Theory of Languages and Automata

Chapter 3- The Church-Turing Thesis

Sharif University of Technology

Turing Machine

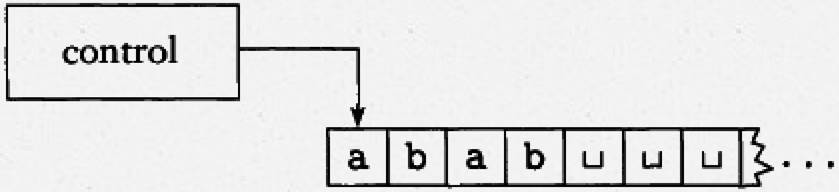
- Several models of computing devices
  - Finite automata
  - Pushdown automata
- Tasks that are beyond the capabilities of these models
  - Much more powerful model
  - Proposed by Alan Turing in 1936

Turing Machine (cont.)

- Similar to a finite automaton
  - ✓ Unlimited memory
- Can do everything that a real computer can do
- Cannot solve certain problems
  - ✓ Beyond the theoretical limits of computation

Tape

- An infinite tape
  - ✓ A tape head to read and write symbols
- Initially contains the input string and blank everywhere else
- Outputs: accept and reject
  - ✓ By entering accepting and rejecting states
  - ✓ If it doesn't enter an accepting or a rejecting state, never halts



Differences with finite automata

1. A Turing machine can both write on the tape and read from it.
2. The read-write head can move both to the left and to the right.
3. The tape is infinite.
4. The special state for rejecting and accepting take effect immediately.

Example

- $B = \{w\#w \mid w \in \{0,1\}^*\}$

- M_1
  - ✓ Accept if its input is a member of B
  - ✓ Reject, otherwise
- Strategy: zigzag to the corresponding places on the two sides of # and determine whether they match

Example (cont.)

- To keep track of which symbols have been checked already, M1 crosses off each symbol as it is examined
- Crossing off all the symbols: going to an accept state

Example (cont.)

- M_1 = “On input string *w*:
  1. Zig-zag across the tape to corresponding positions in either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
  2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*.”

Example (cont.)

- Snapshots of Turing machine M_1 computing on input 011000#011000

```
0 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ ...

x 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ ...

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔ ...

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔ ...

x x 1 0 0 0 # x 1 1 0 0 0 ⊔ ...

x x x x x x # x x x x x x ⊔ ...
                          accept
```

Turing Machine (Formal Definition)

- A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{reject})$, where Q, Σ, Γ are all finite sets and
  1. Q is the set of states,
  2. Σ is the input alphabet not containing the *blank symbol* $\sqcup$,
  3. Γ is the tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
  4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
  5. $q_0 \in Q$ is the start state,
  6. $q_{accept} \in Q$ is the accept state, and
  7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

Computation of a Turing Machine

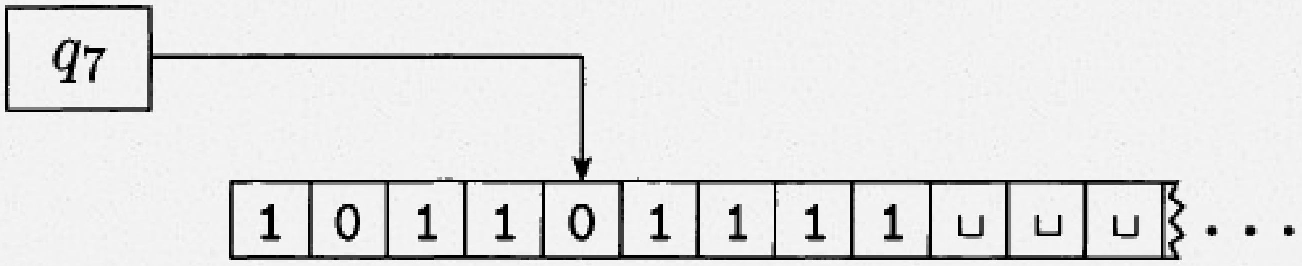
- Initially receives its input on the leftmost n squares of the tape
  - ✓ The rest is blank
- The head starts on the leftmost square of the tape
- The first blank on the tape: the end of the input

Computation of a Turing Machine (cont.)

