Turing Machines and Their Variants
CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Chinese University of Hong Kong

Fall 2016
Looping

Turing machine may not halt

\[ \Sigma = \{0, 1\} \]

input: \( \varepsilon \)

Inputs can be divided into three types:

- **Accept**: \( q_{\text{acc}} \)
- **Reject**: \( q_{\text{rej}} \)
- **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 $C_i$ yields $C_{i+1}$ $C_k$ is accepting or rejecting

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

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**  

4. **Move** right past # and past all \( x \)'s  

5. **If** this entry is different, **reject**  

6. **Write**  

7. **Move** left past # and to right of first \( x \)  

8. **If** you see only \( x \)'s followed by \[\square\], **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
- $x$ $q_a1$ ab#aab
- $xa$ $q_a1$ b#aab
- $xab$ $q_a1$ #aab
- $xab#$ $q_a2$ aab
- $xabq_2$ #xab
- $xa$ $q_3$ b#xab
- $x$ $q_3$ ab#xab
- $q_3$ $xab$#xab
- $x$ $q_0$ ab#xab
- ...

**Diagram:**

The diagram illustrates the transition rules for a Turing machine that checks if two strings are equal. Each state represents a configuration of the machine, and the arrows show the transitions based on the input symbols. The states and transitions are labeled with the corresponding rules for reading and writing symbols on the tape. The final state $q_{rej}$ indicates a rejection if the strings are not equal.
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 \textbf{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:

\[
\begin{align*}
1 & \quad aabbccccc \\
2 & \quad aabbeccc \\
3 & \quad aabbeccc \\
4 & \quad aabbeccc \\
5 & \quad aabbeccc \\
\end{align*}
\]

\[ \Sigma = \{a, b\} \quad \Gamma = \{a, b, c, \alpha, \beta, \epsilon, \Box\} \]
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 $aa^* bb^* cc^*$
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

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

Low-level description of TM:

Scan input from left to right to check it looks like aa* bb* cc*

Move the head to the first symbol of the tape \textbf{How?}

For every a:
- Cross off the \textbf{same number} of b’s and c’s \textbf{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 a ȧaabcccccc

Cross off the same number of b’s and c’s: ȧaabbcdddd
Replace b by b ȧaabbbddd
Move right until you see a c ȧaabbcdddd
Replace c by ε ȧaabbbdddd
Move left just past the last b ȧaabbbdddd
If any uncrossed b’s are left, repeat ȧaabbbdddd

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{a, b, c, a, b, \epsilon, \dot{a}, \dot{a}, \square\} \]
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
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 description:

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

1. Mark the leftmost \# as \( \hat{\#} \) and move right \( \hat{\#}01\hat{0011}\#1 \)

2. Mark the next unmarked \# \( \hat{\#}01\hat{\#0011}\#1 \)
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 $\dot{\#}$
   If they are equal, reject

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

5. Repeat Step 3
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
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 $\hat{12}$

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
Variants of Turing Machines
Transitions may depend on the contents of all cells under the heads

Different tape heads can move independently.
Multitape Turing machine

Multiple tapes are convenient
One tape can serve as temporary storage
How to argue equivalence

Multitape Turing machines are equivalent to single-tape Turing machines.
Simulating multitape Turing machine

\[ M \]
\[ a a \square \square \ldots \]
\[ a b b \square \ldots \]
\[ b a \square \square \ldots \]

\[ \Gamma = \{a, b, \square\} \]

\[ S \]
\[ \# b \dot{a} \# a b b \dot{\square} \# a \dot{a} \# \square \ldots \]

\[ \Gamma = \{a, b, \square, \dot{a}, \dot{b}, \dot{\square}, \#\} \]
Simulating multitape Turing machine

We show how to simulate a multitape Turing machine on a single tape Turing machine.

To be specific, let’s simulate a 3-tape TM

Multitape TM $M$

Single tape TM $S'$

```
#x_1 x_2 \ldots x_r \ldots x_i # y_1 y_2 \ldots y_s \ldots y_j # z_1 z_2 \ldots z_t \ldots z_k
```
Simulating multitape Turing machine

Single-tape TM: Initialization

$S$: On input $w_1 \ldots w_n$:

Replace tape contents by $\#w_1 w_2 \ldots w_n \# \square \# \square$

Remember that $M$ is in state $q_0$
Simulating multitape Turing machine

**Single-tape TM: Simulating multitape TM moves**

Suppose Multitape TM $M$ moves like this:

We simulate the move on single-tape TM $S$ like this
Simulating multitape Turing machine

Given input $w_1 \ldots w_n$:
Replace tape contents by $\# w_1 w_2 \ldots w_n \# \cdot \cdot \cdot$
Remember (in state) that $M$ is in state $q_0$

$S$ simulates a step of $M$:
Make a pass over tape to find $\dot{x}, \dot{y}, \dot{z}$

\[
\begin{array}{c}
\# x_1 x_2 \ldots \dot{x} \ldots x_i \# y_1 y_2 \ldots \dot{y} \ldots y_j \# z_1 z_2 \ldots \dot{z} \ldots z_k
\end{array}
\]

If $M$ at state $q_a$ has transition $\begin{cases} x/x' A \\ y/y' B \\ z/z' C \end{cases}$
update state/tape accordingly

If $M$ reaches accept (reject) state, $S$ accepts (rejects)
To simulate a model $M$ by another model $N$:

Say how the state and storage of $N$ is used to represent the state and storage of $M$.

Say what should be initially done to convert the input of $N$.

Say how each transition of $M$ can be implemented by a sequence of transitions of $N$. 