Chapter 12:
Theory of Computation

Computer Science: An Overview
Eleventh Edition

by
J. Glenn Brookshear

Chapter 12: Theory of Computation

- 12.1 Functions and Their Computation
- 12.2 Turing Machines
- 12.4 A Noncomputable Function
- 12.5 Complexity of Problems

The powers of computers

- **Goal:** To investigate the capabilities of computers
- **Question:** What computers can and can not do?
- Unsolvable v.s. Solvable v.s. Intractable

Functions

- **Function:** A correspondence between a collection of possible input values and a collection of possible output values so that each possible input is assigned a single output
  - **Converting measurements in yards into meters**
  - **Sort function**
  - **Addition function**
  - **etc**

Functions (Cont.)

- **Computing** a function: Determining the output value associated with a given set of input values
  - Compute the addition function to solve an addition problem;
  - Compute the sort function to sort a list
- The ability to compute functions is the ability to solve problems.
- Computer science: find techniques for computing the functions underlying the problems we want to solve

Techniques for computing functions

Inputs and outputs can be predetermined and recorded in a table

Yards (input)	Meters (output)
1	0.9144
2	1.8288
3	2.7432
4	3.6576
5	4.5720
.	.
.	.
.	.

Techniques for computing functions (Cont.)

- A more powerful approach: follow directions provided by an algebraic formula
  - $V = P(1+r)^n$
- Can the sine function be expressed in terms of algebraic manipulations of the degree value?
  - Need good approximation
- Some functions' input/output relationships are too complex to be described by algebraic manipulations.

Computability

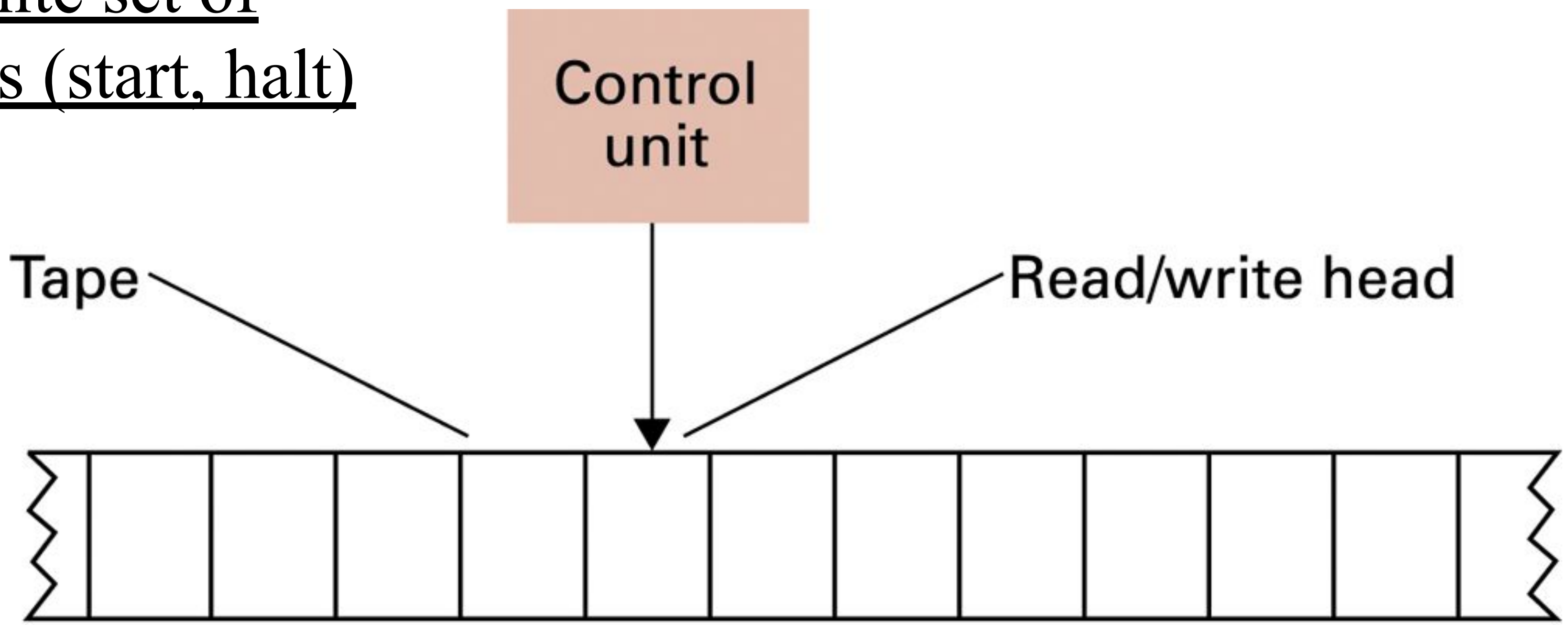
- Functions with increasing complexity need more powerful computing techniques.
- **Question:** Can we always find a system for computing functions, regardless of their complexity?
  - The answer is “No”. That is, no algorithmic system for some very very complex problems.
- **Noncomputable** function: A function that cannot be computed by any algorithm

