Chapter 11: Models of Computation

Invitation to Computer Science,
C++ Version, Third Edition

Objectives

In this chapter, you will learn about:

- A model of a computing agent
- A model of an algorithm
- Turing machine examples
- The Church–Turing thesis
- Unsolvable problems

Introduction

- Some problems do not have any algorithmic solution
- A model of a computer
  - Easy to work with
  - Theoretically as powerful as a real computer
  - Needed to show that something cannot be done by any computer

What Is a Model?

- Models are an important way of studying physical and social phenomena, such as
  - Weather systems
  - Spread of epidemics
  - Chemical molecules
- Models can be used to
  - Predict the behavior of an existing system
  - Test a proposed design

What Is a Model? (continued)

- A model of a phenomenon
  - Captures the essence (mportant properties) of the real thing
  - Probably differs in scale from the real thing
  - Suppresses some of the details of the real thing
  - Lacks the full functionality of the real thing

A Model of a Computing Agent

- A good model for the "computing agent" entity must:
  - Capture the fundamental properties of a computing agent
  - Enable the exploration of the capabilities and limitations of computation in the most general sense

Properties of a Computing Agent

- A computing agent must be able to:
  - Accept input
  - Store information and retrieve it from memory
  - Take actions according to algorithm instructions
    - Choice of action depends on the present state of the computing agent and input item
  - Produce output

The Turing Machine

- A Turning machine includes
  - A (conceptual) tape that extends infinitely in both directions
    - Holds the input to the Turing machine
    - Serves as memory
    - The tape is divided into cells
  - A unit that reads one cell of the tape at a time and writes a symbol in that cell

The Turing Machine (continued)

- Each cell contains one symbol
  - Symbols must come from a finite set of symbols called the alphabet
- Alphabet for a given Turing machine
  - Contains a special symbol *b* (for “blank”)
  - Usually contains the symbols 0 and 1
  - Sometimes contains additional symbols

The Turing Machine (continued)

- Input to the Turning machine
  - Expressed as a finite string of nonblank symbols from the alphabet
- Output from the Turing machine
  - Written on tape using the alphabet
- At any time the unit is in one of *k* states

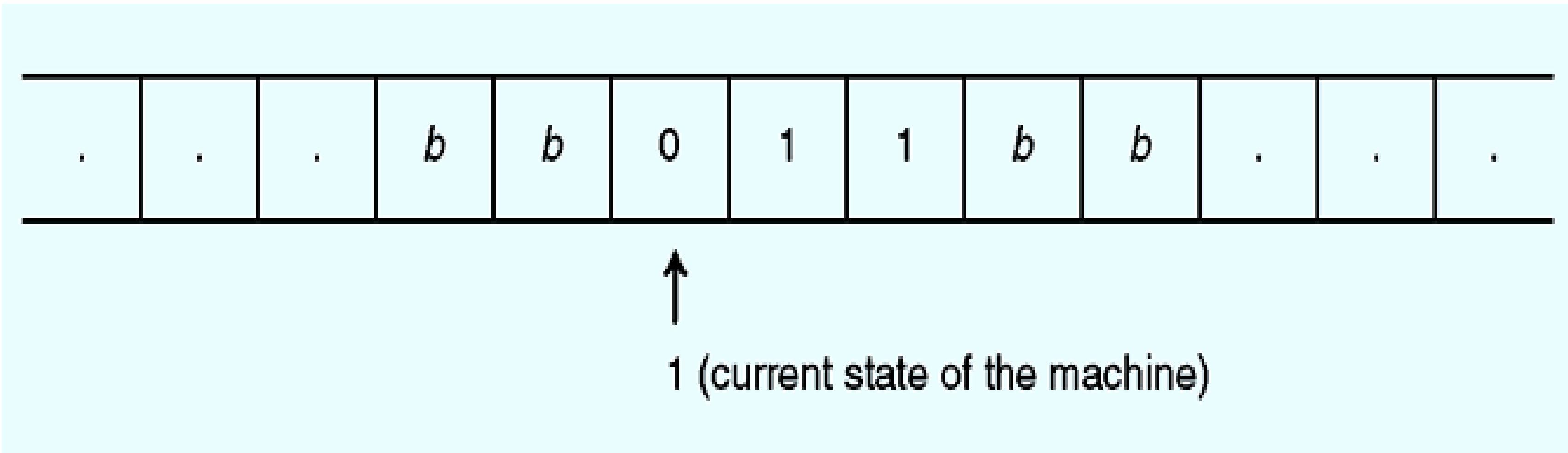

Figure 11.2

A Turing Machine Configuration

The Turing Machine (continued)

