THEORY OF FORMAL LANGUAGES

Turing Machine and Computability

Lecture 3

Prof. K. R. Chowdhary • Professor of Computer Science

A Turing Machine as a Function Computer

Running $M = (Q, \Sigma, \Gamma, \delta, q_0, H)$ on input $w \in \Gamma^*$ produces a configuration sequence $C_0, C_1, \ldots, C_m$. If M halts successfully in C_m, it has computed a result: $f_m(w)$.

Total on w

If $f_m(w)$ is defined for every w, M always halts with an answer.

Partial function

If $f_m(w)$ is undefined for some w (M fails or loops), f_m is a partial function $\Sigma^* \to \Gamma^*$.

Turing Computable

$f : \Sigma^* \to \Gamma^*$ is Turing Computable iff some TM M exists with $f = f_m$.

Since M is deterministic, $w = x$ always gives $f_m(w) = f_m(x)$.

DEFINITION 3.2

The Church–Turing Thesis

> *"A function can be computed by an algorithm if and only if it can be computed by a Turing machine."*

ENCODING INTEGERS ON THE TAPE

A TM computes functions $f : \mathbb{N}_0 \to \mathbb{N}_0$ by representing each integer $i \geq 0$ in unary as the string 1^{i+1}.

$$1^{i_1} 0\, 1^{i_2} 0 \ldots 0\, 1^{i_{k-1}} 0\, 1^{i_k}$$

For k arguments $i_1, \ldots, i_k$, each is written in unary and separated by single 0's. If M halts in state q' with ID $q'1^{m+1}$, then $f(i_1,\ldots,i_k) = m$.

Total vs. Partial Recursive Functions

Total Recursive Function

↔ **Recursive Language**

Every k-tuple of arguments has a defined value. The corresponding TM always halts. Examples: $\lfloor \log n \rfloor$, n^1, 2^n.

Partial Recursive Function

↔ **Recursively Enumerable**

Only some argument tuples have values. The corresponding TM may not halt on every input.

This mirrors the earlier distinction between deciding and accepting a language — a TM computing a total function decides membership, while one computing a partial function may only accept.

EXAMPLE 3.3

Computing x + y in Unary

Represent x and y in unary: $w(x), w(y) \in \{1\}^+$, with $|w(x)| = x$ and $|w(y)| = y$, separated by a single 0 on the tape.

$$q_0 w(x) 0 w(y) \vdash^* q' w(x+y) B$$

The operation: slide the separating 0 to the end and replace it with B, concatenating w(x) and w(y). One extra 1 appears in the process and must be deleted:

$$1^{m+1} . 1^{n+1} \rightarrow 1^{m+n+1} \quad (\text{not } 1^{m+n+2})$$

Q $\{q_0, \ldots, q_7\}, H = \{q_7\}$

Σ $\{0, 1\}$

Γ $\{0, 1, B\}$, leftmostsymbol B

TABLE 3.1

Transition Function for Unary Addition

state	0	1	B
q_0	—	$(q_1, 1, R)$	—
q_1	$(q_2, 1, R)$	$(q_1, 1, R)$	—
q_2	—	$(q_3, 1, R)$	—
q_3	—	$(q_3, 1, R)$	(q_4, B, L)
q_4	—	$(q_5, 1, L)$	—
q_5	—	(q_6, B, L)	—
q_6	—	$(q_6, 1, L)$	(q_7, B, R)
q_7	—	—	—

The three self-loops (q_1, q_3, q_6) consume the runs of 1's in x, y, and the combined result.

