Introduction to the Theory of Computation

Set 7 — Turing Machines

Recap To Date

Finite automata (both deterministic and nondeterministic) machines accept regular languages

Weakness: no memory

Pushdown automata accept context-free languages

Add memory in the form of a stack

Weakness: stack is restrictive

Turing Machines

Similar to a finite automata

+ Unrestricted memory in the form of a tape

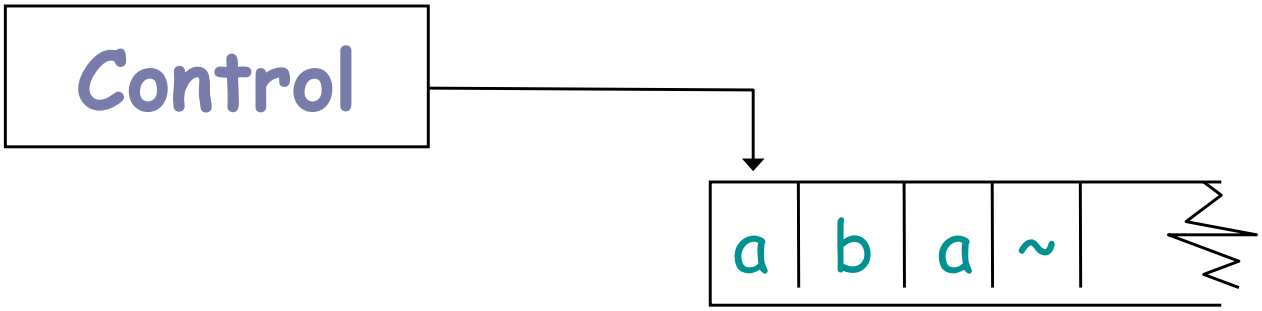
Can do *anything* a real computer can do!

- **Still cannot solve some problems**

Church-Turing thesis:

Any effective computation can be carried out by some Turing machine

Turing Machine Schematic



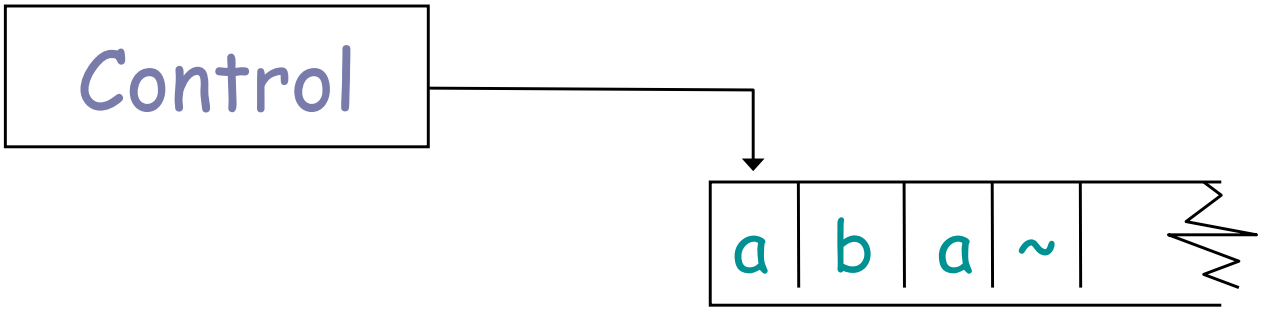
Initially tape contains the input string

Blanks everywhere else on the tape,
denoted by some special symbol in the tape

alphabet: ~, □, ¬, _, ƀ, …

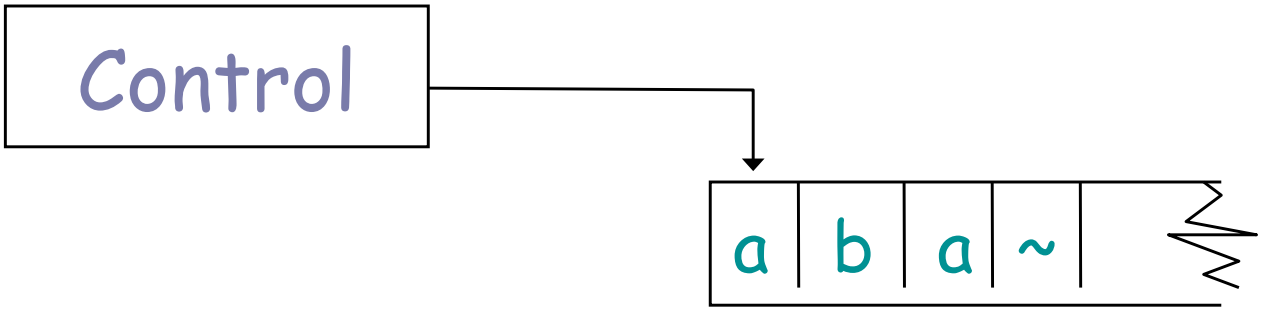
Machine may also *write* information on the tape

Turing Machine Schematic



- Can move tape head to read information written to tape
- Continues computing until output produced
  - Output values are **accept** or **reject**

Turing Machine Schematic



- **Turing machine results**
  - **Accept**
  - **Reject**
  - **Never halts**
- **We may not be able to tell result by observation**

Differences Between TM and FA

TM has tape you can *read* from and *write* to

Read-write head can be moved in either direction

Tape is infinite

Accept and reject states take effect immediately

Example

How can we design a Turing machine to find the middle of a string?

- **If string length is odd, accept and indicate middle symbol**
- **If string length is even, reject string**

Make multiple passes over string *X*-ing out symbols at ends until only middle remains

Processing Input

1. **Check if string is empty or length 1**
   - If empty, return **reject**; If length 1, return **accept**
2. **Write X over first and last non-X symbols**
   - After this, the head will be at the second X
3. **Move left one symbol**
   - If symbol is an X, return **reject** (string is even in length)
4. **Move left one symbol**
   - If symbol is an X, return **accept** (string is not even in length; middle symbol is only non-X)
5. **Go to step 2**

Example

00110~

- First check length of string
- X first and last non-X symbols

X011**X**~

- Move left one symbol

X011**1**X~

- Is symbol an X? No
- Move left one symbol

X0**1**1X~

- Is symbol an X? No
- Write X over first and last non-X symbols

1. Check if string is empty or length 1
   - If empty, return reject; If length 1, return accept
2. Write X over first and last non-X symbols
   - After this, the head will be at the second X
3. Move left one symbol
   - If symbol is an X, return reject (string is even in length)
4. Move left one symbol
   - If symbol is an X, return accept (string is not even in length; middle symbol is only non-X)
5. Go to step 2

