Turing Machines and Time Complexity

Turing Machines

Turing Machines

- (Infinitely long) Tape of 1’s and 0’s

Turing Machines

- (Infinitely long) Tape of 1’s and 0’s
- Able to read and write the tape, and move the tape

Turing Machines

- (Infinitely long) Tape of 1's and 0's
- Able to read and write the tape, and move the tape
- Has initial state, instructions, halt state

Turing Machines

- (Infinitely long) Tape of 1’s and 0’s
- Able to read and write the tape, and move the tape
- Has initial state, instructions, halt state
- Instructions define for each state and value:

Turing Machines

- (Infinitely long) Tape of 1’s and 0’s
- Able to read and write the tape, and move the tape
- Has initial state, instructions, halt state
- Instructions define for each state and value:
  - What value to write?

Turing Machines

- (Infinitely long) Tape of 1's and 0's
- Able to read and write the tape, and move the tape
- Has initial state, instructions, halt state
- Instructions define for each state and value:
  - What value to write?
  - Whether the tape should be moved, and in which direction?

Turing Machines

Turing Machines

M - Turing Machine (The program)

Turing Machines

M - Turing Machine (The program)

$\sum$ - Alphabet of Turing Machine

Turing Machines

M - Turing Machine (The program)

$\sum$ - Alphabet of Turing Machine

x - String at which M begins (Input)

Turing Machines

M - Turing Machine (The program)

Σ - Alphabet of Turing Machine

x - String at which M begins (Input)

y – String at which M halts (Output)

Turing Machines

M - Turing Machine (The program)

∑ - Alphabet of Turing Machine

x - String at which M begins (Input)

y – String at which M halts (Output)

I.e. for each value of x, M computes a function such that:

Turing Machines

M - Turing Machine (The program)

∑ - Alphabet of Turing Machine

x - String at which M begins (Input)

y – String at which M halts (Output)

I.e. for each value of x, M computes a function such that:

$f:\Sigma^* \rightarrow \Sigma^*$

Turing Machines

MULTIPLICATION PROGRAM

q_0,1,q_1,1,L
q_1,$\not{b}$,q_2,*,R
q_2,$\not{b}$,q_3,$\not{b}$,L
q_2,*,q_2,*,R
q_2,1,q_2,1,R
q_2,X,q_2,X,R
q_2,A,q_2,A,R
q_3,1,q_4,$\not{b}$,L
q_3,X,q_4,X,L
q_4,1,q_4,1,L
q_4,X,q_5,X,L
q_5,*,q_8,*,R
q_5,1,q_6,A,L
q_5,A,q_5,A,L
q_6,$\not{b}$,q_7,1,R
q_6,*,q_6,*,L
q_6,1,q_6,1,L
q_7,*,q_7,*,R
q_7,1,q_7,1,R
q_7,X,q_5,X,L
q_7,A,q_7,A,R
q_8,$\not{b}$,q_3,$\not{b}$,L
q_8,1,q_8,1,R
q_8,X,q_8,X,R
q_8,A,q_8,1,R
q_9,*,q_{10},$\not{b}$,S
q_9,1,q_9,1,L
q_{10},$\not{b}$,q_{11},$\not{b}$,S
q_{10},1,q_{10},$\not{b}$,R
q_{10},X,q_{10},$\not{b}$,R
q_{10},A,q_{10},$\not{b}$,R

Turing Machines

M can be given in the form

MULTIPLICATION PROGRAM

q_0,1,q_1,1,L
q_1,$\not{b}$,q_2,*,R
q_2,$\not{b}$,q_3,$\not{b}$,L
q_2,*,q_2,*,R
q_2,1,q_2,1,R
q_2,X,q_2,X,R
q_2,A,q_2,A,R
q_3,1,q_4,$\not{b}$,L
q_3,X,q_4,X,L
q_4,1,q_4,1,L
q_4,X,q_5,X,L
q_5,*,q_8,*,R
q_5,1,q_6,A,L
q_5,A,q_5,A,L
q_6,$\not{b}$,q_7,1,R
q_6,*,q_6,*,L
q_6,1,q_6,1,L
q_7,*,q_7,*,R
q_7,1,q_7,1,R
q_7,X,q_5,X,L
q_7,A,q_7,A,R
q_8,$\not{b}$,q_3,$\not{b}$,L
q_8,1,q_8,1,R
q_8,X,q_8,X,R
q_8,A,q_8,1,R
q_9,*,q_{10},$\not{b}$,S
q_9,1,q_9,1,L
q_{10},$\not{b}$,q_{11},$\not{b}$,S
q_{10},1,q_{10},$\not{b}$,R
q_{10},X,q_{10},$\not{b}$,R
q_{10},A,q_{10},$\not{b}$,R

Turing Machines

M can be given in the form of a set of: (q, s, q’, s’, d)

MULTIPLICATION PROGRAM

q_0,1,q_1,1,L
q_1,ƀ,q_2,*,R
q_2,ƀ,q_3,ƀ,L
q_2,*,q_2,*,R
q_2,1,q_2,1,R
q_2,X,q_2,X,R
q_2,A,q_2,A,R
q_3,1,q_4,ƀ,L
q_3,X,q_4,X,L
q_4,1,q_4,1,L
q_4,X,q_5,X,L
q_5,*,q_8,*,R
q_5,1,q_6,A,L
q_5,A,q_5,A,L
q_6,ƀ,q_7,1,R
q_6,*,q_6,*,L
q_6,1,q_6,1,L
q_7,*,q_7,*,R
q_7,1,q_7,1,R
q_7,X,q_5,X,L
q_7,A,q_7,A,R
q_8,ƀ,q_3,ƀ,L
q_8,1,q_8,1,R
q_8,X,q_8,X,R
q_8,A,q_8,1,R
q_9,*,q_{10},ƀ,S
q_9,1,q_9,1,L
q_{10},ƀ,q_{11},ƀ,S
q_{10},1,q_{10},ƀ,R
q_{10},X,q_{10},ƀ,R
q_{10},A,q_{10},ƀ,R

Turing Machines

M can be given in the form of a set of: (q, s, q’, s’, d)

Where:

MULTIPLICATION PROGRAM

q_0,1,q_1,1,L
q_1,ƀ,q_2,*,R
q_2,ƀ,q_3,ƀ,L
q_2,*,q_2,*,R
q_2,1,q_2,1,R
q_2,X,q_2,X,R
q_2,A,q_2,A,R
q_3,1,q_4,ƀ,L
q_3,X,q_4,X,L
q_4,1,q_4,1,L
q_4,X,q_5,X,L
q_5,*,q_8,*,R
q_5,1,q_6,A,L
q_5,A,q_5,A,L
q_6,ƀ,q_7,1,R
q_6,*,q_6,*,L
q_6,1,q_6,1,L
q_7,*,q_7,*,R
q_7,1,q_7,1,R
q_7,X,q_5,X,L
q_7,A,q_7,A,R
q_8,ƀ,q_3,ƀ,L
q_8,1,q_8,1,R
q_8,X,q_8,X,R
q_8,A,q_8,1,R
q_9,*,q_{10},ƀ,S
q_9,1,q_9,1,L
q_{10},ƀ,q_{11},ƀ,S
q_{10},1,q_{10},ƀ,R
q_{10},X,q_{10},ƀ,R
q_{10},A,q_{10},ƀ,R

Turing Machines

M can be given in the form of a set of: (q, s, q’, s’, d)

