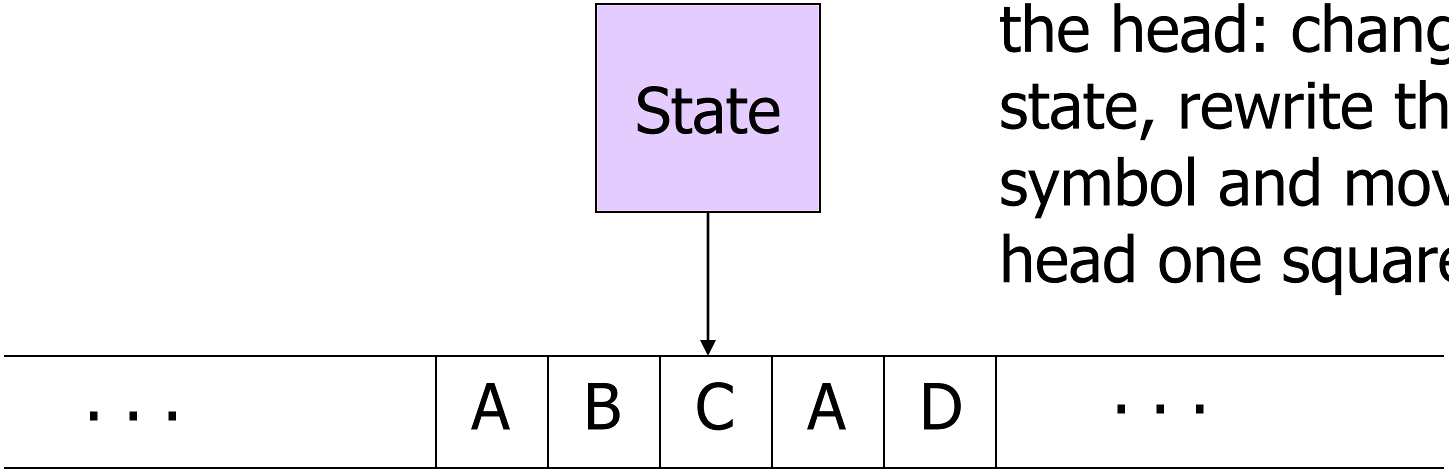
# Introduction to Turing Machines

## The Turing Machine

Informatika

# Picture of a Turing Machine



Action: based on the state and the tape symbol under the head: change state, rewrite the symbol and move the head one square.

Infinite tape with squares containing tape symbols chosen from a finite alphabet

# Turing-Machine Formalism

- **A TM is described by:**
  1. A finite set of *states* (Q, typically).
  2. An *input alphabet* ($\Sigma$, typically).
  3. A *tape alphabet* ($\Gamma$, typically; contains $\Sigma$).
  4. A *transition function* ($\delta$, typically).
  5. A *start state* ($q_0$, in Q, typically).
  6. A *blank symbol* (B, in $\Gamma - \Sigma$, typically).
     - All tape except for the input is blank initially.
  7. A set of *final states* ($F \subseteq Q$, typically).

# Conventions

- **a, b, … are input symbols.**
- **…, X, Y, Z are tape symbols.**
- **…, w, x, y, z are strings of input symbols.**
- **$\alpha$, $\beta$,… are strings of tape symbols.**

# The Transition Function

- **Takes two arguments:**
  1. A state, in Q.
  2. A tape symbol in Γ.
- **δ(q, Z) is either undefined or a triple of the form (p, Y, D).**
  - p is a state.
  - Y is the new tape symbol.
  - D is a *direction*, L or R.

# Actions of the PDA

- **If δ(q, Z) = (p, Y, D) then, in state q, scanning Z under its tape head, the TM:**

  1. Changes the state to p.
  2. Replaces Z by Y on the tape.
  3. Moves the head one square in direction D.
     - D = L: move left; D = R; move right.

# Example: Turing Machine

- **This TM scans its input right, looking for a 1.**
- **If it finds one, it changes it to a 0, goes to final state f, and halts.**
- **If it reaches a blank, it changes it to a 1 and moves left.**

# Example: Turing Machine – (2)

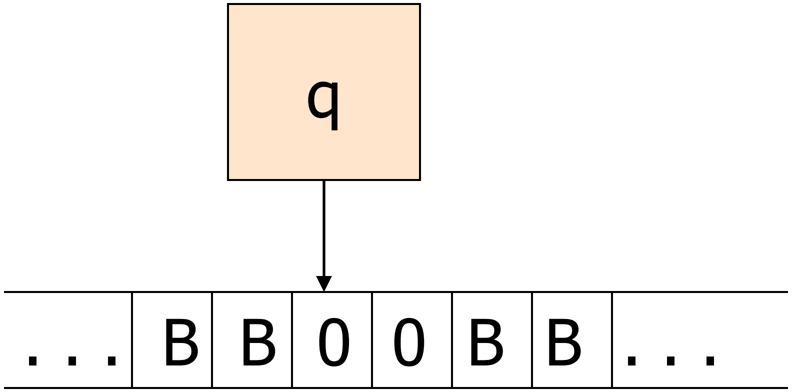
- **States = {q (start), f (final)}.**
- **Input symbols = {0, 1}.**
- **Tape symbols = {0, 1, B}.**
- **δ(q, 0) = (q, 0, R).**
- **δ(q, 1) = (f, 0, R).**
- **δ(q, B) = (q, 1, L).**

