Lecture 11

Ben Rosenberg

June 23, 2021

Turing machines

Recall that we were defining a **Turing machine**.

Definition: A **Deterministic Turing Machine** is a structure:

$$M = (Q, \Sigma, \Gamma, \delta, q_0)$$

such that:

- Q is a finite set of states
- Σ is the input alphabet
- Γ is the *tape alphabet*
- δ is the transition function
- $q_0 \in Q$ is the initial state

For Turing machines, we say that Γ is the **tape alphabet**, and $\Sigma \subset \Gamma$ because of the "blank" symbol $B \in \Gamma \backslash \Sigma$.

The transition function of a Turing machine is a *partial function* defined as:

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

Here, $(Q \times Gamma)$ is the current state-symbol pair, and $\{L, R\}$ is the direction that it moves. The machine reads the current state and symbol, and then overwrites something in its current location before moving.

Our δ is composed of 5-tuples, of the form $[q, a, p, b, D]$ where q is the current state, a is the crrent symbol, p is the next state, b is the symbol to overwrite, and D (the direction) is either L or R for left or right.

Configuration: For a finite state automaton, we had a state-string pair (q, w) where q was the current state and w was the rest of the input string to be read.

For a Turing machine, the configuration consists of the following:

- state
- the *entire* tape content
- the position of the head on the tape

By convention, we write this as $w_1 q w_2$, where $w_1, w_2 \in \Gamma^*$, and $q \in Q$ is the current state. Here, w_1 is the string to the left, and w_2 is the string under the head and to the right (it includes the current input to be processed). It's understood that it is really $w_1 q w_2 B^\infty$, but this is really not how it is supposed to be written (even if there are infinitely many blanks).

Initial configuration: $q_0 w_0, w_0 \in \Sigma^*$

From the current configuration $w_1 q w_2; w_1, w_2 \in \Gamma^*, q \in Q$

The next configuration $x_1 p x_2; x_1, x_2 \in \Gamma^*, p \in Q$ is calculated as follows:

Case 1: Right move.

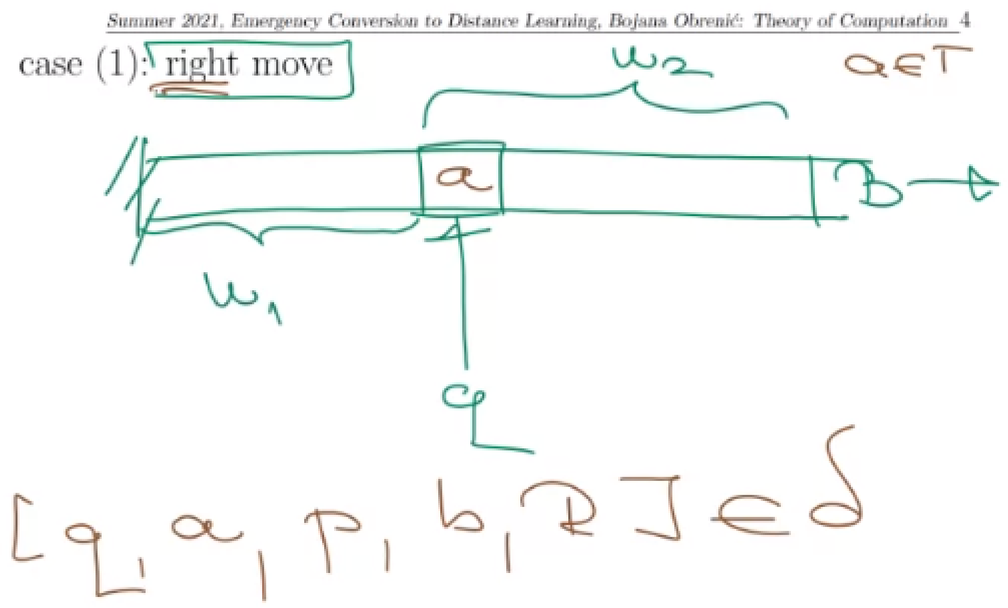
Summer 2021, Emergency Conversion to Distance Learning, Bojana Obrenić: Theory of Computation 4
case (1): right move
w_2
$a \in \Gamma$
a
B
w_1
q
$[q, a, p, b, R] \in \delta$

Figure 1: Diagramming of a right move

We read $[q, a, p, b, R] \in \delta$ as "at state q, on reading a, overwrite a with b and move to state p, and move right on the tape."

So, $x_1 = w_1 b$, and x_2 is defined in terms of w_2 as $w_2 = ax_2$. Again, $w_1, w_2, x_1, x_2 \in \Gamma^*$.

Case 2: Left move.