- The computation proceeds according to the rules
  - ✓ Transition function
- If M tries to move its head to the left off the left-hand end of the tape, the head stays in the same place
- The computation continues until it enters either the accept or reject states
  - ✓ If neither occurs, it goes forever

Configuration

- The current state, the current tape contents, the current head location
  - ✓ Changes occur, as the Turing machine computes



A Turing machine with configuration $1011q_701111$

Configuration (cont.)

- The **Start Configuration** of *M* on input *w* is $\boldsymbol{q_0 w}$.
- In an **Accepting Configuration**, the state of the configuration is $\boldsymbol{q_{accept}}$.
- In a **Rejecting Configuration**, the state of the configuration is $\boldsymbol{q_{reject}}$.
- A **Halting Configuration** is either an accepting configuration or a rejecting configuration.

C1 Yields C2

- Suppose that we have a, b, and c in Γ, as well as u and v in Γ^* and states q_i and q_j. In that case uaq_ibv and uq_jacv are two configurations, Say that

$$uaq_ibv \text{ yields } uq_jacv$$

If in the transition function $\delta(q_i, b)=(q_j, c, L)$. That handles the case where the Turing machine moves leftward. For a rightward move, say that

$$uaq_ibv \text{ yields } uacq_jv$$

if $\delta(q_i, b)=(q_j, c, R)$.

Equivalent Transition Function

- The transition function could have been defined equivalently

$$\delta : Q' \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

- Q' is Q, without q_{accept} and q_{reject}
- M accepts input w, if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists:
  1. C_1 is the start configuration of M on input w,
  2. each C_i yields C_{i+1} , and
  3. C_k is an accepting configuration.

The Language of a Turing Machine

- The collection of strings that M accepts is **the language of M**, or **the language recognized by M**, denoted L(M).

Turing-recognizable language

- **Definition:** Call a language ***Turing-recognizable*** if some Turing machine recognizes it.
- Three outcomes on an input:
  - Accept
  - Reject
  - Loop
- Sometimes distinguishing a machine that is looping from one that is taking a long time, is difficult.

Turing-decidable language

- **Definition:** Call a language ***Turing-decidable*** or simply ***decidable*** if some Turing machine decides it.