Where:

q - Current State

MULTIPLICATION PROGRAM

$q_0,1,q_1,1,L$
$q_1,\not{b},q_2,*,R$
$q_2,\not{b},q_3,\not{b},L$
$q_2,*,q_2,*,R$
$q_2,1,q_2,1,R$
q_2,X,q_2,X,R
q_2,A,q_2,A,R
$q_3,1,q_4,\not{b},L$
q_3,X,q_4,X,L
$q_4,1,q_4,1,L$
q_4,X,q_5,X,L
$q_5,*,q_8,*,R$
$q_5,1,q_6,A,L$
q_5,A,q_5,A,L
$q_6,\not{b},q_7,1,R$
$q_6,*,q_6,*,L$
$q_6,1,q_6,1,L$
$q_7,*,q_7,*,R$
$q_7,1,q_7,1,R$
q_7,X,q_5,X,L
q_7,A,q_7,A,R
$q_8,\not{b},q_3,\not{b},L$
$q_8,1,q_8,1,R$
q_8,X,q_8,X,R
$q_8,A,q_8,1,R$
$q_9,*,q_{10},\not{b},S$
$q_9,1,q_9,1,L$
$q_{10},\not{b},q_{11},\not{b},S$
$q_{10},1,q_{10},\not{b},R$
$q_{10},X,q_{10},\not{b},R$
$q_{10},A,q_{10},\not{b},R$

Turing Machines

M can be given in the form

of a set of: (q, s, q’, s’, d)

Where:

q - Current State

s - Symbol currently under read/ write head

MULTIPLICATION PROGRAM

$q_0, 1, q_1, 1, L$	$q_6, 1, q_6, 1, L$
$q_1, \not{b}, q_2, *, R$	$q_7, *, q_7, *, R$
$q_2, \not{b}, q_3, \not{b}, L$	$q_7, 1, q_7, 1, R$
$q_2, *, q_2, *, R$	q_7, X, q_5, X, L
$q_2, 1, q_2, 1, R$	q_7, A, q_7, A, R
q_2, X, q_2, X, R	$q_8, \not{b}, q_3, \not{b}, L$
q_2, A, q_2, A, R	$q_8, 1, q_8, 1, R$
$q_3, 1, q_4, \not{b}, L$	q_8, X, q_8, X, R
q_3, X, q_4, X, L	$q_8, A, q_8, 1, R$
$q_4, 1, q_4, 1, L$	$q_9, *, q_{10}, \not{b}, S$
q_4, X, q_5, X, L	$q_9, 1, q_9, 1, L$
$q_5, *, q_8, *, R$	$q_{10}, \not{b}, q_{11}, \not{b}, S$
$q_5, 1, q_6, A, L$	$q_{10}, 1, q_{10}, \not{b}, R$
q_5, A, q_5, A, L	$q_{10}, X, q_{10}, \not{b}, R$
$q_6, \not{b}, q_7, 1, R$	$q_{10}, A, q_{10}, \not{b}, R$
$q_6, *, q_6, *, L$	

Turing Machines

M can be given in the form of a set of: (q, s, q', s', d)

Where:

q - Current State

s - Symbol currently under read/ write head

q' - The state in which M is to enter next

MULTIPLICATION PROGRAM

$q_0,1,q_1,1,L$
$q_1,\not b,q_2,*,R$
$q_2,\not b,q_3,\not b,L$
$q_2,*,q_2,*,R$
$q_2,1,q_2,1,R$
q_2,X,q_2,X,R
q_2,A,q_2,A,R
$q_3,1,q_4,\not b,L$
q_3,X,q_4,X,L
$q_4,1,q_4,1,L$
q_4,X,q_5,X,L
$q_5,*,q_8,*,R$
$q_5,1,q_6,A,L$
q_5,A,q_5,A,L
$q_6,\not b,q_7,1,R$
$q_6,*,q_6,*,L$
$q_6,1,q_6,1,L$
$q_7,*,q_7,*,R$
$q_7,1,q_7,1,R$
q_7,X,q_5,X,L
q_7,A,q_7,A,R
$q_8,\not b,q_3,\not b,L$
$q_8,1,q_8,1,R$
q_8,X,q_8,X,R
$q_8,A,q_8,1,R$
$q_9,*,q_{10},\not b,S$
$q_9,1,q_9,1,L$
$q_{10},\not b,q_{11},\not b,S$
$q_{10},1,q_{10},\not b,R$
$q_{10},X,q_{10},\not b,R$
$q_{10},A,q_{10},\not b,R$

Turing Machines

M can be given in the form of a set of: (q, s, q’, s’, d)

Where:

q - Current State

s - Symbol currently under read/ write head

q’ - The state in which M is to enter next

s’ - The symbol to be written in place of s

MULTIPLICATION PROGRAM

$q_0, 1, q_1, 1, L$
$q_1, \not{b}, q_2, *, R$
$q_2, \not{b}, q_3, \not{b}, L$
$q_2, *, q_2, *, R$
$q_2, 1, q_2, 1, R$
q_2, X, q_2, X, R
q_2, A, q_2, A, R
$q_3, 1, q_4, \not{b}, L$
q_3, X, q_4, X, L
$q_4, 1, q_4, 1, L$
q_4, X, q_5, X, L
$q_5, *, q_8, *, R$
$q_5, 1, q_6, A, L$
q_5, A, q_5, A, L
$q_6, \not{b}, q_7, 1, R$
$q_6, *, q_6, *, L$
$q_6, 1, q_6, 1, L$
$q_7, *, q_7, *, R$
$q_7, 1, q_7, 1, R$
q_7, X, q_5, X, L
q_7, A, q_7, A, R
$q_8, \not{b}, q_3, \not{b}, L$
$q_8, 1, q_8, 1, R$
q_8, X, q_8, X, R
$q_8, A, q_8, 1, R$
$q_9, *, q_{10}, \not{b}, S$
$q_9, 1, q_9, 1, L$
$q_{10}, \not{b}, q_{11}, \not{b}, S$
$q_{10}, 1, q_{10}, \not{b}, R$
$q_{10}, X, q_{10}, \not{b}, R$
$q_{10}, A, q_{10}, \not{b}, R$

Turing Machines

M can be given in the form of a set of: (q, s, q’, s’, d)

Where:

q - Current State

s - Symbol currently under read/ write head

q’ - The state in which M is to enter next

s’ - The symbol to be written in place of s

d - In which direction the read/ write head is to move

MULTIPLICATION PROGRAM

$q_0,1,q_1,1,L$
$q_1,\not b,q_2,*,R$
$q_2,\not b,q_3,\not b,L$
$q_2,*,q_2,*,R$
$q_2,1,q_2,1,R$
q_2,X,q_2,X,R
q_2,A,q_2,A,R
$q_3,1,q_4,\not b,L$
q_3,X,q_4,X,L
$q_4,1,q_4,1,L$
q_4,X,q_5,X,L
$q_5,*,q_8,*,R$
$q_5,1,q_6,A,L$
q_5,A,q_5,A,L
$q_6,\not b,q_7,1,R$
$q_6,*,q_6,*,L$
$q_6,1,q_6,1,L$
$q_7,*,q_7,*,R$
$q_7,1,q_7,1,R$
q_7,X,q_5,X,L
q_7,A,q_7,A,R
$q_8,\not b,q_3,\not b,L$
$q_8,1,q_8,1,R$
q_8,X,q_8,X,R
$q_8,A,q_8,1,R$
$q_9,*,q_{10},\not b,S$
$q_9,1,q_9,1,L$
$q_{10},\not b,q_{11},\not b,S$
$q_{10},1,q_{10},\not b,R$
$q_{10},X,q_{10},\not b,R$
$q_{10},A,q_{10},\not b,R$

Turing Machines

M can be given in the form of a set of: (q, s, q’, s’, d)