Example

00110~

- First check if string is empty
- X first and last non-X symbols

X011**X**~

- Move left one symbol

X011**1**X~

- Is symbol an X? No
- Move left one symbol

X0**1**1X~

- Is symbol an X? No
- Write X over first and last non-X symbols

1. Check if string is empty or length 1
   - If empty, return reject; If length 1, return accept
2. Write X over first and last non-X symbols
   - After this, the head will be at the second X
3. Move left one symbol
   - If symbol is an X, return reject (string is even in length)
4. Move left one symbol
   - If symbol is an X, return accept (string is not even in length; middle symbol is only non-X)
5. Go to step 2

Example

XX1XX~

- Move left one symbol

XX1XX~

- Is symbol an X? No
- Move left one symbol

XX1XX~

- Is symbol an X? Yes
- Return accept

1. Check if string is empty or length 1
   - If empty, return reject; If length 1, return accept
2. Write X over first and last non-X symbols
   - After this, the head will be at the second X
3. Move left one symbol
   - If symbol is an X, return reject (string is even in length)
4. Move left one symbol
   - If symbol is an X, return accept (string is not even in length; middle symbol is only non-X)
5. Go to step 2

Formal Definition of a TM

Definition: A Turing Machine is a 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$ where Q, Σ, and Γ are finite sets and

- **Q is the set of states**
- **Σ is the input alphabet not containing a special blank symbol (~)**
- **Γ is the tape alphabet, where ~ $\in \Gamma$ and $\Sigma \subseteq \Gamma$**
- **δ: $Q\times\Gamma \rightarrow Q\times\Gamma\times\{L,R\}$ is the transition function**

Formal Definition of a TM

Definition: A Turing Machine is a 7-tuple

$(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$,

where Q, Σ, and Γ are finite sets and

- $q_0 \in Q$ **is the start state**
- $q_{accept} \in Q$ **is the accept state**
- $q_{reject} \in Q$ **is the reject state,**

where $q_{reject} \neq q_{accept}$

What Is the Transition Function?

Q = set of states, Γ = tape alphabet

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R,S\}$$

Given:

- The current internal state $\in Q$
- The symbol on the current tape cell $\in \Gamma$

Then δ tells us what the TM does:

- Changes to new internal state $\in Q$
- May write a new symbol $\in \Gamma$
  - and move one cell left or right

Computing with a TM M

M receives input $w = w_1 w_2 \ldots w_n \in \Sigma^*$ on "leftmost" n squares of tape

- Rest of tape is blank (all ~ symbols)

Head position begins at leftmost square of input

Computation follows rules of δ

If model is infinite to the right only, head never moves left of leftmost square

- If δ says to move L there, head stays put!

Completing Computation

Continue following δ transition rules until M reaches q_{accept} or q_{reject}

- **Halt at these states**

May never halt if the machine never transitions to one of these states!

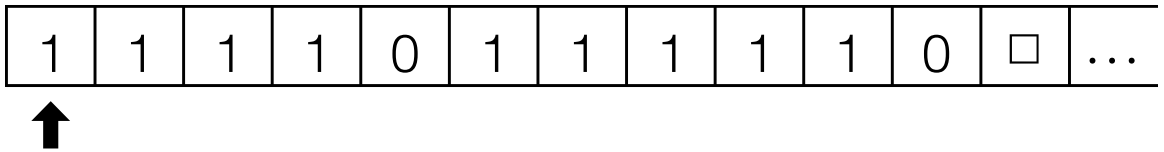
TM Addition Example

We want to create a TM to add two numbers

Use a simple tape alphabet {$0,1$} plus the blank symbol

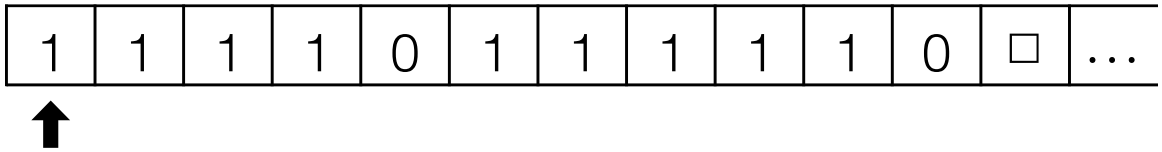
Represent a number n by a string of n+1 1's terminated by a 0

Input to compute 3+4 looks like this:

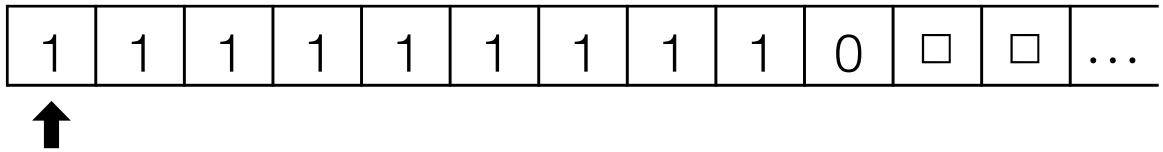


Result of the TM Addition Example

Note that the TM is initially positioned on the leftmost cell of the input.



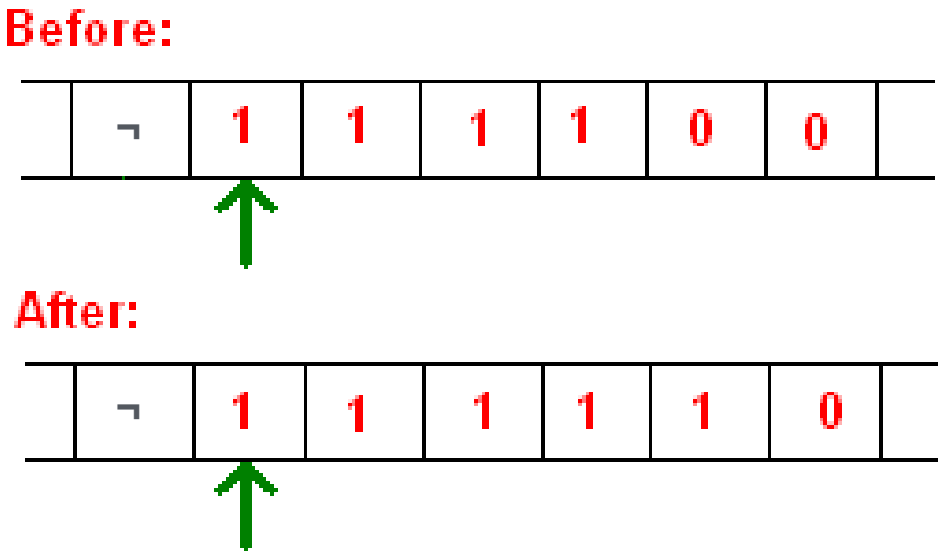
When the TM halts in the accept state, it must also be on the leftmost cell of the output.



