Turing Machines

CSCI 3130 Formal Languages and Automata Theory

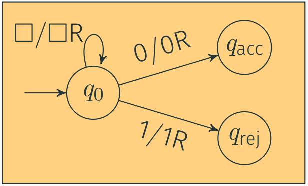
Siu On CHAN

Fall 2022

Chinese University of Hong Kong

Looping

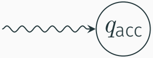
Turing machine may not halt



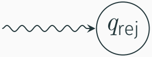
$\Sigma = \{0, 1\}$

input: ε

Inputs can be divided into three types:



Accept



Reject

Infinite loop

Halting

We say M halts on input x if there is a sequence of configurations

$$C_0, C_1, \ldots, C_k$$

C_0 is starting $\quad$ C_i yields C_{i+1} $\quad$ C_k is accepting or rejecting

A TM M is a decider if it halts on every input

A TM M decides a language L if M is a decider and recognizes L

Language L is decidable if it is recognized by a TM that halts on every input

Programming Turing machines: Are two strings equal?

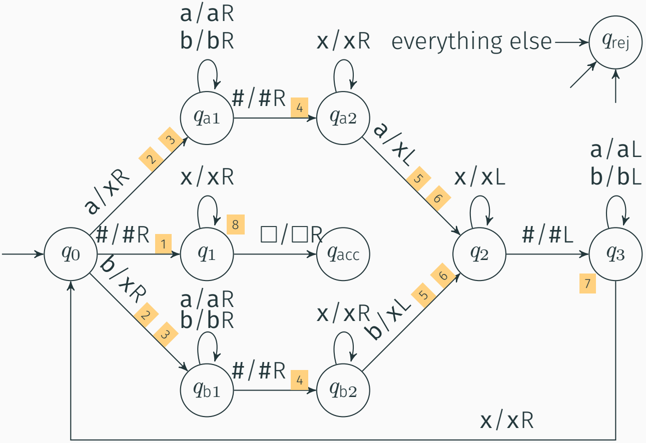
$$L_1 = \{w\#w \mid w \in \{\mathsf{a}, \mathsf{b}\}^*\}$$

Description of Turing Machine

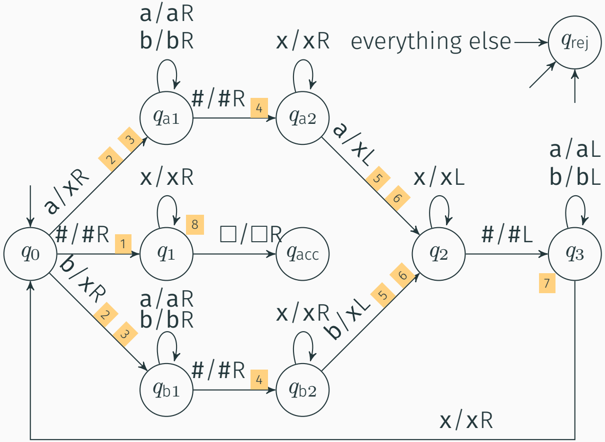
Until you reach #	
Read and remember entry	x$\underline{\text{b}}$baa#xbbaa
Write x	xx$\underline{\text{b}}$aa#xbbaa
Move right past # and past all x's	xxbaa#x$\underline{\text{b}}$baa
If this entry is different, reject	
Write x	xxbaa#x$\underline{\text{x}}$baa
Move left past # and to right of first x	xx$\underline{\text{b}}$aa#xxbaa
If you see only x's followed by $\square$, accept	

Programming Turing machines: Are two strings equal?

$$L_1 = \{w\#w \mid w \in \{\mathsf{a}, \mathsf{b}\}^*\}$$



Programming Turing machines: Are two strings equal?



input:

aab#aab

configurations:

q_0 aab#aab

x q_{a1} ab#aab

xa q_{a1} b#aab

xab q_{a1} #aab

xab# q_{a2} aab

xab q_2 #xab

xa q_3 b#xab

x q_3 ab#xab

q_3 xab#xab

x q_0 ab#xab

⋮

Programming Turing machines

$$L_2 = \{\mathsf{a}^i\mathsf{b}^j\mathsf{c}^k \mid ij = k \text{ and } i, j, k > 0\}$$

High level description of TM:

1 For every a:
2   Cross off the same number of b's and c's
3   Uncross the crossed b's (but not the c's)
4   Cross off this a
5 If all a's and c's are crossed off, accept

Example:

1 aabbcccc
2 aa~~bbcc~~cc
3 aabb~~cc~~cc
4 ~~a~~abb~~cc~~cc
5 ~~a~~abb~~cc~~cc
2 ~~a~~a~~bbcccc~~
3 ~~aa~~bb~~cccc~~

$\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ Γ = {a, b, c, ~~a~~, ~~b~~, ~~c~~, □}

$$L_2 = \{\mathsf{a}^i\mathsf{b}^j\mathsf{c}^k \mid ij = k \text{ and } i, j, k > 0\}$$

Low-level description of TM:

Scan input from left to right to check it looks like $\mathsf{aa}^*\mathsf{bb}^*\mathsf{cc}^*$

Move the head to the first symbol of the tape

For every a:

- Cross off the same number of b's and c's
- Restore the crossed off b's (but not the c's)
- Cross off this a

If all a's and c's are crossed off, accept

Programming Turing machines

$$L_2 = \{\mathsf{a}^i\mathsf{b}^j\mathsf{c}^k \mid ij = k \text{ and } i, j, k > 0\}$$

Low-level description of TM:

Scan input from left to right to check it looks like $\mathsf{aa}^*\mathsf{bb}^*\mathsf{cc}^*$

Move the head to the first symbol of the tape   How?