# Simulation of TM

δ(q, 0) = (q, 0, R)

δ(q, 1) = (f, 0, R)

δ(q, B) = (q, 1, L)

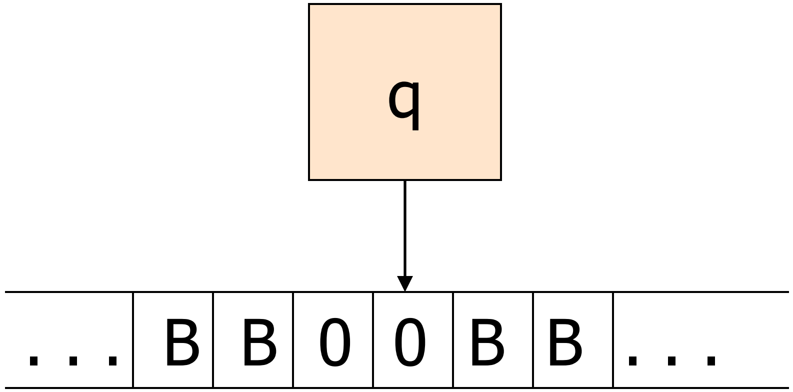


# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

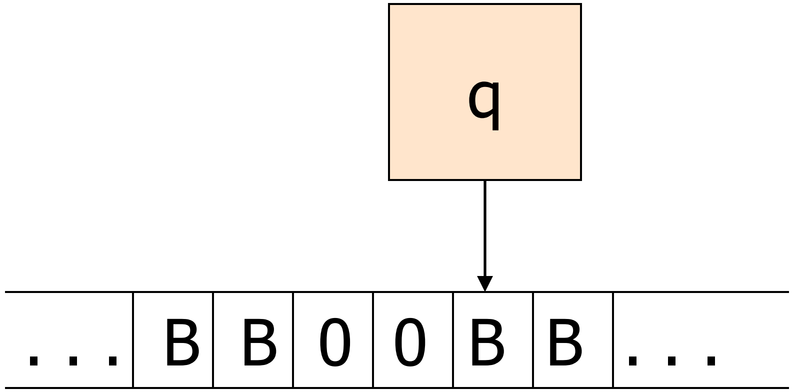


# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

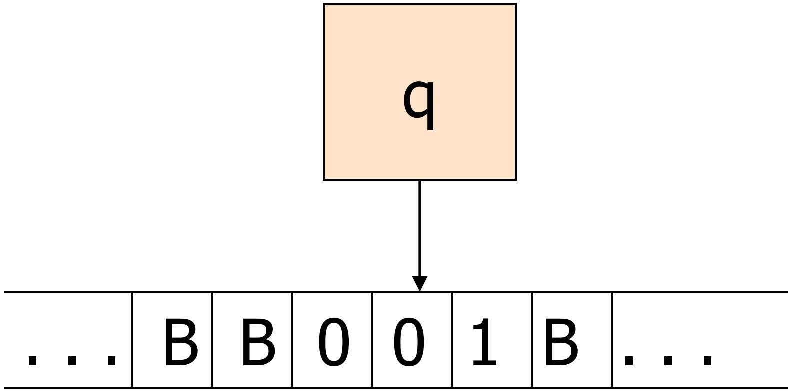


# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$



q

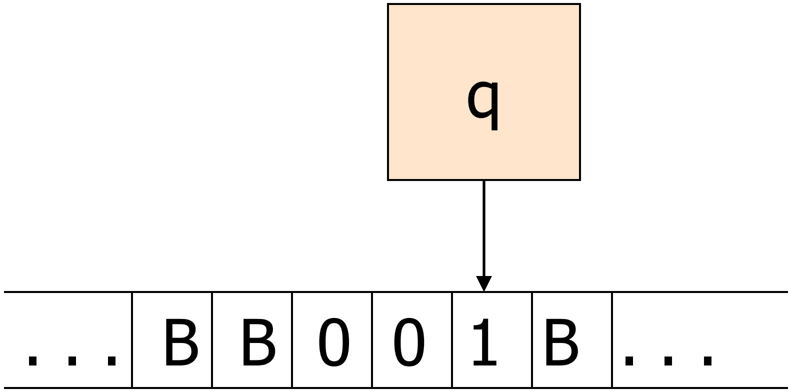
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| . . . | B | B | 0 | 0 | 1 | B | . . . |

# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

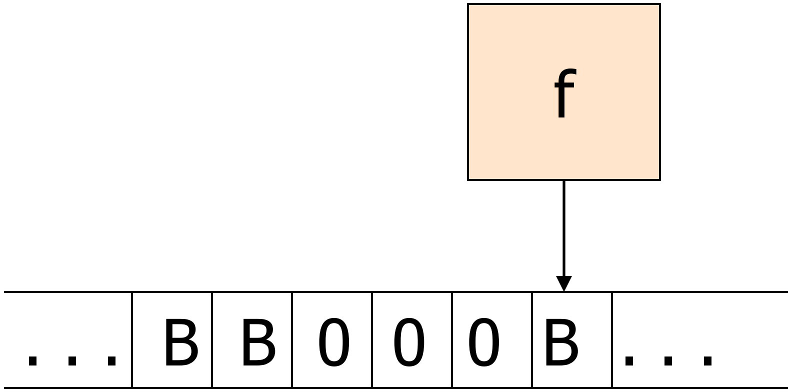


# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$



No move is possible. The TM halts and accepts.