- Each operation involves:
  - Write a symbol in the cell (replacing the symbol already there)
  - Go into a new state (could be same state)
  - Move one cell left or right

The Turing Machine (continued)

- Each instruction says something like:

  if (you are in state i) and (you are reading symbol j) then

    write symbol *k* onto the tape

    go into state *s*

    move in direction *d*

The Turing Machine (continued)

- A shorthand notation for instructions
  - Five components
    - Current state
    - Current symbol
    - Next symbol
    - Next state
    - Direction of move
  - Form

    (current state, current symbol, next symbol, next state, direction of move)

The Turing Machine (continued)

- A clock governs the action of the machine
- Conventions regarding the initial configuration when the clock begins
  - The start-up state will always be state 1
  - The machine will always be reading the leftmost nonblank cell on the tape
- The Turing machine has the required features for a computing agent

A Model of an Algorithm

- Instructions for a Turing machine are a model of an algorithm
  - Are a well-ordered collection
  - Consist of unambiguous and effectively computable operations
  - Halt in a finite amount of time
  - Produce a result

Turing Machine Examples: A Bit Inverter

- A bit inverter Turing machine
  - Begins in state 1 on the leftmost nonblank cell
  - Inverts whatever the current symbol is by printing its opposite
  - Moves right while remaining in state 1
- Program for a bit inverter machine

  (1,0,1,1,R)

  (1,1,0,1,R)

A Bit Inverter (continued)

- A state diagram
  - Visual representation of a Turing machine algorithm
  - Circles are states
  - Arrows are state transitions

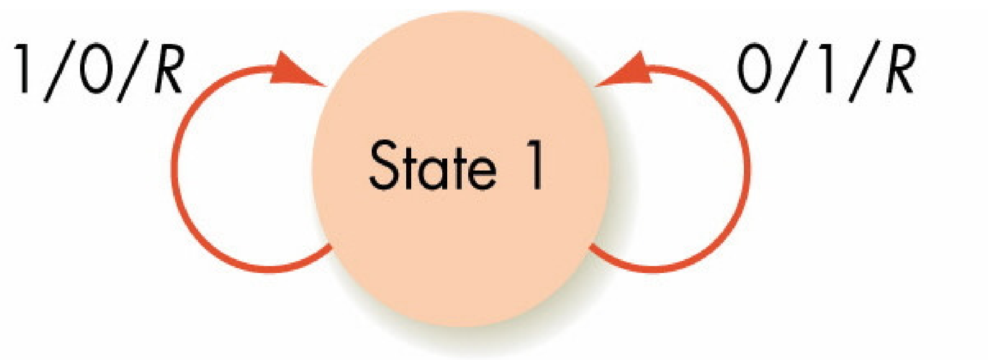

Figure 11.4
State Diagram for the Bit Inverter Machine

A Parity Bit Machine

- Odd parity bit
  - Extra bit attached to the end of a string of bits
  - Set up so that the number of 1s in the whole string, including the parity bit, is odd
    - If the string has an odd number of 1s, parity bit is set to 0
    - If the string has an even number of 1s, parity bit is set to 1