Transition Diagram for Unary Addition

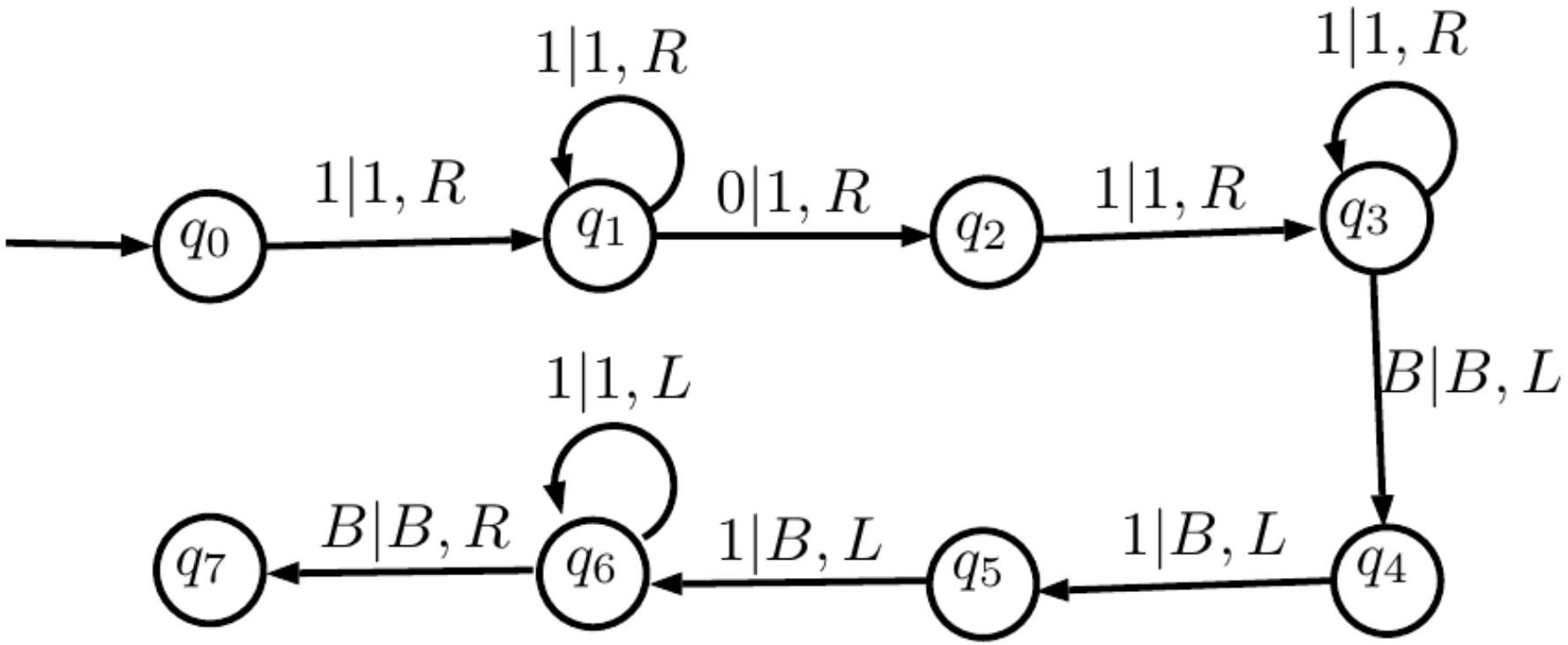


Figure 3.1 — TM to add two integers in unary format

WORKED TRACE

Computing 2 + 3 = 5

w(2) = 111, w(3) = 1111. Initial tape: B q_0 111 0 1111.

$Bq_0111\ 01111$	$\vdash B11111q_3111B$	$\vdash B111111q_51BB$	$\vdash B1q_611111BBB$
$\vdash B1q_1110\ 1111B$	$\vdash B111111q_311B$	$\vdash B11111q_61BBB$	$\vdash Bq_6111111BBB$
$\vdash B11q_1100\ 1111B$	$\vdash B1111111q_31B$	$\vdash B1111q_611BBB$	$\vdash qB\ 111111BBB$
$\vdash B111q_1001\ 111B$	$\vdash B11111111q_3B$	$\vdash B111q_6111BBB$	$\vdash Bq_7111111BBB$
$\vdash B1111q_2111\ 1B$	$\vdash B1111111q_41B$	$\vdash B11q_61111BBB$	

Final: $Bq_7111111BBB$ — five 1's remain: wt(5) = 11111 = 1^5.