- Turing machines that halt on all inputs
  - Never loop
  - Deciders

Example 1

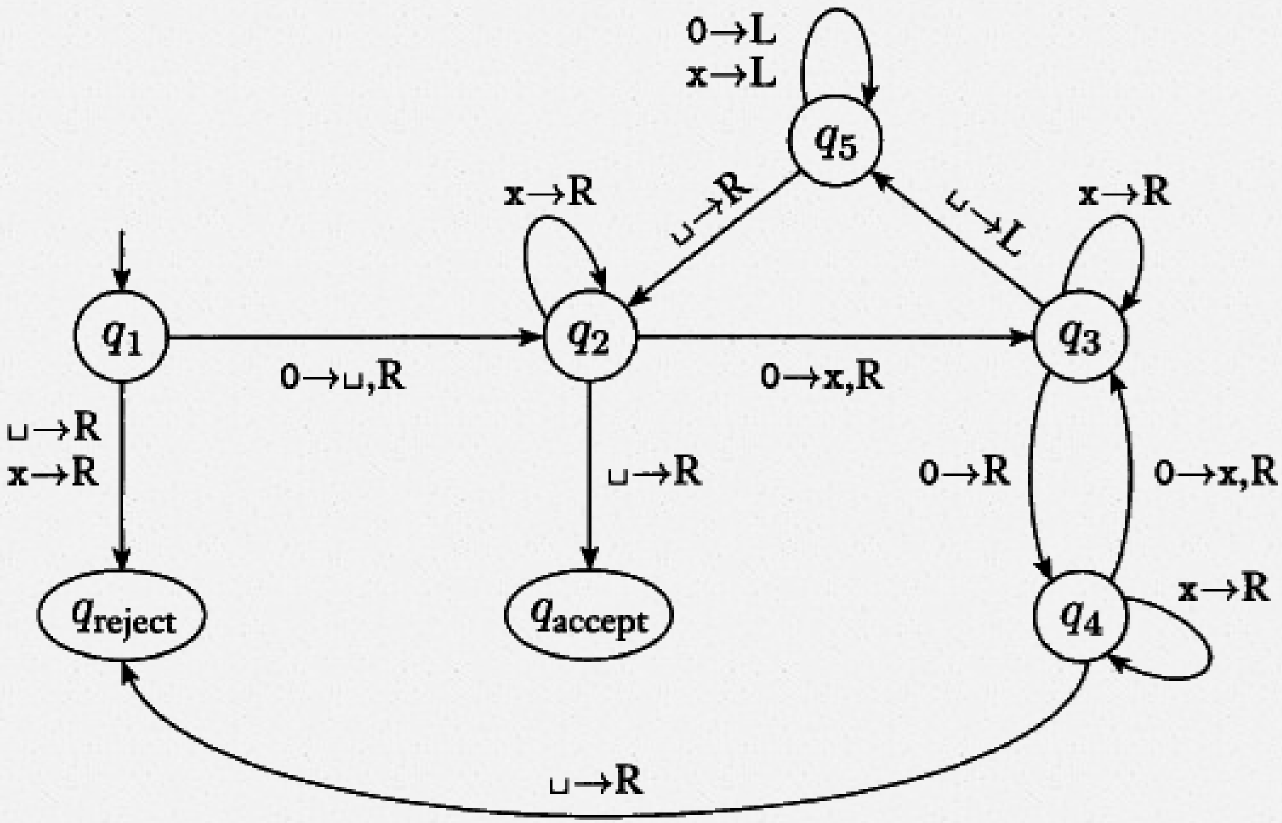
- $A = \left\{0^{2^n} \middle| n \geq 0\right\}$

- M_1 = "On input string w:
  1. Sweep left to right across the tape, crossing off every other 0.
  2. If in stage 1 the tape contained a single 0, *accept*.
  3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, *reject*.
  4. Return the head to the left-hand end of the tape.
  5. Go to stage 1."

Example 1 (formal description)

- $M_1 = (Q, \sum, \Gamma, \delta, q_1, q_{accept}, q_{reject})$
  - $Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$
  - $\sum = \{0\}$, and
  - $\Gamma = \{0, x, \sqcup\}$
  - We describe δ with a state diagram
  - The start, accept, and reject states are q_1, q_{accept}, and q_{reject}.

Example 1 (state diagram)



Example 1 (sample run on input 0000)