MULTIPLICATION PROGRAM

$q_0,1,q_1,1,L$	$q_6,1,q_6,1,L$
$q_1,\not{b},q_2,*,R$	$q_7,*,q_7,*,R$
$q_2,\not{b},q_3,\not{b},L$	$q_7,1,q_7,1,R$
$q_2,*,q_2,*,R$	q_7,X,q_5,X,L
$q_2,1,q_2,1,R$	q_7,A,q_7,A,R
q_2,X,q_2,X,R	$q_8,\not{b},q_3,\not{b},L$
q_2,A,q_2,A,R	$q_8,1,q_8,1,R$
$q_3,1,q_4,\not{b},L$	q_8,X,q_8,X,R
q_3,X,q_4,X,L	$q_8,A,q_8,1,R$
$q_4,1,q_4,1,L$	$q_9,*,q_{10},\not{b},S$
q_4,X,q_5,X,L	$q_9,1,q_9,1,L$
$q_5,*,q_8,*,R$	$q_{10},\not{b},q_{11},\not{b},S$
$q_5,1,q_6,A,L$	$q_{10},1,q_{10},\not{b},R$
q_5,A,q_5,A,L	$q_{10},X,q_{10},\not{b},R$
$q_6,\not{b},q_7,1,R$	$q_{10},A,q_{10},\not{b},R$
$q_6,*,q_6,*,L$	

Where:

q - Current State

s - Symbol currently under read/ write head

q’ - The state in which M is to enter next

s’ - The symbol to be written in place of s

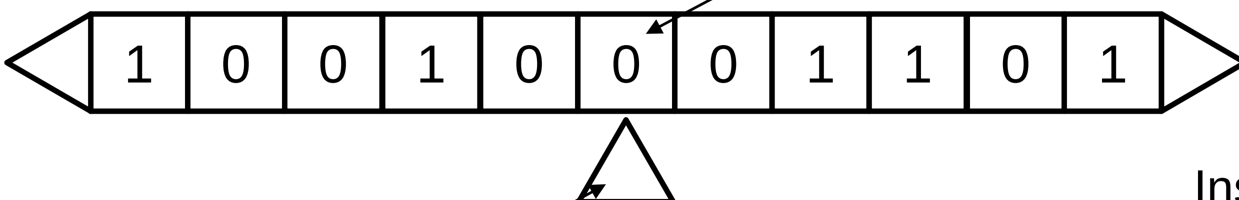
d - In which direction the read/ write head is to move

(such that D = {left, right, stop})

Turing Machines

A Turing Machine Conceptualization using a State Diagram:

Tape with 0’s and 1’s



Read/ Write Function

Instructions

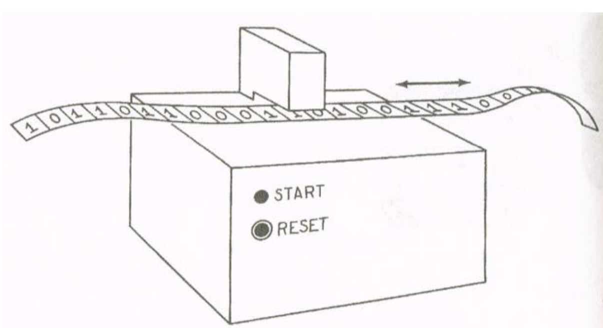


Figure 31.1 A Turing machine conceptualization

Initial State

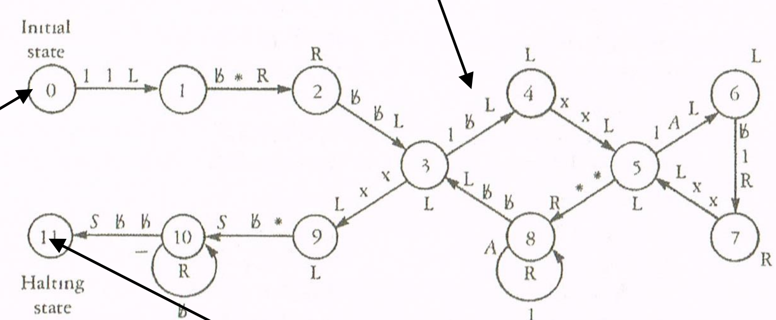


Figure 31.3 The unary multiplication program

Halt State

Turing Machines

A Turing Machine Conceptualization using a State Diagram:

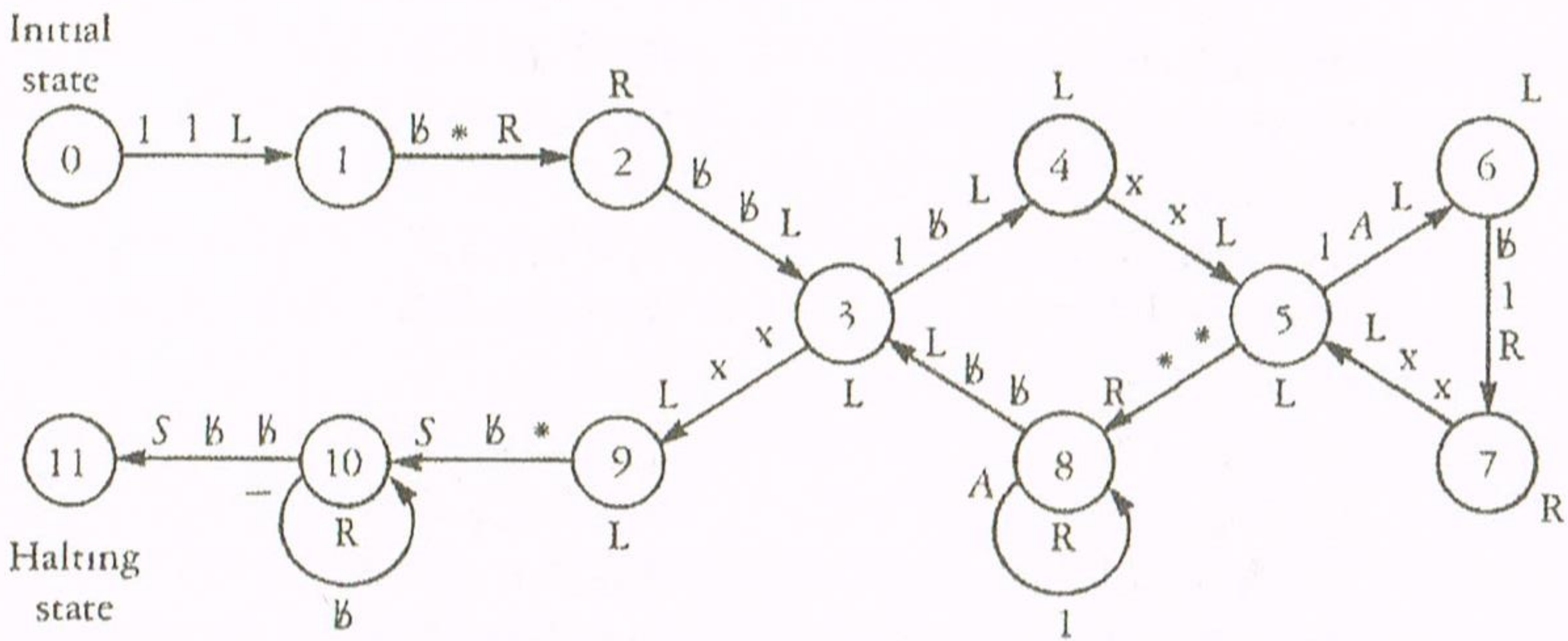


Figure 31.3 The unary multiplication program

Turing Machines

Turing Machines

- Are ‘Turing Complete’

Turing Machines

- Are ‘Turing Complete’
- I.e. they can perform every possible computation (given enough time and space)

