**Chapter 12**

**Models of Computation**

**A Guide to This Instructor’s Manual:**

We have designed this Instructor’s Manual to supplement and enhance your teaching experience through classroom activities and a cohesive chapter summary.

This document is organized chronologically, using the same headings that you see in the textbook. Under the headings, you will find: lecture notes that summarize the section, Teaching Tips, Class Discussion Topics, and Additional Projects and Resources. Pay special attention to teaching tips and activities geared toward quizzing your students and enhancing their critical thinking skills.

In addition to this Instructor’s Manual, our Instructor’s Resources also contain PowerPoint Presentations, Test Banks, and other supplements to aid in your teaching experience.

|  |
| --- |
| **At a Glance** |

**Instructor’s Manual Table of Contents**

* Overview
* Learning Objectives
* Teaching Tips and Quick Quizzes
* Class Discussion Topics
* Additional Projects
* Additional Resources
* Key Terms
* Solutions to End-of-Chapter Exercises

|  |
| --- |
| Lecture Notes |

**Overview**

Chapter 12 examines the nature of computation and some important results from the theory of computation. It introduces models of computing agents and explains how Turing machines are good models of computing agents, and they also capture the features of algorithms. It describes how Turing machines work, and the thesis that Turing machines, as a model, capture all algorithms. The chapter ends with a discussion of unsolvable problems: in particular, the halting problem.

# **Learning Objectives**

* Explain the purpose of constructing a model
* List the required features of a computing agent
* Describe the components of a Turing machine, and explain how it is a good model of a computing agent
* List the features of an algorithm, and explain how a Turing machine program matches them
* • Simulate the operation of a simple Turing machine on specific inputs
* Construct a simple Turing machine from a specification, both writing the rules and drawing the state diagram
* State the Church-Turing thesis and explain what it means
* Justify why computer scientists believe the Church-Turing thesis is true
* Explain what an unsolvable problem is
* Describe the halting problem, what its inputs and outputs are
* Outline the proof that the halting problem cannot be solved by any Turing machine

# **Teaching Tips**

**12.1 Introduction**

1. Emphasize that there are problems that have no algorithmic solution, while there are other problems that very well may have one, but we haven’t been creative enough to figure them out.
2. Talk about stripping away the bells and whistles of modern computers to get to the core and find its fundamental features, and we can build a model of a computing agent with which to study what computing can and cannot do.

**12.2 What Is a Model?**

1. List the features of a model: it captures the essence of the real thing, it may differ in scale (physical size, time frame) from the real thing, it omits some details of the real thing, and it lacks some functionality of the real thing. Give details of different kinds of models: mathematical, physical, and computational.
2. Models can be physical objects like planes or people, but they can also be mathematical models written on paper with a pencil.
3. Models may be used to learn about the real thing: changes to the model produce effects transferrable to the real thing. Models may be safer to work with, or more convenient (in time scale or size) to work with than the real thing. Models may be used to predict the behavior of existing systems, and to prototype systems before building them.

**12.3 A Model of a Computing Agent**

1. Discuss the construction of a good model of a computing agent. Emphasize the need for it to provide an ability to explore the capabilities and limitations of computation in the most general sense. Essentially, it must capture the fundamental properties of computing while suppressing the lower-level details.
2. Any computing agent must have the ability to follow the instructions of an algorithm. The instructions must be presented in a form the computing agent can comprehend, but it does not matter the language in which those instructions are presented.
3. Introduce the term **Turing machine**, and describe how it fits the essential features of a computing agent. A Turing machine has a one-dimensional tape of infinite extent, used for input, memory, and output.
4. Introduce the term **tape alphabet** to describe the set of symbols that may be written on the tape. At any given moment, the Turing machine is located at one particular cell of the tape. The machine can read the symbol written in that cell, write a symbol to the cell, and move one cell to the left or right.Turing machines also have a finite number of internal states.
5. Introduce the term **Turing machine instructions**. Turing machines perform just one kind of operation. Given the current symbol on the tape and the current state, the machine can transition to a new state, overwrite the symbol with a new one, and move one cell to the left or right. Turing machine instructions may be encoded as a 5-tuple (current state, current symbol, next symbol, next state, and move direction).

|  |  |
| --- | --- |
| ***Teaching Tip*** | If students are to really understand Turing machines, building and testing them is very useful. JFLAP is a free Turing machine simulator, available at: <http://www.jflap.org> |

1. Introduce the term **Turing machine program**: A set of instructions that tell what to do in each situation.
2. Work through several examples of Turing machines, with the instructions written as 5-tuples. Ask students to identify the purpose of each machine. After several examples, ask students to walk through a new Turing machine on their own.