$q_1 0000$

$\sqcup q_2 000$

$\sqcup \mathrm{x} q_3 00$

$\sqcup \mathrm{x} 0 q_4 0$

$\sqcup \mathrm{x} 0 \mathrm{x} q_3 \sqcup$

$\sqcup \mathrm{x} 0 q_5 \mathrm{x} \sqcup$

$\sqcup \mathrm{x} q_5 0 \mathrm{x} \sqcup$

$\sqcup q_5 \mathrm{x} 0 \mathrm{x} \sqcup$

$q_5 \sqcup \mathrm{x} 0 \mathrm{x} \sqcup$

$\sqcup q_2 \mathrm{x} 0 \mathrm{x} \sqcup$

$\sqcup \mathrm{x} q_2 0 \mathrm{x} \sqcup$

$\sqcup \mathrm{xx} q_3 \mathrm{x} \sqcup$

$\sqcup \mathrm{xxx} q_3 \sqcup$

$\sqcup \mathrm{xx} q_5 \mathrm{x} \sqcup$

$\sqcup \mathrm{x} q_5 \mathrm{xx} \sqcup$

$\sqcup q_5 \mathrm{xxx} \sqcup$

$q_5 \sqcup \mathrm{xxx} \sqcup$

$\sqcup q_2 \mathrm{xxx} \sqcup$

$\sqcup \mathrm{x} q_2 \mathrm{xx} \sqcup$

$\sqcup \mathrm{xx} q_2 \mathrm{x} \sqcup$

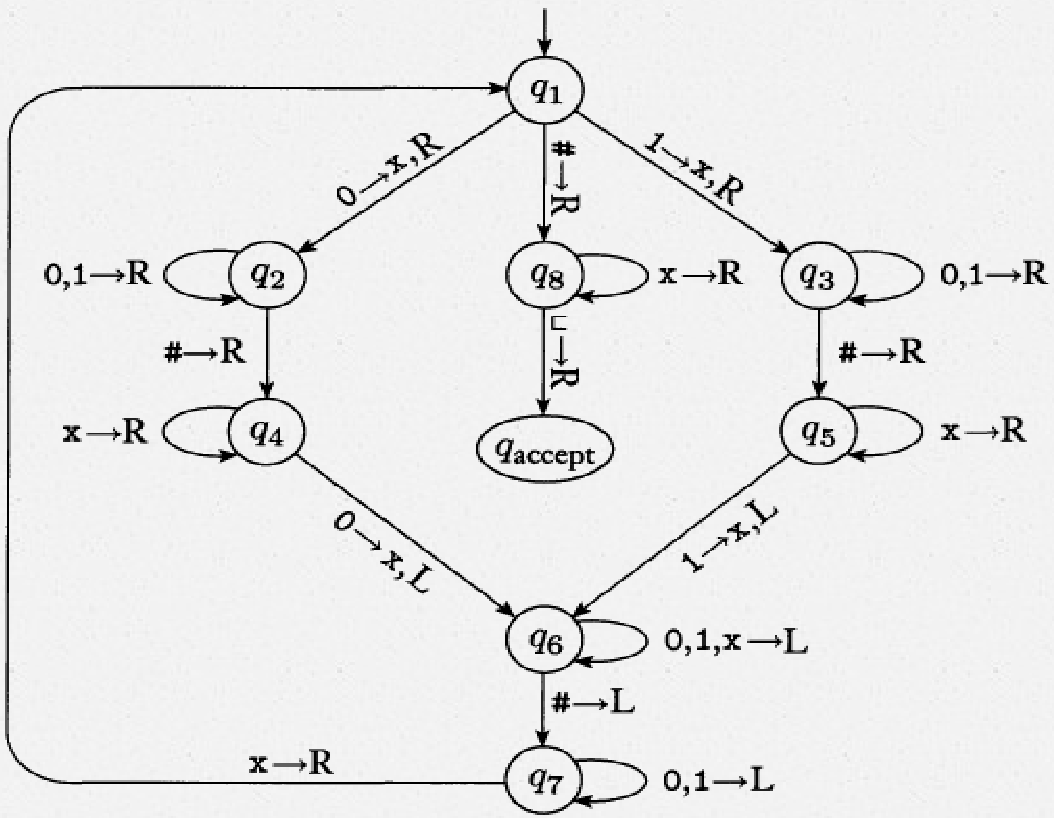
$\sqcup \mathrm{xxx} q_2 \sqcup$

$\sqcup \mathrm{xxx} \sqcup q_{\text{accept}}$

Example 2

- $B = \{w\#w \mid w \in \{0,1\}^*\}$
- $M_2 = (Q, \sum, \Gamma, \delta, q_1, q_{accept}, q_{reject})$
  - $Q = \{q_1, \dots, q_{14}, q_{accept}, q_{reject}\}$,
  - $\sum = \{0,1,\#\}$, and $\Gamma = \{0,1,\#, x, \sqcup\}$.
  - We describe δ with a state diagram
  - The start, accept, and reject states are q_1, q_{accept}, and q_{reject}.

Example 2 (state diagram)



Example 3

- $C = \left\{a^i b^j c^k \,\middle|\, i \times j = k \; and \; i, j, k \geq 1\right\}$.
- M_3 = “On input string w:
  1. Scan the input from left to right to determine whether it is a member of $a^+b^+c^+$ and *reject* if it isn’t.
  2. Return the head to the left-hand end of the tape.
  3. Cross off an *a* and scan to the right until a *b* occurs. Shuttle between the *b*’s and the *c*’s, crossing off one of each until all *b*’s are gone. If all *c*’s have been crossed off and *b*’s remain, *reject*.
  4. Restore the crossed off *b*’s and repeat stage 3 if there is another *a* to cross off. If all *a*’s have been crossed off, determine whether all *c*’s also have been crossed off. If yes, *accept*; otherwise, *reject*.”