# Instantaneous Descriptions of a Turing Machine

- Initially, a TM has a tape consisting of a string of input symbols surrounded by an infinity of blanks in both directions.
- The TM is in the start state, and the head is at the leftmost input symbol.

# TM ID's – (2)

- **An ID is a string $\alpha q\beta$, where $\alpha\beta$ is the tape between the leftmost and rightmost nonblanks (inclusive).**
- **The state q is immediately to the left of the tape symbol scanned.**
- **If q is at the right end, it is scanning B.**
  - If q is scanning a B at the left end, then consecutive B's at and to the right of q are part of $\alpha$.

# TM ID’s – (3)

- As for PDA’s we may use symbols ⊦ and ⊦* to represent “becomes in one move” and “becomes in zero or more moves,” respectively, on ID’s.
- Example: The moves of the previous TM are q00⊦0q0⊦00q⊦0q01⊦00q1⊦000f

# Formal Definition of Moves

1. **If $\delta(q, Z) = (p, Y, R)$, then**
    - $\alpha qZ\beta \vdash \alpha Yp\beta$
    - If Z is the blank B, then also $\alpha q \vdash \alpha Yp$
2. **If $\delta(q, Z) = (p, Y, L)$, then**
    - For any X, $\alpha XqZ\beta \vdash \alpha pXY\beta$
    - In addition, $qZ\beta \vdash pBY\beta$

# Languages of a TM

- A TM defines a language by final state, as usual.
- L(M) = {w | $q_0 w \vdash^* I$, where I is an ID with a final state}.
- Or, a TM can accept a language by halting.
- H(M) = {w | $q_0 w \vdash^* I$, and there is no move possible from ID I}.

# Equivalence of Accepting and Halting

1. **If L = L(M), then there is a TM M’ such that L = H(M’).**
2. **If L = H(M), then there is a TM M” such that L = L(M”).**

# Proof of 1: Acceptance -> Halting

- **Modify M to become M' as follows:**
  1. For each accepting state of M, remove any moves, so M' halts in that state.
  2. Avoid having M' accidentally halt.
     - Introduce a new state s, which runs to the right forever; that is δ(s, X) = (s, X, R) for all symbols X.
     - If q is not accepting, and δ(q, X) is undefined, let δ(q, X) = (s, X, R).

# Proof of 2: Halting -> Acceptance

- **Modify M to become M" as follows:**
  1. Introduce a new state f, the only accepting state of M".
  2. f has no moves.
  3. If δ(q, X) is undefined for any state q and symbol X, define it by δ(q, X) = (f, X, R).

# Recursively Enumerable Languages

- **We now see that the classes of languages defined by TM's using final state and halting are the same.**
- **This class of languages is called the *recursively enumerable languages*.**
  - Why? The term actually predates the Turing machine and refers to another notion of computation of functions.

# Recursive Languages

- **An *algorithm* is a TM that is guaranteed to halt whether or not it accepts.**
- **If L = L(M) for some TM M that is an algorithm, we say L is a *recursive language*.**
  - Why? Again, don't ask; it is a term with a history.

