Turing Machines

CSCI 2670

UGA

Fall 2014

Outline

- Turing Machines (TMs)
- Multitape Turing Machines
- Nondeterministic Turing Machines
- Enumerators
- The Church-Turing Thesis

Turing Machines

- Regular and CF languages are both recognized by a particular type of machine.
  - Regular: Finite Automata (Deterministic, Nondeterministic).
  - Context Free: Pushdown Automata.
- Both types of machine are limited.
  - Regular: No memory.
  - Context Free: A restricted type of memory (first-in, last-out).
- Neither model is acceptable as a general model of computation
  - Clearly some machine should be able to recognize strings in $\{w\#w | w \in \{0,1\}^*\}$ (which is not a CFL).
  - PDAs and DFAs can't.
- **Turing Machines** (TMs) are.
  - They are the canonical model for computation.
  - To be computable means to be computable by a Turing Machine (this is the **Church-Turing Thesis**).

Turing Machines: Informal

- Like DFAs and PDAs, TMs consists of a set Q of states.
- At any stage of computation, the TM occupies a single state.
- A TM processes strings over a given input alphabet Σ.
- The strings are written on a read-write tape (infinite to the right), which is divided into cells.
- The TM has a tape head; at any stage of computation, the tape head appears over a single tape cell.
- Initially, the tape head appears over the leftmost cell; the input string appears written on the tape.
- In each step, the tape head reads a symbol, writes a symbol, and then moves one cell to the left or right.
- Special accepting and rejecting states exist. The machine halts immediately after entering one of these.

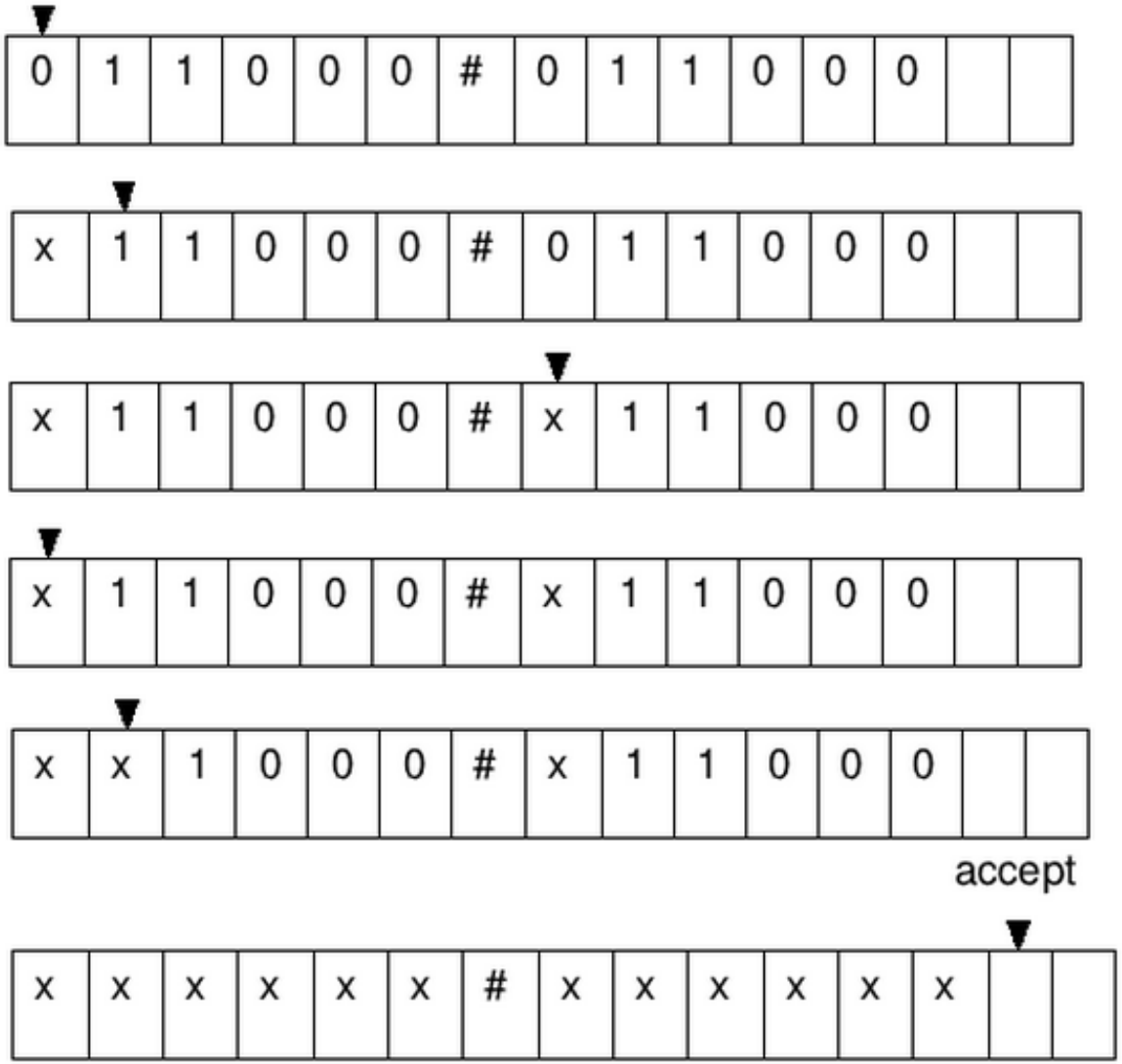
Turing Machines: Informal

Example

M_1 accepts strings of $\{w\#w | w \in \{0,1\}^*\}$. It works as follows:

1. At the start, the input string appears on the tape; the tape-head is over the leftmost cell.
2. Mark the symbol a below the tape head, remembering if it's a 0 or 1.
3. Move right, finding the first unmarked symbol b after the #. If $a \neq b$, or if an empty cell is found before a #, then reject.
4. Otherwise, mark b and return to the left, stopping at the first marked symbol after the #. Then move one cell to the right.
5. Repeat this process until all symbols to the left of the # are marked. Then move to the right, looking for unmarked symbols after the #.
6. If any are found, reject. If instead an empty cell is encountered, accept.

Turing Machines: Informal



Turing Machines: Formal

Definition

A **Turing machine** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ such that

1. Q is a finite set of states;
2. Σ is the input alphabet not containing the blank symbol $\sqcup$;
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$;
4. $\delta : (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})$ is the transition function;
5. $q_0 \in Q$ is the start state;
6. $q_{accept} \in Q$ is the accept state;
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

Turing Machines: Questions

Examine the formal definition of a Turing machine to answer the following questions, and explain your reasoning.

1. Can a Turing machine ever write the blank symbol $\sqcup$ on its tape?
2. Can the tape alphabet Γ be the same as the input alphabet Σ?
3. Can a Turing machine's head ever be in the same location in two successive steps?
4. Can a Turing machine contain just a single state?

Configurations

Definition

The state, current tape contents, and tape head position constitute a machine **configuration**. An **accepting configuration** (**rejecting configuration**) is one in which the machine is in state q_{accept} (q_{reject}); these are **halting configurations**. The **start configuration** has the tape head in the leftmost position, and the input string is written on the tape.