Example 4

- $E = \{\#x_1\#x_2\# \ldots \#x_l | each\ x_i \in \{0,1\}^* \ and\ x_i \neq x_j\ for\ each\ i \neq j\}$.
- M_4 = "On input *w*:
  1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
  2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only x_1 was present, so *accept*.
  3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
  4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
  5. Go to stage 3."

Multitape Turing Machine

- Like an ordinary machine with several tapes
- Each tape has its own head
- Initially the input appears on tape 1
- Transition function:

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$$

$$k \text{ is the number of tapes}$$

Multitape Turing Machine

- The expression

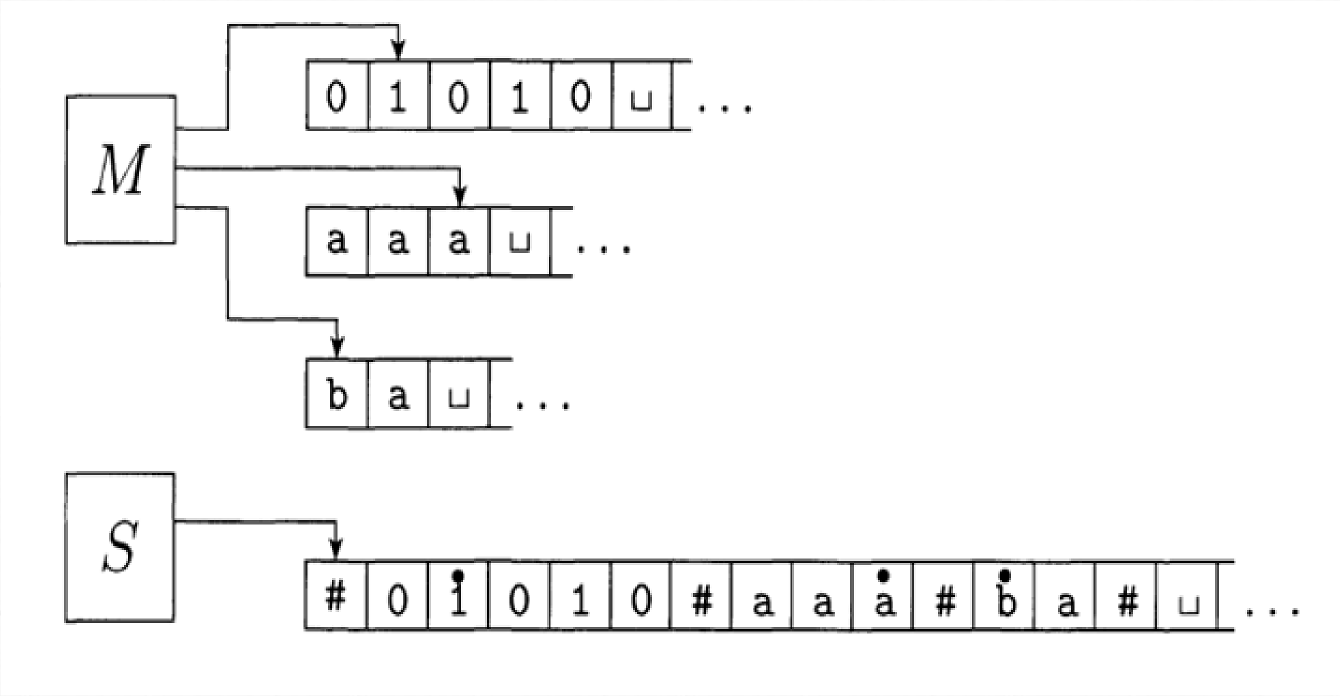
$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots b_k, L, R, \ldots, L)$$

means that, if the machine is in state q_i and heads 1 through k are reading symbols $a_1$through a_k, the machine goes to state q_j, writes symbols $b_1$through b_k, and directs each head to move left or right, or to stay put, as specified.

