Techniques for Turing Machine Construction

Techniques

Storage in the finite control

The finite control can be used to hold a finite amount of information. To do so, the state is written as a pair of elements, one exercising control and the other storing a symbol. It should be emphasized that this arrangement is for conceptual purposes only. No modification in the definition of the Turing machine has been made.

Example 7.3 Consider a Turing machine M that looks at the first input symbol, records it in its finite control, and checks that the symbol does not appear elsewhere on its input. Note that M accepts a regular set, but M will serve for demonstration purposes:

$$M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, [q_0, B], B, F),$$

where Q is $\{q_0, q_1\} \times \{0, 1, B\}$. That is, Q consists of the pairs $[q_0, 0]$, $[q_0, 1]$, $[q_0, B]$, $[q_1, 0]$, $[q_1, 1]$, and $[q_1, B]$. The set F is $\{[q_1, B]\}$. The intention is that the first component of the state controls the action, while the second component "remembers" a symbol.

We define δ as follows.

1) a) $\delta([q_0, B], 0) = ([q_1, 0], 0, R)$. b) $\delta([q_0, B], 1) = ([q_1, 1], 1, R)$.
Initially, q_0 is the control component of the state, and M moves right. The first component of M's state becomes q_1, and the first symbol seen is stored in the second component.

2) a) $\delta([q_1, 0], 1) = ([q_1, 0], 1, R)$. b) $\delta([q_1, 1], 0) = ([q_1, 1], 0, R)$.
If M has a 0 stored and sees a 1 or vice versa, then M continues to move to the right.

3) a) $\delta([q_1, 0], B) = ([q_1, B], 0, L)$, b) $\delta([q_1, 1], B) = ([q_1, B], 0, L)$.
M enters the final state $[q_1, B]$ if it reaches a blank symbol without having first encountered a second copy of the leftmost symbol.

If M reaches a blank in state $[q_1, 0]$, or $[q_1, 1]$, it accepts. For state $[q_1, 0]$ and symbol 0 or for state $[q_1, 1]$ and symbol 1, δ is not defined. Thus if M encounters the tape symbol stored in its state, M halts without accepting.

In general, we can allow the finite control to have k components, all but one of which store information.

Multiple Tracks

We can imagine that the tape of the Turing machine is divided into k tracks, for any finite k. This arrangement is shown in Fig. 7.4, with $k = 3$. The symbols on the tape are considered k-tuples, one component for each track.

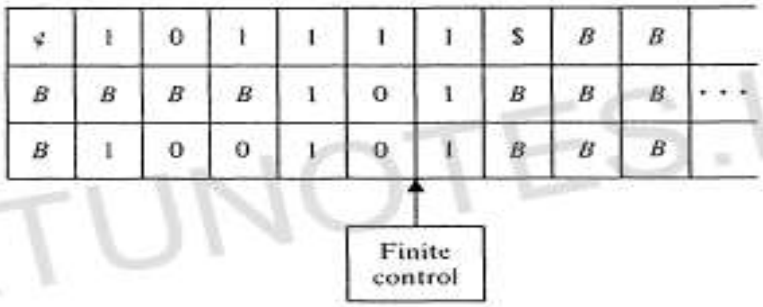

Fig. 7.4 A three-track Turing machine.

Example 7.4 The tape in Fig. 7.4 belongs to a Turing machine that takes a binary input greater than 2, written on the first track, and determines whether it is a prime. The input is surrounded by ¢ and $ on the first track. Thus, the allowable input symbols are [¢, *B*, *B*], [0, *B*, *B*], [1, *B*, *B*], and [$, *B*, *B*]. These symbols can be identified with ¢, 0, 1, and $, respectively, when viewed as input symbols. The blank symbol can be identified with [*B*, *B*, *B*].

To test if its input is a prime, the TM first writes the number two in binary on the second track and copies the first track onto the third. Then the second track is subtracted, as many times as possible, from the third track, effectively dividing the third track by the second and leaving the remainder.

If the remainder is zero, the number on the first track is not a prime. If the remainder is nonzero, the number on the second track is increased by one. If the second track equals the first, the number on the first track is a prime, because it cannot be divided by any number lying properly between one and itself. If the second is less than the first, the whole operation is repeated for the new number on the second track.

Checking off Symbols

Checking off symbols is a useful trick for visualizing how a TM recognizes languages defined by repeated strings, such as

$$\{ww \mid w \text{ in } \Sigma^*\}, \qquad \{wcy \mid w \text{ and } y \text{ in } \Sigma^*,\ w \neq y\} \qquad \text{or} \qquad \{ww^R \mid w \text{ in } \Sigma^*\}.$$

It is also useful when lengths of substrings must be compared, such as in the languages

$$\{a^i b^i \mid i \geq 1\} \qquad \text{or} \qquad \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}.$$

We introduce an extra track on the tape that holds a blank or $\surd$. The $\surd$ appears when the symbol below it has been considered by the TM in one of its comparisons.