Non-Determinism

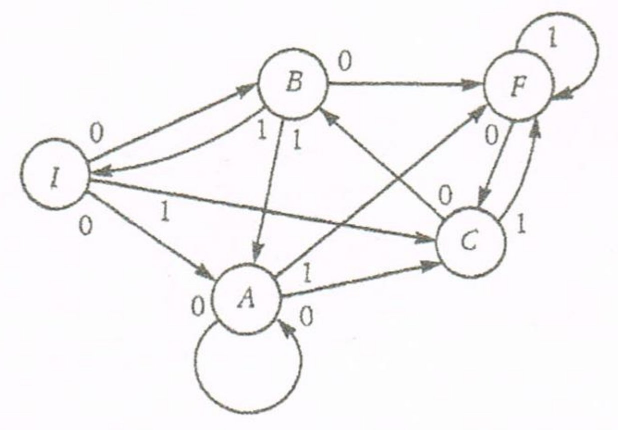


Figure 26.1 A nondeterministic automaton

Non-Determinism

(Automata that guess correctly)

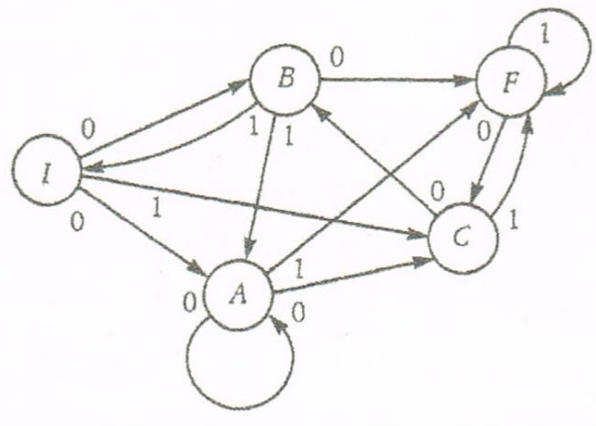


Figure 26.1 A nondeterministic automaton

Non-Determinism

(Automata that guess correctly)

- Don't exist – but are useful in theory (can simplify problems)

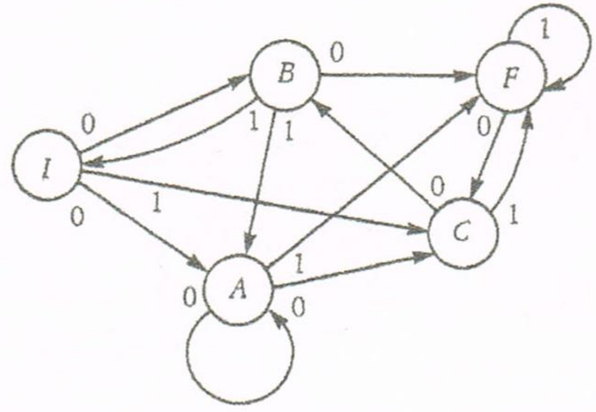


Figure 26.1 A nondeterministic automaton

Non-Determinism

(Automata that guess correctly)

- Don’t exist – but are useful in theory (can simplify problems)
- For some states and inputs, there may be multiple options

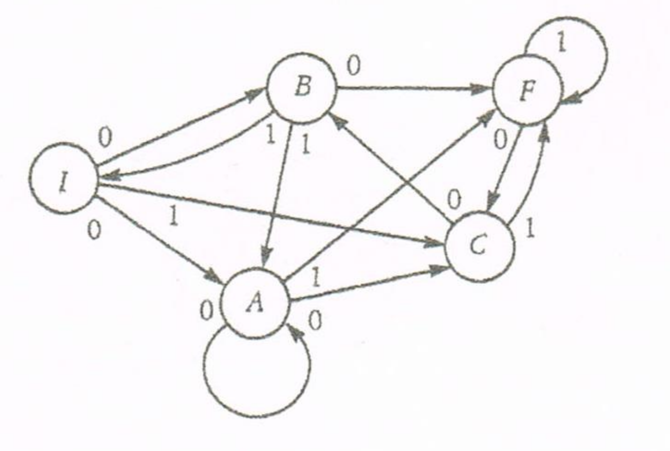


Figure 26.1 A nondeterministic automaton

Non-Determinism

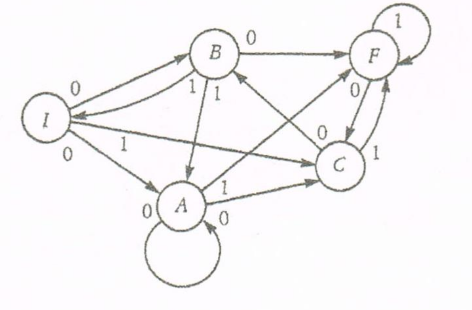


Figure 26.1 A nondeterministic automaton

Non-Determinism

- d – Transition Function

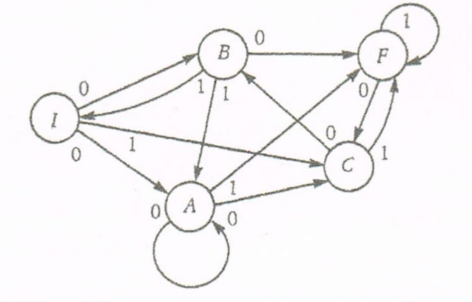


Figure 26.1 A nondeterministic automaton

Non-Determinism

- d – Transition Function
- ∑ - Finite Alphabet

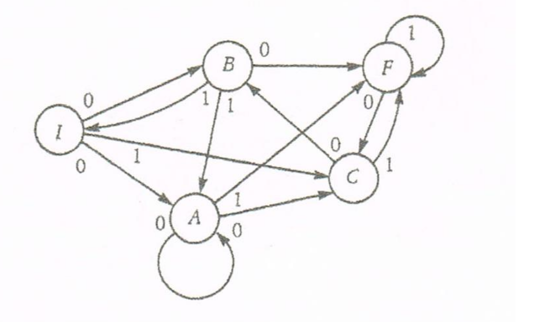


Figure 26.1 A nondeterministic automaton

Non-Determinism

- d – Transition Function
- ∑ - Finite Alphabet
- Finite set of Q states

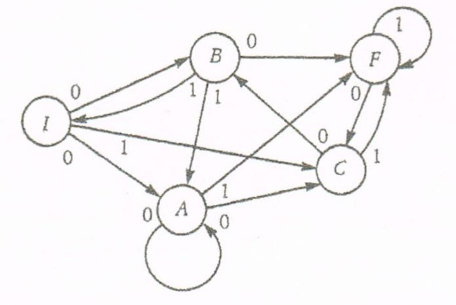


Figure 26.1 A nondeterministic automaton

Non-Determinism

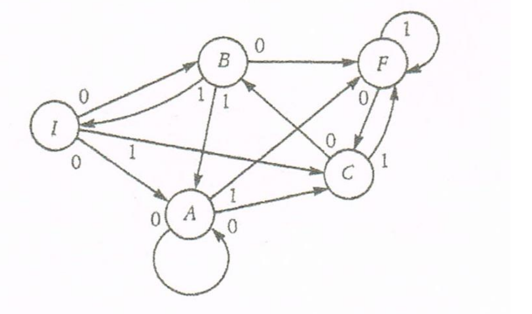


Figure 26.1 A nondeterministic automaton

- d – Transition Function
- ∑ - Finite Alphabet
- Finite set of Q states
- P(Q) - Power set of Q (Set of all subsets of Q)

Non-Determinism

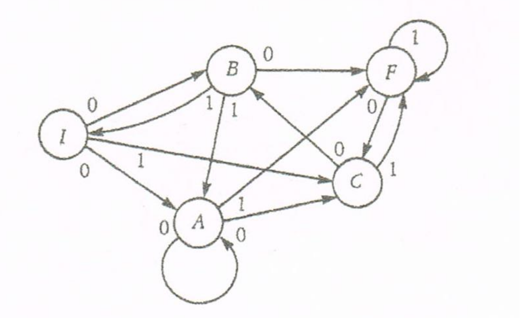


Figure 26.1 A nondeterministic automaton

- d – Transition Function
- ∑ - Finite Alphabet
- Finite set of Q states
- P(Q) - Power set of Q (Set of all subsets of Q)
- Such that d: Q X ∑ → P(Q)

Non-Determinism

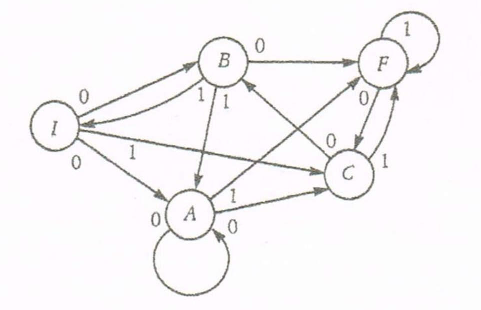


Figure 26.1 A nondeterministic automaton

- d – Transition Function
- ∑ - Finite Alphabet
- Finite set of Q states
- P(Q) - Power set of Q (Set of all subsets of Q)
- Such that d: Q X ∑ → P(Q)
- E.g. d maps the input (I,0) → state set {A, B}

What is an NP Problem?