Theorem: Every multitape Turing machine has an equivalent single-tape Turing machine.

Example

- Representing three tapes with one



Simulation Procedure

- S = "On input $w = w_1 \ldots w_n$:
  1. First S puts its tape into the format that represents all k tapes of M. The formatted tape contains
  $$\dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$$
  2. To simulate a single move, S scans its tape from the first #, which marks the left-hand end, to the (k+1)st #, which marks the right-hand end, in order to determine the symbols under the virtual heads. Then S makes a second pass to update the tapes according to the way that M's transition function dictates.
  3. If at any point S moves one of the virtual heads to the right onto a #, this action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape. So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost #, one unit to the right. Then it continues the simulation as before."

Nondeterministic Turing Machine

- At any point in a computation the machine may proceed according to several possibilities.
- Transition function

$$\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$$

- Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Example

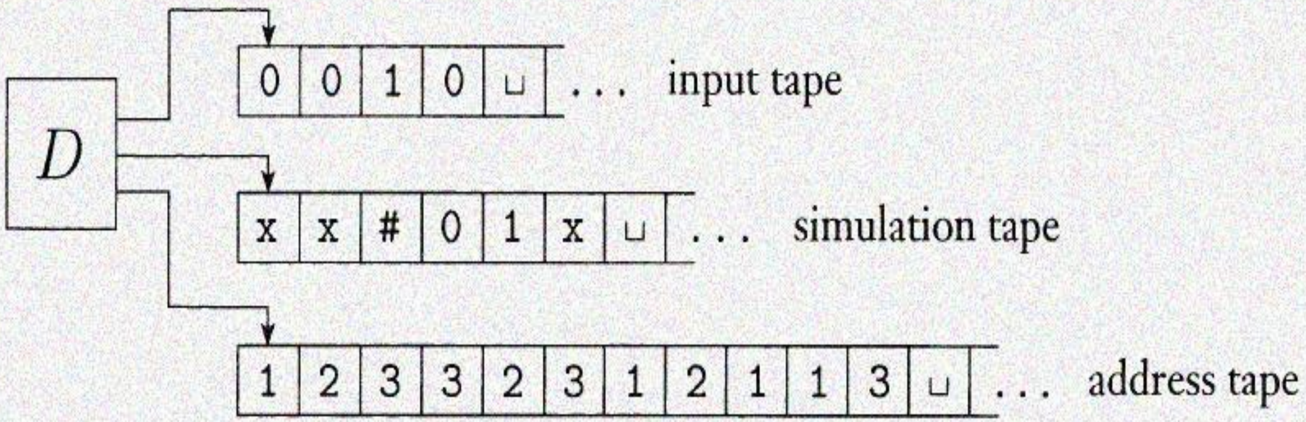


FIGURE 3.17
Deterministic TM *D* simulating nondeterministic TM *N*

Simulation Procedure

1. Initially tape 1 contains the input w, and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2.
3. Use tape 2 to simulate *N* with input w on one branch of its nondeterministic computation. Before each step of *N* consult the next symbol on tape 3 to determine which choice to make among those allowed by *N's* transition function. If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, *accept* the input.
4. Replace the string on tape 3 with the lexicographically next string. Simulate the next branch of N's computation by going to stage 2.