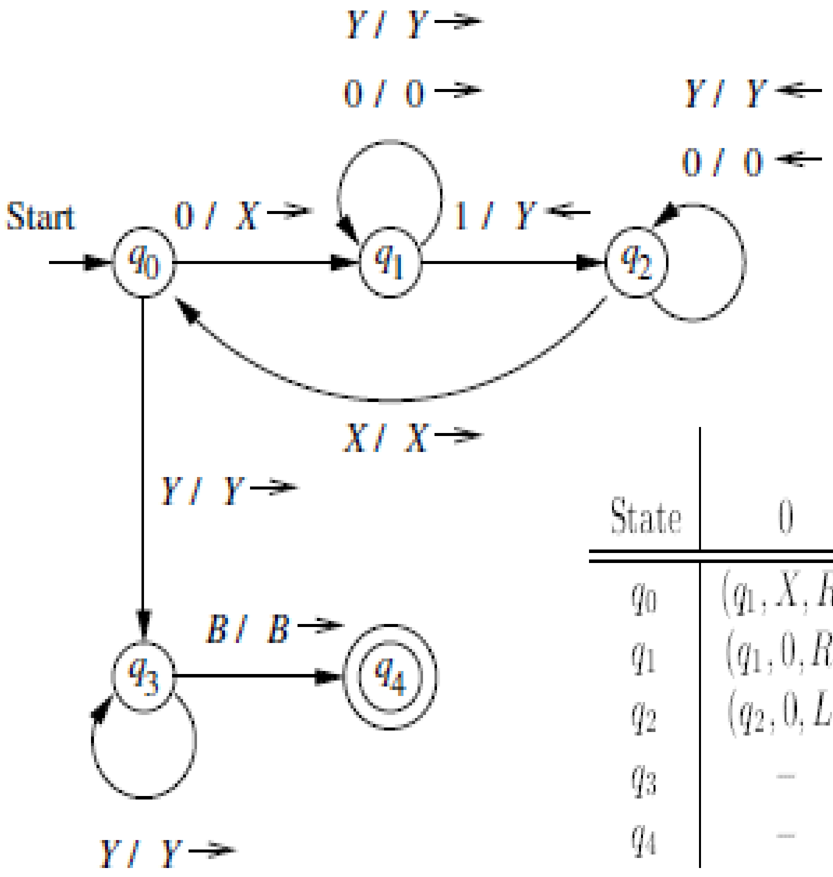
# Example: Recursive Languages

- **Every CFL is a recursive language.**
  - Use the CYK algorithm.
- **Every regular language is a CFL (think of its DFA as a PDA that ignores its stack); therefore every regular language is recursive.**
- **Almost anything you can think of is recursive.**

# Turing-Machine Formalism

- **A TM is described by:**
  1. A finite set of *states* (Q, typically).
  2. An *input alphabet* (Σ, typically).
  3. A *tape alphabet* (Γ, typically; contains Σ).
  4. A *transition function* (δ, typically).
  5. A *start state* ($q_0$, in Q, typically).
  6. A *blank symbol* (B, in Γ– Σ, typically).
     - All tape except for the input is blank initially.
  7. A set of *final states* (F ⊆ Q, typically).

**Example 8.3:** Figure 8.10 shows the transition diagram for the Turing machine of Example 8.2, whose transition function was given in Fig. 8.9. □



| State | 0 | 1 | X | Y | B |
|---|---|---|---|---|---|
| | Symbol | | | | |
| $q_0$ | $(q_1, X, R)$ | – | – | $(q_3, Y, R)$ | – |
| $q_1$ | $(q_1, 0, R)$ | $(q_2, Y, L)$ | – | $(q_1, Y, R)$ | – |
| $q_2$ | $(q_2, 0, L)$ | – | $(q_0, X, R)$ | $(q_2, Y, L)$ | – |
| $q_3$ | – | – | – | $(q_3, Y, R)$ | $(q_4, B, R)$ |
| $q_4$ | – | – | – | – | – |

Figure 8.10: Transition diagram for a TM that accepts strings of the form $0^n1^n$

# Latihan

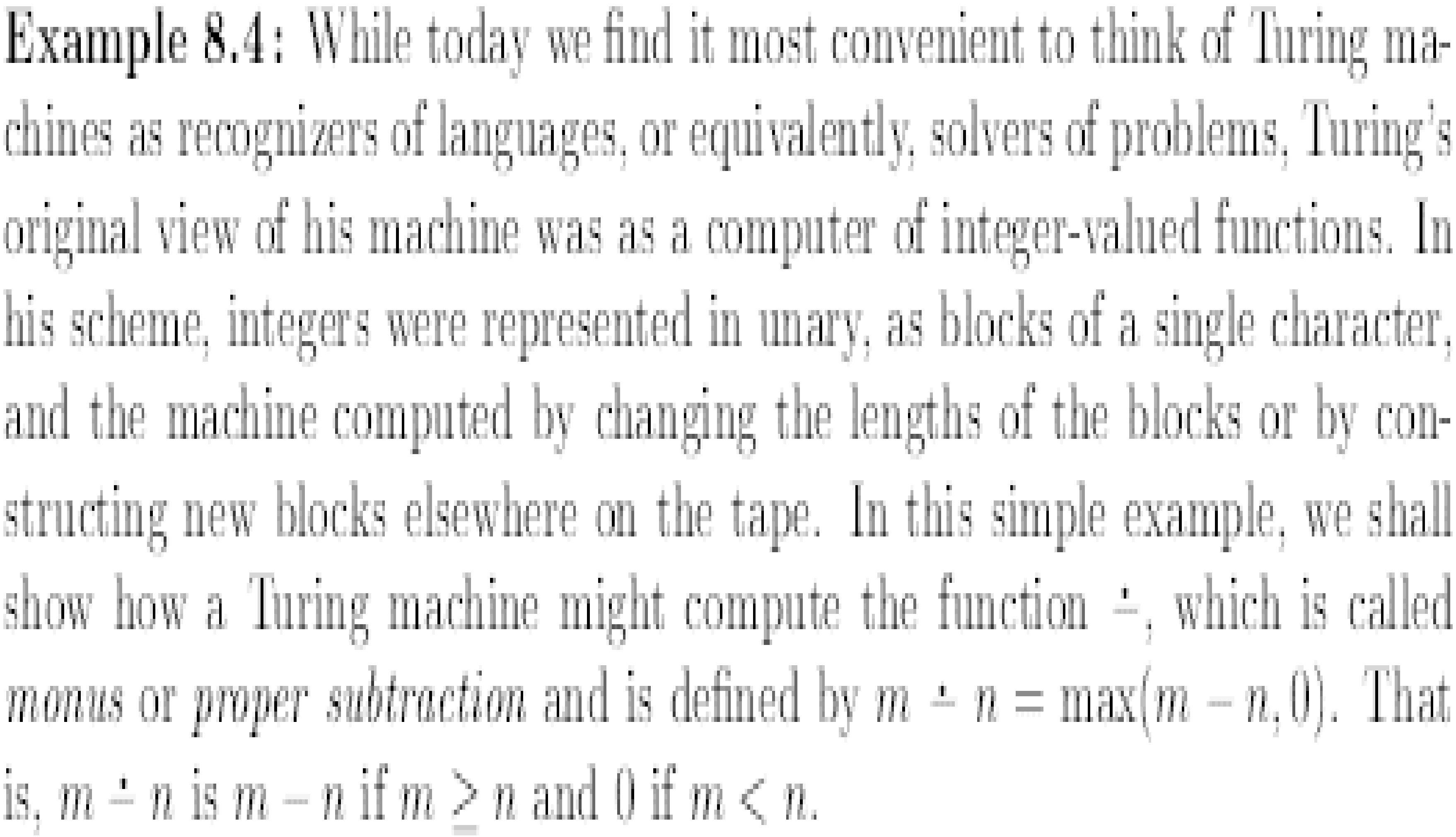
- **Buatlah sebuah Turing Machine yang bisa menerima $a^n b^n c^n$**