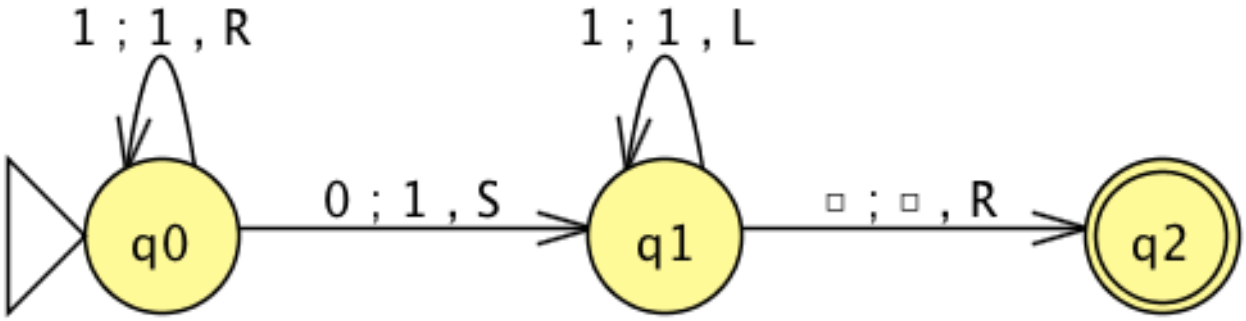
Breaking Down the Addition Problem

(Computer Scientists like to simplify)

A successor TM appends a 1 to the right end of a string of 1's



The Successor Subroutine



The TM starts in the initial state q_0, positioned on the leftmost of a string of 1's

If it sees a 1, it writes a 1, moves right, and stays in state q_0

If it sees a 0, it writes a 1 and moves to q_1

It then returns to the left

Successor Subroutine State Transitions

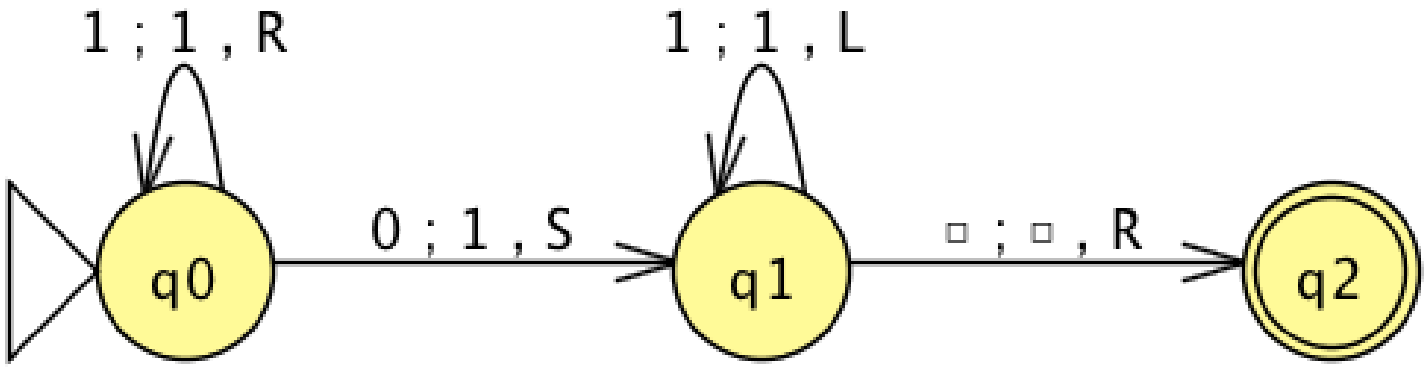
< original state, input, output, new state, action >

< q_0, 1, 1, q_0, R >

< q_0, 0, 1, q_1, S >

< q_1, 1, 1, q_1, L >

< q_1, □, □, q_2, R >

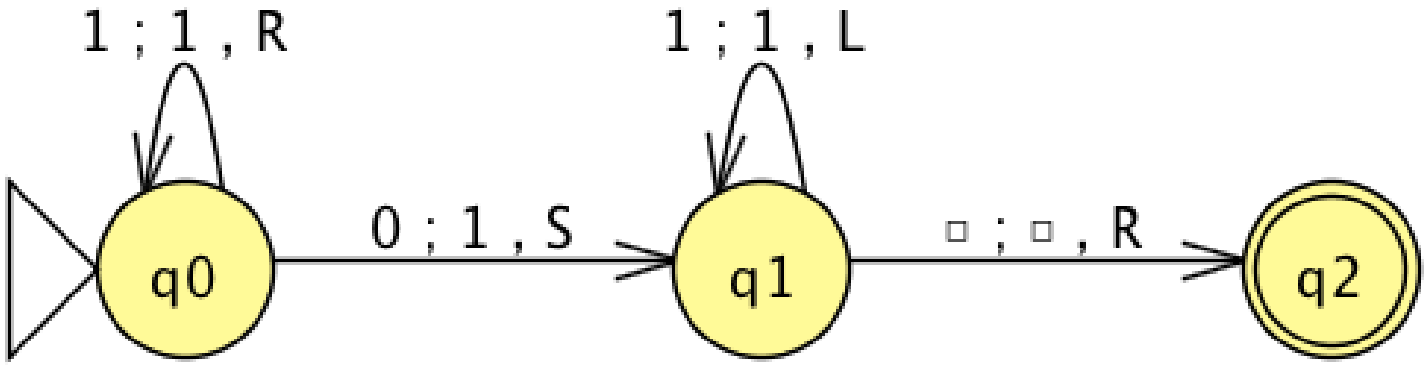


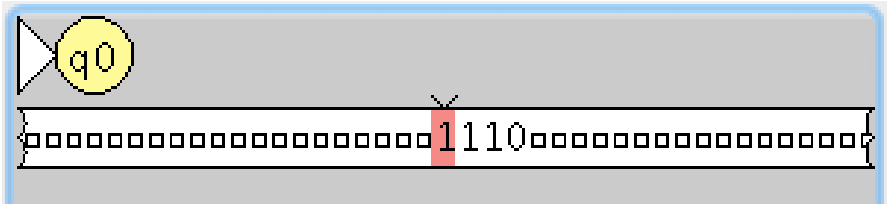
TM State Interpretation

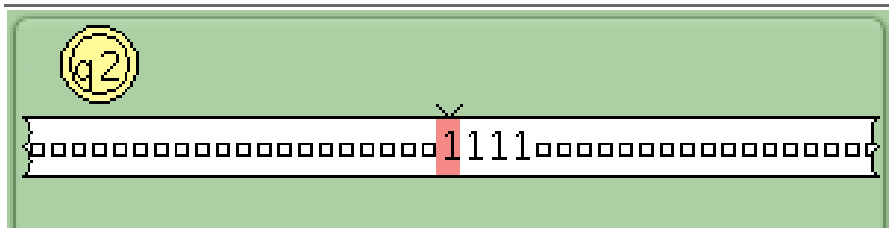
q_0 – the TM has seen only 1's so far and is scanning right

q_1 – the TM has seen its first 0 and is scanning left

q_2 – the TM has returned to the leftmost 1 and halts.