Example 7.5 Consider a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, which recognizes the language $\{wcw \mid w \text{ in } (\mathbf{a} + \mathbf{b})^+\}$. Let

$$Q = \{[q, d] \mid q = q_1, q_2, \ldots, q_9 \quad \text{and} \quad d = a, b, \text{ or } B\}.$$

The second component of the state is used to store an input symbol,

$$\Sigma = \{[B, d] \mid d = a, b, \text{ or } c\}.$$

The input symbol $[B, d]$ is identified with d. Remember that the two "tracks" are just conceptual tools; that is, $[B, d]$ is just another "name" for d:

$$\Gamma = \{[X, d] \mid X = B \text{ or } \surd \quad \text{and} \quad d = a, b, c, \text{ or } B\},$$

$$q_0 = [q_1, B], \quad \text{and} \quad F = \{[q_9, B]\};$$

$[B, B]$ is identified with B, the blank symbol. For $d = a$ or b and $e = a$ or b we define δ as follows.

1) $\delta([q_1, B], [B, d]) = ([q_2, d], [\surd, d], R)$.
   M checks the symbol scanned on the tape, stores the symbol in the finite control, and moves right.
2) $\delta([q_2, d], [B, e]) = ([q_2, d], [B, e], R)$.
   M continues to move right, over unchecked symbols, looking for c.
3) $\delta([q_2, d], [B, c]) = ([q_3, d], [B, c], R)$.
   On finding c, M enters a state with first component q_3.

4) $\delta([q_3, d], [\surd, e]) = ([q_3, d], [\surd, e], R)$.
M moves right over checked symbols.

5) $\delta([q_3, d], [B, d]) = ([q_4, B], [\surd, d], L)$.
M encounters an unchecked symbol. If the unchecked symbol matches the symbol stored in the finite control, M checks it and begins moving left. If the symbols disagree, M has no next move and so halts without accepting. M also halts if in state q_3 it reaches $[B, B]$ before finding an unchecked symbol.

6) $\delta([q_4, B], [\surd, d]) = ([q_4, B], [\surd, d], L)$.
M moves left over checked symbols.

7) $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$.
M encounters the symbol c.

8) $\delta([q_5, B], [B, d]) = ([q_6, B], [B, d], L)$.
If the symbol immediately to the left of c is unchecked, M proceeds left to find the rightmost checked symbol.

9) $\delta([q_6, B], [B, d]) = ([q_6, B], [B, d], L)$.
M proceeds left.

10) $\delta([q_6, B], [\surd, d]) = ([q_1, B], [\surd, d], R)$.

M encounters a checked symbol and moves right to pick up another symbol for comparison. The first component of state becomes q_1 again.

11) $\delta([q_5, B], [\surd, d]) = ([q_7, B], [\surd, d], R)$.

M will be in state $[q_5, B]$ immediately after crossing c moving left. (See rule 7.) If a checked symbol appears immediately to the left of c, all symbols to the left of c have been checked. M must test whether all symbols to the right have been checked. If so, they must have compared properly with the symbols to the left of c, so M will accept.

12) $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$.

M moves right over c.

13) $\delta([q_8, B], [\surd, d]) = ([q_8, B], [\surd, d], R)$.

M moves to the right over checked symbols.

14) $\delta([q_8, B], [B, B]) = ([q_9, B], [\surd, B], L)$.

If M finds $[B, B]$, the blank, it halts and accepts. If M finds an unchecked symbol when its first component of state is q_8, it halts without accepting.

SHIFTING OVER

Shifting over

A Turing machine can make space on its tape by shifting all nonblank symbols a finite number of cells to the right. To do so, the tape head makes an excursion to the right, repeatedly storing the symbols read in its finite control and replacing them with symbols read from cells to the left. The TM can then return to the vacated cells and print symbols of its choosing. If space is available, it can push blocks of symbols left in a similar manner.

Example 7.6 We construct part of a Turing machine, $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, which may occasionally have a need to shift nonblank symbols two cells to the right. We suppose that M's tape does not contain blanks between nonblanks, so when it reaches a blank it knows to stop the shifting process. Let Q contain states of the form $[q, A_1, A_2]$ for $q = q_1$ or q_2, and A_1 and A_2 in Γ. Let X be a special symbol not used by M except in the shifting process. M starts the shifting process in state $[q_1, B, B]$. The relevant portions of the function δ are as follows.

1) $\delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, R)$ for A_1 in $\Gamma - \{B, X\}$.

   M stores the first symbol read in the third component of its state. X is printed on the cell scanned, and M moves to the right.