Here is an example of an accepting computation by $M$. Its input is 0011. Initially, $M$ is in state $q_0$, scanning the first 0, i.e., $M$'s initial ID is $q_0 0011$. The entire sequence of moves of $M$ is:

$$\begin{array}{l} q_0 0011 \vdash Xq_1 011 \vdash X0q_1 11 \vdash Xq_2 0Y1 \vdash q_2 X0Y1 \vdash \\ Xq_0 0Y1 \vdash XXq_1 Y1 \vdash XXYq_1 1 \vdash XXq_2 YY \vdash Xq_2 XYY \vdash \\ XXq_0 YY \vdash XXYq_3 Y \vdash XXYYq_3 B \vdash XXYYBq_4 B \end{array}$$

For another example, consider what $M$ does on the input 0010, which is not in the language accepted.

$$\begin{array}{l} q_0 0010 \vdash Xq_1 010 \vdash X0q_1 10 \vdash Xq_2 0Y0 \vdash q_2 X0Y0 \vdash \\ Xq_0 0Y0 \vdash XXq_1 Y0 \vdash XXYq_1 0 \vdash XXY0q_1 B \end{array}$$



**Example 8.4:** While today we find it most convenient to think of Turing machines as recognizers of languages, or equivalently, solvers of problems, Turing's original view of his machine was as a computer of integer-valued functions. In his scheme, integers were represented in unary, as blocks of a single character, and the machine computed by changing the lengths of the blocks or by constructing new blocks elsewhere on the tape. In this simple example, we shall show how a Turing machine might compute the function $\dot{-}$, which is called *monus* or *proper subtraction* and is defined by $m \dot{-} n = \max(m - n, 0)$. That is, $m \dot{-} n$ is $m - n$ if $m \geq n$ and 0 if $m < n$.

A TM that performs this operation is specified by

$$M = (\{q_0, q_1, \ldots, q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B)$$

Note that, since this TM is not used to accept inputs, we have omitted the seventh component, which is the set of accepting states. $M$ will start with a tape consisting of $0^m 1 0^n$ surrounded by blanks. $M$ halts with $0^{m \dot{-} n}$ on its tape, surrounded by blanks.