Theorem

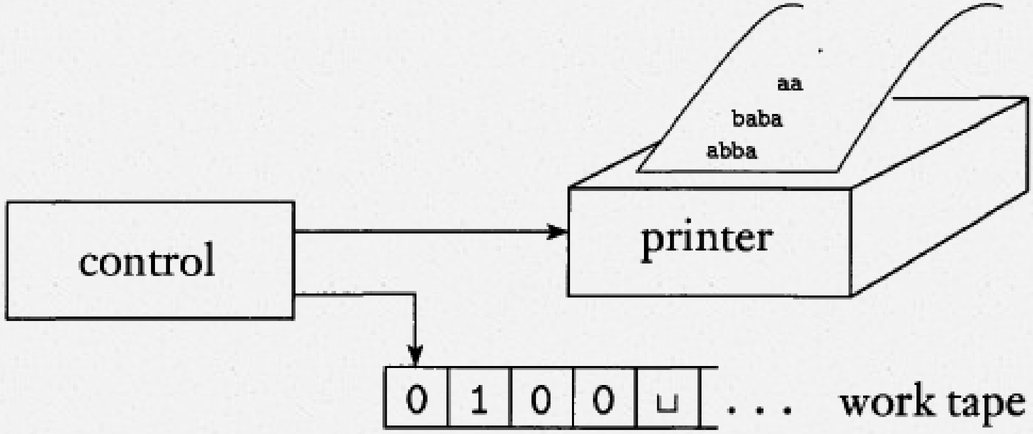
- A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.
- **PROOF:** Any deterministic TM is automatically a nondeterministic TM, and so one direction of this theorem follows immediately. The other direction follows from Theorem 3.16.

Theorem (cont.)

- We can modify the proof of Theorem 3.16 so that if N always halts on all branches of its computation, D will always halt. We call a nondeterministic Turing machine a ***decider*** if all branches halt on all inputs. Exercise 3.3 asks you to modify the proof in this way to obtain the following corollary to previous theorem.
- **COROLLARY:** A language is decidable if and only if some nondeterministic Turing machine decides it.

Enumerators

- Loosely defined: a Turing Machine with an attached printer
  - ✓ Printer: an output device
- Every time the Turing Machine wants to add a string to the list, it sends the string to the printer



Enumerators

- An enumerator E starts with a blank input tape.
- If it doesn’t halt, it may print an infinite list of strings.
- The language enumerated by E: the collection of all the strings that it eventually prints out.
- A language is Turing-recognizable **if and only if** some enumerator enumerates it.

Theorem

- A language is Turing-recognizable **if and only if** some enumerator enumerates it.
- **PROOF** First we show that if we have an enumerator *E* that enumerates a language A, a TM M recognizes A. The TM *M* works in the following way.
- *M = "On input w:*

1. Run *E*. Every time that *E* outputs a string, compare it with w.
2. If w ever appears in the output of *E*, *accept."*

Clearly, *M* accepts those strings that appear on *E's* list. Now we do the other direction. If TM *M* recognizes a language A, we can construct the following enumerator *E* for A. Say that $s_1, s_2, s_3, \ldots$ is a list of all possible strings in Σ*.

- *E* = "Ignore the input.

1. Repeat the following for i = 1, 2, 3 ....
2. Run *M* for i steps on each input, $s_1, s_2, \ldots, s_i$.
3. If any computations accept, print out the corresponding s_j".

Equivalence with Other Models

- **All** variants of Turing Machine models share the essential feature of Turing Machines: unrestricted access to unlimited memory!

Definition of Algorithm

- Informally, an algorithm is a collection of simple instructions for carrying out some task.
- Even though algorithms have had a long history, the notion of algorithm itself was not defined precisely until the twentieth century.
- The definition came in the 1936 papers of Alonzo Church and Alan Turing

Hilbert’s 23 Problems

- **Hilbert's problems** are twenty-three problems in mathematics published by German mathematician David Hilbert in 1900.
- They were all unsolved at the time, and several proved to be very influential for 20th-century mathematics.

Continuum Hypothesis

- By Cantor's Theorem, we have $2^{\aleph_a} \geq \aleph_{a+1}$ for any ordinal a.
- Do we have equality or not?
- The famous *Continuum Hypothesis* asserts that $2^{\aleph_0} = \aleph_1$.
- This was one of the problems posed in 1900 to the mathematical community by David Hilbert, to guide the development of mathematics in the twentieth century.

Continuum Hypothesis (con.)

- Godel proved $2^{\aleph_0} = \aleph_1$ cannot be disproved in ZFC.
- Thirty years later, however, Cohen, showed that $2^{\aleph_0} = \aleph_1$ cannot be proved in ZFC either. By a new technique known as *forcing*, he constructed a model of ZFC in which $2^{\aleph_0} = \aleph_2$.