What is an NP Problem?

- Can be solved by a **non-deterministic** Turing Machine

What is an NP Problem?

- Can be solved by a **non-deterministic** Turing Machine
- In **polynomial time**

What is an NP Problem?

- Can be solved by a **non-deterministic** Turing Machine
- In **polynomial time**

I.e.

What is an NP Problem?

- Can be solved by a **non-deterministic** Turing Machine
- In **polynomial time**

I.e.

- Where there is a solution

What is an NP Problem?

- Can be solved by a **non-deterministic** Turing Machine
- In **polynomial time**

I.e.

- Where there is a solution
- That solution must be *verifiable* in polynomial time (by a deterministic Turing Machine)

Big-O Notation

Big-O Notation

- Each program has a maximum runtime

Big-O Notation

- Each program has a maximum runtime
- The worst-case time complexity of an algorithm on an input of size n – maximum time taken

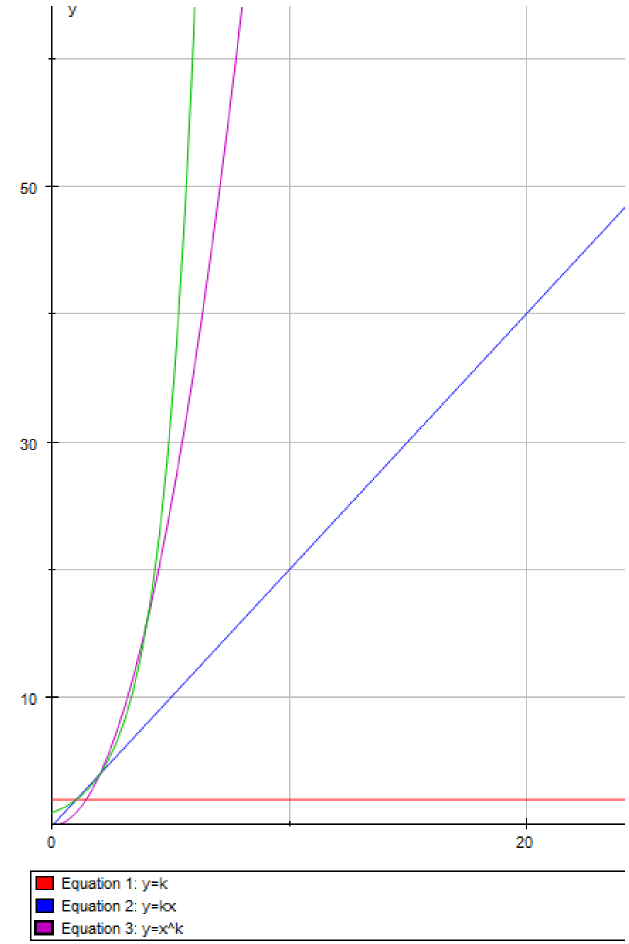
Big-O Notation

- Each program has a maximum runtime
- The worst-case time complexity of an algorithm on an input of size n – maximum time taken
- Big-O notation ignores exact runtimes for simplification

Big-O Notation

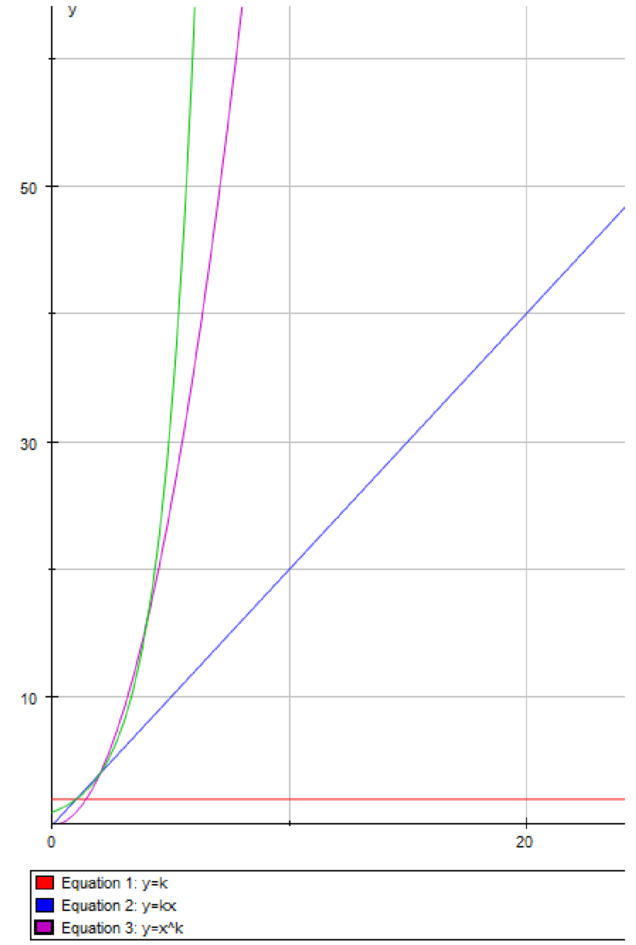
- Each program has a maximum runtime
- The worst-case time complexity of an algorithm on an input of size n – maximum time taken
- Big-O notation ignores exact runtimes for simplification
- E.g. (7.25n^2 – 1.14n + 2.83)/2 → O(n^2) – Regardless of the values of constants – runtime will always be quadratic