Turing machines

- To understand capabilities and limitations of machines, many researchers have proposed and studied various computational devices.
- Alan M. Turing in 1936 proposed the Turing machines, which is still used today as a tool for studying the power of algorithmic processes.

Figure 11.2 The components of a Turing machine

- a finite set of symbols
- a finite set of states (start, halt)



Turing Machine Operation

- Inputs at each step
  - State
  - Value at current tape position
- Actions at each step
  - Write a value at current tape position
  - Move read/write head
  - Change state

Figure 11.3 A Turing machine for incrementing a value

Current state	Current cell content	Value to write	Direction to move	New state to enter
START	*	*	Left	ADD
ADD	0	1	Right	RETURN
ADD	1	0	Left	CARRY
ADD	*	*	Right	HALT
CARRY	0	1	Right	RETURN
CARRY	1	0	Left	CARRY
CARRY	*	1	Left	OVERFLOW
OVERFLOW	*	*	Right	RETURN
RETURN	0	0	Right	RETURN
RETURN	1	1	Right	RETURN
RETURN	*	*	No move	HALT

Figure 11.3 A Turing machine for incrementing a value

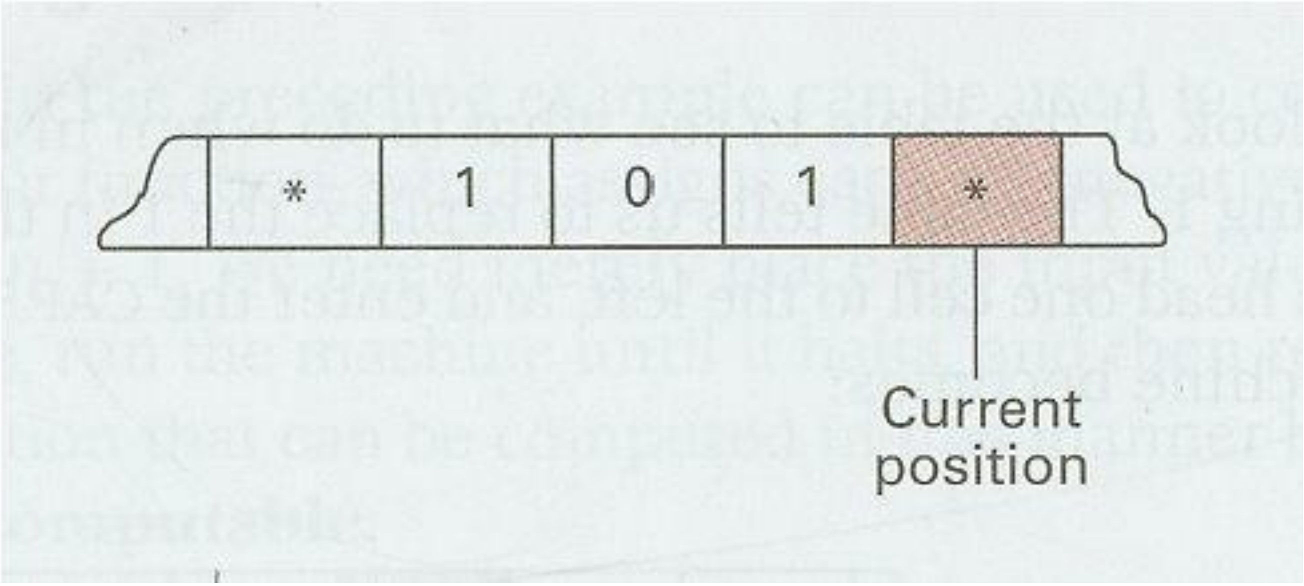


Figure 11.3 A Turing machine for incrementing a value