Hilbert’s 1st & 10th Problems

- Paul Cohen received the Fields Medal during 1966 for his work on the first problem.
- The negative solution of the tenth problem during 1970 by Yuri Matiyasevich (completing work of Martin Davis, Hilary Putnam, and Julia Robinson) generated similar acclaim.
- Aspects of these problems are still of great interest today.

Hilbert's 10^{th} Problem

$$6.x.x.x.y.z.z = 6x^3yz^2$$

is a term with coefficient 6, and

$$6x^3yz^2 + 3xy^2 - x^3 - 10$$

is a polynomial with four terms over the variables x, y, and z. For this discussion,

- we consider only coefficients that are integers. A ***root*** of a polynomial is an assignment of values to its variables so that the value of the polynomial is 0. This polynomial has a root at $x = 5$, y = 3, and z = 0. This root is an ***integral root*** because all the variables are assigned integer values. Some polynomials have an integral root and some do not!

HILBERT’s Problem(Cont.)

- Hilbert's tenth problem was to devise an algorithm that tests whether a polynomial has an integral root. He did not use the term *algorithm* but rather “ a process according to which it can be determined by a finite number of operations.”
- Interestingly, in the way he phrased this problem, Hilbert explicitly asked that an algorithm be "devised." Thus he apparently assumed that such an algorithm must exist-someone need only find it.

Church-Turing Thesis

Intuitive notion of algorithms	equals	*Turing machine algorithms*

D = {p| p is a polynomial with an integral root}. Hilbert's tenth problem asks in essence whether the set D is decidable.
D1 = {p| p is a polynomial over x with an integral root}.
Here is a TM *M1* that recognizes *D1 :*
M1 = "The input is a polynomial p over the variable x.
1. Evaluate p with x set successively to the values 0, 1, -1, 2, -2, 3, 3.... If at any point the polynomial evaluates to 0, *accept.*"
For the multivariable case, we can present a similar TM M that recognizes D. Here M goes through all possible settings of its variables to integral values.
Both M1 and M are recognizers but not deciders

Church-Turing Thesis (Cont.)

- We can convert M1 to be a decider for D1 because we can calculate bounds within which the roots of a single variable polynomial must lie and restrict the search to these bounds. In fact, one can prove that the roots of such a polynomial must lie between the values

$$\pm k \, c_{max} / c_1$$

where k is the number of terms in the polynomial, c_{max} is coefficient with the largest absolute value, and c_1 is the coefficient of the highest order term.

Terminology for describing Turing Machines

- The input to a Turing Machine is a string.
- If we want to provide an object, rather than a string, it must be represented as a string.
- The notion for the encoding of an object O into its representation as a string is <O>
- Break the algorithm into stages
- The first line of the algorithm describes the input to the machine

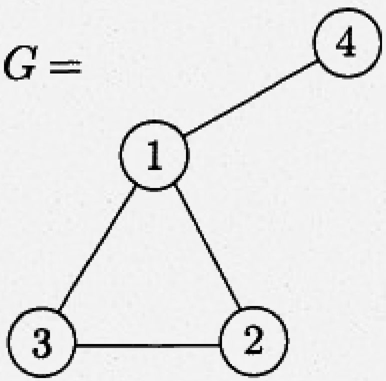
Example

- A = {⟨G⟩|G is a connected undirected graph}
- The following is a high-level description of a TM M that decides A.
- *M* = "On input (G), the encoding of a graph G:
  1. Select the first node of G and mark it.
  2. Repeat the following stage until no new nodes are marked:
  3. For each node in G, mark it if it is attached by an edge to a node that is already marked.
  4. Scan all the nodes of G to determine whether they all are marked. If they are, *accept;* otherwise, *reject.*"

Example (Cont.)

- How <G> encodes the graph G as a string

$G =$



$\langle G \rangle =$

(1,2,3,4)((1,2),(2,3),(3,1),(1,4))

- A list of the nodes of G followed by a list of the edges of G