**Quick Quiz 1**

1. What are the four essential features of a computing agent?

Answer: taking input, storing and retrieving data from memory, performing actions based on input and state, and producing output

1. (True or False) A Turing machine uses its tape for input, output, and memory.

Answer: True

1. List the parts of a Turing machine instruction.

Answer: current state, current symbol, symbol to write, new state, and direction of movement.

1. Describe what the following Turing machine instruction means: (3,1,0,5,L).

Answer: if the machine is in state 3 and reads a 1 on the tape, overwrite the 1 with a 0, change to state 5, and move one cell to the left.

**12.4 A Model of an Algorithm**

1. Remind student of the features of an algorithm.
2. Emphasize that a Turing machine program fits the criteria of an algorithm.
3. The issue of Turing machines and halting is an important one; spend time going over the issue. Note that just like real programs, Turing machine programs can be written that loop infinitely, but we don’t consider those correct algorithms.
4. A Turing machine is a model of a computing agent. Because the Turing machine program is usually combined intrinsically with the machine, we consider a Turing machine with its program to be a model of an algorithm.

**12.5 Turing Machine Examples**

1. The purpose of this section is to show that while a Turing Machine may not do anything earthshaking, they can still accomplish some worthwhile things.
2. A bit inverter takes a string of 0s and 1s, and turns every 0 into a 1, and every 1 into a 0. A Turing machine for this task needs only one state and can simply move to the right changing each bit as it goes, until it reads a blank symbol.
3. Introduce the term **state diagram**, a visual representation of a Turing machine algorithm. Draw the state diagram for the bit inverter and work through an example with it.
4. Introduce the term **parity** bit. A parity bit machine takes a string of 0s and 1s and adds an extra bit to the end of the string.
5. The goal of the bit is to ensure that the number of 1s in the whole string (including the new one) is odd: odd parity. Give short and long examples of strings with odd and even parity, and show what would be added in each case.
6. Introduce the term **unary representation**, for representing nonnegative integers. Draw students’ attention to the fact that this representation uses 1 to represent 0, and 11 to represent 1, and so on. A unary incrementer needs to add a 1 to the string. Show both alternative implementations to solve this problem, and work through examples. Demonstrate that we can still evaluate Turing machine algorithms for efficiency.
7. Unary addition takes in two strings of 1s separated by a blank and leaves the sum of the two numbers, in unary, on the tape. Develop the require steps: filling in the blank with a 1 and removing two other 1s from somewhere, and then complete the state diagram.

**Quick Quiz 2**

1. In order to solve the odd parity bit problem, the Turing machine needed \_\_\_ states.

Answer: 3

1. (True or False) Turing machines cannot implement arithmetic.

Answer: False

1. (True or False) A Turing machine may use states to distinguish between blanks it needs to overwrite and blanks that indicate it should halt.

Answer: True

**12.6 The Church-Turing Thesis**

1. Introduce the **Church-Turing thesis** and explain its claim that every symbol manipulation task that has an algorithmic solution can be carried out by a Turing machine.
2. Explain that a thesis is something that cannot be proven, but that no one has disproven it in all the years of trying.
3. Introduce the terms **computability**, and **uncomputable** or **unsolvable** problems. A problem may be shown to be computable by giving a Turing machine for it. Showing something is not computable requires more work.