2) $\delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, R)$ for A_1 and A_2 in $\Gamma - \{B, X\}$.

   M shifts the symbol in the third component to the second component, stores the symbol being read in the third component, prints an X, and moves right.

3) $\delta([q_1, A_1, A_2], A_3) = ([q_1, A_2, A_3], A_1, R)$ for A_1, A_2, and A_3 in $\Gamma - \{B, X\}$.

   M now repeatedly reads a symbol A_3, stores it in the third component of state, shifts the symbol previously in the third component, A_2, to the second component, deposits the previous second component, A_1, on the cell scanned, and moves right. Thus a symbol will be deposited two cells to the right of its original position.

4) $\delta([q_1, A_1, A_2], B) = ([q_1, A_2, B], A_1, R)$ for A_1 and A_2 in $\Gamma - \{B, X\}$.

   When a blank is seen on the tape, the stored symbols are deposited on the tape.

5) $\delta([q_1, A_1, B], B) = ([q_2, B, B], A_1, L)$.

   After all symbols have been deposited, M sets the first component of state to q_2 and moves left to find an X, which marks the rightmost vacated cell.

6) $\delta([q_2, B, B], A) = ([q_2, B, B], A, L)$ for A in $\Gamma - \{B, X\}$.

   M moves left until an X is found. When X is found, M transfers to a state that we have assumed exists in Q and resumes its other functions.

Subroutine

As with programs, a "modular" or "top-down" design is facilitated if we use subroutines to define elementary processes. A Turing machine can simulate any type of subroutine found in programming languages, including recursive procedures and any of the known parameter-passing mechanisms. We shall here describe only the use of parameterless, nonrecursive subroutines, but even these are quite powerful tools.

The general idea is to write part of a TM program to serve as a subroutine; it will have a designated initial state and a designated return state which temporarily has no move and which will be used to effect a return to the calling routine. To design a TM that "calls" the subroutine, a new set of states for the subroutine is made, and a move from the return state is specified. The call is effected by entering the initial state for the subroutine, and the return is effected by the move from the return state.

Example 7.7 The design of a TM M to implement the total recursive function "multiplication" is given below. M starts with 0^m10^n on its tape and ends with 0^{mn} surrounded by blanks. The general idea is to place a 1 after 0^m10^n and then copy the block of n 0's onto the right end m times, each time erasing one of the m 0's. The result is 10^n10^{mn}. Finally the prefix 10^n1 is erased, leaving 0^{mn}. The heart of the algorithm is a subroutine COPY, which begins in an ID $0^m1q_10^n10^i$ and eventually enters an ID $0^m1q_50^n10^{i+n}$. COPY is defined in Fig. 7.5. In state q_1, on seeing a 0, M changes it to a 2 and enters state q_2. In state q_2, M moves right, to the next blank, deposits the 0, and starts left in state q_3. In state q_3, M moves left to a 2. On reaching a 2, state q_1 is entered and the process repeats until the 1 is encountered, signaling that the copying process is complete. State q_4 is used to convert the 2's back to 0's, and the subroutine halts in q_5.

	0	1	2	B
q_1	$(q_2, 2, R)$	$(q_4, 1, L)$		
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$		$(q_3, 0, L)$
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_1, 2, R)$	
q_4		$(q_5, 1, R)$	$(q_4, 0, L)$	

Fig. 7.5 δ for subroutine COPY.

To complete the program for multiplication, we add states to convert initial ID $q_0 0^n 1 0^n$ to $B 0^{n-1} 1 q_1 0^n 1$. That is, we need the rules

$$\delta(q_0, 0) = (q_6, B, R),$$
$$\delta(q_6, 0) = (q_6, 0, R),$$
$$\delta(q_6, 1) = (q_1, 1, R).$$

Additional states are needed to convert an ID $B^i 0^{m-i} 1 q_5 0^n 1 0^{ni}$ to $B^{i+1} 0^{m-i-1} 1 q_1 0^n 1 0^{ni}$, which restarts COPY, and to check whether $i = m$, that is, all m 0's have been erased. In the case that $i = m$, the leading $10^n 1$ is erased and the computation halts in state q_{12}. These moves are shown in Fig. 7.6.

	0	1	2	B
q_5	$(q_7, 0, L)$			
q_7		$(q_8, 1, L)$		
q_8	$(q_9, 0, L)$			(q_{10}, B, R)
q_9	$(q_9, 0, L)$			(q_0, B, R)
q_{10}		(q_{11}, B, R)		
q_{11}	(q_{11}, B, R)	(q_{12}, B, R)		

Fig. 7.6 Additional moves for TM performing multiplication.

Note that we could make more than one call to a subroutine if we rewrote the subroutine using a new set of states for each call.