$01010q_7010111$ indicates a configuration: 01010010111 is written on the tape, and the tape head is in state q_7 and appears above the 6th symbol.

Let $a, b, c \in \Gamma$ and $u, v \in \Gamma^*$.

- Configuration uaq_ibv **yields** uq_jacv if $\delta(q_i, b) = (q_j, c, L)$.
- Configuration uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$.

The left end of the tape is handled differently.

- (Leftmost) q_ibv yields q_jcv if $\delta(q_i, b) = (q_j, c, L)$.
- (Leftmost) q_ibv yields cq_jv if $\delta(q_i, b) = (q_j, c, R)$.

The right edge of the tape contents needs no special handling.

Turing Machines: Acceptance

Definition

A Turing Machine M accepts a string w if there exists a sequence of configurations $C_1, C_2, \ldots C_n$ such that

1. C_1 is the starting configuration of M on input w,
2. C_n is an accepting configuration,
3. for each $1 \leq i < n$, C_i yields C_{i+1}.

$L(M)$, **language of** M (the language **recognized** by M), is the set of strings accepted by M.

Turing Recognizable and Decidable Languages

Definition

A language L is **Turing–recognizable** (also called **recursively enumerable**) if there is a Turing machine M such that $L(M) = L$.

- It is possible for a TM to never reach a halting configuration. On some given input w, it might instead **loop**.
- There are three possible outcomes for a TM on an input: **accept**, **reject**, **loop**.
- A TM M that halts on every input is called a **decider**.
- A decider that recognizes L also is said to **decide** L.

Definition

A language L is **Turing–decidable** or simply **decidable**) if there is a Turing machine M that decides L.

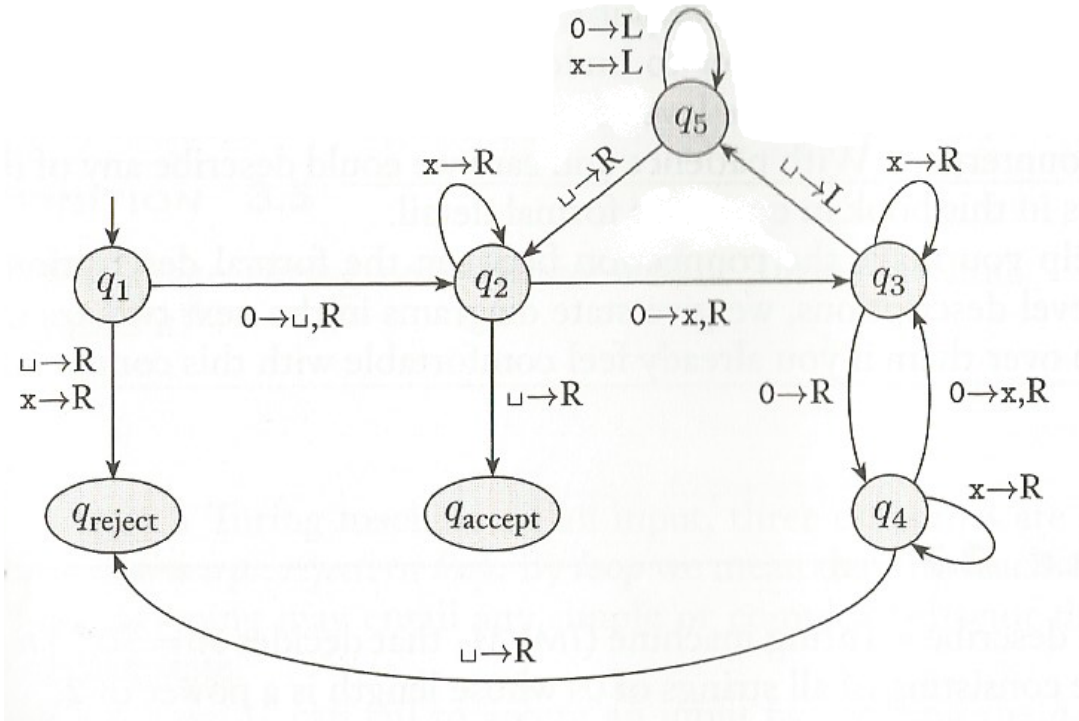
Example

Example

A Turing machine M_2 that decides $A = \{0^{2^n} | n \geq 0\}$, the language consisting of all strings of 0s whose length is a power of 2.
M_2 = "On input string w:

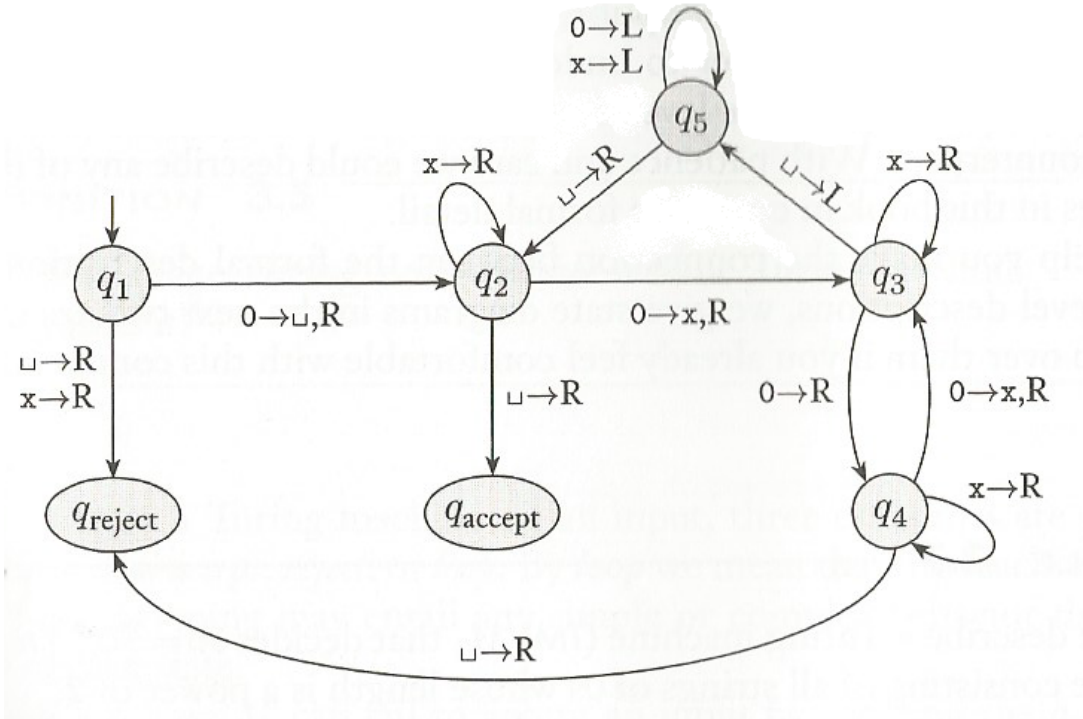
1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, accept.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject.
4. Return the head to the left–hand end of the tape.
5. Go to stage 1."