Big-O Notation



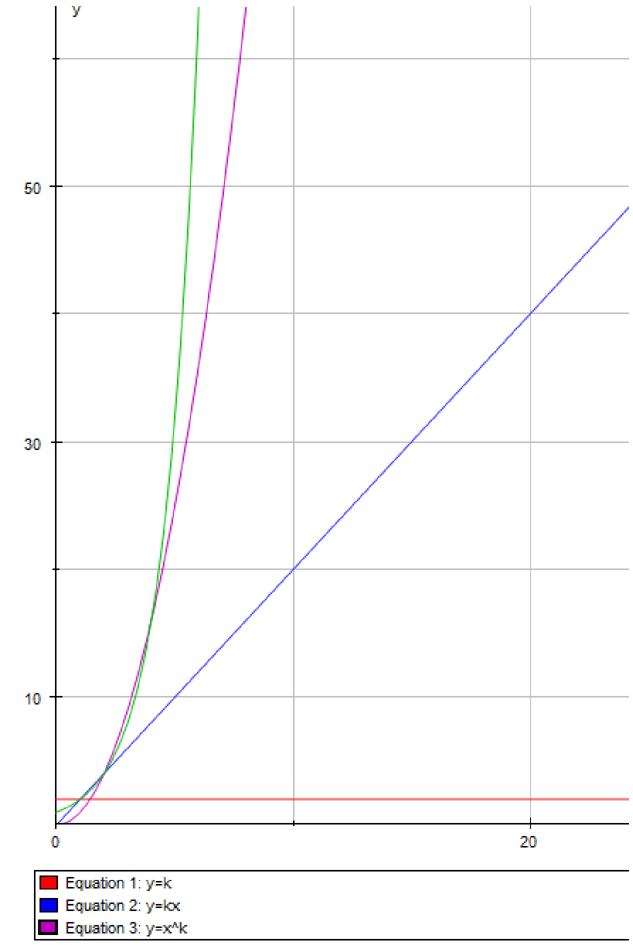
Big-O Notation

- O(k) Constant



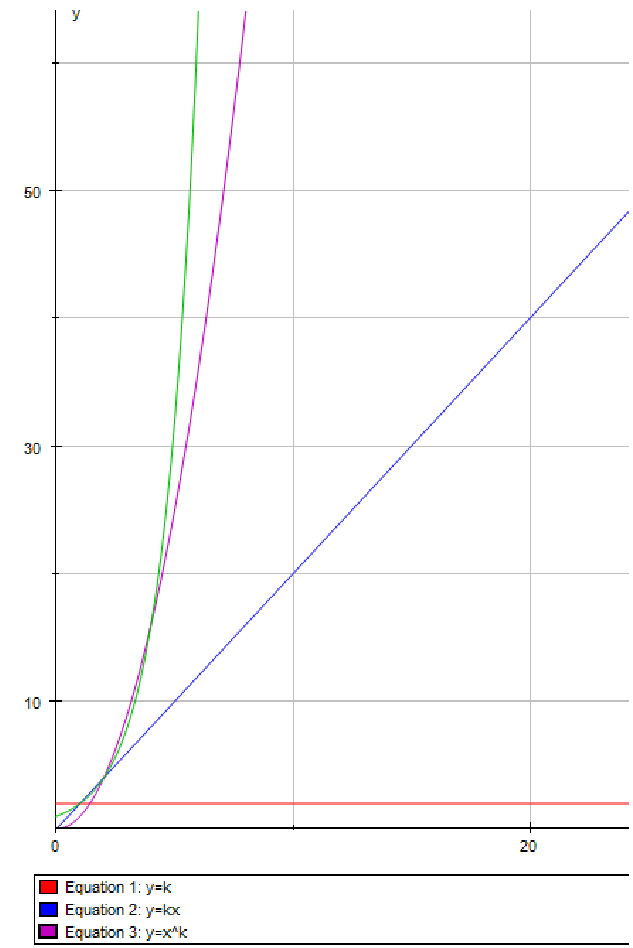
Big-O Notation

- O(k) Constant
- O(kn) Linear



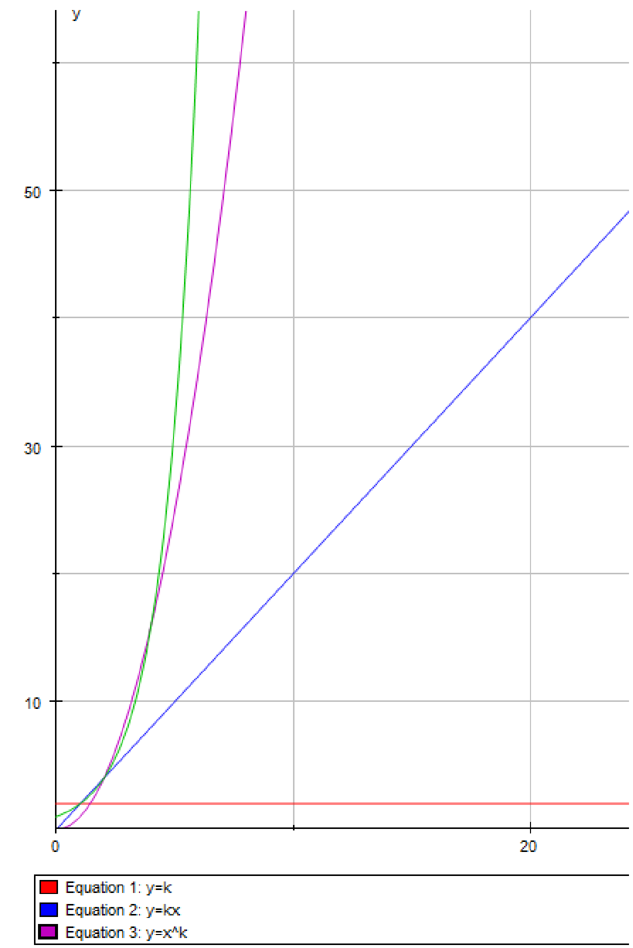
Big-O Notation

- O(k) Constant
- O(kn) Linear
- O(n^k) Polynomial



Big-O Notation

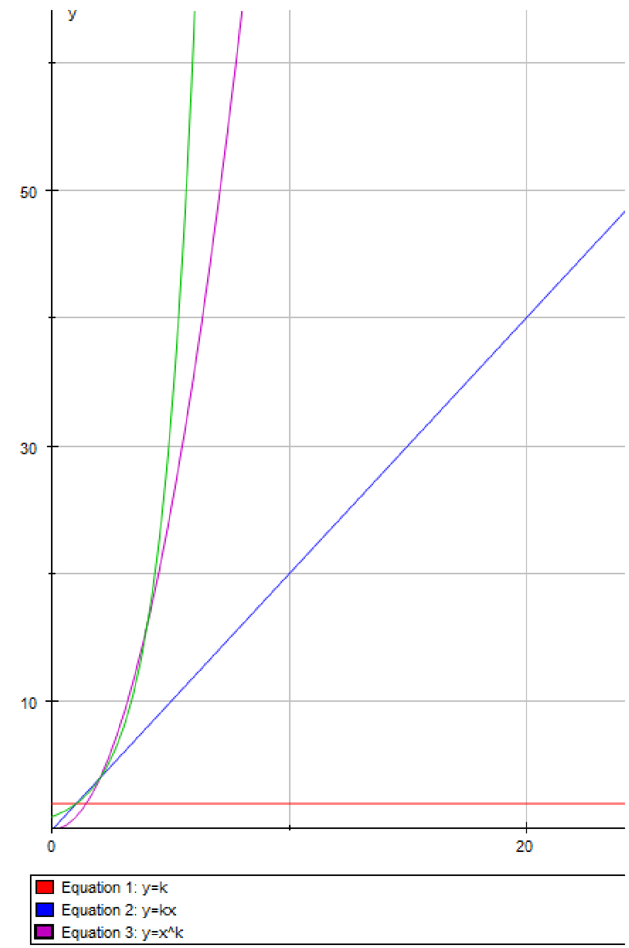
- O(k) Constant
- O(kn) Linear
- O(n^k) Polynomial
- O(k^n) Exponential



Big-O Notation

- O(k) Constant
- O(kn) Linear
- O(n^k) Polynomial
- O(k^n) Exponential

(Ranked in order of runtime as n tends toward infinity)



What is an NP-Complete Problem?

What is an NP-Complete Problem?

- Solved by a computer only by sometimes waiting an extraordinarily long time for a solution

What is an NP-Complete Problem?