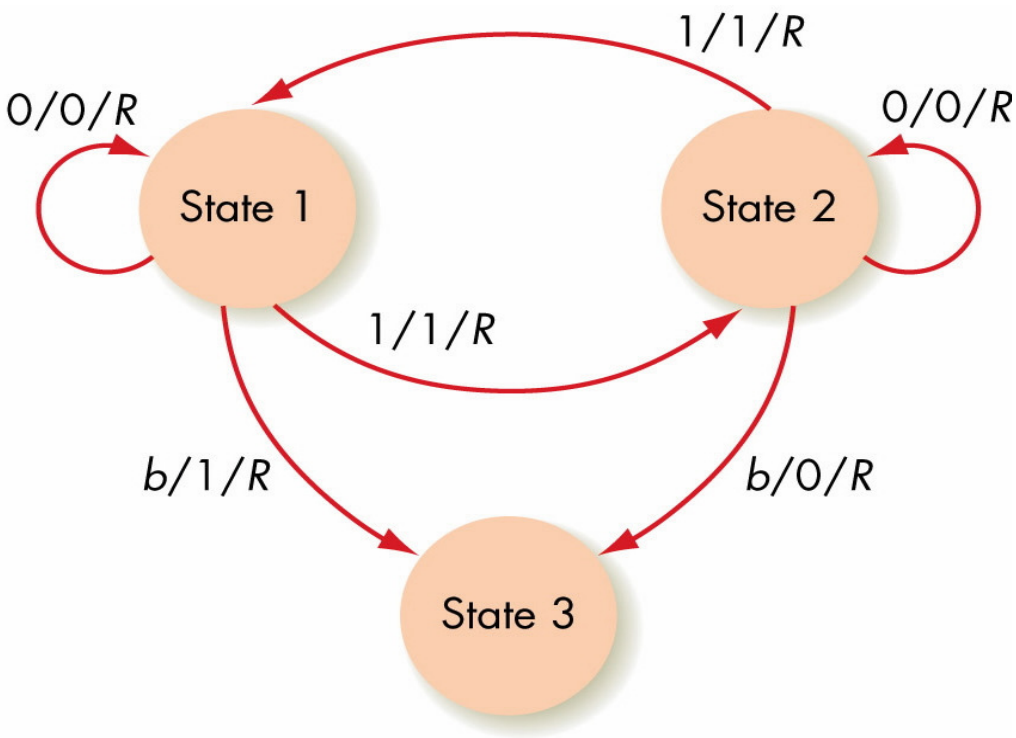

Figure 11.5

State Diagram for the Parity Bit Machine

A Parity Bit Machine (continued)

- Turing machine program for a parity bit machine

  (1,1,1,2,R)

  (1,0,0,1,R)

  (2,1,1,1,R)

  (2,0,0,2,R)

  (1,b,1,3,R)

  (2,b,0,3,R)

Machines for Unary Incrementing

- Unary representation of numbers
  - Uses only one symbol: 1
  - Any unsigned whole number n is encoded by a sequence of n + 1 1s
- An incrementer
  - A Turing machine that adds 1 to any number

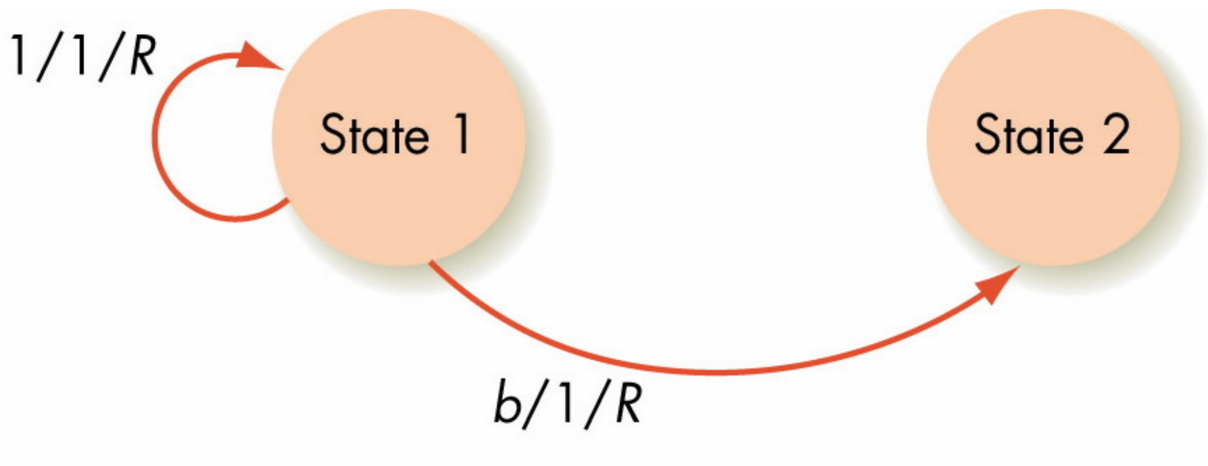

Figure 11.6
State Diagram for Incrementer

Machines for Unary Incrementing (continued)