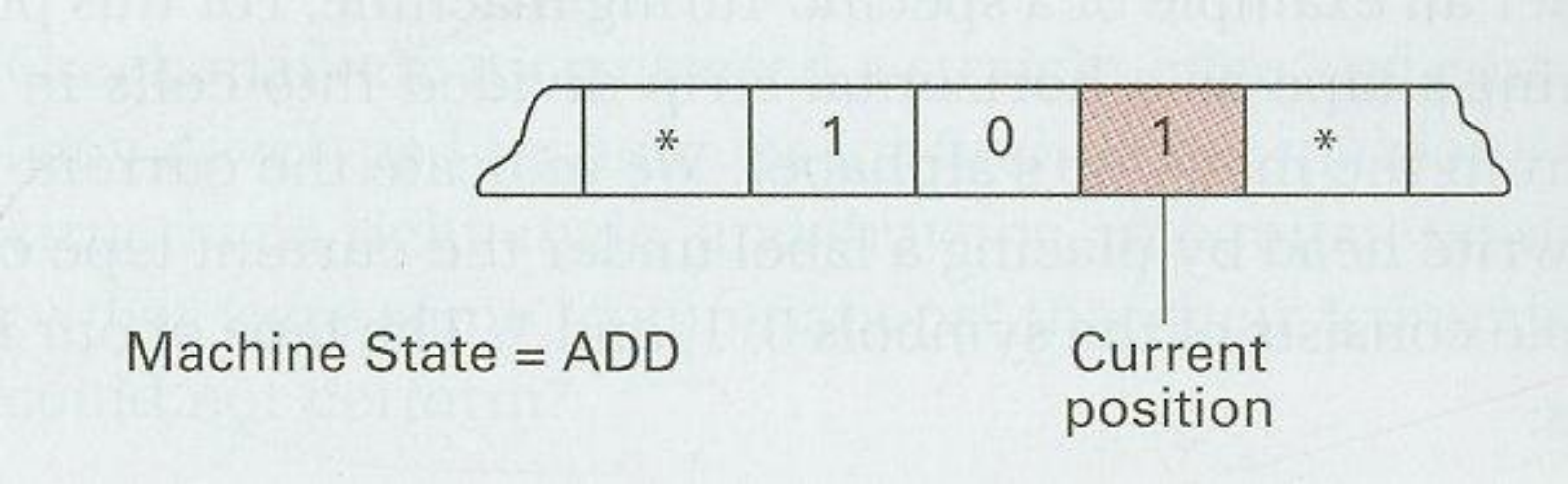


Figure 11.3 A Turing machine for incrementing a value

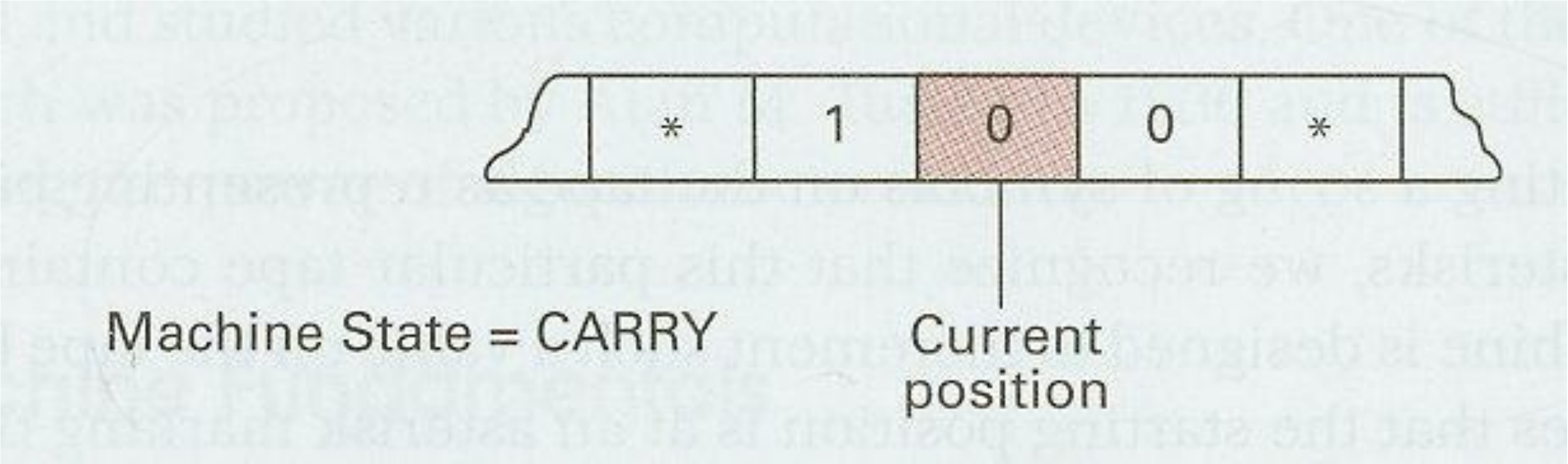


Figure 11.3 A Turing machine for incrementing a value

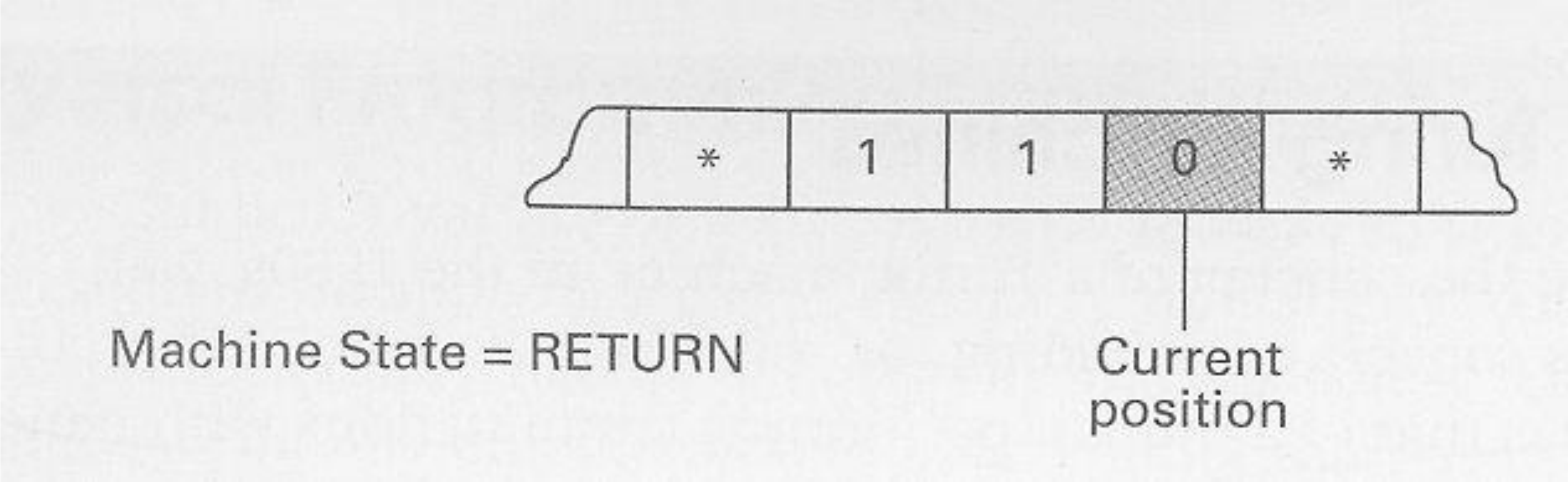


Figure 11.3 A Turing machine for incrementing a value

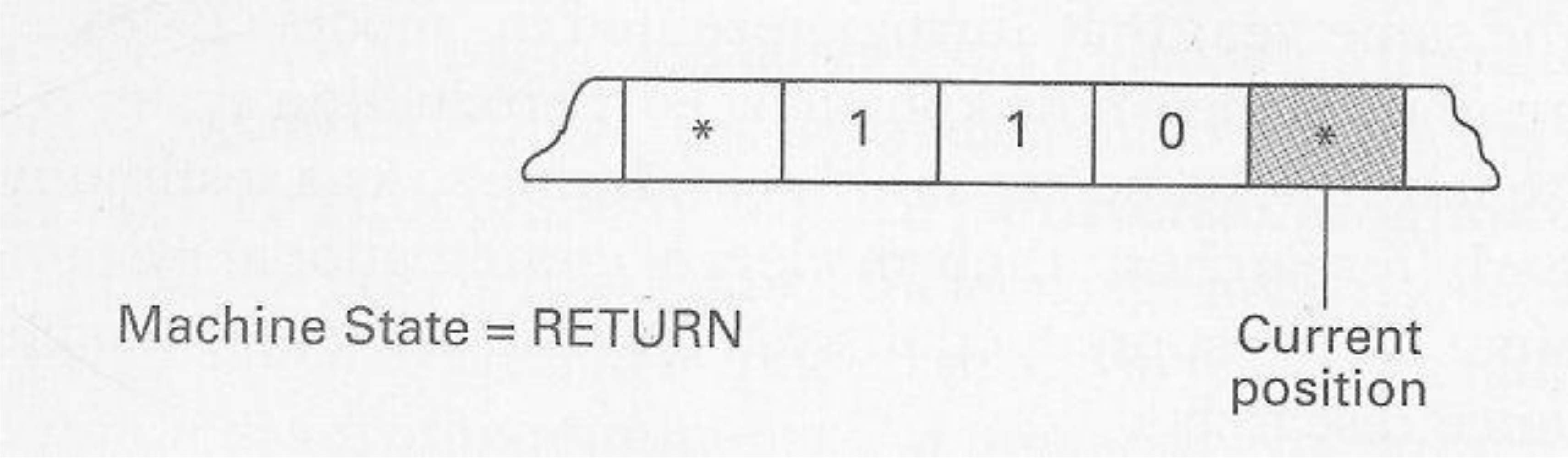
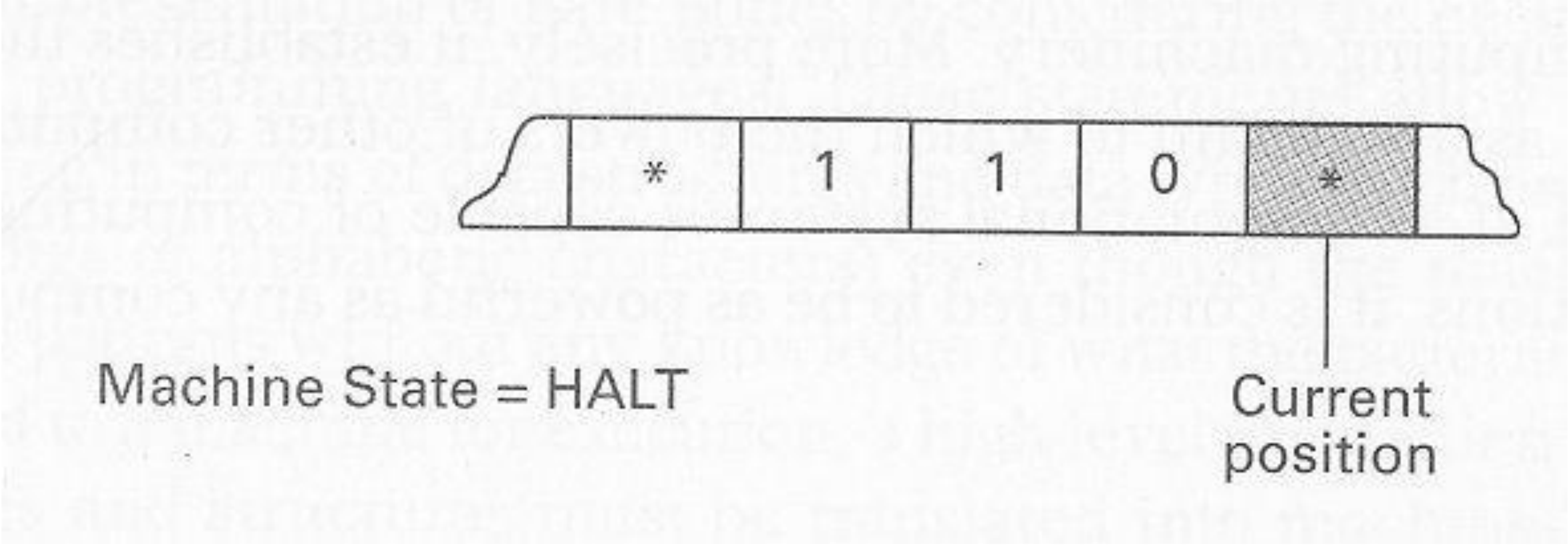