- Solved by a computer only by sometimes waiting an extraordinarily long time for a solution
- Application of Cook’s Theorem – states that the Boolean Satisfiability Problem is NP-Complete

Satisfiability Problem (SAT)

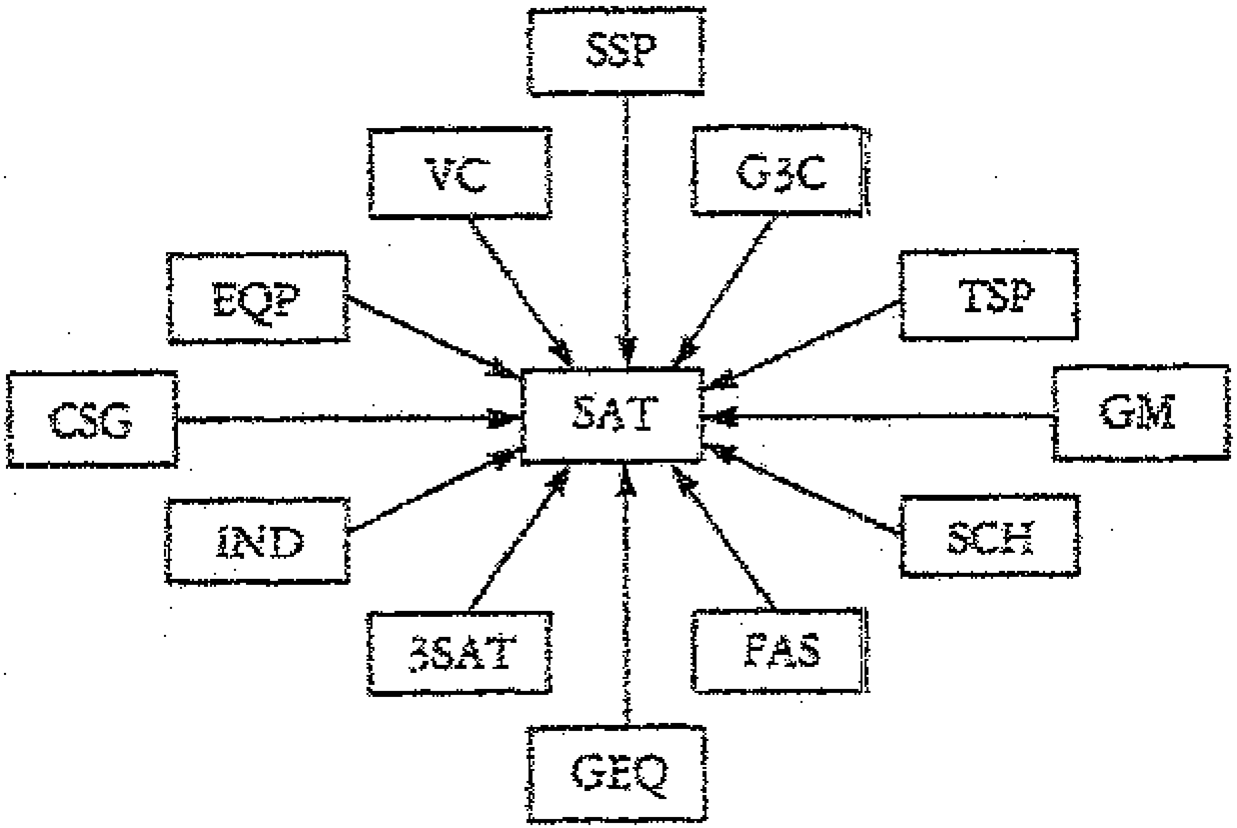
Satisfiability Problem (SAT)

- Asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE.

Satisfiability Problem (SAT)

- Asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE.
- A generic transformation from every NP Problem

Satisfiability Problem (SAT)



Satisfiability Problem (SAT)

Satisfiability Problem (SAT)

- Take the Travelling Salesperson Problem (TSP) – can be mapped to SAT, such that:

Satisfiability Problem (SAT)

- Take the Travelling Salesperson Problem (TSP) – can be mapped to SAT, such that:
  - The transformation can be computed in polynomial time

Satisfiability Problem (SAT)

- Take the Travelling Salesperson Problem (TSP) – can be mapped to SAT, such that:
  - The transformation can be computed in polynomial time
  - The instance of TSP has a solution only if the corresponding instance of SAT does

Satisfiability Problem (SAT)

Satisfiability Problem (SAT)

- Therefore ➔ SAT is at least as hard as any other problem in NP

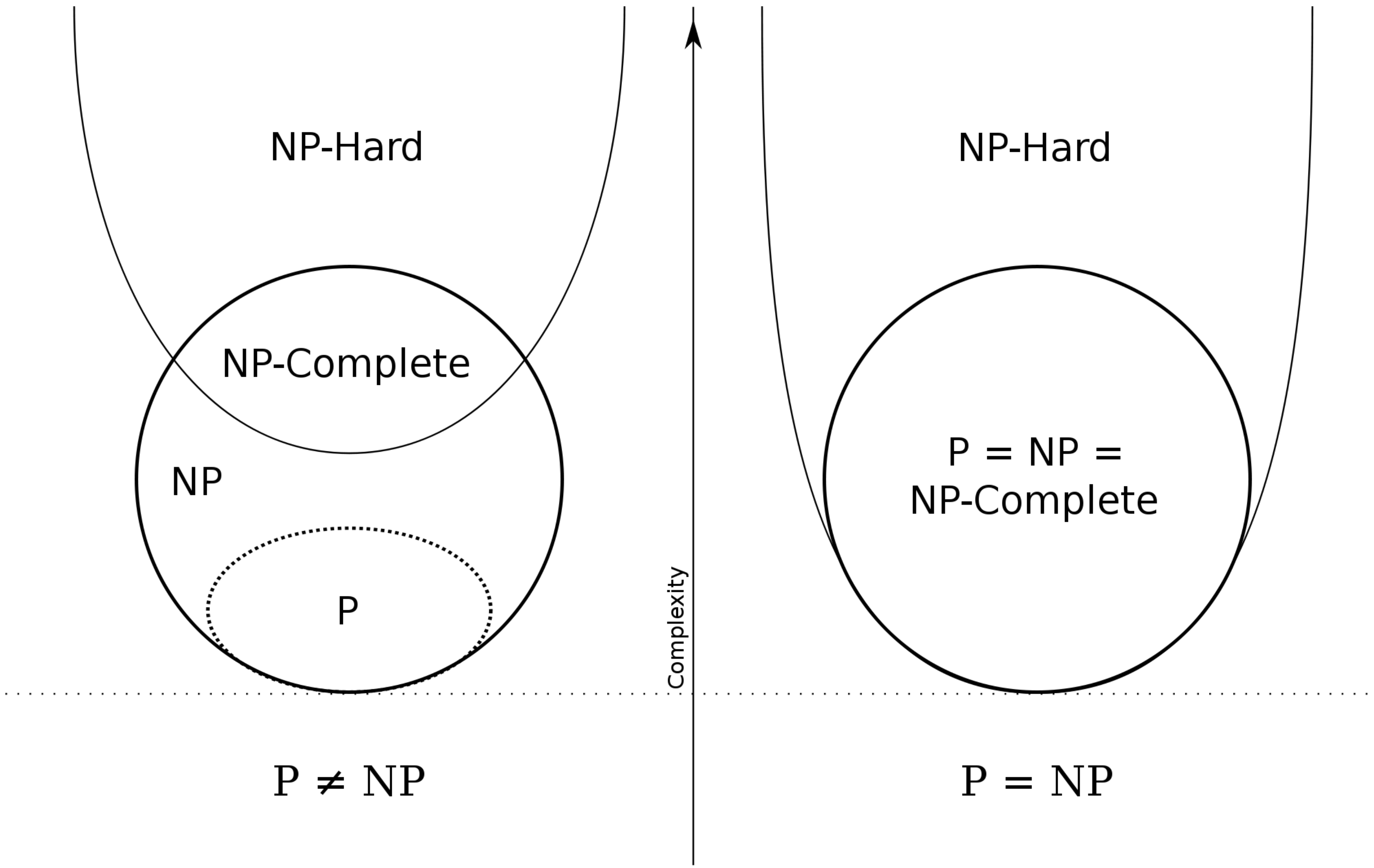
Satisfiability Problem (SAT)

- Therefore ➔ SAT is at least as hard as any other problem in NP
- If you can find an algorithm to solve SAT, it can also solve all problems in NP