Example



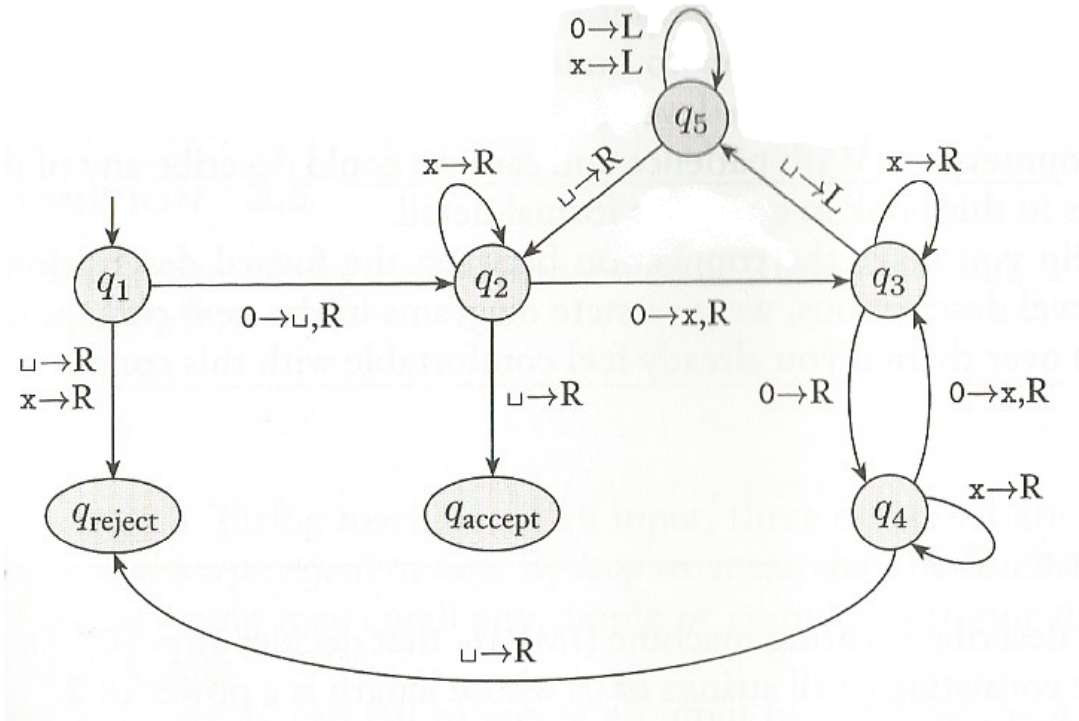
- This is a state diagram for a Turing Machine M_2.
- A label $a \to b, R$ on edge (q_i, q_j) means that if M_2 is in state q_i and is reading an a, then it should write a b, move right, and then enter state q_j.
- $a \to R$ is shorthand for $a \to a, R$.

Example



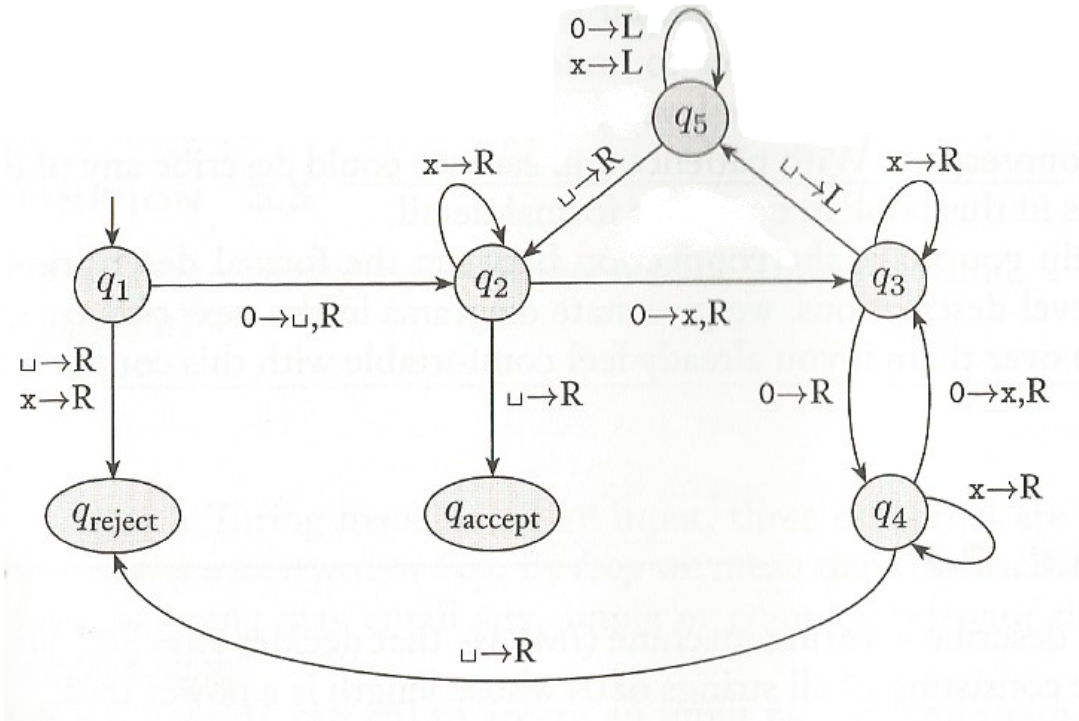
- The machine M_2 counts 0s, determining whether the input is a power of 2 (0^{2^n} for some $n \geq 0$).
- It moves to the right, crossing off every second 0.
- If it reads a single 0, followed by the empty space, it accepts ($0^1 = 0^{2^0}$).
- Note that it replaces the first 0 with a ␣.

Example



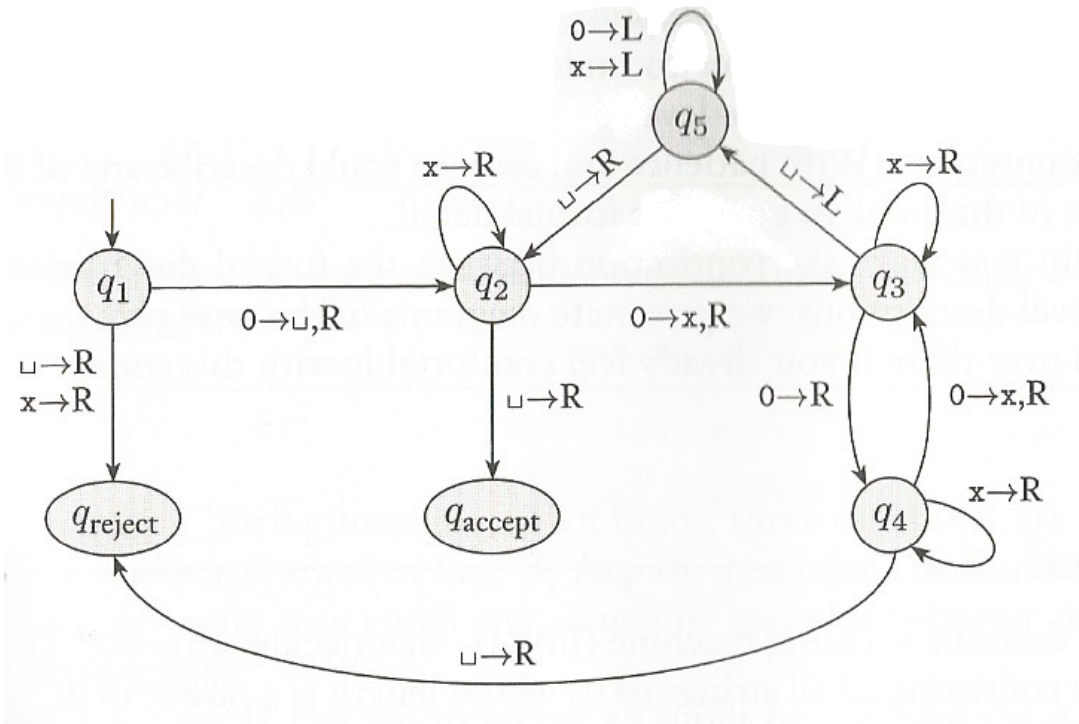
- If in state q_2, it reads another 0, it x-es it out and moves to state 3.
- It then alternates between q_3 and q_4, x-ing out every other 0.
- Existing x's are passed over.

