Other Models of Computation

Models of computation:

- Turing Machines
- Recursive Functions
- Post Systems
- Rewriting Systems

Church's Thesis:

All models of computation are equivalent

Turing's Thesis:

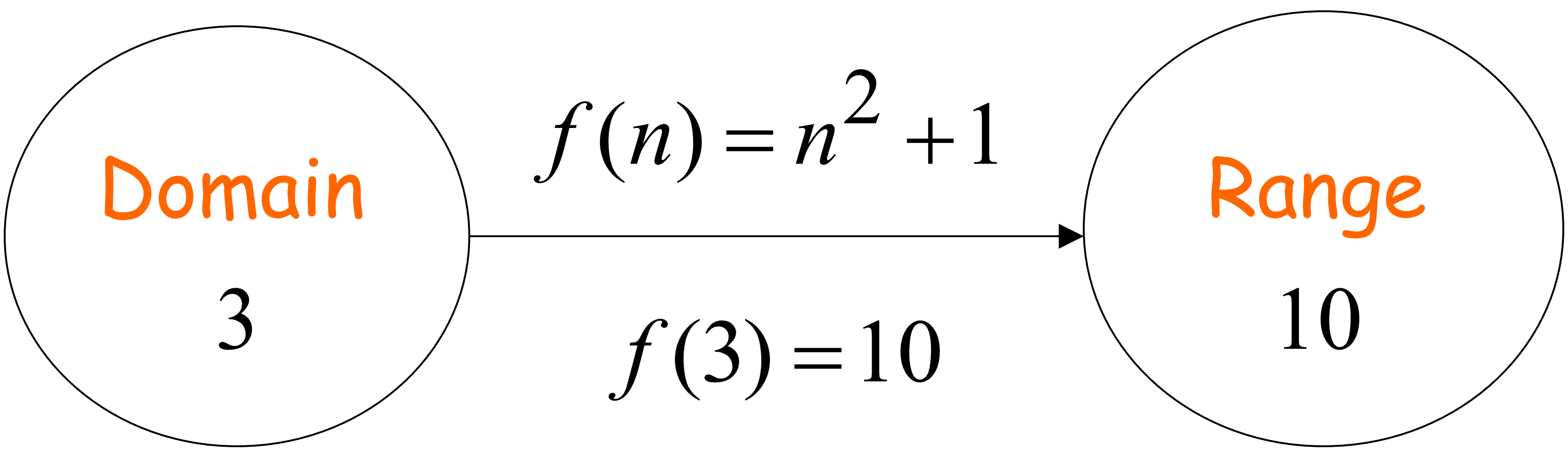
A computation is mechanical if and only if it can be performed by a Turing Machine

Church's and Turing's Thesis are similar:

Church-Turing Thesis

Recursive Functions

An example function:



We need a way to define functions

We need a set of basic functions

Basic Primitive Recursive Functions

Zero function: $z(x) = 0$

Successor function: $s(x) = x + 1$

Projection functions: $p_1(x_1, x_2) = x_1$

$p_2(x_1, x_2) = x_2$

Building complicated functions:

Composition: $f(x,y) = h(g_1(x,y), g_2(x,y))$

Primitive Recursion:

$$f(x,0) = g_1(x)$$

$$f(x,y+1) = h(g_2(x,y), f(x,y))$$

Any function built from
the basic primitive recursive functions
is called:

Primitive Recursive Function

A Primitive Recursive Function: $add(x, y)$

$$add(x,0) = x$$

(projection)

$$add(x, y+1) = s(add(x, y))$$

(successor function)

$$
\begin{aligned}
add(3,2) &= s(add(3,1)) \\
&= s(s(add(3,0))) \\
&= s(s(3)) \\
&= s(4) \\
&= 5
\end{aligned}
$$

Another Primitive Recursive Function:

$$mult(x, y)$$

$$mult(x, 0) = 0$$

$$mult(x, y+1) = add(x, mult(x, y))$$

Theorem:

The set of primitive recursive functions is countable

Proof:

Each primitive recursive function can be encoded as a string

Enumerate all strings in proper order

Check if a string is a function

Theorem

there is a function that
is not primitive recursive

Proof:

Enumerate the primitive recursive functions

$$f_1, f_2, f_3, \ldots$$

Define function $g(i) = f_i(i) + 1$

g differs from every f_i

g is not primitive recursive

END OF PROOF

A specific function that is <u>not</u> Primitive Recursive:

Ackermann's function:

$$A(0, y) = y + 1$$
$$A(x, 0) = A(x - 1, 1)$$
$$A(x, y + 1) = A(x - 1, A(x, y))$$

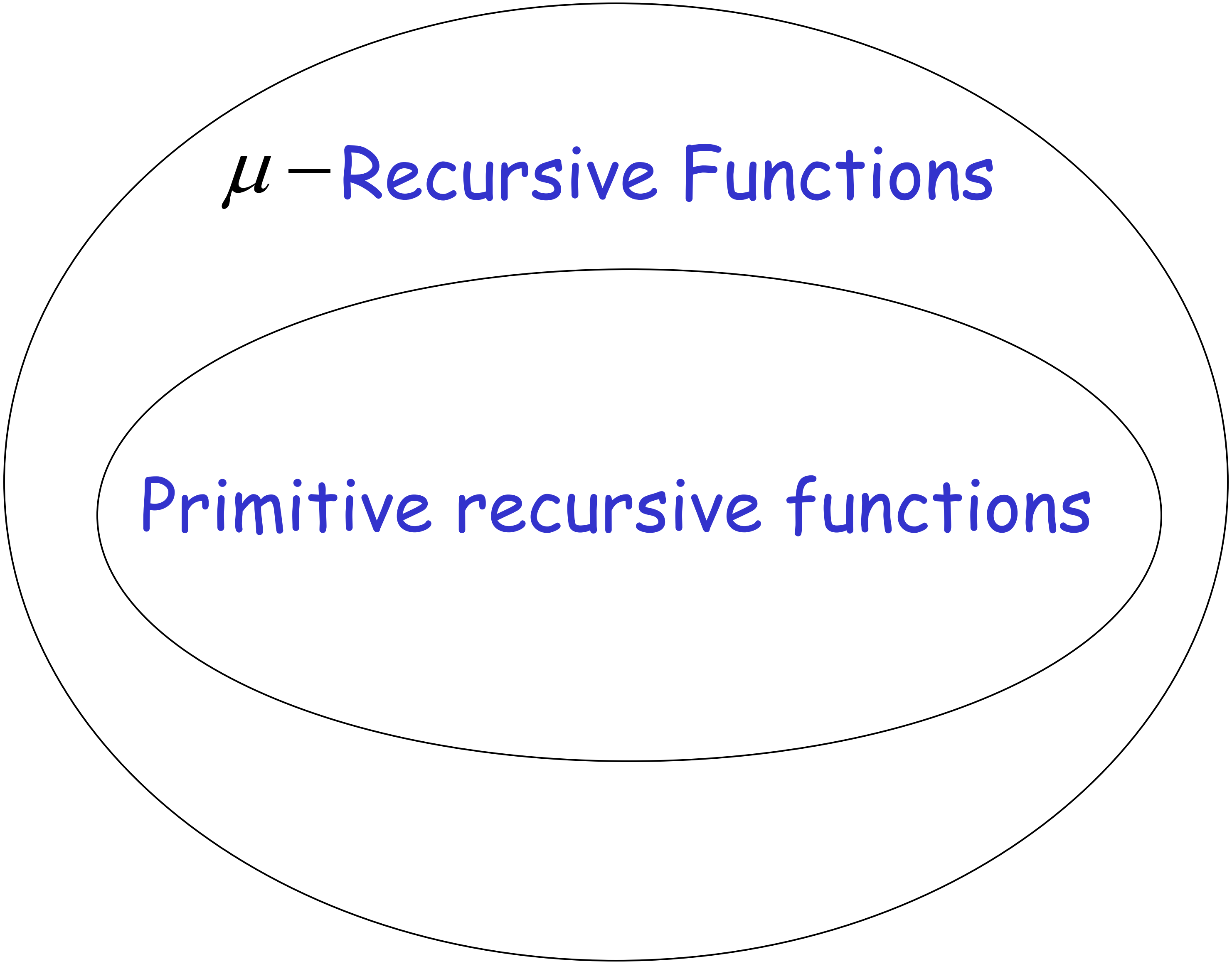
Grows very fast,
faster than any primitive recursive function