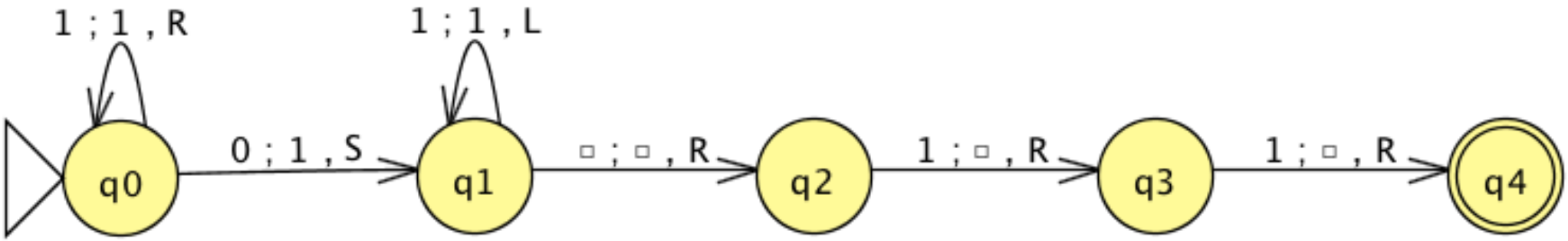




From Successor TM to Addition TM

The successor TM will join the two blocks of n+1 1's and m+1 1's into a single block of n+m+3 1's

To complete the computation, knock off two 1's from left end (states q_2 and q_3)



TM Configurations

The *configuration* of a Turing machine is the current setting

- Current state
- Current tape contents
- Current tape location

Notation uqv

- Current state = q
- Current tape contents = uv
  - Only ~ symbols after last symbol of v
- Current tape location = first symbol of v

Configuration C_1 Yields C_2

C_1 yields C_2 if the TM can legally go from C_1 to C_2 in one step

- Assume $a,b \in \Gamma$ and $u,v \in \Gamma^*$
- uaq_ibv yields uq_kacv if $\delta(q_i,b)=(q_k,c,L)$
- uaq_ibv yields $uacq_kv$ if $\delta(q_i,b)=(q_k,c,R)$

Special cases if head is at beginning or end of tape

- $q_ibv\sim$ yields $q_kcv\sim$ if $\delta(q_i,b)=(q_k,c,L)$ and tape head is at beginning of tape
- $uaq_i\sim$ yields $uacq_k\sim$ if $\delta(q_i,\sim)=(q_k,c,R)$

Special Configurations

Start configuration

- $q_0 w$

Halting configurations

- Accepting configuration: $uq_{accept}v$
- Rejecting configuration: $uq_{reject}v$

  $u, v \in \Gamma^*$

Strings Accepted by a TM

A Turing machine M accepts input sequence w if a sequence of configurations C_1, C_2, …, C_k exist, where

- **C_1 is the start configuration of M on input w**
- **Each C_i yields C_{i+1} for i = 1, 2, …, k-1**
- **C_k is an accepting configuration**

Language of a TM

The language of M, denoted L(M), is

$$L(M) = \{w \mid M \text{ accepts } w\}$$

A language is called Turing-recognizable if some Turing machine recognizes it

M halts and **Accepts** for every string in the language

M can either **Reject** or **Run Forever** for strings not in the language

Deciders

A Turing machine is called a decider if *every* string in Σ^* is either accepted or rejected

A language is called Turing-decidable if some Turing machine decides it

- **These languages are often just called decidable**

Deciders will always halt

Example

Write a TM that accepts all strings of the form 10100100010000 1…

- **Start with a 1**
- **End with a 1**
- **Progressively more 0's between consecutive 1's**

`1010010001`

Design

Check first symbol is a 1

- If not reject

Move right and check if second symbol is a 0

- If not reject
- If so, replace with X and begin recursion

1x10010001~

Recursion (High Level)

Go back and forth on either side of each `1`

- Replace `0` on right side of `1` with an X
- Replace X on left side of `1` with a Y

After all X's on left side of `1` are replaced with Y's, there should be exactly one `0` on the right side that has not been X'ed

- If not, reject
- If so, replace with X and recurse

`1Y1YY1XXX1~`

Exit Condition

If you begin to look for the next group of 0's and reach a ~ then accept

1010010001

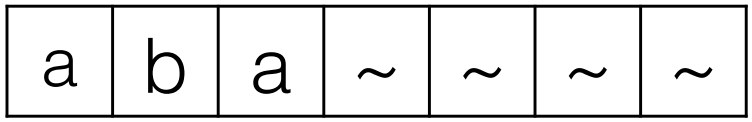
Design a Turing machine that accepts…

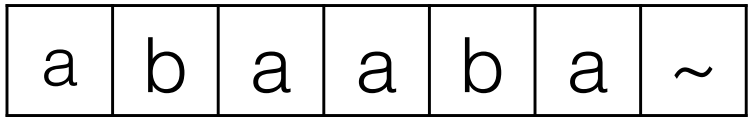
1. any string in $\{a,b\}^*$ after first making a copy on the tape
2. language $\{w \in \{a,b\}^* \mid |w| \text{ is even}\}$
3. " $\{ a^n b^m \mid n>m \}$
4. " $\{a^n b^m a^{n+m} \mid n \geq 0 \text{ and } m \geq 1\}$
5. " $\{ ww^R \mid w \in \{a,b\}^* \}$
6. " $\{ w \in \{a,b\}^* \mid w \text{ has more a's than b's} \}$

TM 1

Design a Turing machine to accept any string in $\{a,b\}^*$ after making a copy of it on the tape

- **The tape will start with w**
- **After TM processes the string, the tape should read ww**





TM 1 Design

Read symbol

- If a, replace with x, move to end and write x if first time, otherwise write a
- If b, replace with y, move to end and write y if first time, otherwise write b

Move left to first x or y

Move left to next x or y

- If x, replace with a and move right
- If y, replace with b and move right

Read symbol at tape head

- If x, replace with a and accept
- If y, replace with b and accept
- If neither, loop to beginning

a	b	a	~	~	~	~

TM 1 Design

Read symbol

- If a, replace with x, move to end and write x if first time, otherwise write a
- If b, replace with y, move to end and write y if first time, otherwise write b

Move left to first x or y

Move left to next x or y

- If x, replace with a and move right
- If y, replace with b and move right

Read symbol at tape head

- If x, replace with a and accept
- If y, replace with b and accept
- If neither, loop to beginning