Example



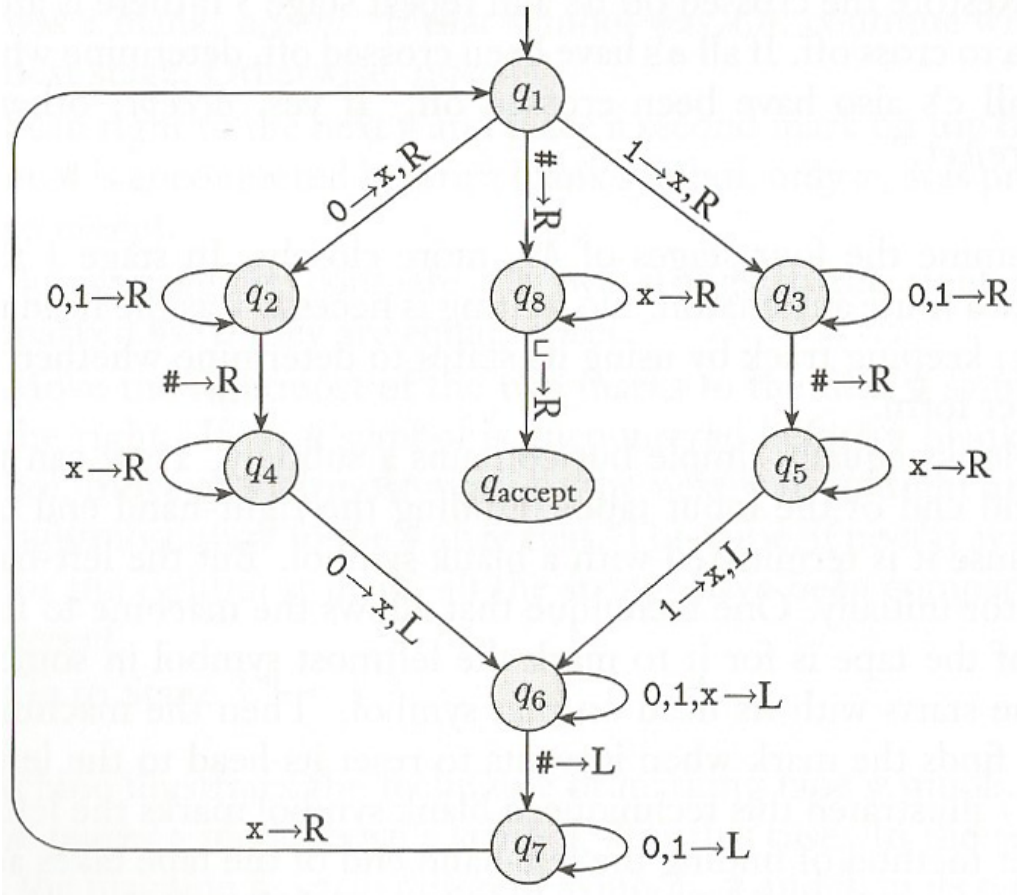
- If it reaches a blank while in state q_4, it rejects.
- If it reaches a blank while in state q_3, it moves to state q_5.
- In q_5, it moves left, moving to state q_2 only when a ␣ is found.
- From here, it repeats the previous steps.

Example



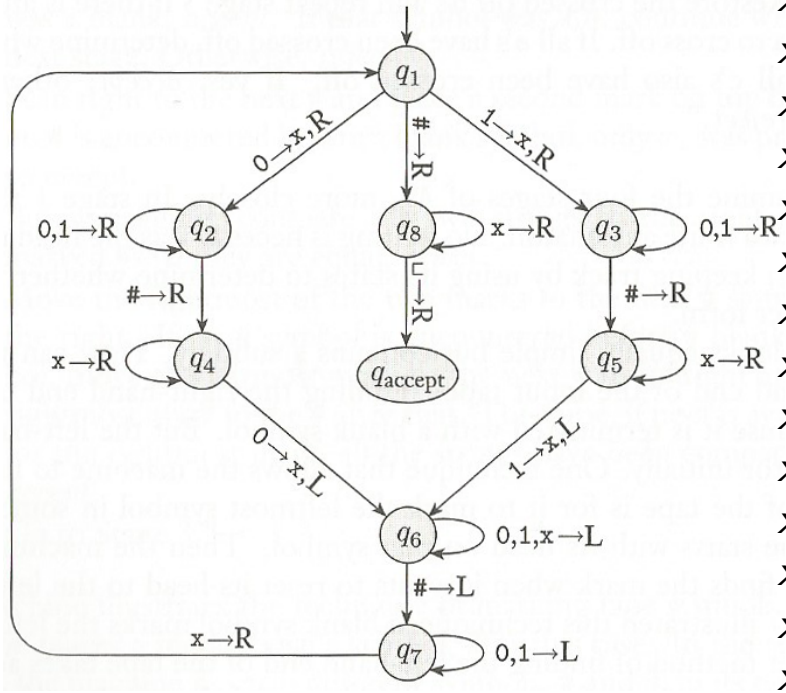
$q_1 0000␣$
$␣q_2 000␣$
$␣xq_3 00␣$
$␣x0q_4 0␣$
$␣x0xq_3 ␣$
$␣x0q_5 x␣$
$␣xq_5 0x␣$
$␣q_5 x0x␣$
$q_5 ␣ x0x␣$
$␣q_2 x0x␣$
$␣xq_2 0x␣$
$␣xxq_3 x␣$
$␣xxxq_3 ␣$
$␣xxq_5 x␣$
$␣xq_5 xx␣$
$␣q_5 xxx␣$
$q_5 ␣ xxx␣$
$␣q_2 xxx␣$
$␣xq_2 xx␣$
$␣xxq_2 x␣$
$␣xxxq_2 ␣$
$␣xxx ␣ q_{accept}$

Example



- This machine M_1 recognizes $\{w\#w | w \in \{0,1\}^*\}$.
- The diagram is abbreviated. No reject state is given.
- $\Sigma = \{0, 1, \#\}$ and $\Gamma = \Sigma \cup \{x, \sqcup\}$.
- Each branch ensures a 0 or 1 is matched with a symbol after the #.