EXAMPLE 3.4

Counting the Steps for m + n

$q_0 \to q_1$	1	$q_3 \to q_4$	1
$q_1 \to q_1$	$m - 1$	$q_4 \to q_5$	1
$q_1 \to q_2$	1	$q_5 \to q_6$	1
$q_2 \to q_3$	1	$q_6 \to q_6$	$m + n + 1$
$q_3 \to q_3$	$n - 1$	$q_6 \to q_7$	1

Total steps = $2(m + n + 3)$ → $O(n)$ for input $|w| = n$

EXAMPLE 3.5

Subtraction Is Also Turing Computable

$$f(x, y) = x - y$$

- Represent x and y as unary numbers on the tape, as with addition.
- Find the beginning of y; for each 1 found in y, go back to the start of x and delete one 1.
- For the subtraction to be valid, x must be greater than y.

The construction mirrors the addition machine, but each matched pair of 1's is deleted rather than merged — a similar marking strategy to earlier examples.

EXAMPLE 3.6

Incrementing a Binary Number by 1

The number is stored in reverse on the tape (LSB first). 13 = 1101 is stored as 1011, with initial configuration $Bq_0 1011B$.

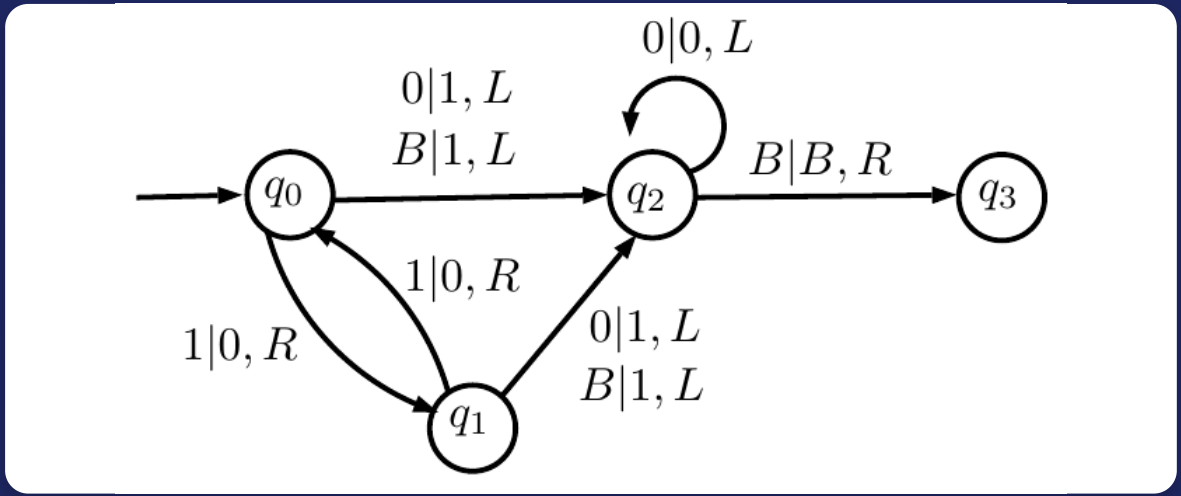
1. Moving right, every 1 is flipped to 0. When a 0 is found, flip it to 1 and return to the start without further changes.

2. If every bit was 1, they all become 0; the trailing B becomes 1, and the head returns to the start.

3. If the tape held only a B (empty number), it becomes 1 and the head returns to the first bit.

Examples: 101110 → 101111 • 110111 → 111000 • 111111 → 1000000

Increment TM: Diagram & Table



state	0	1	B
q_0	$(q_2, 1, L)$	$(q_1, 0, R)$	$(q_2, 1, L)$
q_1	$(q_2, 1, L)$	$(q_0, 0, R)$	$(q_2, 1, L)$
q_2	$(q_2, 0, L)$	—	(q_3, B, R)
q_3	—	—	—

Figure 3.2 & Table 3.2 — Turing Machine to increment a binary number

Carrying works right to left: consecutive 1's become 0's (the carry propagates), and the first 0 encountered flips to 1, absorbing the carry — exactly like adding 1 by hand.