$\mu -$Recursive Functions

$$\mu y(g(x,y)) = \text{smallest } y \text{ such that } g(x,y) = 0$$

Ackerman's function is a

$\mu -$Recursive Function


$\mu-$Recursive Functions
Primitive recursive functions

Post Systems

- Have Axioms

- Have Productions

Very similar with unrestricted grammars

Example: Unary Addition

Axiom: $1+1=11$

Productions:

$$V_1 + V_2 = V_3 \quad \rightarrow \quad V_1 1 + V_2 = V_3 1$$

$$V_1 + V_2 = V_3 \quad \rightarrow \quad V_1 + V_2 1 = V_3 1$$

A production:

$$V_1 + V_2 = V_3 \quad \rightarrow \quad V_1 1 + V_2 = V_3 1$$

$$1+1=11 \;\Rightarrow\; 11+1=111 \;\Rightarrow\; 11+11=1111$$

$$V_1 + V_2 = V_3 \quad \rightarrow \quad V_1 + V_2 1 = V_3 1$$

Post systems are good for proving mathematical statements from a set of Axioms

Theorem:

A language is recursively enumerable
if and only if
a Post system generates it

Rewriting Systems

They convert one string to another

- Matrix Grammars

- Markov Algorithms

- Lindenmayer-Systems

Very similar to unrestricted grammars

Matrix Grammars

Example:

$$P_1: \quad S \to S_1 S_2$$

$$P_2: \quad S_1 \to aS_1, \; S_2 \to bS_2c$$

$$P_3: \quad S_1 \to \lambda, \; S_2 \to \lambda$$

Derivation:

$$S \Rightarrow S_1 S_2 \Rightarrow aS_1 bS_2 c \Rightarrow aaS_1 bbS_2 cc \Rightarrow aabbcc$$

A set of productions is applied simultaneously

$$P_1: \quad S \to S_1 S_2$$

$$P_2: \quad S_1 \to aS_1, \;\; S_2 \to bS_2c$$

$$P_3: \quad S_1 \to \lambda, \;\; S_2 \to \lambda$$

$$L = \{a^n b^n c^n : n \geq 0\}$$

Theorem:

A language is recursively enumerable
if and only if
a Matrix grammar generates it

Markov Algorithms

Grammars that produce λ

Example: $ab \rightarrow S$

$aSb \rightarrow S$

$S \rightarrow .\lambda$

Derivation:

$$aaabbb \Rightarrow aaSbb \Rightarrow aSb \Rightarrow S \Rightarrow \lambda$$

$$ab \to S$$

$$aSb \to S$$

$$S \to .\lambda$$

$$L = \{a^n b^n : n \geq 0\}$$

In general: $L = \{w: \ w \overset{*}{\Rightarrow} \lambda\}$

Theorem:

A language is recursively enumerable
if and only if
a Markov algorithm generates it

Lindenmayer-Systems

They are parallel rewriting systems

Example: $a \rightarrow aa$

Derivation: $a \Rightarrow aa \Rightarrow aaaa \Rightarrow aaaaaaaa$

$$L = \{a^{2^n} : n \geq 0\}$$

Lindenmayer-Systems are not general
As recursively enumerable languages

Extended Lindenmayer-Systems: $(x, a, y) \rightarrow u$

context

Theorem:

A language is recursively enumerable
if and only if an
Extended Lindenmayer-System generates it