Example



$$
\begin{array}{lcll}
q_1001\#001\sqcup & \Rightarrow & xq_201\#001\sqcup & \Rightarrow \\
x0q_21\#001\sqcup & \Rightarrow & x01\#q_4001\sqcup & \Rightarrow \\
x01q_6\#x01\sqcup & \Rightarrow & x0q_71\#x01\sqcup & \Rightarrow \\
xq_701\#x01\sqcup & \Rightarrow & q_7x01\#x01\sqcup & \Rightarrow \\
xq_101\#x01\sqcup & \Rightarrow & xxq_21\#x01\sqcup & \Rightarrow \\
xx1q_2\#x01\sqcup & \Rightarrow & xx1\#q_4x01\sqcup & \Rightarrow \\
xx1\#xq_401\sqcup & \Rightarrow & xx1\#q_6xx1\sqcup & \Rightarrow \\
xx1q_6\#xx1\sqcup & \Rightarrow & xxq_71\#xx1\sqcup & \Rightarrow \\
xq_7x1\#xx1\sqcup & \Rightarrow & xxq_11\#xx1\sqcup & \Rightarrow \\
xxxq_3\#xx1\sqcup & \Rightarrow & xxx\#q_5xx1\sqcup & \Rightarrow \\
xxx\#xq_5x1\sqcup & \Rightarrow & xxx\#xxq_51\sqcup & \Rightarrow \\
xxx\#xq_6xx & \Rightarrow & xxx\#q_6xxx & \Rightarrow \\
xxxq_6\#xxx & \Rightarrow & xxq_7x\#xxx & \Rightarrow \\
xxxq_1\#xxx & \Rightarrow & xxx\#q_8xxx & \Rightarrow \\
xxx\#xq_8xx & \Rightarrow & xxx\#xxq_8x\sqcup & \Rightarrow \\
xxx\#xxxq_8\sqcup & \Rightarrow & xxx\#xxxq_{accept}\sqcup & \Rightarrow
\end{array}
$$

Example

Example

A Turing machine M_3 is doing some elementary arithmetic. It decides the language $C = \{a^i b^j c^k | i \times j = k \text{ and } i, j, k \geq 1\}$.
M_3 = "On input string w:

1. Scan the input from left to right to be sure that it is a member of $a^* b^* c^*$ and reject if it is not.
2. Return the head to the left–hand end of the tape.
3. Cross off an a and scan to the right until a b occurs. Shuttle between the b's and the c's, crossing off one of each until all b's are gone.
4. Restore the crossed off b's and repeat stage 3 if there is another a to cross off. If all a's are crossed off, check on whether all c's also are crossed off. If yes, accept; otherwise, reject."

Example

Example

A Turing machine M_4 is solving what is called the element distinctness problem. It is given a list of strings over $\{0, 1\}$ separated by $\#s$ and its job is to accept if all the strings are different. The language is
$E = \{\#x_1\#x_2\#\cdots\#x_m |$ each $x_i \in \{0, 1\}^*$ and $x_i \neq x_j$ for each $i \neq j\}$
$M_4 =$ "On input string w:

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, accept. If that symbol was a $\#$, continue with the next stage. Otherwise, reject.
2. Scan right to the next $\#$ and place a second mark on top of it. If no $\#$ is encountered before a blank symbol, only x_1 was present, so accept.
3. By zig–zagging, compare the two strings to the right of the marked $\#s$. If they are equal, reject.
4. Move the rightmost of the two marks to the next $\#$ symbol to the right. If no $\#$ symbol is encountered before a blank symbol, move the leftmost mark to the next $\#$ to its right and the rightmost mark to the $\#$ after that. This time, if no $\#$ is available for the rightmost mark, all the strings have been compared, so accept.
5. Go to stage 3."

Multi-Tape Turing Machines

Definition

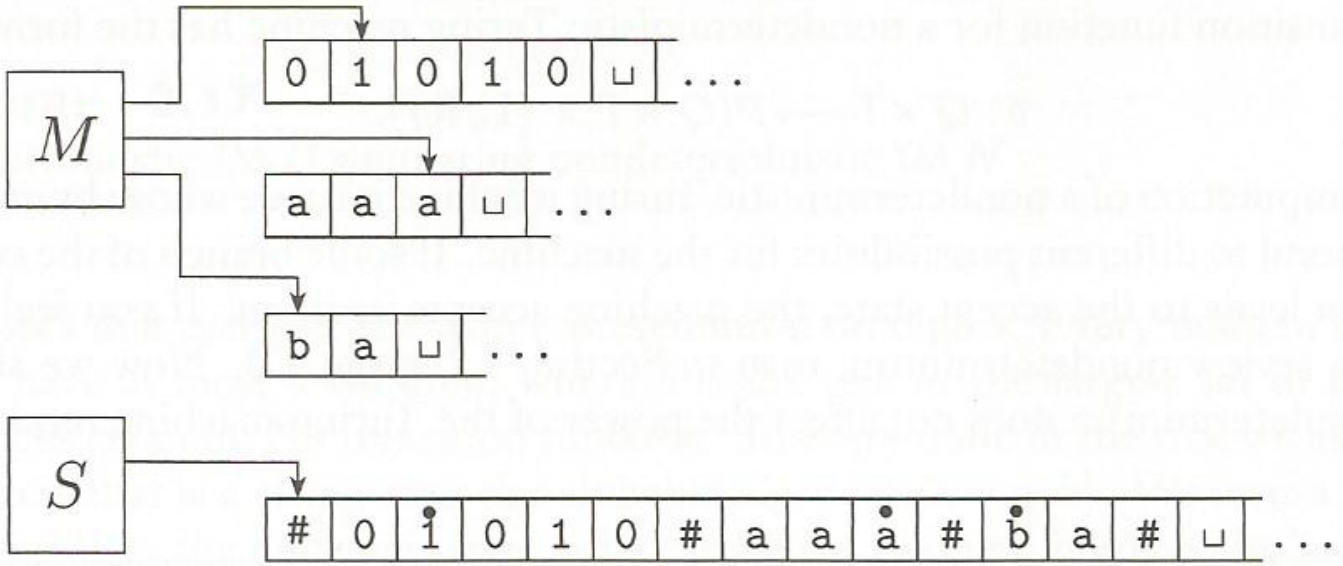
A k-**tape Turing machine** is a TM that utilizes k tapes and k tape heads. The transition function δ is defined appropriately.

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

- S means that a tape head may "stay put" rather than move L or R.
- The input string is initially written on the first tape.
- The other tapes are empty.