EXAMPLE 3.7

Concatenation of Two Strings

Compute the binary function of concatenation over {a,b}*. Initial configuration q_0BuBvB; final configuration q_fBuvB. Either or both of u, v may be null.

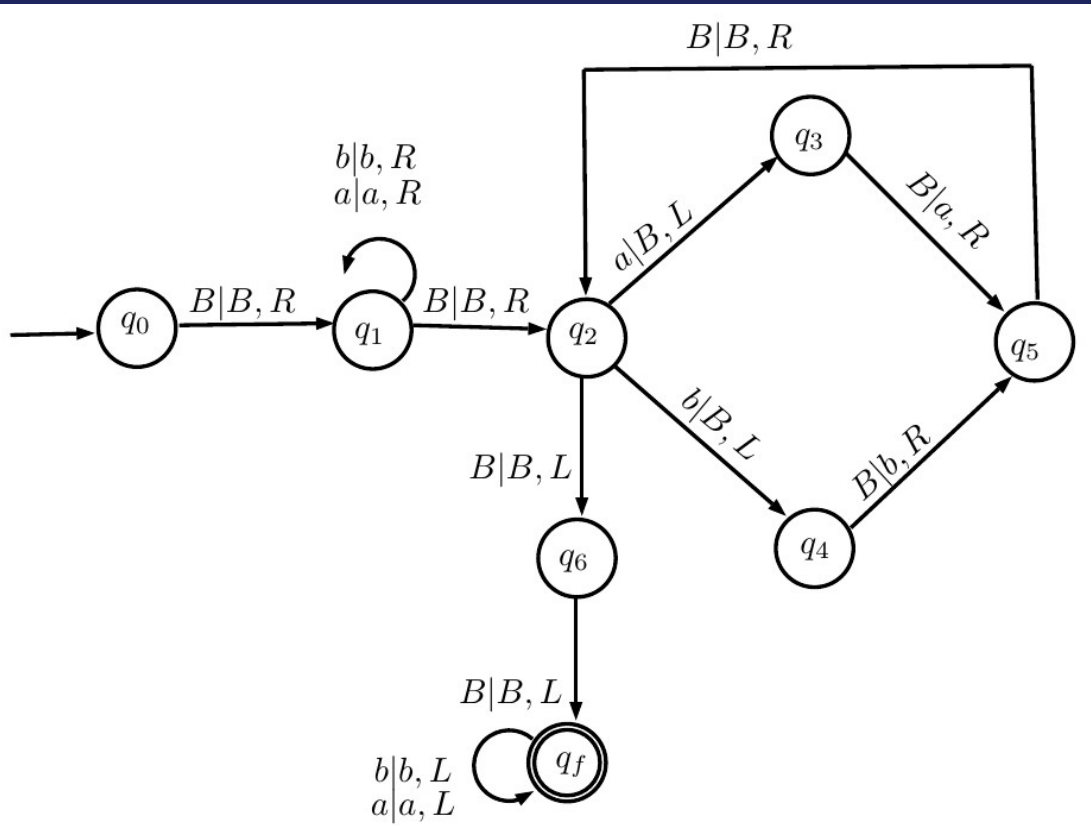


Figure 3.3 — TM to concatenate two strings

Key Takeaways

1. A TM is Turing Computable for f if $f = f_m$ for some machine M; total functions correspond to Recursive languages, partial ones to Recursively Enumerable languages.

2. The Church–Turing Thesis: algorithmic computability and Turing computability are the same notion.

3. Unary addition takes O(n) steps and total 2(m+n+3) elementary instructions; a similar marking strategy computes subtraction.

4. Binary increment and string concatenation show TMs can manipulate structured data, not just accept or reject languages.