Looping

Turing machine may not halt

Inputs can be divided into three types:

Accept

Reject

Infinite loop

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

input: \( \varepsilon \)
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

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 \( x\)  
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 \( \square \), 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
xa $q_{a1}$ ab#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

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

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:
1. \( aabbbc \)
2. \( aabbeccc \)
3. \( aabbeccc \)
4. \( aabbeccc \)
5. \( aabbeccc \)
6. \( aabbeccc \)
7. \( aabbeccc \)

\[ \Sigma = \{ a, b \} \quad \Gamma = \{ a, b, c, \epsilon, \square \} \]
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 \( a^* b^* c^* \)

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 \( a^* b^* c^* \)

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

For every \( a \):

Cross off the 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 $a$

Cross off the same number of $b$'s and $c$'s:
Replace $b$ by $\hat{b}$
Move right until you see a $c$
Replace $c$ by $\epsilon$
Move left just past the last $\hat{b}$
If any uncrossed $b$'s are left, repeat

$$\Sigma = \{a, b, c\} \quad \Gamma = \{a, b, c, a, \hat{b}, \epsilon, \hat{a}, \hat{a}, \Box\}$$
\[ L_3 = \{ \#x_1\#x_2\ldots\#x_m \mid x_i \in \{0, 1\}^* \text{ and } x_i \neq x_j \text{ for every } i \neq j \} \]

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 \( \dot{\#}01\#0011\#1 \)

2. Mark the next unmarked \# \( \dot{\#}01\#0011\#1 \)
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 \( \dot{\#}01\#0011\#1 \)

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 \( \dot{\#}01\#0011\dot{\#}1 \)

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

$(1, 2, 3, 4)((1, 4), (2, 3), (3, 4), (4, 2))$

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 \( \dot{1}2 \)

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

Different tape heads can move independent
Multiple tapes are convenient

One tape can serve as temporary storage
Multitape Turing machines are equivalent to single-tape Turing machines.
Simulating multitape Turing machine

\[ M \]

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

\[ S \]

\[ \Gamma = \{a, b, \square, \dot{a}, \dot{b}, \dot{\square}, \#\} \]
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**

On input $w_1 \ldots w_n$:

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

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:

\[
\begin{array}{c}
\text{b} \quad \text{a} \quad \square \quad \square \quad \ldots \\
\text{a} \quad \text{b} \quad \text{a} \quad \square \quad \ldots \\
\text{a} \quad \text{a} \quad \text{b} \quad \square \quad \ldots \\
\end{array}
\]

\[
\begin{array}{c}
\text{b} \quad \text{a} \quad \square \quad \square \quad \ldots \\
\text{a} \quad \text{b} \quad \text{b} \quad \square \quad \ldots \\
\text{a} \quad \text{a} \quad \text{b} \quad \square \quad \ldots \\
\end{array}
\]

We simulate the move on single-tape TM \( S \) like this:

\[
\begin{array}{c}
\# \quad \text{b} \quad \text{a} \quad \square \quad \# \quad \text{a} \quad \text{b} \quad \text{a} \quad \# \quad \text{a} \quad \text{a} \quad \text{b} \quad \# \quad \square \\
\end{array}
\]

\[
\begin{array}{c}
\# \quad \text{b} \quad \text{\dot{a}} \quad \# \quad \text{a} \quad \text{b} \quad \text{b} \quad \square \quad \# \quad \text{a} \quad \text{\dot{a}} \quad \text{b} \quad \# \quad \square \\
\end{array}
\]
Simulating multitape Turing machine

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

Replace tape contents by $\# \dot{w}_1 \dot{w}_2 \ldots \dot{w}_n \# \dot{\Box} \# \dot{\Box}$

Remember (in state) that $M$ is in state $q_0$

$S$ simulates a step of $M$:

Make a pass over tape to find $\dot{x}$, $\dot{y}$, $\dot{z}$

If $M$ at state $q_a$ has transition

\[
\begin{cases}
  x / x' \text{A} \\
  y / y' \text{B} \\
  z / z' \text{C}
\end{cases}
\]

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 $N$