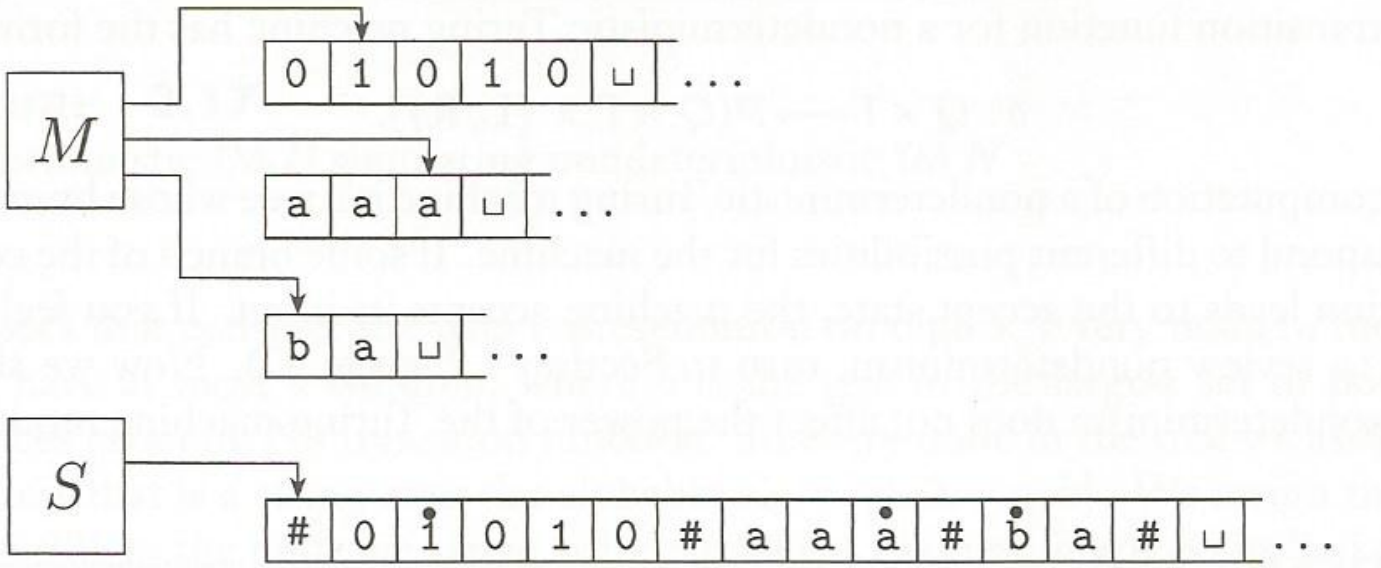
- Though it may seem otherwise, multitape machines are no more powerful than normal Turing machines.

Multi-Tape Turing Machines



- We can simulate a k-tape machine on a single tape machines.
- At any step, only a finite initial fragment of each tape is used.
- These fragments can be encoded on a single tape, divided by #s.
- To mark the position of the tape heads, we mark the corresponding cell symbol (a becomes $\dot{a}$).
- This adds new symbols to the tape alphabet (but does not harm).

Multi-Tape Turing Machines



- To simulate the behavior of the multi-tape machine, the single tape machine makes multiple passes across its tape, updating the virtual tapes appropriately.
- If a virtual tape head moves onto an unread blank space, the contents of the single tape must be shifted appropriately to make room.

Multi-Tape Turing Machines

Theorem

Every multitape Turing Machines has an equivalent single tape Turing machine.

Corollary

A language L is Turing recognizable if and only if there exists some multitape Turing Machine M such that $L(M) = L$.

Nondeterministic Turing Machines

Just as there are nondeterministic finite automata and pushdown automata, there are nondeterministic versions of Turing Machines.

Definition

A **nondeterministic Turing machine** (NTM) M is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ defined exactly like a deterministic Turing machine, save that δ has the following form.

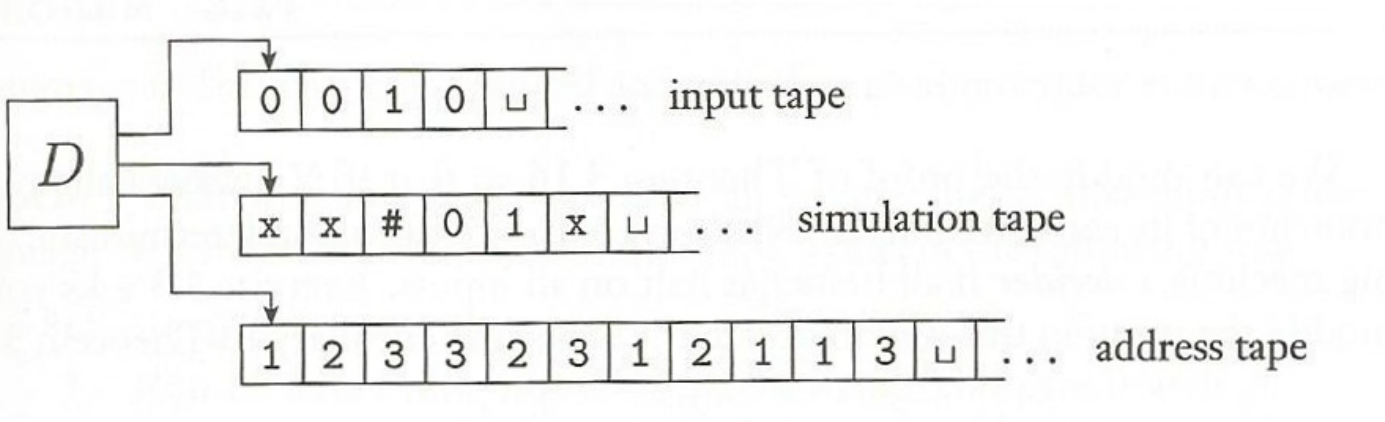
$$\delta : (Q \times \Gamma) \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

- The computations of an NTM take the form of a tree, with each configuration potentially branching off into many different configurations.
- Acceptance is defined as it is for deterministic machines: M accepts w if there exists a sequence of configurations C_1, C_2, ... C_n such that
  1. C_1 is the starting configuration of M on input w,
  2. C_n is an accepting configuration,
  3. for each $1 \leq i < n$, C_i yields C_{i+1}.

Note: only one branch of the computation tree need accept w.

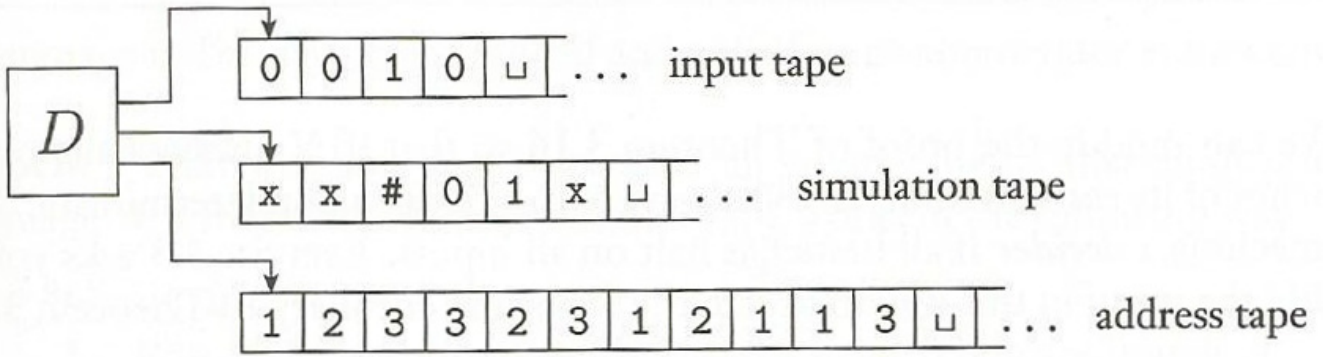
Equivalence between NTMs and DTMs

An NTM N can be simulated with deterministic TM (DTM) M.