We read $[q, a, p, b, L] \in \delta$ and see that the new w_1 is $x_1 c, c \in \Gamma^*$, and $x_2 = cby$ where $w_2 = ay$. Again, $x_1, x_2, w_1, w_2, y \in \Gamma^*$ and $a, c \in \Gamma$.

Case 3: No move.

We have no move when we get $[q, a, \_, \_, \_] \notin \delta$, which cannot execute. We say that the machine **halts** in this circumstance.

When does a Turing machine halt? When it does not have a tuple that applies. The current state and symbol are enough to determine whether or not the tuple applies.

Examples:

Problem 1: Turing machine over alphabet $\{0, 1\}$ that always halts.

$M = (Q, \Sigma, \Gamma, \delta, q)$

$Q = \{q\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

δ : empty; no transitions

Problem 2: Turing machine over alphabet $\{0, 1\}$ that never halts (it *diverges*).

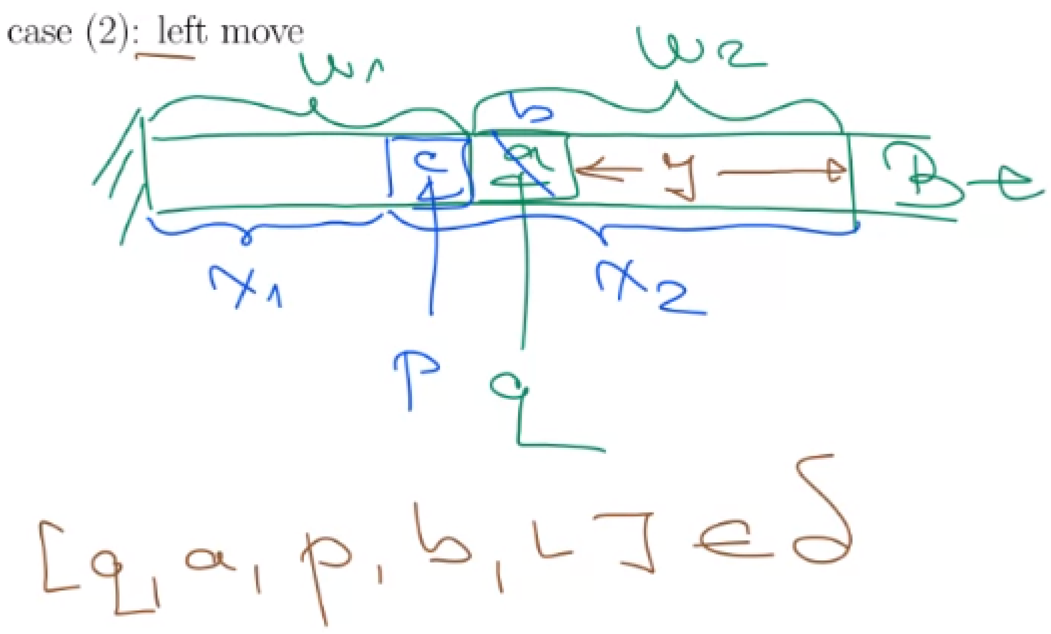

Figure 2: Diagramming of a left move

$M = (Q, \Sigma, \Gamma, \delta, q)$

$Q = \{q\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

$\delta : [q, 0, q, 0, R]; [q, 1, q, 1, R]; [q, B, q, B, R]$ (Note that q is the singular state here, which indicates that we should move to the right.)

We'll set our machine to "escape" to the right: all δ-tuples will be have $D = R$.

Def (informal): *Skipping* is moving over inputs without regard for what is there.

The above machine continues skipping inputs and moving to the right.

Def (informal): A Turing machine "escapes" if, for a new state, we continue skipping the state, and moving to the right.

Problem 3: Turing machine over alphabet $\{0, 1\}$ that turns its input string into its bit-wise complement and then halts.

$M = (Q, \Sigma, \Gamma, \delta, q)$

$Q = \{q\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

$\delta : [q, 0, q, 1, R]; [q, 1, q, 0, R]$

Here we are skipping over the input, with a minor twist (instead of writing the original symbol, we write its complement). When we reach a B, we halt.

Problem 4: Turing machine over alphabet $\{0, 1\}$ that flips the rightmost zero of its input string into 1 and then halts; if the input string does not contain any zewros then it halts without modifying the input.

Ex. `0100010011` $\to$ `0100010111`

$M = (Q, \Sigma, \Gamma, \delta, q)$

$Q = \{q, p, s, t\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

$\delta : [q, 0, p, 0, R]; [q, 1, q, 1, R]; [p, 0, p, 0, R]; [p, 1, p, 1, R]; [p, B, s, B, L]; [s, 0, t, 1, R]; [s, 1, s, 1, L]$

Plan: skip over input (go to the right until we hit a B). Then, turn back to the left, skipping over input until we get to 0. Then, we flip that zero and halt.