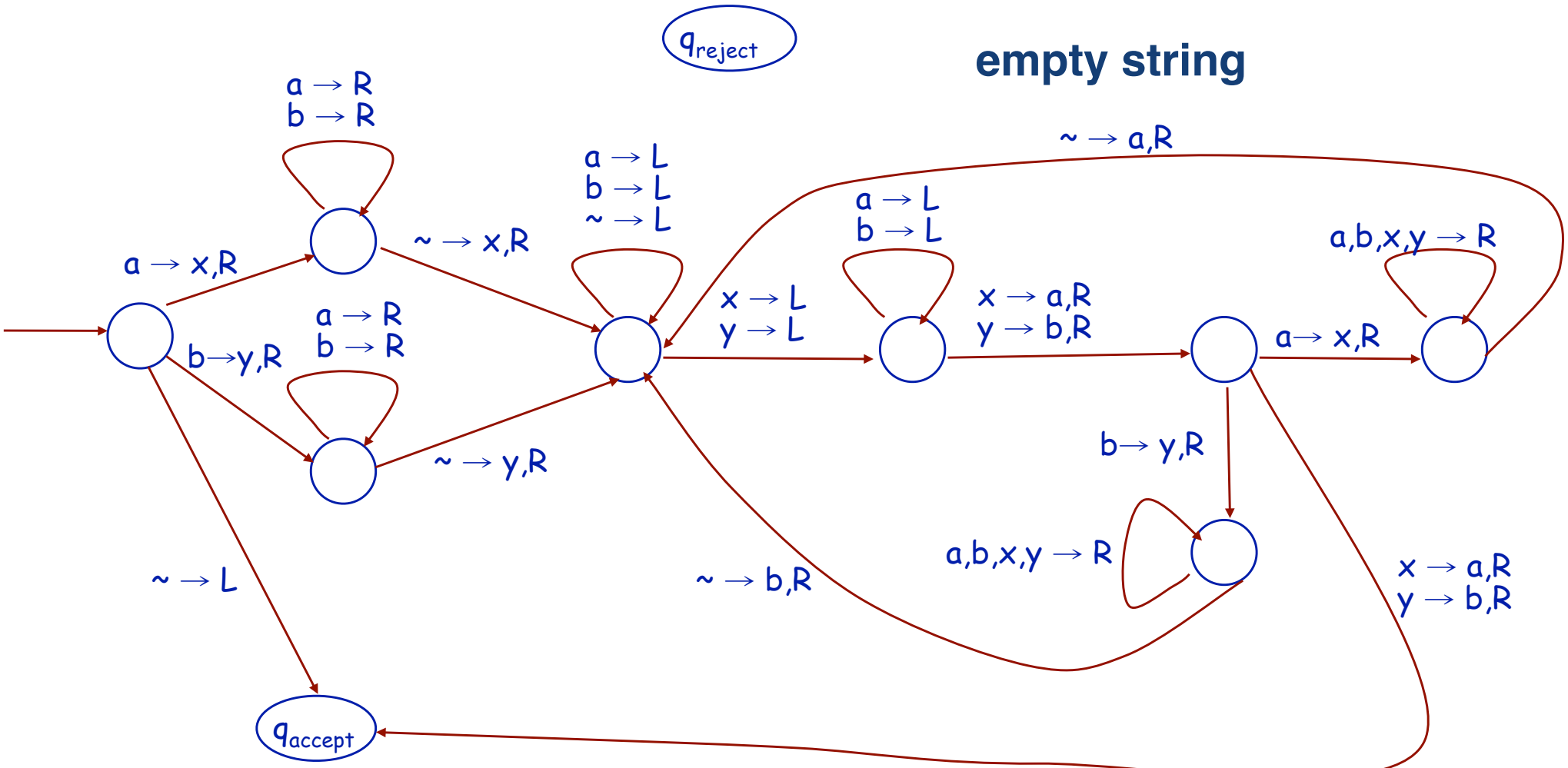
a	b	a	a	b	a	~

accept

a	b	a	~	~	~	~

Copy Machine

start state q_1; accept state q_{accept}; reject state q_{reject}



scan right to end; scan left to x or y; write a,b,x,y

Variants of Turing Machines

Robustness of model

- Varying the model
  does not change the power

Simple variant of TM model

- Add “stay put” direction

Other variants

- More tapes (Multitape)
- Nondeterministic

Multitape Turing Machines

Same as standard Turing Machine, but have several tapes

TM definition changes only in definition of δ

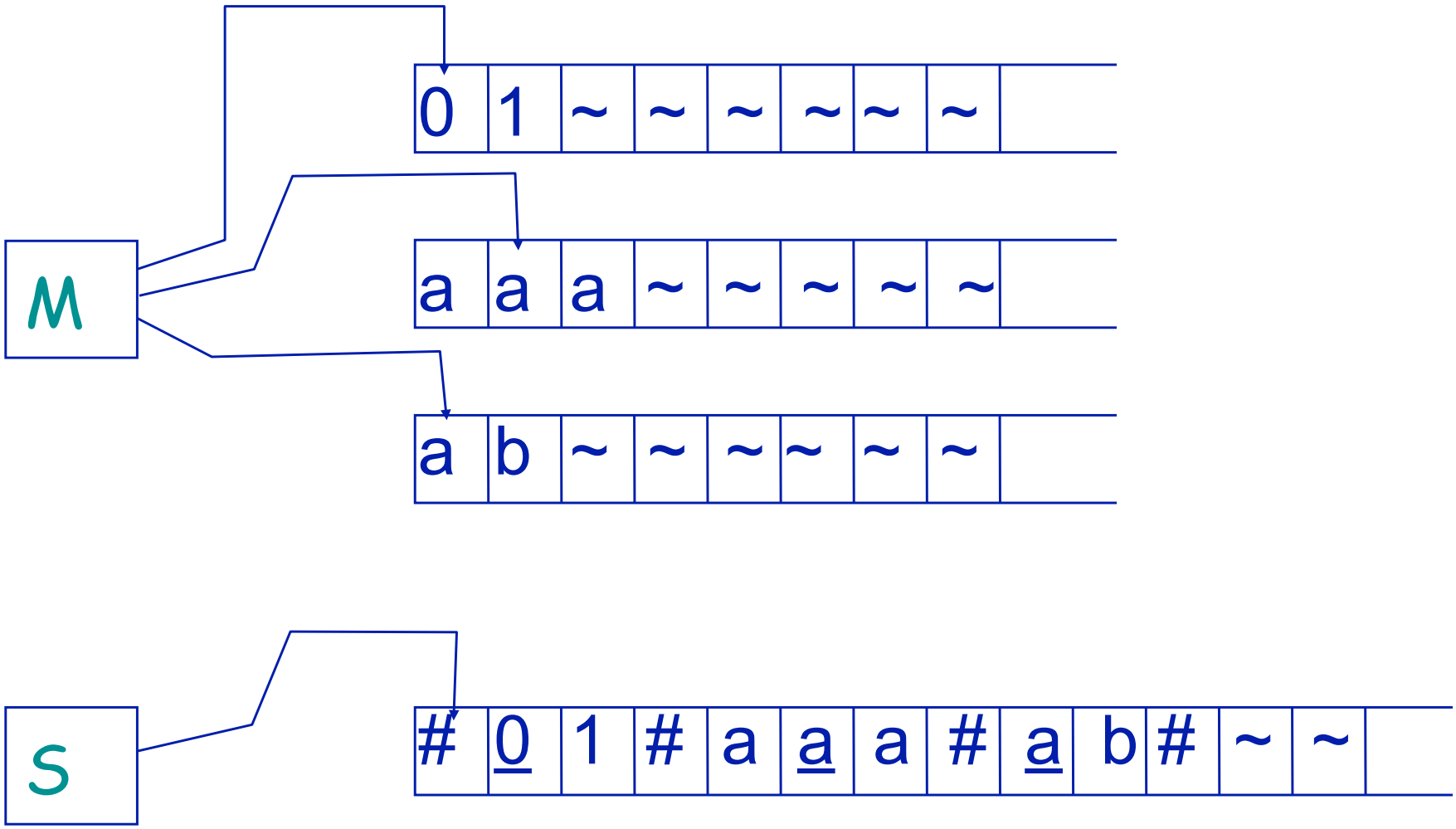
$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$

