CS121 Section 6: Turing Machines

Max Guo

Harvard University

October 13, 2020

Definition of a Turing Machine (Barak)

A Turing Machine M, as defined in Barak's textbook, contains the following:

- k states
- Alphabet $\Sigma \supseteq \{0, 1, \triangleright, \varnothing\}$
- Transition function $\delta_M : [k] \times \Sigma \to [k] \times \Sigma \times \{L, R, S, H\}$.

Definition of a Turing Machine (Barak) (cont.)

On input $x \in \{0,1\}^*$, the output of M on x, $M(x)$, is the result of the following:

- Initialize tape T to be the sequence

$$\triangleright, x_0, x_1, \ldots, x_{n-1}, \varnothing, \varnothing, \ldots$$

- Initialize $i = 0$ (head position), $s = 0$ (state).
- Repeat:
  1. Let $(s', \sigma', D) = \delta_M(s, T[i])$.
  2. Let $s = s'$, $T[i] = \sigma'$.
  3. Move i based on D: if $D = R$ then $i = i + 1$. If $D = L$ then $i = max(i - 1, 0)$.
  4. If $D = H$, then halt and return $y = T[0] \ldots T[i] \in \{0,1\}^*$, where i is the final head position.
- If the Turing Machine does not halt, denote $M(x) = \perp$.

Example Turing Machine

Consider the function $f : \{0,1\}^* \to \{0,1\}$ such that $f(x) = 1$ if and only if $|x|$ is even. Construct a Turing Machine that computes f.

Example Turing Machine (cont.)

States:

0. *EVEN* if the current number of inputs is even.
1. *ODD* if the current number of inputs is odd.
2. $CLEAR_0$ if we've found our answer - head back to the beginning and output 0
3. $CLEAR_1$ same as state 2 but output 1.
4. $OUTPUT_0$ to output 0.
5. $OUTPUT_1$ to output 1.

Example Turing Machine (cont.)

States/Inputs	$\triangleright$	0	1	$\varnothing$
0 (EVEN)				
1 (ODD)				
2 (CLEAR_0)				
3 (CLEAR_1)				
4 (OUTPUT_0)				
5 (OUTPUT_1)				

0. *EVEN*, *ODD* if the current number of inputs is even/odd.
1. $CLEAR_0$, $CLEAR_1$ if we've found our answer - head back to the beginning and output 0/1.
2. $OUTPUT_0$, $OUTPUT_1$ to output 0/1.

Example Turing Machine (cont.)

States/Inputs	$\triangleright$	0	1	$\varnothing$
0 (EVEN)	invalid	(1, 0, R)	(1, 1, R)	(3, $\varnothing$, L)
1 (ODD)	invalid	(0, 0, R)	(0, 1, R)	(2, $\varnothing$, L)
2 (CLEAR_0)	(4, $\triangleright$, R)	(2, 0, L)	(2, 1, L)	invalid
3 (CLEAR_1)	(5, $\triangleright$, R)	(3, 0, L)	(3, 1, L)	invalid
4 (OUTPUT_0)	invalid	(-, 0, H)	(-, 0, H)	(-, 0, H)
5 (OUTPUT_1)	invalid	(-, 1, H)	(-, 1, H)	(-, 1, H)

Definition of a Turing Machine (Sipser)

A Turing Machine as defined in Sipser's textbook is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:

- Q is the set of states
- Σ is the input alphabet, not containing the blank symbol $\varnothing$
- Γ is the tape alphabet, where $\varnothing \in \Gamma$ and $\Sigma \subseteq \Gamma$.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- $q_0 \in Q$ is the start state.
- q_{accept} is the accept state
- q_{reject} is the reject state, where $q_{reject} \neq q_{accept}$.

Differences between Sipser and Barak

	Barak	Sipser
States	$[k]$	Q
Input Alphabet	$\{0, 1\}$	General Σ
Directions D	$\{L, R, S, H\}$	$\{L, R\}$
Return Methods	Halt, $T[0] \ldots T[i]$	Reach state q_{accept} or q_{reject}
Return Values	$\{0, 1\}^*$	$\{0, 1\}$

Table 1: Differences between Barak and Sipser's definitions of Turing Machines

Simulating Sipser TM with Barak TM

To simulate a Sipser TM with a Barak TM, we must expand upon the corresponding states for q_{accept} and q_{reject} and also account for general input alphabet.