Computational Complexity

Time Complexity: The number of steps during a computation

Space Complexity: Space used during a computation

Time Complexity

- We use a multitape Turing machine

- We count the number of steps until a string is accepted

- We use the $O(k)$ notation

Example: $L = \{a^n b^n : n \geq 0\}$

Algorithm to accept a string w :

- Use a two-tape Turing machine
- Copy the a on the second tape
- Compare the a and b

$$L = \{a^n b^n : n \geq 0\}$$

Time needed:

- Copy the a on the second tape $\qquad O(|w|)$
- Compare the a and b $\qquad O(|w|)$

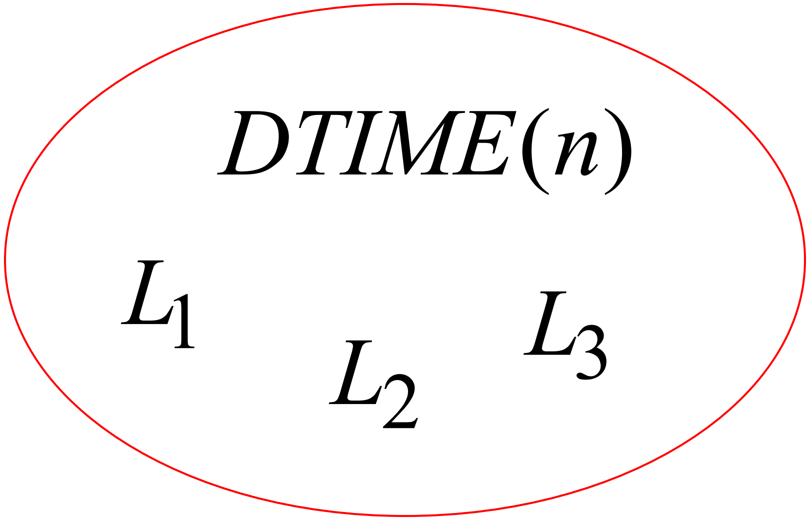
Total time: $\qquad O(|w|)$

$$L = \{a^n b^n : n \geq 0\}$$