Figure 11.3 A Turing machine for incrementing a value



Church-Turing Thesis

- A function that can be computed by a Turing machine is said to be Turing computable.
- Turing's conjecture (Church-Turing Thesis): The functions that are computable by a Turing machine are exactly the functions that can be computed by any algorithmic means.
  - Widely accepted today

The Halting Problem

- Given any program with its inputs, return 1 if the program is self-terminating, or 0 if the program is not.

The Polynomial Problems

- The question of whether a solvable problem has a practical solution.
- Polynomial problems **P**: a problem is a polynomial problem if the problem is in O(f(n)), where f(n) is either a polynomial itself or bounded by a polynomial
  - E.g., $O(n^3)$, O(nlg n)
  - Searching a list, sorting a list
- The problems in **P** are characterized as having practical solutions.

The Polynomial Problems (Cont.)

- Consider the problem of listing all possible subcommittees that can be formed from a group of *n* people.
  - There are 2^n-1 such subcommittees
  - Any algorithm that solves this problem must have at least 2^n-1 steps
  - This problem is not in **P**.
- The non-polynomial problems are called "intractable".

The traveling salesman problem

- **Traveling salesman problem (TSP):** visit each of his clients in different cities without exceeding his travel budget (the length of the path does not exceed his allowed mileage)
  - An exponential time algorithm: consider the potential paths in a systematic manner

A nondeterministic algorithm for the traveling salesman problem

Pick one of the possible paths, and compute its total distance

If (this distance is not greater than the allowable mileage)

then (declare a success)