Equivalence of Machines

Theorem: **Every multitape Turing machine has an equivalent single tape Turing machine**

Proof method: **construction**

Equivalent Machines



Simulating k-tape Behavior

Single tape start string is

$$\#w\#\underline{\sim}\#\ldots\#\underline{\sim}\#$$

Each move proceeds as follows:

- Start at leftmost slot
- Scan right to $(k+1)^{st}$ # to find symbol at each virtual tape head
- Do a second pass making updates indicated by the k-tape transition function
- When a virtual head moves onto a #, shift the string to right

Corollary

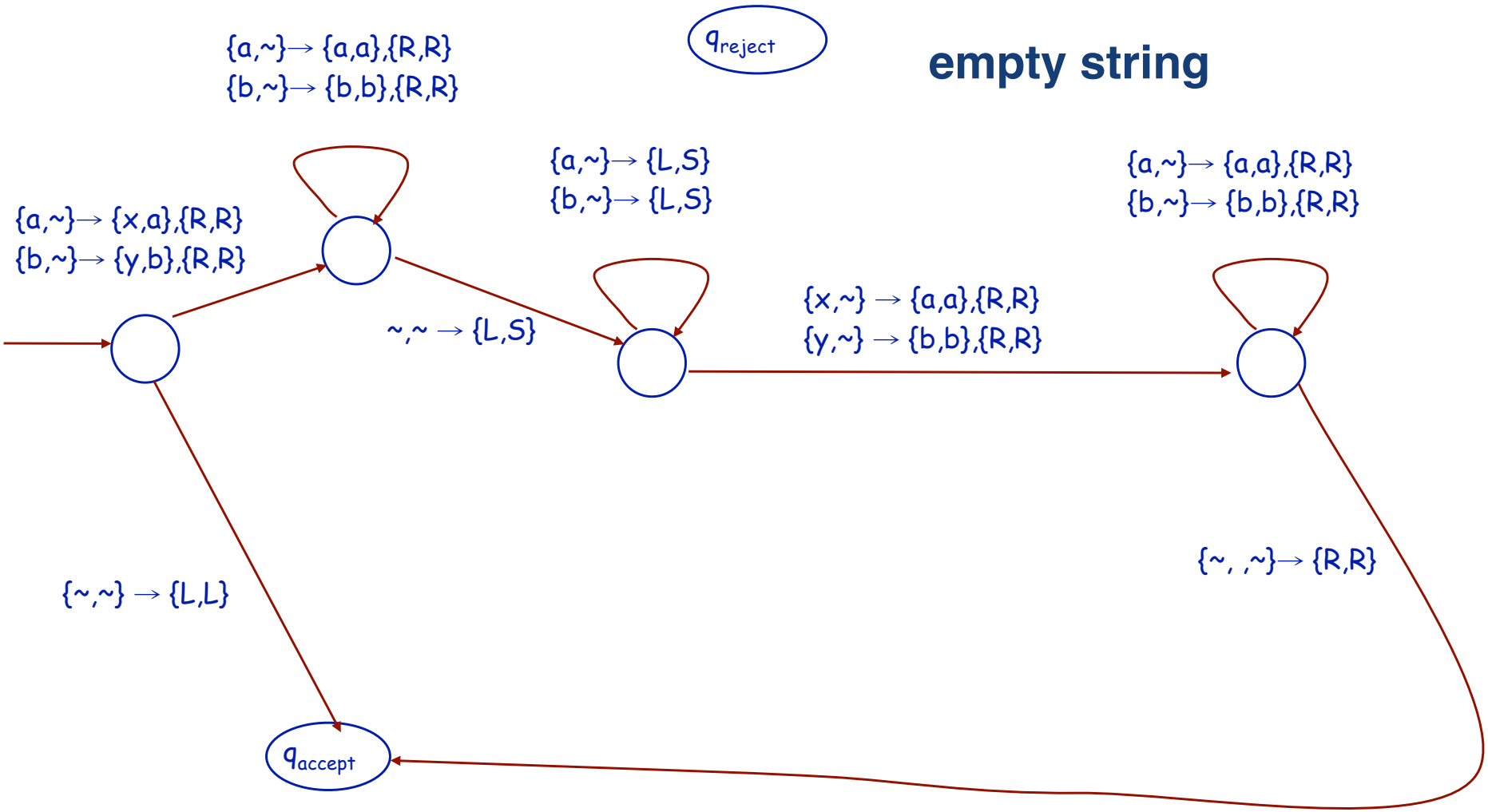
Corollary: A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.

Example

Using 2-tape Turing machine, write a copy machine

- **Copy tape 1 to tape 2**
- **Move tape 1 to beginning**
- **Copy tape 1 to tape 2**
- **Accept**

Copy Machine



Nondeterministic Turing Machines

Same as standard Turing machines, but may have one of several choices at any point