For string of length n

time needed for acceptance: $O(n)$

Language class: $DTIME(n)$



A Deterministic Turing Machine accepts each string of length n in time $O(n)$

$$DTIME(n)$$

$$\{a^n b^n : n \geq 0\}$$

$$\{ww\}$$

In a similar way we define the class

$$DTIME(T(n))$$

for any time function: $T(n)$

Examples: $DTIME(n^2), DTIME(n^3), \ldots$

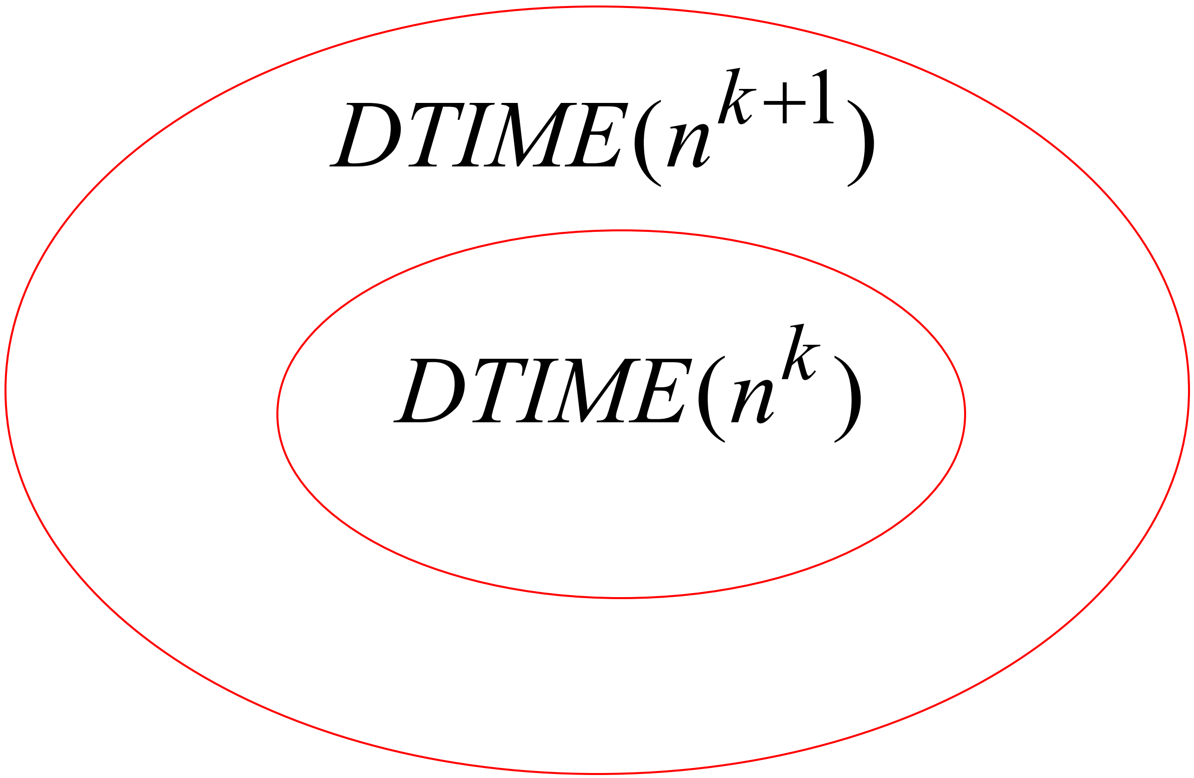
Example: The membership problem for context free languages

$$L = \{w : w \text{ is generated by grammar } G\}$$

$$L \in DTIME(n^3)$$ (CYK - algorithm)

Polynomial time

Theorem: $DTIME(n^{k+1}) \subset DTIME(n^k)$



Polynomial time algorithms: $DTIME(n^k)$

Represent tractable algorithms:

For small k we can compute the result fast

The class P

$$P = \cup DTIME(n^k) \qquad \text{for all } k$$

- Polynomial time

- All tractable problems

P

CYK-algorithm

$\{a^n b^n\}$

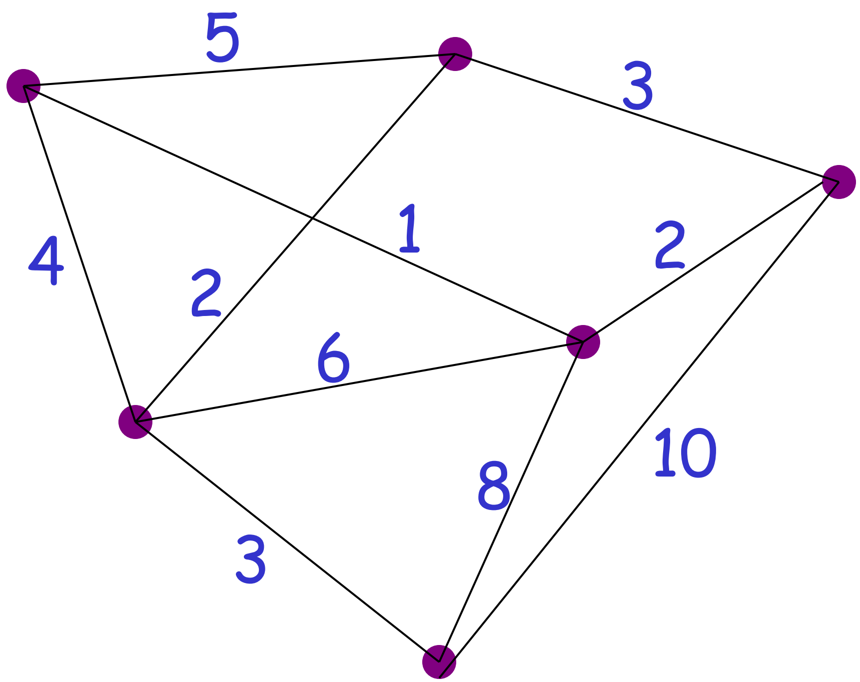
$\{ww\}$

Exponential time algorithms: $DTIME(2^n)$

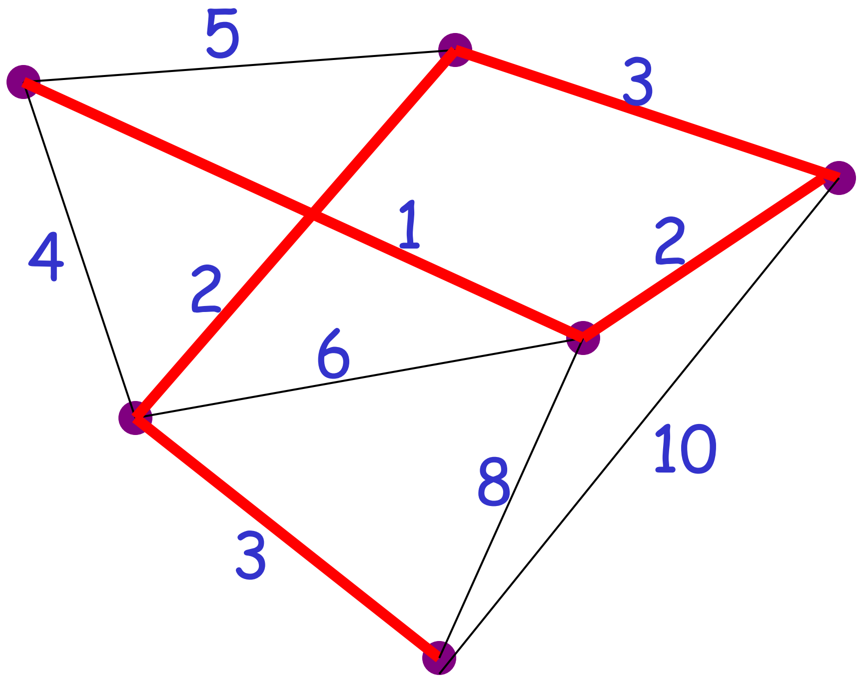
Represent intractable algorithms:

Some problem instances
may take centuries to solve

Example: the Traveling Salesperson Problem



Question: what is the shortest route that connects all cities?



Question: what is the shortest route that connects all cities?

A solution: search exhuastively all hamiltonian paths

L = {shortest hamiltonian paths}

$$L \in DTIME(n!) \approx DTIME(2^n)$$

Exponential time

Intractable problem

Example: The Satisfiability Problem

Boolean expressions in
Conjunctive Normal Form:

$$t_1 \wedge t_2 \wedge t_3 \wedge \Lambda \wedge t_k$$

$$t_i = x_1 \vee \bar{x}_2 \vee x_3 \vee \Lambda \vee \bar{x}_p$$

Variables

Question: is expression satisfiable?