**12.7 Unsolvable Problems**

1. Define the **halting problem**, essentially 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.
2. Introduce the term **proof by contradiction**. State that we will assume the opposite of what we want to prove, and then show that a consequence of the assumption is a contradiction. Go through the steps of the halting problem proof starting on page 622.
   1. Assume P is a Turing machine that solves the halting problem. The input to P is an encoding of a set of Turing machine instructions, and an encoding of the input for that machine. If P leaves a single 1 on the tape, then the input instructions halt on the given input. Otherwise P will leave a single 0 on the tape.
   2. Modify P to make machine Q. Q acts just like P, except when Q reaches the state where P was to write a 1 and halt, Q goes into an infinite loop.
   3. Build a machine S that first copies its input on the tape, and then runs just like Q.
   4. What happens when S is run on its own encoding?—A contradiction!

|  |  |
| --- | --- |
| ***Teaching Tip*** | If students find the halting problem proof difficult to understand in terms of Turing machines, show them that the same proof pertains if we assume algorithm P exists, written in pseudocode or a programming language they know. Then write Q and S in pseudocode or that language. |

1. Note that this proof only applies to the most general problem.
2. There are some specific Turing machines, algorithms, and programs, that the halting problem may be answered. The proof states that no *one algorithm*exists that can answer the halting problem for *every possible algorithm*.

**Quick Quiz 3**

1. (True or False) The halting problem pertains to whether a Turing machine halts when given no input on its tape.

Answer: False (it is more general than that)

1. To prove the halting problem, we must use the proof technique called: \_\_\_\_\_\_\_\_\_\_\_\_\_.

Answer: proof by contradiction

# **Class Discussion Topics**

1. As a computing agent, in what ways is a Turing machine different from a human being? Are there any features Turing machines lack that are important for understanding what humans can express or compute algorithmically?
2. Think about a variation on a Turing machine that does not have an infinite tape. Instead, its tape is N cells long. Can you think of problems, such a machine could not solve, that could be described algorithmically?
3. What does it mean for the field of computer science that there are unsolvable problems?

# **Additional Projects**

1. Given a Turing machine program described as a set of 5-tuples or a state diagram, determine the result of the program on a set of provided inputs, and describe the function implemented by the program.
2. Working with a partner, design a Turing machine that can compute one of these operations:
   1. The unary decrement operation
   2. An operation to check if a binary input string represents an unsigned integer that is even or odd
   3. An operation to see if the input string consists of some number of 0s followed by some number of 1s (excluding strings like 0011100 and 1100011010)
   4. Using unary to perform addition and subtraction of various integers

# **Additional Resources**

1. An online biography of Alan Turing: <http://www.turing.org.uk/bio/part1.html>
2. An online biography of Alonzo Church: <http://www.gap-system.org/~history/Biographies/Church.html>
3. There are many physical implementations of Turing machines demonstrated online. Here are two: <http://www.youtube.com/watch?v=E3keLeMwfHY&feature=related> and <http://www.youtube.com/watch?v=cYw2ewoO6c4&feature=related>

**Key Terms**

* **Church-Turing thesis**: The thesis that if there exists an algorithm to do a symbol manipulation task, then there exists a Turing machine to do that task.
* **Computability**: That can be done by symbol manipulation algorithms.
* **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.
* **Proof by contradiction**: Assume the opposite of what you want to prove and follow the logical consequences until you reach a contradiction, thereby showing that your assumption is incorrect.
* **Parity bits**: Used to detect errors that occur as a result of electronic interference when transmitting various strings.
* **Recognizer**: A Turing machine that decides whether the string of characters initially on its tape matches a certain pattern.
* **State diagram**: A visual representation of a Turing machine algorithm.
* **Tape alphabet**: The allowable symbols that can be written on a Turing machine tape.
* **Turing Award**: The most prestigious technical award given by the Association for Computing Machinery; it is for “contributions of lasting and major technical importance to the computer field.”
* **Turing machine**: A theoretical model of a computing agent in which symbols are written in cells of a hypothetical infinite tape and are read and changed by a read/write unit according to the input symbol, the state of the read/write head, and the set of rules for the Turing machine.
* **Turing machine instruction**: Given the current tape symbol and the state of the read/write unit, it describes the symbol to write, the next state, and the direction of movement of the read/write unit.
* **Turing machine program**: A set of Turing machine instructions that allow a Turing machine to carry out a certain task.
* **Unary representation**: Using only one symbol.
* **Uncomputable/unsolvable**: A problem that cannot be solved by an algorithm.