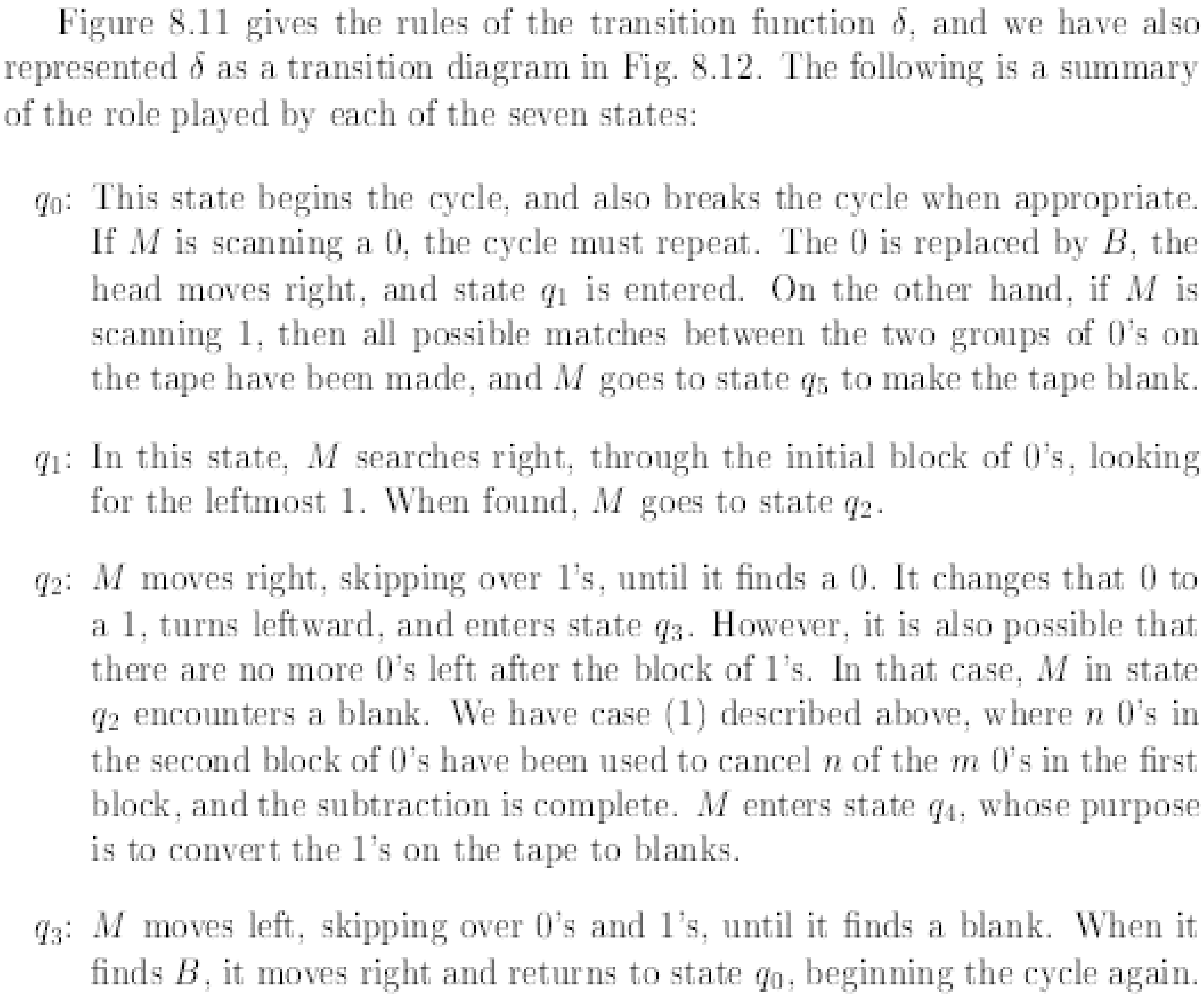


Figure 8.11 gives the rules of the transition function $\delta$, and we have also represented $\delta$ as a transition diagram in Fig. 8.12. The following is a summary of the role played by each of the seven states:

$q_0$: This state begins the cycle, and also breaks the cycle when appropriate. If $M$ is scanning a 0, the cycle must repeat. The 0 is replaced by $B$, the head moves right, and state $q_1$ is entered. On the other hand, if $M$ is scanning 1, then all possible matches between the two groups of 0's on the tape have been made, and $M$ goes to state $q_5$ to make the tape blank.

$q_1$: In this state, $M$ searches right, through the initial block of 0's, looking for the leftmost 1. When found, $M$ goes to state $q_2$.

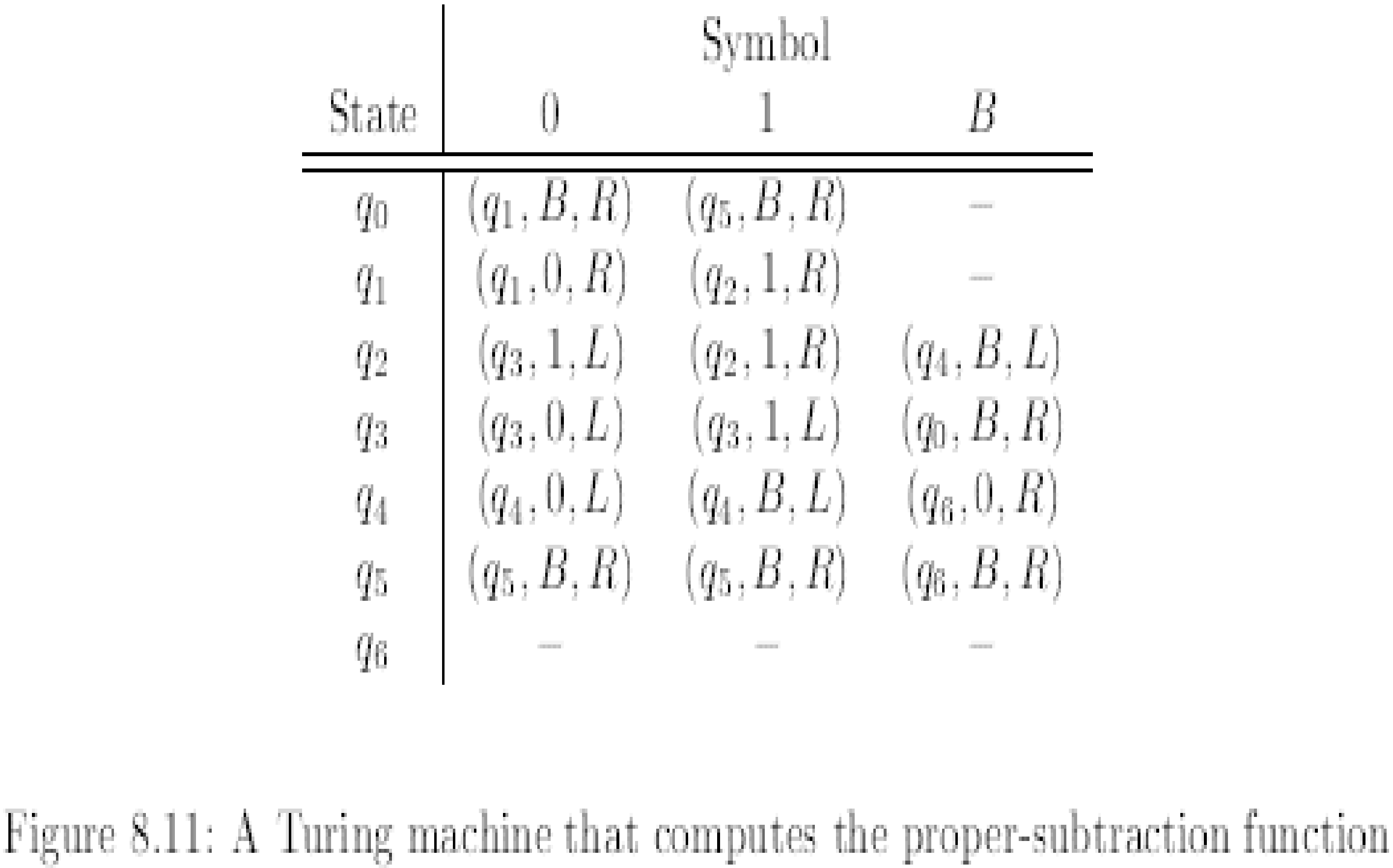
$q_2$: $M$ moves right, skipping over 1's, until it finds a 0. It changes that 0 to a 1, turns leftward, and enters state $q_3$. However, it is also possible that there are no more 0's left after the block of 1's. In that case, $M$ in state $q_2$ encounters a blank. We have case (1) described above, where $n$ 0's in the second block of 0's have been used to cancel $n$ of the $m$ 0's in the first block, and the subtraction is complete. $M$ enters state $q_4$, whose purpose is to convert the 1's on the tape to blanks.