Example: $(\overline{x}_1 \vee x_2) \wedge (x_1 \vee x_3)$

Satisfiable: $x_1 = 0,\ \ x_2 = 1,\ \ x_3 = 1$

$$(\overline{x}_1 \vee x_2) \wedge (x_1 \vee x_3) = 1$$

Example: $(x_1 \vee x_2) \wedge \overline{x}_1 \wedge \overline{x}_2$

Not satisfiable

$$L = \{w : \text{expression } w \text{ is satisfiable}\}$$

For n variables: $L \in DTIME(2^n)$

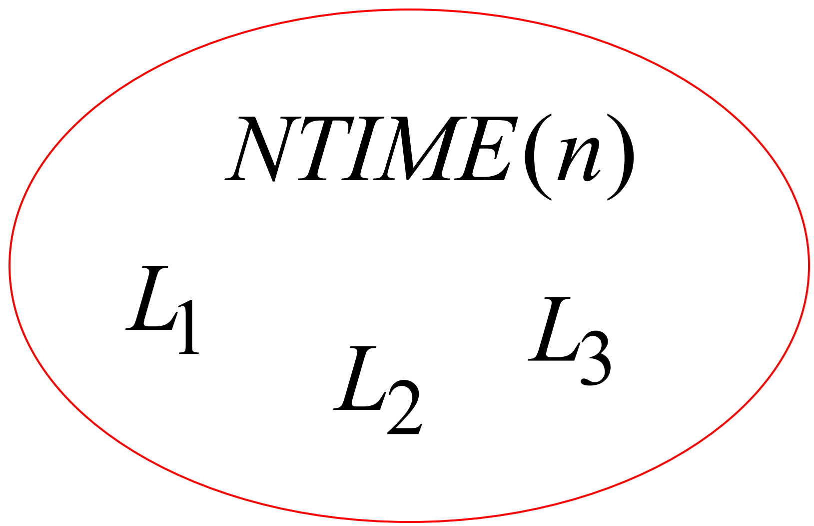
exponential

Algorithm:

search exhaustively all the possible binary values of the variables

Non-Determinism

Language class: $NTIME(n)$



A Non-Deterministic Turing Machine accepts each string of length n in time $O(n)$

Example: $L = \{ww\}$

Non-Deterministic Algorithm
to accept a string ww :

- Use a two-tape Turing machine

- Guess the middle of the string and copy w on the second tape

- Compare the two tapes

$$L = \{ww\}$$

Time needed:

- Use a two-tape Turing machine

- Guess the middle of the string and copy w on the second tape $O(|w|)$

- Compare the two tapes $O(|w|)$

Total time: $O(|w|)$

$$NTIME(n)$$

$$L = \{ww\}$$

In a similar way we define the class

$$NTIME(T(n))$$

for any time function: $T(n)$

Examples: $NTIME(n^2), NTIME(n^3), \ldots$

Non-Deterministic Polynomial time algorithms:

$$L \in NTIME(n^k)$$

The class NP

$$P = \cup NTIME(n^k) \qquad \text{for all} \quad k$$

Non-Deterministic Polynomial time

Example: The satisfiability problem

$$L = \{w : \text{expression } w \text{ is satisfiable}\}$$

Non-Deterministic algorithm:

- Guess an assignment of the variables
- Check if this is a satisfying assignment

$$L = \{w : \text{expression } w \text{ is satisfiable}\}$$

Time for n variables:

- Guess an assignment of the variables $O(n)$
- Check if this is a satisfying assignment $O(n)$

Total time: $O(n)$

$$L = \{w : \text{expression } w \text{ is satisfiable}\}$$

$$L \in NP$$

The satisfiability problem is an NP- Problem

Observation:

$$P \subseteq NP$$

Deterministic Polynomial

Non-Deterministic Polynomial

Open Problem: $P = NP\,?$

WE DO NOT KNOW THE ANSWER

Open Problem: $P = NP$?

Example: Does the Satisfiability problem have a polynomial time deterministic algorithm?

WE DO NOT KNOW THE ANSWER

NP-Completeness

A problem is NP-complete if:

- It is in NP
- Every NP problem is reduced to it

  (in polynomial time)

Observation:

If we can solve any NP-complete problem in Deterministic Polynomial Time (P time) then we know:

$$P = NP$$

Observation:

If we prove that
we cannot solve an NP-complete problem
in Deterministic Polynomial Time (P time)
then we know:

$$P \neq NP$$

Cook's Theorem:

The satisfiability problem is NP-complete

Proof:

Convert a Non-Deterministic Turing Machine
to a Boolean expression
in conjunctive normal form

Other NP-Complete Problems:

- The Traveling Salesperson Problem
- Vertex cover
- Hamiltonian Path

All the above are reduced
to the satisfiability problem

Observations:

It is unlikely that NP-complete problems are in P

The NP-complete problems have exponential time algorithms

Approximations of these problems are in P