$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L,R\})$$

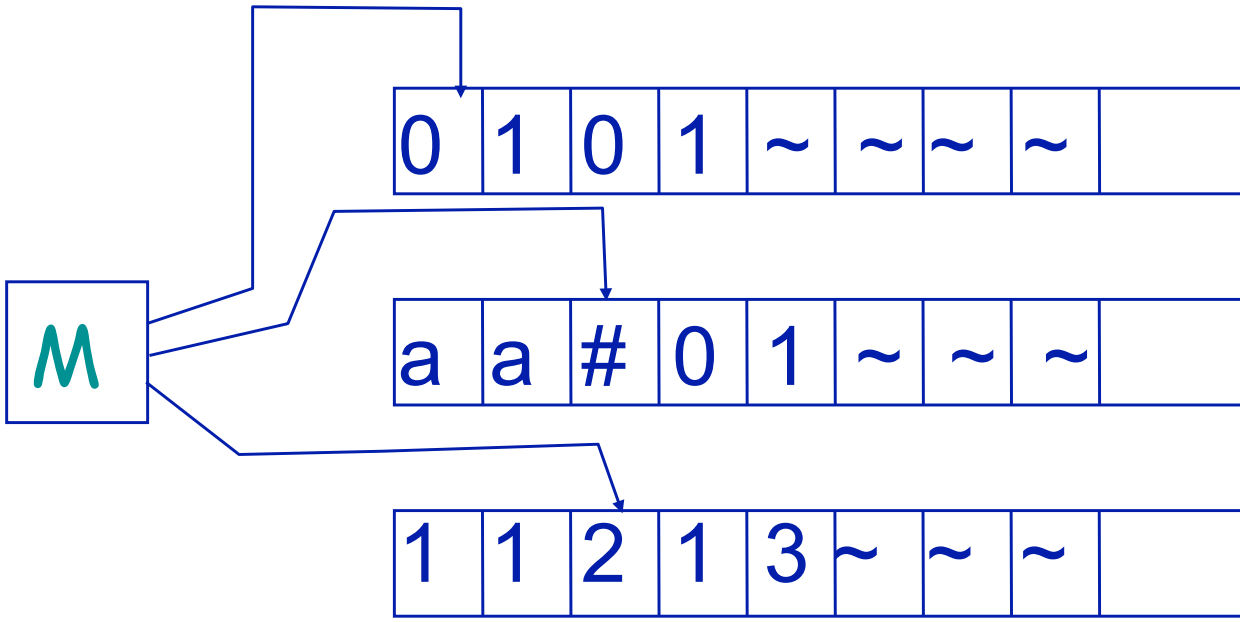
Equivalence of Machines

Theorem: **Every nondeterministic Turing machine has an equivalent deterministic Turing machine**

Proof method: **construction**

Proof idea: **Use a 3-tape Turing machine to deterministically simulate the nondeterministic TM. First tape keeps copy of input, second tape is computation tape, third tape keeps track of choices.**

Proof Idea

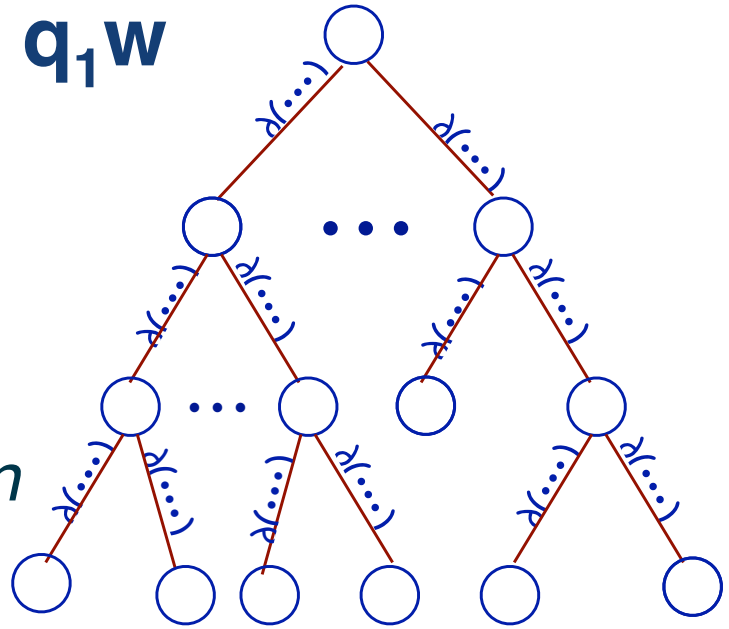


Try new decision paths until the string is accepted

Intuition

Consider the nondeterministic calculations as a tree

- Each node represents a configuration
- A node for configuration C_1 has one child for each configuration C_2 such that C_1 yields C_2
- Root of tree is configuration q_1w
- A configuration may appear more than once in the tree



Tree Representation

Example

Nondeterministic TM that accepts

$$\{ ww \mid w \in \{a,b\}^* \}$$

- **Use 2 tapes**
- **Copy input to tape 2**
- **Position heads at beginning of tapes**
- **Move both heads right simultaneously**

Nondeterministic Solution

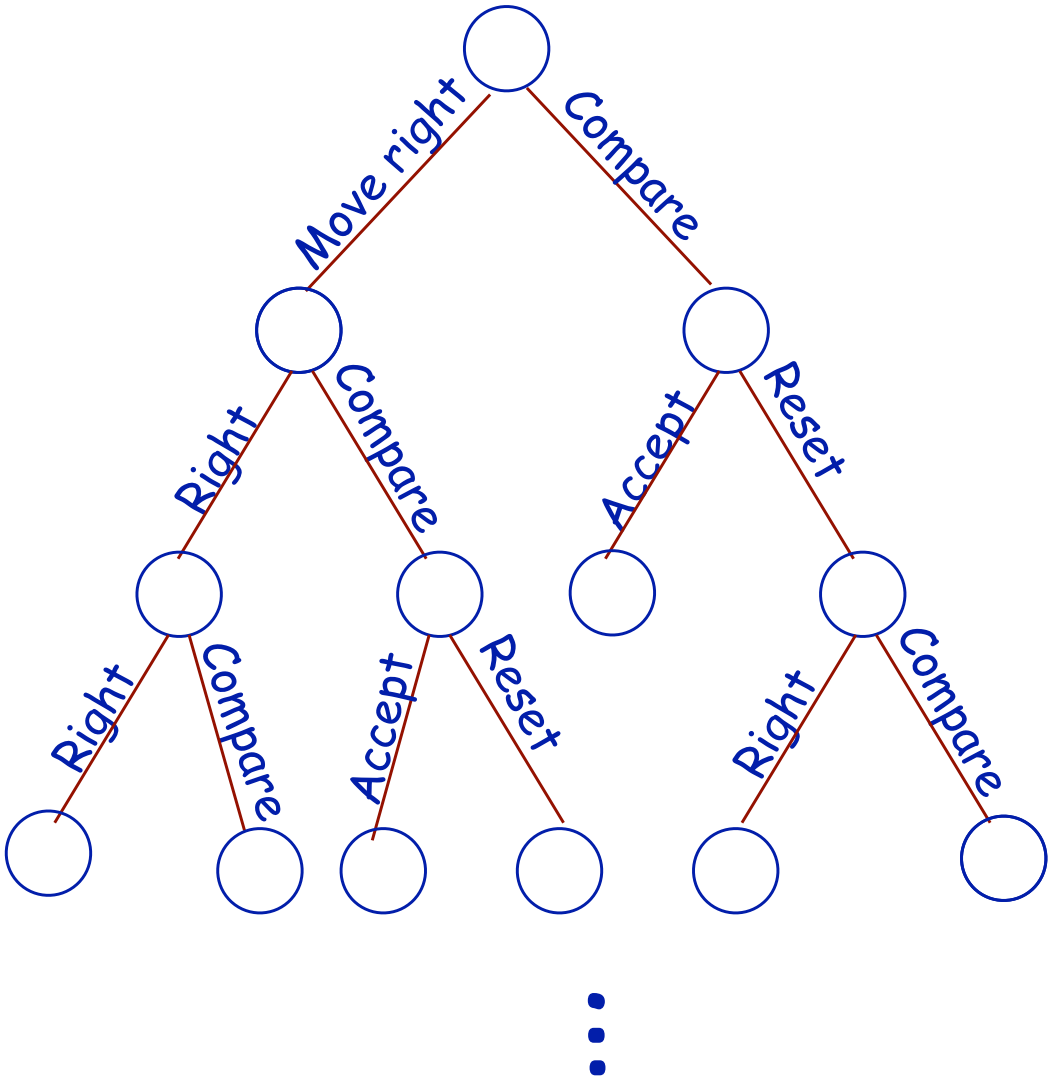
Nondeterministically choose the midpoint