# **Solutions to End-of-Chapter Exercises**

1. Answers may vary. Factors that might be included are the incubation period, population density, time until death or recovery, probability of infection if exposed.
2. This is a subjective question, but students should be able to back up their answers with reasonable arguments. For example: In the first case the manufacturer is likely to be found accountable because it did not take what are considered standard testing steps (prototype vehicles). In the second case, the manufacturer may not be found accountable because the prototype is presumed to be closer to the real thing than a simulation, so it could be argued that if the fault is not detected in the prototype, it would not have been detected in a simulation.
3. a. modeling long-term effects on the body of some drug

b. modeling nutritional content of a food product under development

c. modeling potential risks/hazards and their effect on policy holders

1. a. yes - it takes and stores input (alarm time setting), takes action according to   
    instructions (alarm, music, snooze alarm, etc.), produces output

b. no - it is a physical device that does not follow instructions

c. no - same as above

d. yes (it is clear that it has all four properties)

1. The next configuration is

|  |  |  |  |
| --- | --- | --- | --- |
| b | 0 | 0 | b |
| ↑ |  |  |  |
| 2 |  |  |  |

1. No. This machine moves only to the right, so in the previous configuration it had to be looking at the position now occupied by the 0. Neither instruction writes a 0, therefore the 0 character must have been present in the previous configuration. However, had the machine been in state 1 looking at a 0, it would have halted because there is no appropriate instruction.

**7.** No. There are two different instructions that have the form (3,0,-,-,-).

**8.**  b 1 0 0 0 1 b

**9.** b 1 1 1 b

**10.** The Turing machine cycles forever back and forth over the two leftmost nonblank cells, alternating between states 1 and 2.

**11.** The Turing machine never halts, but keeps writing

1 0 0 1

1 0 0 0 1

1 0 0 0 0 1

etc. on the tape.

**12.** It passes over the original binary string, leaving it unchanged, and then moves forever right, changing blanks to 1s.

**13.** (1,1,0,2,L) change leftmost 1 to 0 and halt

**14.** No - this Turing machine only works on this particular input string, or a shorter string; it will not work on a longer string.

**15.** a. (1,1,1,1,R) pass right over all the 1s

(1,b,b,2,L) find the right end of the string, move left

(2,1,0,2,R) change rightmost 1 to 0

b. The solution to part (a) will work on any unary string

**16.** (1,1,b,1,L) remove a single 1 from the string (n > 0, so there are at least two 1s in the string to begin with)

**17.** (1,1,b,2,R) remove a single 1 from the string

(2,b,0,1,R) if n = 0, then tape contained a single 1, write a 0

**18**. (1,1,1,1,R) pass over any 1s

(1,0,1,1,R) change any 0s to 1s

**19.** (1,1,1,2,R) leave 1 alone, go to state 2

(2,1,0,1,R) change 1 to 0, go to state 1

**20.** (1,1,1,2,R) Even parity state reading 1, change state.

(1,0,0,1,R) Even parity state reading 0, don’t change state.

(2,1,1,1,R) Odd parity state reading 1, change state.

(2,0,0,2,R) Odd parity state reading 0, don’t change state.

(1,b,0,3,R) End of string in even parity state, write 0 and go to state 3.

(2,b,1,3,R) End of string in odd parity state, write 1 and go to state 3.

**21.** (1,1,1,2,L) Pass to the left over start of string

(2,b,1,3,L) Add the first 1 at the lefthand end of the string

(3,b,1,4,L) Add the second 1 at the lefthand end of the string

(4,b,1,5,L) Add the third 1 at the lefthand end of the string

**22.** (1,1,1,1,R) in state 1, find the right end of the string

(1,0,0,1,R)

(1,b,0,2,L) attach a 0 on the right end, prepare to move left

(2,1,1,2,L) in state 2, find the left end of the string

(2,0,0,2,L)

(2,b,0,1,R) attach a 0 on the left end, prepare to move right

**23.** (1,1,1,2,R) read initial 1

(2,b,1,5,R) number is 0, write the second 1 and halt

(2,1,1,3,R) number was not 0, move right

(3,b,1,5,R) if number is 1, write the third 1 and halt

(3,1,1,4,R) if number > 1, leave third 1 and move right

(4,1,b,4,R) erase any additional 1s

**24**. (1,1,0,1,R) change leftmost 1 to 0

(1,1,1,1,R) move right over any 1s

(1,b,b,2,L)

