *s* to *t}.*
- We can easily obtain an exponential time algorithm for the HAMPATH problem by modifying the ***brute-force*** algorithm for PATH problem. a checker to verify that the potential path is Hamiltonian will be added to the PATH. No one knows whether HAMPATH is solvable in polynomial time.

-The *HAMPATH* problem does have a feature called *polynomial verifiability* that is important for understanding its complexity.

- *verifying* the existence of a Hamiltonian path may be much easier than *determining* its existence.



HAMPATH using (NTM)

- The following is a nondeterministic Turing machine (NTM) that decides the *HAMPATH problem in nondeterministic polynomial time. Recall that,* we defined the time of a nondeterministic machine to be the time by the longest computation branch.

- *N1 = "On input (G, s, t), where G is a directed graph with nodes s and t:*

1. Write a list of m numbers, *pi, . . ,*p_m*,, where m is the number of nodes in G. Each number in the list is non-deterministically selected to be between 1 and m.*
2. Check for repetitions in the list. If any are found, *reject.*
3. Check whether *s = p1 and t =* p_m*. If either fail, reject.*
4. For each i between l and *m - 1, check whether* (p_i, p_{i+1}) *is an* edge of G. If any are not, *reject. Otherwise, all tests have been* passed, so *accept."*

The class NP

- Another polynomially verifiable problem is compositeness.
- A natural number is ***composite*** if it is the product of two integers greater than 1.
  - COMPOSITES={x|x = pq, for integers p,q>1}
- We can easily verify that a number is composite, all that is needed is a divisor of that number. Recently, a polynomial time algorithm for testing whether a number is prime or composite was discovered, but it is considerably more complicated than the preceding method for verifying compositeness.
- Try to find that polynomial algorithm????

Verifier

- A verifier of a language, A, is an algorithm, V, such that
  - A = { w | V accepts <w, c> for some string c}

  where c is a certificate (additional information that provides proof), |c| is polynomial in terms of |w|.

- For the *HAMPATH* problem, a certificate for a string *(G,* s, *t)* ∈ *HAMPATH* simply is the Hamiltonian path from s to *t.*

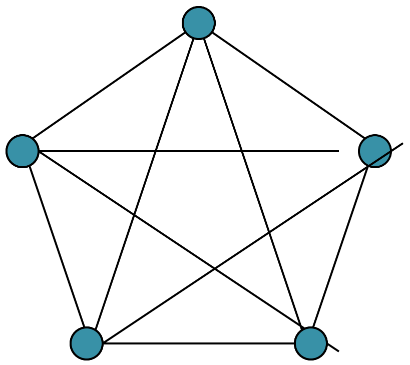
- For the *COMPOSITES* problem, a certificate for the composite number x simply is one of its divisors.

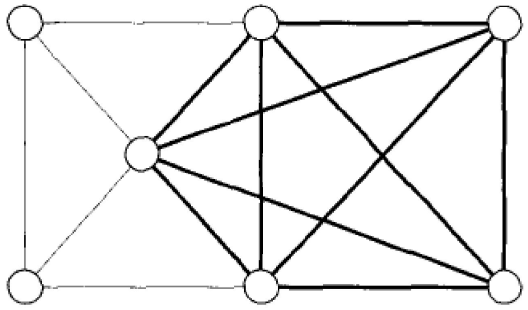
- In both cases the verifier can check in polynomial time that the input is in the language when it is given the certificate.

- ***NP is the class of languages that have polynomial time (in terms of the length of w) verifiers.***

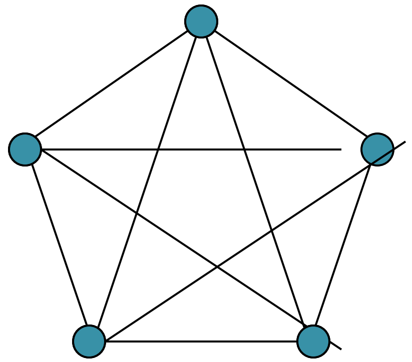
NP problem: Clique

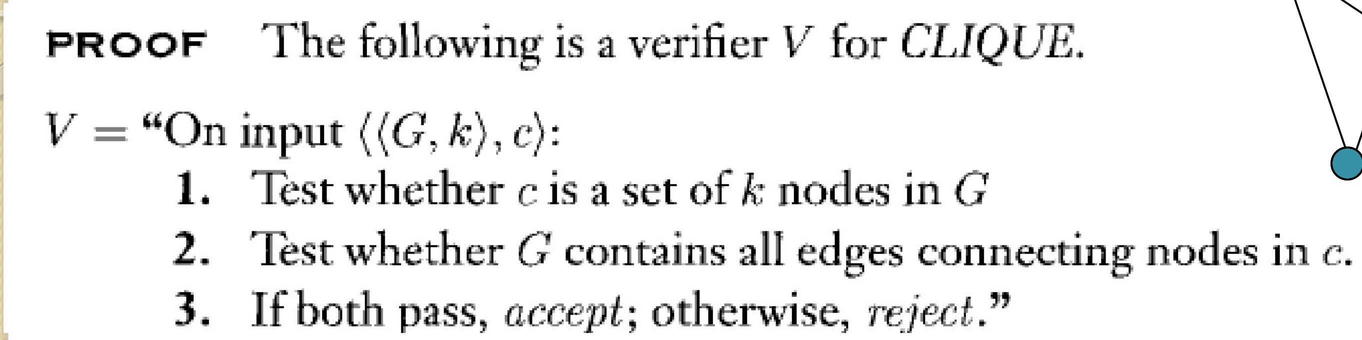
- CLIQUE = { <G,k> | G is a graph with a clique of size k }
- A clique is a subset of vertices that are all connected
- Why is CLIQUE in NP?