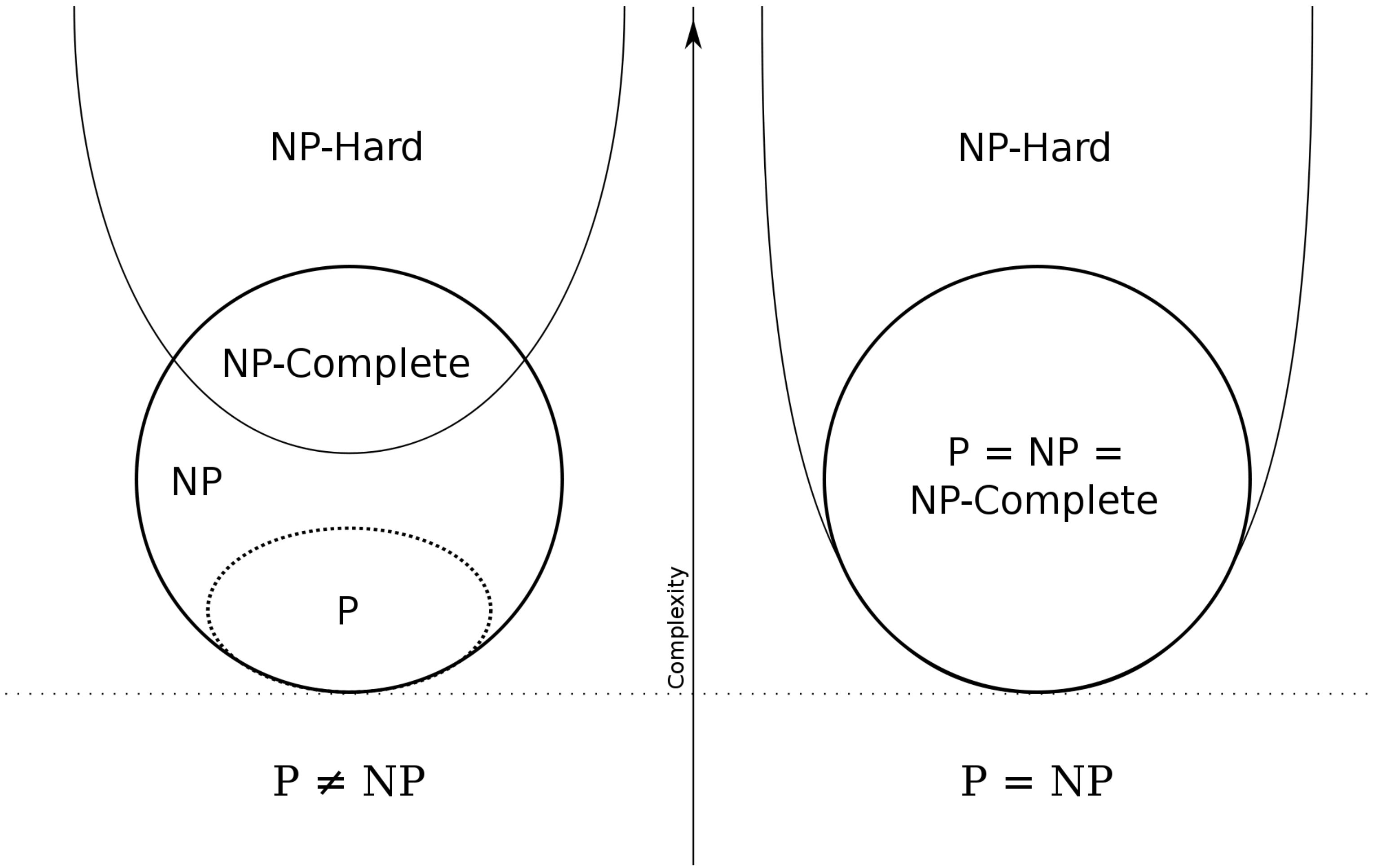
Satisfiability Problem (SAT)

- Therefore ➔ SAT is at least as hard as any other problem in NP
- If you can find an algorithm to solve SAT, it can also solve all problems in NP
- We call SAT “NP-Hard” – At least as hard as the hardest problems in NP

NP-Completeness



NP-Completeness



NP-Complete Problems are both in NP and also in NP-Hard

Partition Problem (PRT)

Partition Problem (PRT)

- In Decision Form: Given a set of positive integers, is there a partition of the set into two parts so that both sum to the same number?

Partition Problem (PRT)

- In Decision Form: Given a set of positive integers, is there a partition of the set into two parts so that both sum to the same number?
- 17, 5, 31, 12, 9, 20, 8

Partition Problem (PRT)

- In Decision Form: Given a set of positive integers, is there a partition of the set into two parts so that both sum to the same number?
- 17, 5, 31, 12, 9, 20, 8
- Computer guesses: 17, 5, 9, 20

Partition Problem (PRT)

- In Decision Form: Given a set of positive integers, is there a partition of the set into two parts so that both sum to the same number?
- 17, 5, 31, 12, 9, 20, 8
- Computer guesses: 17, 5, 9, 20
- Sum = 51

Partition Problem (PRT)

- In Decision Form: Given a set of positive integers, is there a partition of the set into two parts so that both sum to the same number?
- 17, 5, 31, 12, 9, 20, 8
- Computer guesses: 17, 5, 9, 20
- Sum = 51
- Complement = 31, 12, 8; Sum = 51

Partition Problem (PRT)

Partition Problem (PRT)

- For each yes instance of PRT (I.e. an instance that has a solution) – there must be a yes computation of minimum length

Partition Problem (PRT)

- For each yes instance of PRT (I.e. an instance that has a solution) – there must be a yes computation of minimum length
- One of these minimum computation lengths over all instances of size n would be a maximum

Partition Problem (PRT)

- For each yes instance of PRT (I.e. an instance that has a solution) – there must be a yes computation of minimum length
- One of these minimum computation lengths over all instances of size n would be a maximum
- This could be calculated exactly, but it suffices to say that PRT never requires more than $O(n^3)$ steps – so it may be completed in polynomial time

Partition Problem (PRT)

Partition Problem (PRT)

- We can alter PRT so it is no longer in NP

Partition Problem (PRT)

- We can alter PRT so it is no longer in NP
- “Each number in one part must be at least as large as any number in the other part”

Partition Problem (PRT)

- We can alter PRT so it is no longer in NP
- “Each number in one part must be at least as large as any number in the other part”
  - Sort the inputs into decreasing order; find their sum

Partition Problem (PRT)

- We can alter PRT so it is no longer in NP
- "Each number in one part must be at least as large as any number in the other part"
  - Sort the inputs into decreasing order; find their sum
  - Scan the numbers from first to last, accumulating a partial sum

Partition Problem (PRT)

- We can alter PRT so it is no longer in NP
- “Each number in one part must be at least as large as any number in the other part”
  - Sort the inputs into decreasing order; find their sum
  - Scan the numbers from first to last, accumulating a partial sum
  - If the partial sum even equals one half of the total, then output yes

The Class P

The Class P

- This new problem is in the class P

The Class P

- This new problem is in the class P
  - Can be solved and polynomial time, by a deterministic computer

P = NP?

P = NP?

- P must be a subset of NP (All problems in P can also be solved by a non-deterministic computer)

P = NP?

- P must be a subset of NP (All problems in P can also be solved by a non-deterministic computer)
- Therefore ➔ The actual question is “Are there any decision problems in NP that do not have a polynomial-time deterministic solution?”

P = NP?

P = NP?

- Hasn’t been proved either way

Further Reading

Further Reading

- Satisfyability Problem

Further Reading

- Satisfyability Problem
- Cook’s Theorem (the theorem that establishes SAT must be NP-Complete by mapping every problem in NP to SAT in polynomial time)