- A program for incrementer

  (1,1,1,1,R)

  (1,b,1,2,R)

- An alternative program for incrementer

  (1,1,1,1,L)

  (1,*b*,1,2,L)

A Unary Addition Machine

- A Turing machine can be written to add two numbers, using unary representation
- The Turing machine program

  (1,1,b,2,R)

  (2,1,b,3,R)

  (3,1,1,3,R)

  (3,b,1,4,R)

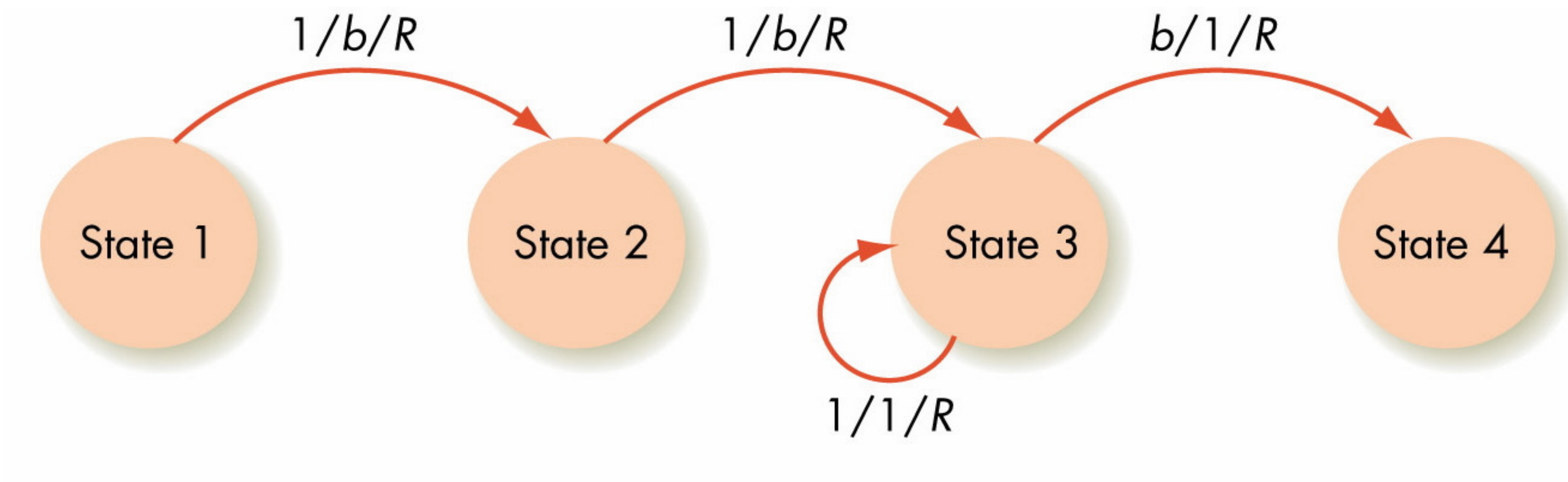

Figure 11.8

State Diagram for the Addition Machine

The Church–Turing Thesis

- Church–Turing thesis
  - If there exists an algorithm to do a symbol manipulation task, then there exists a Turing machine to do that task

The Church–Turing Thesis (continued)

- Two parts to writing a Turing machine for a symbol manipulation task
  - Encoding symbolic information as strings of 0s and 1s
  - Writing the Turing machine instructions to produce the encoded form of the output