Simulating Sipser TM with Barak TM (cont.)

Consider the function $f : \{a, b, c\}^* \rightarrow \{0, 1\}$ that accepts expressions of the form $(abc)^*$. Write a Sipser TM that computes this function, and then convert it to a Barak TM.

Simulating Sipser TM with Barak TM (cont.)

States	**State Names/Inputs**	$\triangleright$	**0**	**1**	$\varnothing$
0	Start	invalid	(1, 0, R)	(7, 1, R)	(8, $\varnothing$, L)
1	a	invalid	(7, 0, R)	(2, 1, R)	(7, $\varnothing$, L)
2	First_One_a	invalid	(4, 0, R)	(7, 1, R)	invalid
3	First_One_b	invalid	(7, 0, R)	(5, 1, R)	invalid
4	b	invalid	(7, 0, R)	(3, 1, R)	(7, $\varnothing$, L)
5	Second_One_b	invalid	(6, 0, R)	invalid	invalid
6	c	invalid	(1, 0, R)	(7, 1, R)	(8, $\varnothing$, L)
7	CLEAR_0	(9, $\triangleright$, R)	(7, 0, L)	(7, 1, L)	(7, $\varnothing$, L)
8	CLEAR_1	(10, $\triangleright$, R)	(8, 0, L)	(8, 1, L)	(8, $\varnothing$, L)
9	OUTPUT_0	invalid	(-, 0, H)	(-, 0, H)	(-, 0, H)
10	OUTPUT_1	invalid	(-, 1, H)	(-, 1, H)	(-, 1, H)

NAND-TM

NAND-TM programs are sequences of lines consisting of:

- Scalar and array variables.
- Lines of the form. `variable = NAND(variable1, variable2)`.
- A `MODANDJUMP` instruction at the end.
- Input and output array variables.
- Index variable i.

NAND-TM (cont.)

Theorem

For every $F : \{0,1\}^* \to \{0,1\}^*$, F is computable by a NAND-TM program P if and only if there is a Turing Machine M that computes F.

NAND-TM Syntactic Sugar:

- Inner loops: `while` and `for` loops
- Multiple index variables
- Arrays with higher dimensions.

NAND-TM Syntactic Sugar

Show that NAND-TM can implement 2 dimensional arrays, so that we can use them as syntactic sugar.

NAND-TM Syntactic Sugar (cont.)

$$embed(x, y) = \frac{1}{2}(x + y)(x + y + 1) + x.$$

	0	1	2	3	4
0					
1					
2					
3					
4					

NAND-RAM

Theorem

For every $F : \{0,1\}^* \to \{0,1\}^*$*, F is computable by a NAND-TM program P if and only if F is computable by a NAND-RAM program.*

NAND-RAM properties:

- Everything that NAND-TM possesses, plus:
- Variables can be integer-valued
- Basic arithmetic operations
- Indexed access in arrays

Big Idea

Using equivalence results such as those between Turing and RAM machines, we can “have our cake and eat it too”.

- If we want to prove something can’t be done, use a Turing machine
- If we want to prove something can be done, use a high level language (e.g. NAND-RAM, Python, C).

Practice Problem 1

Consider the function $f : \{0,1\}^* \to \{0,1\}$ such that $f(x) = 1$ if and only if $|x| = n$ is even and $x_0 = x_{n/2}$. In other words, the first bit of x is equal to the first bit of the second half of x. Construct a Turing Machine that computes f.

Practice Problem 2

Suppose that $F : \{0,1\}^* \to \{0,1\}$ is a computable function. Prove that G is computable in each of the following situations:

1. For every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, $G(x_0 \dots x_{n-1}) = F(x_{n-1} \dots x_0)$.
2. For every $x \in \{0,1\}^*$, $G(x) = 1$ iff there exists a list $u_0, \dots, u_{t-1}$ of non-empty strings such that $F(u_i = 1)$ for every $i \in [t]$ and $x = u_0 u_1 \dots u_{t-1}$.

Hint: Use the "have our cake and eat it too" paradigm!