(1,X,X,2,L) right end of 1s found, move left

(2,1,X,3,L) mark rightmost 1 with X, move left

(3,1,1,4,L) more 1s remain in first half

(4,1,1,4,L) move to left end of 1s, start again

(4,0,0,1,R)

(3,0,0,5,R) no more 1s, midpoint found

(5,X,1,5,R) change markers in right half to 1s

**25**. (1,1,X,2,R) mark a 1 in first number, move right

(2,1,1,2,R)

(2,b,b,3,R) separating blank found

(3,X,X,3,R) move right over markers in second number

(3,b,b,4,L) first number larger, prepare to erase second number markers

(4,X,b,4,L) erase second number markers

(4,b,b,5,L) separating blank found

(5,1,1,5,L)

(5,X,1,5,L) restore first number

(3,1,X,6,L) second number at least as big as first, set marker,

move left to start again

(6,X,X,6,L) move left over second number markers

(6,b,b,7,L) separating blank found, move left to first number

(7,1,1,7,L) move left over first number

(7,X,X,1,R) first number markers found, start over

(1,b,b,8,L) first number exhausted, so second number is larger

(8,X,X,8,L)

(8,b,b,9,R) find left end of string

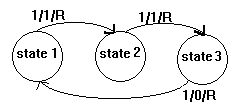
(9,X,b,9,R) erase first number

(9,b,b,10,R) separating blank

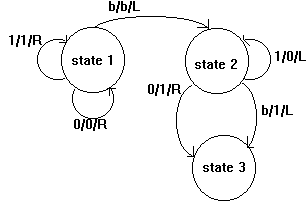
(10,X,1,10,R) restore second number

**26**. Yes, it simply erases the single 1 that represents the 0 and halts, leaving on the tape the representation of the second number.

**27**.



**28**.



**29.** The TM sweeps to the right, then to the left, changing 1s to 0s, then back to the right, changing 0s to 1s. Space efficiency is n since no new cells are used. Time efficiency is about 3n for the three sweeps of the input. The exact expression is 3n + 2, the 2 being the two turnarounds.

**30**. Multiply by using repeated additions. Use the first of the two numbers to count the number of additions. For each 1 in the first number (except the final one that is the extra representation digit), do a copy of the second number. Then add the copies together one by one.

**31**. a. yes

b. (1,1,1,1,R) move right to separating blank

(1,b,1,2,R) fill blank with 1

(2,1,1,2,R) move to far right end

(2,b,b,3,L)

(3,1,b,4,L) erase a 1

(4,1,b,5,L) erase a 1

(1,1,b,2,R) erase 1 on left end

(2,1,1,2,R) move to blank

(2,b,1,3,R) fill blank

(3,1,1,3,R) move to far right end

(3,b,b,4,L)

(4,1,b,5,L) erase a 1

c. The one given in the chapter seems most efficient. Both of the other two require traveling all the way to the far right end.

d. algorithm from the chapter: n + 2 ( n + 1 to traverse the representation of n plus 1 to fill in the blank)

algorithm 1: n + m + 3 (to traverse the tape) + 3 (to recognize the far right end and delete the two 1s)

algorithm 2: n + m + 3 (to traverse the tape) + 2 (to recognize the far right end and delete a 1)

The first is most efficient.

e. n + m + 3

**32**. (1,1,b,2,R) first character is a 1, blank it out and go to state 2

(1,0,b,3,R) first character is a 0, go to state 3

(2,1,1,2,R)

(2,0,0,2,R) in state 2, find the right end of the string

(2,b,b,4,L)

(3,1,1,3,R)

(3,0,0,3,R) in state 3, find the right end of the string

(3,b,b,6,L)

(4,1,b,5,L) last character is a 1 (matching first character), blank it out and go left in state 5

(4,0,0,4,R) last character is a 0, not a palindrome, halt with nonblank tape

(6,0,b,5,L) last character is a 0 (matching first character), blank it out and go left in state 5

(6,1,1,6,R) last character is a 1, not a palindrome, halt with nonblank tape

(5,1,1,5,L)

(5,0,0,5,L) OK so far, go to left end and start again  
 (5,b,b,1,R)

**33.** a.The language consists of strings of one or more 1's.

b. (1,1,b,2,R) One 1 has been read

(2,1,b,2,R) Read any additional 1's. Any 0's on the tape cause the machine to   
 halt with nonblank tape.