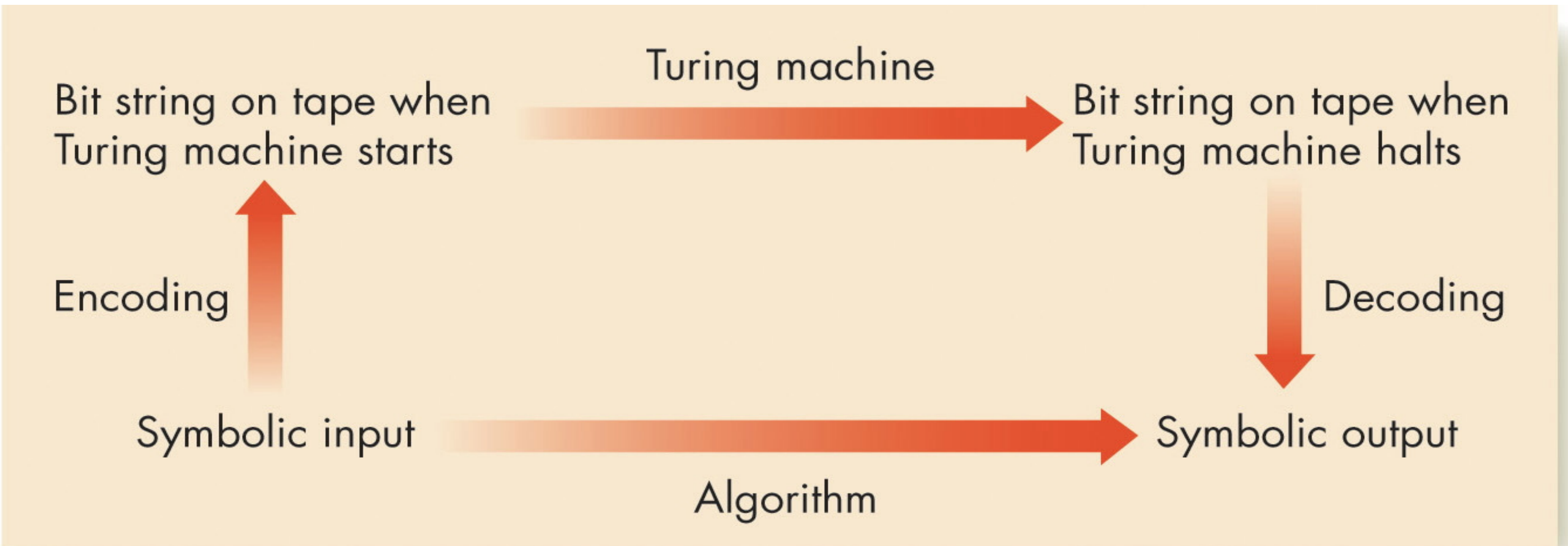

Figure 11.9

Emulating an Algorithm by a Turing Machine

The Church–Turing Thesis (continued)

- Based on the Church–Turing thesis
  - The Turing machine can be accepted as an ultimate model of a computing agent
  - A Turing machine program can be accepted as an ultimate model of an algorithm

The Church–Turing Thesis (continued)

- Turing machines define the limits of computability

- An uncomputable or unsolvable problem
  - A problem for which we can prove that no Turing machine exists to solve it

Unsolvable Problems

- The halting problem
  - Decide, given any collection of Turing machine instructions together with any initial tape contents, whether that Turing machine will ever halt if started on that tape

Unsolvable Problems (continued)

- To show that no Turing machine exists to solve the halting problem, use a proof by contradiction approach
  - Assume that a Turing machine exists that solves this problem
    - Show that this assumption leads to an impossible situation

Unsolvable Problems (continued)

- Practical consequences of other unsolvable problems related to the halting problem
  - No program can be written to decide whether any given program always stops eventually, no matter what the input
  - No program can be written to decide whether any two programs are equivalent (will produce the same output for all inputs)

Unsolvable Problems (continued)

- Practical consequences of other unsolvable problems related to the halting problem (continued)
  - No program can be written to decide whether any given program run on any given input will ever produce some specific output

Summary of Level 4

- Topics examined in Level 4: The Software World
  - Java: procedural high-level programming language
  - Other high-level languages: other procedural languages, special-purpose languages, functional languages, logic-based languages

Summary of Level 4 (continued)

- Topics examined in Level 4: The Software World (continued)
  - Series of tasks that a language compiler must perform to convert high-level programming language instructions into machine language code
  - Problems that can never be solved algorithmically

Summary

- Models are an important way of studying physical and social phenomena
- Church-Turing thesis: If there exists an algorithm to do a symbol manipulation task, then there exists a Turing machine to do that task
- The Turing machine can be accepted as an ultimate model of a computing agent

Summary

- A Turing machine program can be accepted as an ultimate model of an algorithm
- Turing machines define the limits of computability
- An uncomputable or unsolvable problem: we can prove that no Turing machine exists to solve the problem