- M uses 3 tapes:
- **Tape 1** (read only)—Records the input string. It's reused many times.
- **Tape 2**—used as N's tape.
- **Tape 3**: a counter. It stores an integer number $d_1 d_2 d_3 \ldots d_n$, where each d_i indicates a choice to make at step i. E.g., pick the 1st branch, the second, etc.). The maximum value for d_i is the largest number of choices given by δ_N.

Equivalence between NTMs and DTMs



1. The DTM M begins with tape 2, 3 empty; tape 1 holds the input .
2. Wipe tape 2, and copy the input string from tape 1 to tape 2.
3. Simulate the NTM N on tape 2.
   - 3.1 At each step i, determine the value v of cell d_i on tape 3.
   - 3.2 If v is a valid transition choice for N, then update tape 2 appropriately.
   - 3.3 If not, abort the branch: GOTO step 4.
   - 3.4 Also abort if the transition represents reject.
4. Increment the value on tape 3: GOTO step 2.

Equivalence between NTMs and DTMs

Theorem

Every nondeterministic Turing Machine has an equivalent deterministic Turing machine.

Corollary

A language L is Turing recognizable if and only if there exists some nondeterministic Turing Machine N such that $L(N) = L$.

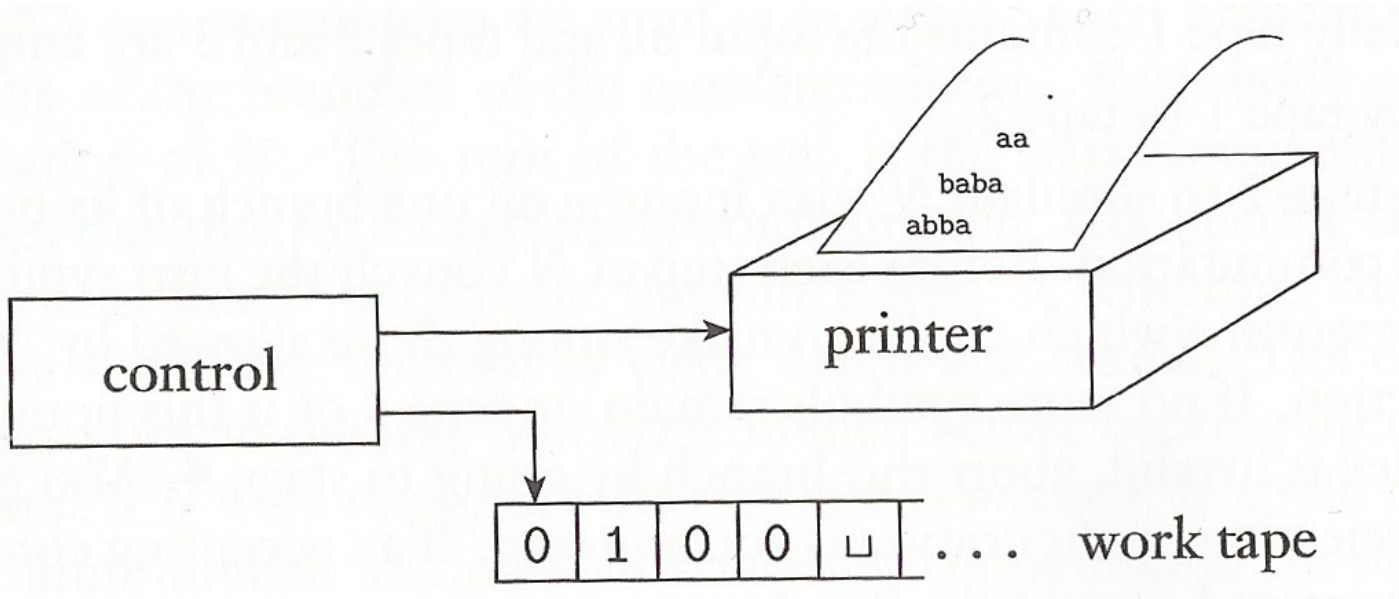
A NTM is a **decider** if **every** branch of computation halts on **all** inputs (that is, the machine never loops on any inputs).

Corollary

A language L is decidable if and only if there exists some nondeterministic Turing Machine that decides it.

Enumerators

1. An enumerator is a TM that starts with a blank tape and then "enumerates" strings from Σ^*.
2. It need never halt; it can generate an infinite sequence of strings, perhaps with repetitions.
3. You can think of a printer being attached to the machine.
4. At certain stages of computation, a signal is sent to the printer, and the tape contents (a string) are printed out.
5. Clearly, the strings printed constitute a language.



Equivalence between Enumerators and DTMs

A TM M can be constructed to recognize a language enumerated from an enumerator E. M works as follows:

- On input w, M simulates the execution of E.
- If at any point E enumerates a string v, M compares v to w.
- If $v = w$, M accepts and halts.
- Otherwise it continues simulating E.

Similarly, an enumerator E can be constructed to enumerate the strings accepted by TM M. Let s_1, s_2, ... be a list of all the strings of Σ^* (these can be generated in an ordered fashion). E works as follows:

- For $i = 1, 2, 3 \ldots$:
  - Simulate M for i steps on input s_1, s_2, ... s_i.
  - If M accepts s_j within i steps, print out s_j.
- E never halts, and it outputs all and only string in $L(M)$.

Equivalence between Enumerators and DTMs

Theorem

A language is Turing recognizable if and only if some enumerator enumerates it.

The Church-Turing Thesis

- We are interested in determining what problems are solvable by purely "mechanical" means, that is, by an "algorithm", or "effective procedure"
- Historically, however, these terms were not well-defined.
- Particularly in the 1920s and 1930s, researchers studied the nature and limits of computation. Several models were developed (including TMs).
- Surprisingly, they are equivalent. A problem solvable in one is solvable in the others.
- Turing Machines are the most commonly encountered model.
- The claim that the notion of algorithm is precisely captured by Turing machines is the **Church-Turing Thesis**.

Terminology for Describing Turing Machines

We continue to speak of Turing machines, but our real focus from now on is on algorithms. That is, the Turing machine merely serves as a precise model for the definition of algorithm.