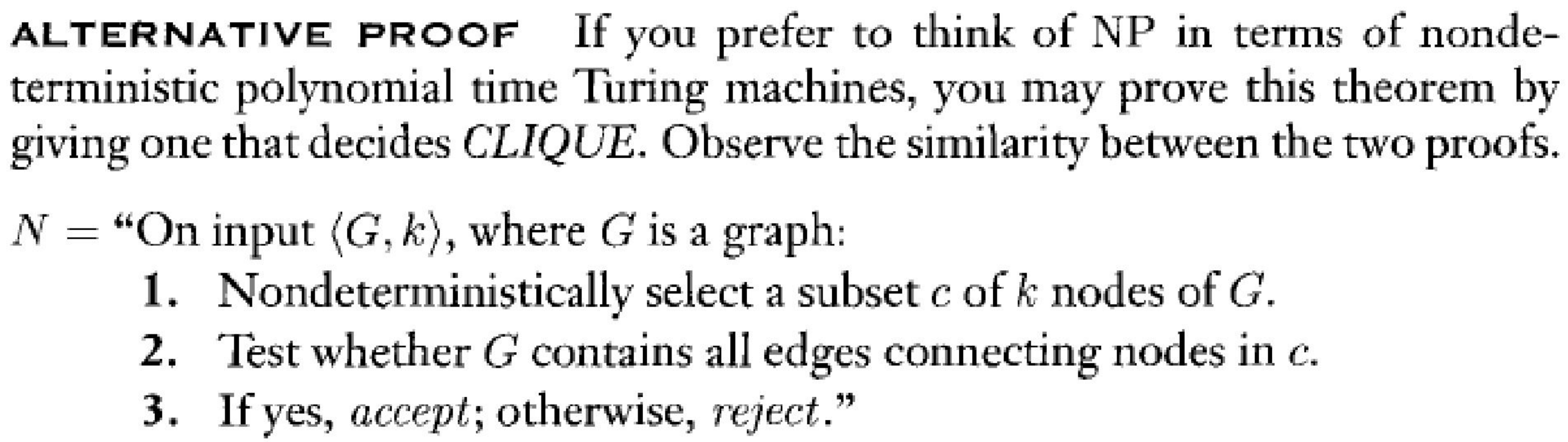
NP problem: Clique





PROOF The following is a verifier V for *CLIQUE*.

```
V = "On input ⟨⟨G, k⟩, c⟩:
    1. Test whether $c$ is a set of $k$ nodes in $G$
    2. Test whether $G$ contains all edges connecting nodes in $c$.
    3. If both pass, accept; otherwise, reject."
```



ALTERNATIVE PROOF If you prefer to think of NP in terms of nondeterministic polynomial time Turing machines, you may prove this theorem by giving one that decides *CLIQUE*. Observe the similarity between the two proofs.

```
N = "On input ⟨G, k⟩, where $G$ is a graph:
    1. Nondeterministically select a subset $c$ of $k$ nodes of $G$.
    2. Test whether $G$ contains all edges connecting nodes in $c$.
    3. If yes, accept; otherwise, reject."
```

Co-NP class

- Some problems are not polynomial verifiable, as the $\overline{\text{HAMPATH}}$, $\overline{\text{CLIQUE}}$ problems. *Till now we do not know a verifier in a polynomial time for the direct nonexistence, however we have an exponential time algorithm for making the determination of the existence.*

- **The class co-NP**: class of problems whose complement is in the class NP.

- $\overline{\text{HAMPATH}}$ and $\overline{\text{CLIQUE}} \notin$ NP class.

Non-Robustness of TM Complexity

- Computability: all variations on TMs have the same computing power
  - If there is a multi-tape TM that can decide *L*, there is a regular TM that can decide *L*.
  - If there is a nondeterministic TM that can decide *L*, there is a deterministic TM that can decide *L*.
- Complexity: variations on TM can solve problems in different times
  - Is a multi-tape TM *faster* than a regular TM?
  - Is a nondeterministic TM *faster* than a regular TM?

Multi-Tape vs. One-Tape TM

Are there problems that are in **TIME**(t(n)) for a multi-tape TM, but not in **TIME**(t(n)) for a one-tape TM?

Is Class P is sensitive to the TM computational model????

$\Downarrow$

Class P is robust .

P and NP classes

- ***Are there problems that are in TIME(t(n)) for a DTM, but not in NTIME(t(n)) for the NTM? No.***
- ***Are there problems that are in NTIME(t(n)) for a NTM, but not in TIME(t(n)) for the DTM? Yes.***

- ***Is Class NP is sensitive to the TM computational model????***

$\Downarrow$

Class NP is robust .

P vs. NP QUESTIONS

- P ⊆ NP
  - unknown if the classes are unequal
- If P = NP, then all problems in NP can be solved in polynomial time, if we are clever enough to find the right algorithm
- P = the class of languages for which membership can be *decided* quickly "in a polynomial time".
- NP = the class of languages for which membership can be *verified* quickly "in a polynomial time"..

Thanks for listining