**34.** a. The language consists of all strings that start with zero or more pairs of 0's followed by a single 1.

b. (1,0,b,2,R)

(2,0,b,1,R) Cycle back and forth between states 1 and 2 to read pairs of 0's

(1,1,b,3,R) This should be the single 1 at the end, make sure there are no more 1's  
(3,b,b,4,R) no more 1's, machine halts

**35.** a. The language consists of all strings beginning with two 0's, followed by 1 or more 1's, followed by two 0's.

b. (1,0,b,2,R) Read the two 0's at the front

(2,0,b,3,R)

(3,1,b,4,R) Read the single required 1

(4,1,b,4,R) Read any additional 1's

(4,0,b,5,R) Read the two 0's at the back

(5,0,b,6,R)

**36.** a. The language consists of strings beginning with some positive number of 0's followed by the same number of 1's

b. (1,0,b,2,R) Blank out first 0

(2,0,0,2,R) Move right over remaining 0's and 1's

(2,1,1,3,R)

(3,1,1,3,R)

(3,b,b,4,L) At far right end, move left

(4,1,b,5,L) Blank out matching 1

(5,1,1,6,L) Go to the left end and start again

(6,1,1,6,L)

(6,0,0,7,L)

(7,0,0,7,L)

(7,b,b,1,R)

**37.** Yes. The Church-Turing thesis says nothing about halting. But even the unsolvability of the halting problem says that there is no algorithm to test *all* program/input pairs, however there may well be a way to test a specific program/input pair.

**38**. It proves the existence of clearly-stated problems that have no algorithmic solution.

**39**. The first one.

**40**. The algorithm for the solution is to run the Turing machine on the tape and wait for 10 steps. If the machine has halted within that time, the answer is yes, otherwise the answer is no. It is the limited wait time that makes this problem solvable where the general halting problem is not.

**Challenge Work**

**1**. This is quite a complex problem. Students might be guided to an automata theory textbook for additional help. The general ideas follow:

a. Assume that one binary string is placed on the first track and the second binary string is placed on the second track, with their left ends aligned. Then the head must simply move from left to right across the two strings, comparing pairs of symbols. If at any point they do not match, then the machine should go into a HALT-NO MATCH state. If the head reaches the right end and hits matching blanks on the two tapes, then it should go into a HALT-MATCH state.

b. Assume that the tape initially contains the first binary string, a separating blank, and then the second binary string. The machine must move back and forth comparing corresponding elements in the two strings. Markers will need to be used to indicate positions that have already been compared. As before, if corresponding digits don't match then the machine should go into a HALT-NO\_MATCH state. If, on the other hand, every digit matches and the strings are the same length, then the machine should go into a HALT\_MATCH state.

c. This is a fairly easy proof. One can simulate a one-track machine trivially on a two-track machine by setting up the rules to carry out the original algorithm on the first track while ignoring whatever is on the second track.

d. This is much harder than part (c), but the idea is to put the contents of the first track, followed by a separating blank, followed by the contents of the second track all on a single tape. The single-track machine then uses several steps to carry out each step of the multitrack machine, namely reading a single symbol from the left half, laying down a marker, running to the right to read the corresponding symbol from the second half, and resetting the symbols in each half appropriately. As in part (b), there will be much sweeping back and forth between the locations on the single tape that represent corresponding positions of the two tracks. In addition, if the two-track machine ever writes symbols to the left of the starting location, then the one-track machine must copy the second half of the tape over farther to the right to create additional room after the separating blank.

**2**. Students might be interested to read or view "Breaking the Code," the play based on Alan Turing's life. A comparison of its perhaps fictionalized account with more biographical accounts would be interesting. Also, the 2014 movie "The Imitation Game" depicts Alan Turing's work during World War II.