- Mark this point on tape 2 and return tape 2's head to beginning

Compare strings

- If tape head points to ~ on tape 1 and midpoint marker on tape 2 then accept
- Otherwise, if all possible midpoints have been tried then reject
- Otherwise, try a new midpoint

Tree Representation



Deterministic Equivalent

Assume midpoint is at beginning

- If so accept
- If not …

Assume midpoint is after first symbol

- If so accept
- If not …

Assume midpoint is after second symbol

- If so accept
- If not …

etc. …

How Should It Search the Tree?

Breadth first search

- Search all possibilities involving k steps before searching any possibilities involving (k+1) steps

What's wrong with depth first search?

- If some sequence of choices results in never reaching a halting state, we will never get to the accept state

When Does It Halt?

When it reaches an accept state

- Return **accept**

Will It Halt on Strings in the Language?

Yes

- Let b be the largest number of children of any node
  - Can we be sure b is finite?
- Let k be the minimum number of steps it takes to get to the accept state
- This method will take at most b^k steps to get to the accept state

What about strings not in the language?

Won’t halt

- That’s okay

Equivalence of Approaches

Corollary: A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.

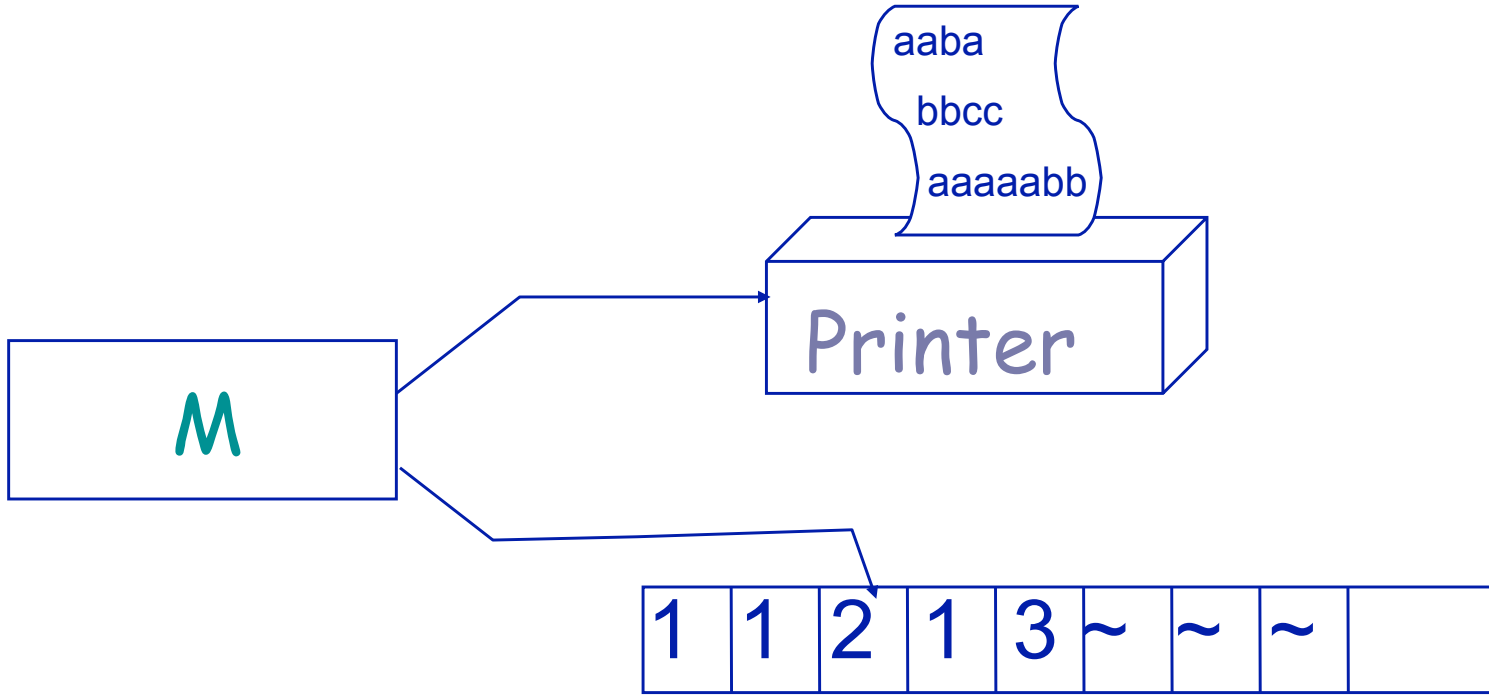
Equivalence of Approaches

Corollary: A language is Turing-decidable if and only if some nondeterministic Turing machine decides it.

Proof: Very similar to the proof we just outlined

Enumerators

Instead of reading an input and processing it, enumerators start with an empty tape and print out strings from Σ^*



Machine Equivalence

Theorem: **A language is Turing-recognizable if and only if it is enumerated by some enumerator.**

Proof technique:

Construction in each direction

TM Accepts Enumerator Language

TM = "On input w:

- **Run enumerator E. Every time E prints a string, compare it to w.**
- **If w appears in the output, accept."**

Enumerator Accepts TM Language

Let $s_1, s_2, s_3, \ldots$ be all the strings in Σ^*

E = "Ignore the input.

- For i = 1, 2, 3, …
  - Run M for i steps on each input $s_1, s_2, \ldots, s_i$
  - Whenever M accepts a string, print it "

What Is an Algorithm?

Intuitively, an algorithm is anything that can be simulated by a Turing machine (Church-Turing Thesis)

- Inputs can be represented as strings
  - Graphs
  - Polynomials
  - Automata
  - Etc.