$q_3$: $M$ moves left, skipping over 0's and 1's, until it finds a blank. When it finds $B$, it moves right and returns to state $q_0$, beginning the cycle again.

$q_4$: Here, the subtraction is complete, but one unmatched 0 in the first block was incorrectly changed to a $B$. $M$ therefore moves left, changing 1's to $B$'s, until it encounters a $B$ on the tape. It changes that $B$ back to 0, and enters state $q_6$, wherein $M$ halts.

$q_5$: State $q_5$ is entered from $q_0$ when it is found that all 0's in the first block have been changed to $B$. In this case, described in (2) above, the result of the proper subtraction is 0. $M$ changes all remaining 0's and 1's to $B$ and enters state $q_6$.

$q_6$: The sole purpose of this state is to allow $M$ to halt when it has finished its task. If the subtraction had been a subroutine of some more complex function, then $q_6$ would initiate the next step of that larger computation.



| | Symbol | | |
|---|---|---|---|
| State | $0$ | $1$ | $B$ |
| $q_0$ | $(q_1, B, R)$ | $(q_5, B, R)$ | $-$ |
| $q_1$ | $(q_1, 0, R)$ | $(q_2, 1, R)$ | $-$ |
| $q_2$ | $(q_3, 1, L)$ | $(q_2, 1, R)$ | $(q_4, B, L)$ |
| $q_3$ | $(q_3, 0, L)$ | $(q_3, 1, L)$ | $(q_0, B, R)$ |
| $q_4$ | $(q_4, 0, L)$ | $(q_4, B, L)$ | $(q_6, 0, R)$ |
| $q_5$ | $(q_5, B, R)$ | $(q_5, B, R)$ | $(q_6, B, R)$ |
| $q_6$ | $-$ | $-$ | $-$ |