For every a:

- Cross off the same number of b's and c's   How?
- Restore the crossed off b's (but not the c's)
- Cross off this a

If all a's and c's are crossed off, accept

Implementation details:

Move the head to the first symbol of the tape:	
Put a special marker on top of the first a	ȧabbcccc
Cross off the same number of b's and c's:	ȧabbcccc
Replace b by ~~b~~	ȧa~~b~~bcccc
Move right until you see a c	ȧa~~b~~bcccc
Replace c by ~~c~~	ȧa~~b~~b~~c~~ccc
Move left just past the last ~~b~~	ȧa~~bbc~~ccc
If any uncrossed b's are left, repeat	ȧa~~bbc~~ccc
	ȧa~~bbcc~~cc

$\Sigma = \{a, b, c\} \qquad \Gamma = \{a, b, c, \not{a}, \not{b}, \not{c}, \dot{a}, \dot{\not{a}}, \square\}$

Programming Turing machines: Element distinctness

$L_3 = \{\texttt{\#}x_1\texttt{\#}x_2 \ldots \texttt{\#}x_m \mid x_i \in \{\texttt{0}, \texttt{1}\}^* \text{ and } x_i \neq x_j \text{ for every } i \neq j\}$

Example: $\texttt{\#01\#0011\#1} \in L_3$

High-level description of TM:

On input w

For every pair of blocks x_i and x_j in w

Compare the blocks x_i and x_j

If they are the same, reject

Accept

Programming Turing machines: Element distinctness

$L_3 = \{\#x_1\#x_2 \ldots \#x_m \mid x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for every } i \neq j\}$

Low-level desrciption:

0. If input is ε, or has exactly one #, accept
1. Mark the leftmost # as $\dot{\#}$ and move right $\dot{\#}$01#0011#1
2. Mark the next unmarked # $\dot{\#}$01$\dot{\#}$0011#1

Programming Turing machines: Element distinctness

$$L_3 = \{\#x_1\#x_2\ldots\#x_m \mid x_i \in \{\texttt{0},\texttt{1}\}^* \text{ and } x_i \neq x_j \text{ for every } i \neq j\}$$

3. Compare the two strings to the right of $\dot{\#}$
   If they are equal, reject

   $\dot{\#}\underline{\texttt{01}}\dot{\#}\underline{\texttt{0011}}\#\texttt{1}$

4. Move the right $\dot{\#}$
   If not possible, move the left $\dot{\#}$ to the next #
   and put the right $\dot{\#}$ on the next #
   If not possible, accept

   $\dot{\#}\texttt{01}\#\texttt{0011}\dot{\#}\texttt{1}$

5. Repeat Step 3

   $\dot{\#}\underline{\texttt{01}}\#\texttt{0011}\dot{\#}\underline{\texttt{1}}$

   $\#\texttt{01}\dot{\#}\texttt{0011}\dot{\#}\texttt{1}$

   $\#\texttt{01}\dot{\#}\underline{\texttt{0011}}\dot{\#}\underline{\texttt{1}}$

How to describe Turing Machines

Unlike for DFAs, NFAs, PDAs, we rarely give complete state diagrams of Turing Machines

We usually give a high-level description

unless you're asked for a low-level description or even state diagram

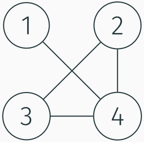
We are interested in algorithms behind the Turing machines

Programming Turing machines: Graph connectivity

$L_4 = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$

How do we feed a graph into a Turing Machine?

How to encode a graph G as a string $\langle G \rangle$?



`(1,2,3,4)((1,4),(2,3),(3,4),(4,2))`

Conventions for describing graphs:

`(`nodes`)(`edges`)`
no node appears twice
edges are pairs `(`first node`,` second node`)`

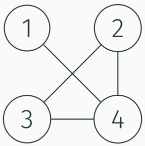
Programming Turing machines: Graph connectivity

$L_3 = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$

High-level description:

On input $\langle G \rangle$

0. Verify that $\langle G \rangle$ is the description of a graph
   No node/edge repeats; Edge endpoints are nodes
1. Mark the first node of G
2. Repeat until no new nodes are marked:
   2.1 For each node, mark it if it is adjacent to an already marked node
3. If all nodes are marked, accept; otherwise reject



Programming Turing machines: Graph connectivity

Some low-level details:

0. Verify that $\langle G \rangle$ is the description of a graph

No node/edge repeats: Similar to Element distinctness

Edge endpoints are nodes: Also similar to Element distinctness

1. Mark the first node of G

Mark the leftmost digit with a dot, e.g. 12 becomes $\dot{1}2$

2. Repeat until no new nodes are marked:

2.1 For each node, mark it if it is attached to an already marked node

For every dotted node u and every undotted node v:

Underline both u and v from the node list

Try to match them with an edge from the edge list

If not found, remove underline from u and/or v and try another pair