Turing Machines

CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN
Fall 2020

Chinese University of Hong Kong
Looping

Turing machine may not halt

\[ Σ = \{0, 1\} \]

input: \( ε \)

Inputs can be divided into three types:

- \( q_{\text{acc}} \) Accept
- \( q_{\text{rej}} \) Reject
- Infinite loop
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
- $C_i$ yields $C_{i+1}$
- $C_k$ 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?

\[ L_1 = \{ w\#w \mid w \in \{a, b\}^* \} \]

Description of Turing Machine

1. **Until** you reach #
2. **Read** and remember entry
3. **Write** x
4. **Move** right past # and past all x’s
5. **If** this entry is different, reject
6. **Write** x
7. **Move** left past # and to right of first x
8. **If** you see only x’s followed by □, accept
Programming Turing machines: Are two strings equal?

\[ L_1 = \{ w\#w \mid w \in \{a, b\}^* \} \]
Programming Turing machines: Are two strings equal?

**Input:**
aab#aab

**Configurations:**
- $q_0$: aab#aab
- $x$: qa1 ab#aab
- $xa$: q1 b#aab
- $xab$: qa1 #aab
- $xab#$: qa2 aab
- $xab q_2$: #xab
- $xa$: q2 b#xab
- $x$: q3 ab#xab
- $q_3$: xab#xab
- $x$: q0 ab#xab
Programming Turing machines

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

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:

\[
\begin{align*}
1 & \quad aabbcc \quad \text{aabbcccc} \\
2 & \quad bbcc \quad aabbc \quad \text{aabbc} \\
3 & \quad bbcc \quad aabbc \quad \text{aabbc} \\
4 & \quad bbcc \quad aabbc \quad \text{aabbc} \\
5 & \quad bbcc \quad aabbc \quad \text{aabbc} \\  
\end{align*}
\]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ a, b, c, \epsilon, \square \} \]
\[ L_2 = \{a^i b^j c^k \mid ij = k \text{ and } i, j, k > 0\} \]

Low-level description of TM:

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

Move the head to the first symbol of the tape

For every \texttt{a}:

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

If all \texttt{a}'s and \texttt{c}'s are crossed off, accept
Programming Turing machines

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

Low-level description of TM:

Scan input from left to right to check it looks like \( a a^* b b^* c c^* \)

Move the head to the first symbol of the tape \( \text{ How? } \)

For every \( a \):

- Cross off the \textbf{same number} of \( b \)'s and \( c \)'s \( \text{ How? } \)
- 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
Programming Turing machines

Implementation details:

Move the head to the first symbol of the tape:
Put a special marker on top of the first \texttt{a} \texttt{àaabbccccc}

Cross off the same number of \texttt{b}'s and \texttt{c}'s: \texttt{àaabbbc} \texttt{ccc}
Replace \texttt{b} by \texttt{b} \texttt{àaabb} \texttt{cccc}
Move right until you see a \texttt{c} \texttt{àaabb} \texttt{èccc}
Replace \texttt{c} by \texttt{è} \texttt{àaabb} \texttt{èccc}
Move left just past the last \texttt{b} \texttt{àaabb} \texttt{èccc}
If any uncrossed \texttt{b}'s are left, repeat \texttt{àaabb} \texttt{èCCC}

\[\Sigma = \{a, b, c\} \quad \Gamma = \{a, b, c, a, b, c, \dot{a}, \dot{a}, \Box\}\]
Programming Turing machines: Element distinctness

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

Example: \( \#01\#0011\#1 \in L_3 \)

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

Low-level description:

0. If input is \( \varepsilon \), or has exactly one \#, accept

1. Mark the leftmost \# as \( \dot{\#} \) and move right

\[ \#01\#0011\#1 \]

2. Mark the next unmarked \#

\[ \#01\dot{\#}0011\#1 \]
Programming Turing machines: Element distinctness

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

3. Compare the two strings to the right of \( \# \) \( \#01\#0011\#1 \)
   If they are equal, reject

4. Move the right \( \# \) \( \#01\#0011\#1 \)
   If not possible, move the left \( \# \) to the next \( \# \)
   and put the right \( \# \) on the next \( \# \)
   If not possible, accept

5. Repeat Step 3
   \( \#01\#0011\#1 \)
   \( \#01\#0011\#1 \)
   \( \#01\#0011\#1 \)
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 $\langle G \rangle$?

$\langle (1, 2, 3, 4), ((1, 4), (2, 3), (3, 4), (4, 2)) \rangle$

Conventions for describing graphs:

(nodes)(edges)
no node appears twice
edges are pairs (first node, second node)
\[ L_3 = \{ \langle G \rangle \mid G \text{ is a connected undirected graph} \} \]

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
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