The problem is that there might not be any zeroes. To rectify this, we can, instead of skipping, we can *discover* if there is a zero, and *remember the fact that we have seen zero in the state*. If we have seen a zero, then turn back; otherwise, we just halt.

Note that we can't remember exactly *which* state we've seen the zero, as there are only finitely many states. If we were to try to make n states to keep track of each position in the string, the string might have a length of $n + 1$: we cannot write a machine in terms of a string! The input is *unbounded* and we have only finitely many states.

Note: Whenever we have an L transition, we need to be sure that we will not break the machine by going before the start of the input string.

Problem 5: Turing machine over alphabet $\{0, 1\}$ that shifts its input by one cell to the right, writes 1 into the leftmost cell, turns the input string into its bit-wise complement, returns the head to the leftmost cell, and then halts.

Ex. `10101110` $\to$ `110101110` $\to$ `101010001`, and return the head to the leftmost position and halt.

$M = (Q, \Sigma, \Gamma, \delta, q)$

$Q = \{q, n, x, s, t, f\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B, n\}$

Idea: We can always remember the previous symbol in the state, as there are only finitely many symbols in Σ.

New issue: How do we know where the leftmost symbol is? Idea: We can place a sentinel character there, which will tell us the location of the leftmost position.

$$\delta \quad : \quad \underbrace{[q, 0, n, N, R]; [q, 1, x, N, R]}_{\text{sentinel}}; \underbrace{[n, 0, n, 1, R]; [n, 1, x, 1, R]; [x, 0, n, 0, R]; [x, 1, x, 0, R]}_{\text{shift and flip}};$$

$$\underbrace{[n, B, S, B, L]; [x, B, S, B, L]}_{\text{turn back}}; \underbrace{[s, 0, s, 0, L]; [s, 1, s, 1, L]; [s, N, t, 1, R]}_{\text{look for left end}}; \underbrace{[t, 0, f, 0, L]; [t, 1, f, 1, L]}_{\text{park}}$$

Issue: we missed the case in which the entire string was blank. In this case, we are halting because there is no transition for q on B. So, we make one: $[q, B, t, N, R]$. Then, in t we need to add a transition on B: $[t, B, f, B, L]$.

Practice:

Problem 6: interpret input string as the binary representation of a positive number, and increment its input by 1

Problem 7: (a,b,c,d): examine input string and halt if the input string satisfies the form $a^n b^n c^n d^n$ but diverges otherwise

Halting problem

Defintion 2: Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0)$ accepts a language L by halting if for every string $w \in \Sigma^*$, M behaves as follows:

$$w \in L \implies M(w) \searrow \text{ (M halts on w)}$$
$$w \notin L \implies M(w) \nearrow \text{ (M diverges on w)}$$

If the string is good, the machine will halt; if the string is not good, it will not halt. But we can't tell. This is the **halting problem**.

Problem 8: Turing machine over alphabet $\{0, 1\}$ that accepts by halting those strings that contain `11` as a substring.

$M = (Q, \Sigma, \Gamma, \delta, q)$

$Q = \{q, n, f\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

$\delta : [q, 0, q, 0, R]; [q, 1, n, 1, R]; [q, B, q, B, R]; [n, 1, f, 1, R]; [n, 0, q, 0, R]; [n, B, q, B, R]$

This accepts by halting $(0 \cup 1)^* 11 (0 \cup 1)^*$.

Definition 3: Turing machine

$$M_1 = (Q, \Sigma, \Gamma, \delta, q_0, \mathbf{F})$$

accepts a language L by final state if, for every string $w \in \Sigma^*$, M_1 behaves as follows:

$$w \in L \implies M_1(w) \searrow \text{ in a final state}$$
$$w \notin L \implies \text{ either } M_1(w) \nearrow \text{ or } M_1(w) \searrow \text{ in a non-final state}$$

Problem 9: accepts by final state exactly those strings that begin and end with 0

$M = (Q, \Sigma, \Gamma, \delta, q, F)$

$Q = \{q, \}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B, \}$

$F = \{\}$

$\delta : [q, 0, p, 0, R]; [p, 0, p, 0, R]; [p, 1, p, 1, R]; [p, B, s, B, L]; [s, 0, f, 0, L]; [s, 1, h, 1, R]; [h, B, h, B, R]$

Algorithms (to be continued next time)

Algorithm 1:

Input: Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0)$ that accepts by halting some language L;

Output: Turing machine $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F)$ that accepts L by final state

Algorithm 2:

Input: Turing machine $M = (Q, \Sigma, \Gamma, \delta, q, F)$ that accepts L by final state;

Output: Turing machine $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_0)$ that accepts by halting some language L