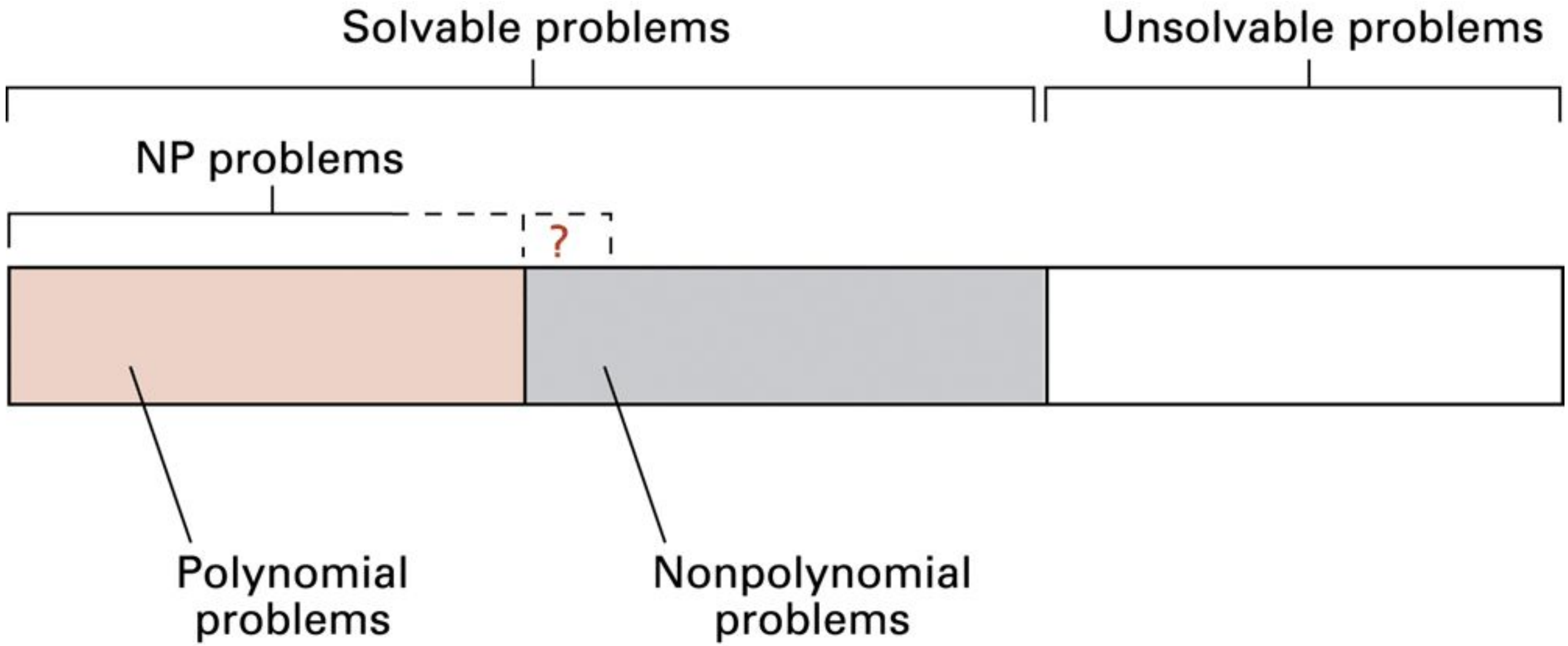
else (declare nothing)

=> A nondeterministic algorithm based on "lucky guess".

P versus NP

- **Class P:** All problems in any class O(f(n)), where f(n) is a polynomial
- **Class NP:** All problems that can be solved by a nondeterministic algorithm in polynomial time
  - **Nondeterministic algorithm** = an "algorithm" whose steps may not be uniquely and completely determined by the process state
- We know that **P** is a subset of **NP**.
- Whether the class **NP** is bigger than class **P** is currently unknown.
  - **NP** = **P**?

Figure 11.12 A graphic summation of the problem classification



NP-Complete problems

- NP-complete problems: NP problems to which all other NP problems can be reduced in polynomial time
- The traveling salesman problem is a NP-complete problem
- 3-Coloring problem: Given an undirected graph G = (V, E), determine whether G can be color with three color with the requirement
  - Each vertex is assigned one color and no two adjacent vertices have the same color