Figure 8.11: A Turing machine that computes the proper-subtraction function

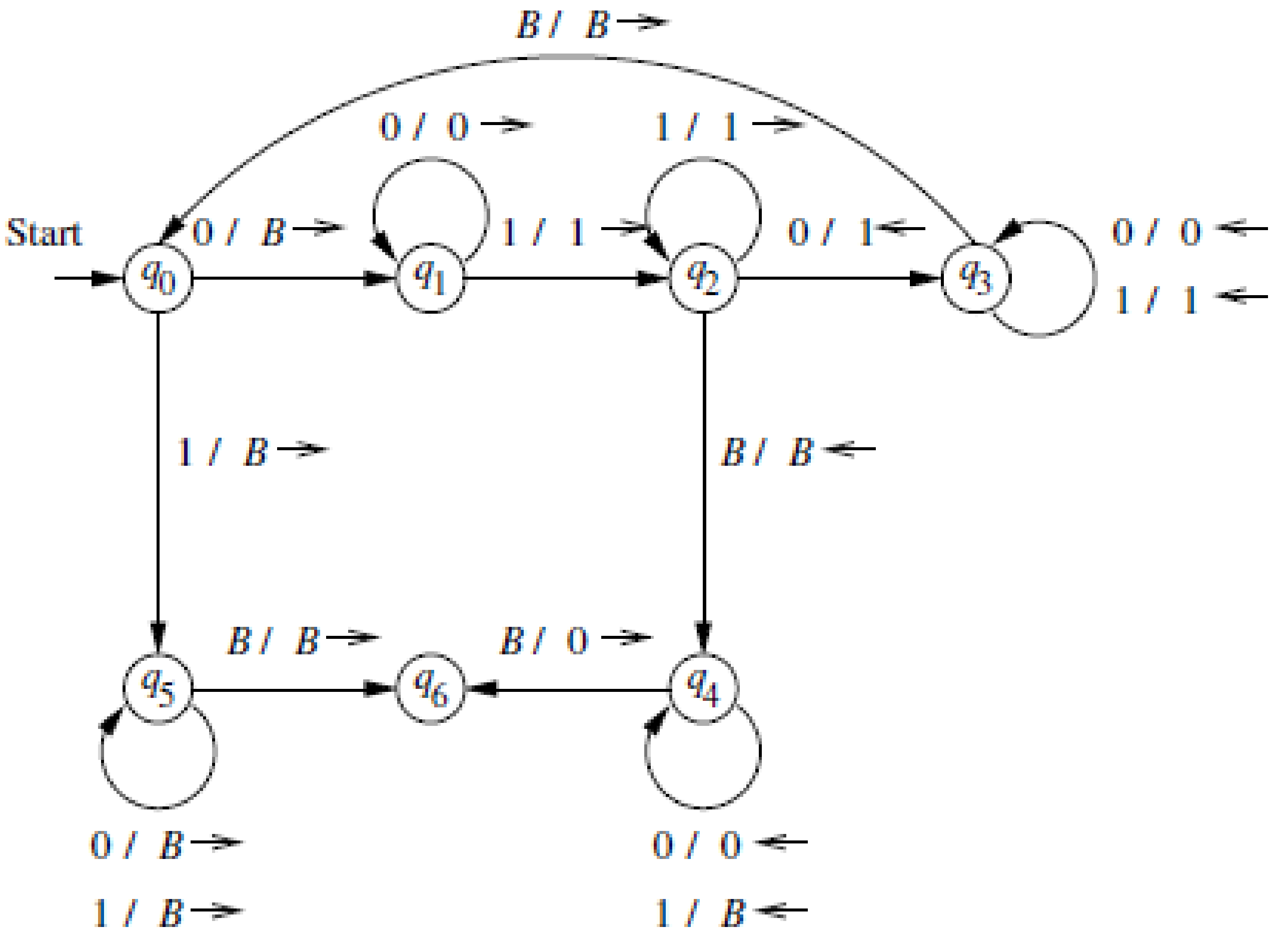
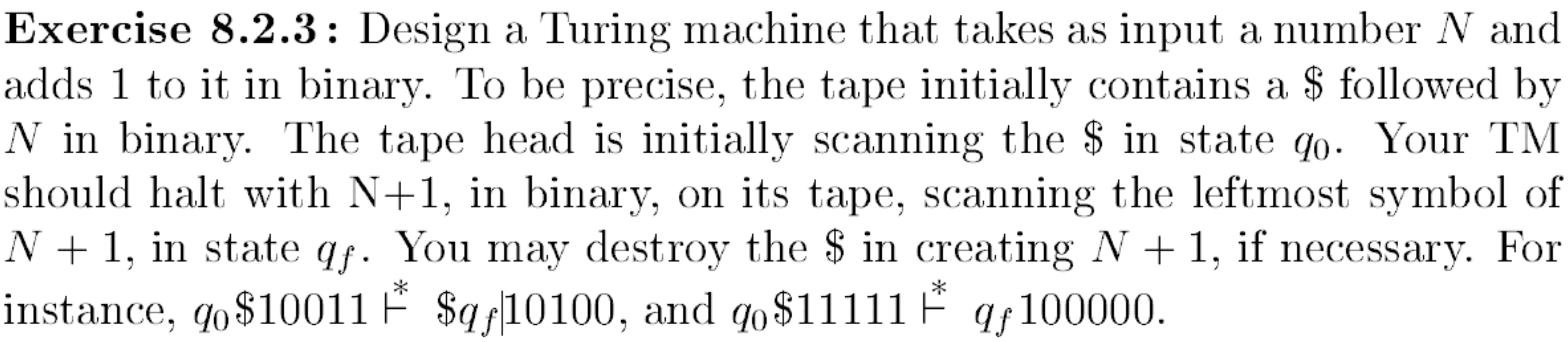


Figure 8.12: Transition diagram for the TM of Example 8.4



**Exercise 8.2.3 :** Design a Turing machine that takes as input a number $N$ and adds 1 to it in binary. To be precise, the tape initially contains a \$ followed by $N$ in binary. The tape head is initially scanning the \$ in state $q_0$. Your TM should halt with N+1, in binary, on its tape, scanning the leftmost symbol of $N+1$, in state $q_f$. You may destroy the \$ in creating $N+1$, if necessary. For instance, $q_0\$10011 \overset{*}{\vdash} \$q_f|10100$, and $q_0\$11111 \overset{*}{\vdash} q_f100000$.