- **formal description**: based on the formal definition of Turing machine, and it is the lowest, most detailed, level of description.
- **implementation description**: using English prose to describe the way that the Turing machine moves its head and the way that it stores data on its tape.
- **high-level description**: using English prose to describe an algorithm, ignoring the implementation details.

Examples

Example

Give implementation-level descriptions of a Turing machine that decides the following language over the alphabet $\{0, 1\}$.

$$\{w | w \text{ contains an equal number of 0s and 1s } \}$$

Examples

$$\{w | w \text{ contains an equal number of 0s and 1s } \}$$

$M =$ "On input string w:

1. Scan the tape and mark the first 0 which has not been marked. If no unmarked 0 is found, go to stage 4. Otherwise, move the head back to the front of the tape.
2. Scan the tape and mark the first 1 which has not been marked. If no unmarked 1 is found, reject.
3. Move the head back to the front of the tape and go to stage 1.
4. Move the head back to the front of the tape. Scan the tape to see if any unmarked 1s remain. If none are found, accept; otherwise, reject."

Examples

Example

Give a high level description of a TM that accepts the language A, where A consists of all strings that representing undirected graphs that are connected.

$$A = \{\langle G \rangle | G \text{ is a connected undirected graph } \}$$

Examples

$$A = \{\langle G \rangle | G \text{ is a connected undirected graph } \}$$

$M =$ "On input string $\langle G \rangle$:

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in G, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of G to determine whether they all are marked. If they are, accept; otherwise, reject."

Nondeterministic TMs: Example

Example

Give a high level description of a nondeterministic TM that accepts the following language.

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

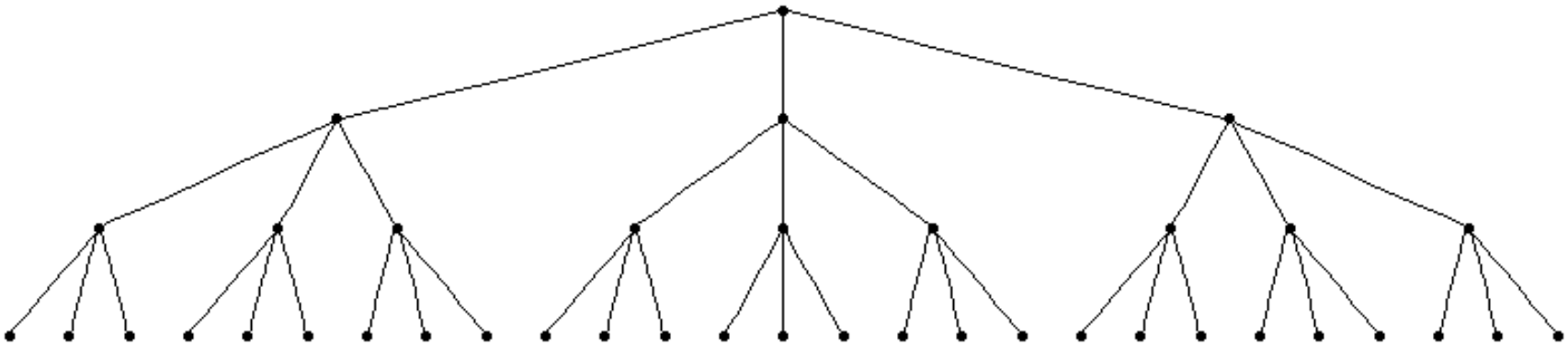
You may assume two tapes are used.

Nondeterministic TMs: Example

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

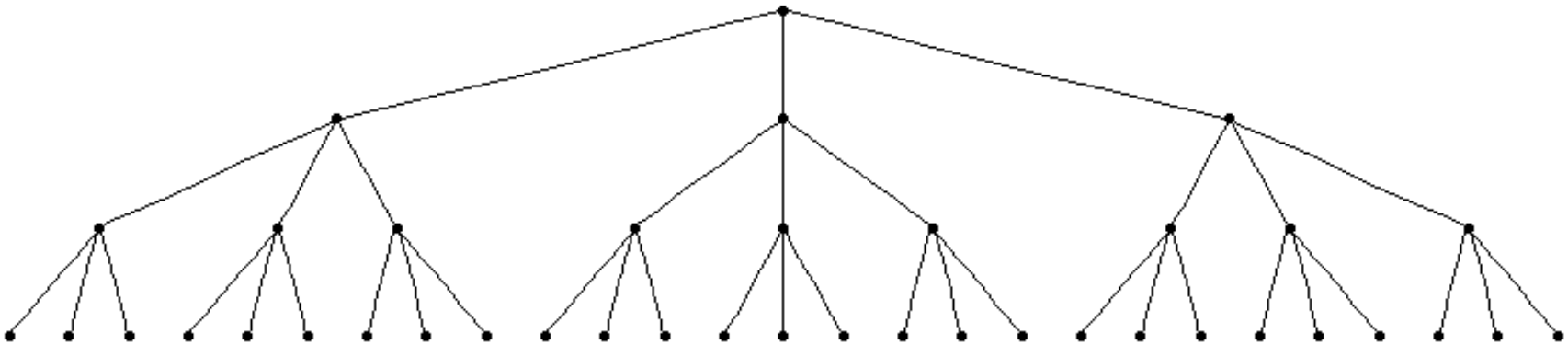
- Copy the input from tape 1 to tape 2.
- Reposition both tape heads at the left of the tape.
- Then scan right, moving both heads each turn.
- Nondeterministically pick the midpoint of ww. Mark that cell.
- Reposition both tape heads at the left of the tape.
- Keep tape head 2 fixed, but scan right with tape head 1.
- Keep moving right until the marked cell is reached.
- Then begin moving both tapeheads to the right, comparing cell contents.
- If cell contents differ before the marked cell is reached on tape 2, reject.
- If tapehead 1 reaches a $\sqcup$ and tape head 2 reaches the marked cell, then accept. Otherwise reject.

Nondeterministic TMs: Example



- The nondeterministic computations of the TM form a tree.
- Each node represents a configuration.
- Each edge represents a transition from one configuration to another.
- The TM accepts w if any branch end in an accepting configuration.
- It rejects if all branches end in a rejecting configuration.

Nondeterministic TMs: Example



- In general, some branches can be infinitely long.
- In such cases, the machine does not reject the string.
- When simulating the NTM, use a **breadth first search** (BFS).
- A BFS is guaranteed to take at most b^k to reach an accept state.
  - k is the minimum depth of an accept state.
  - b is the maximum number of children of any node.

More TMs

Example

Give high level descriptions of TMs which can be used to show that the collection of decidable languages is closed under the operation of

1. union.
2. concatenation.
3. intersection.
4. complementation.
5. star.

More TMs

Example

Show that the collection of decidable languages is closed under the operation of Union.

For any two decidable languages L_1 and L_2, let M_1 and M_2 be the TMs that decide them. We construct a TM M' that decides the union of L_1 and L_2:
"On input w:

1. Run M_1 on w, if it accepts, accept.
2. Run M_2 on w, if it accepts, accept. Otherwise, reject."

M' accepts w if either M_1 or M_2 accept it. If both reject, M' rejects.

More TMs

Example

Show that the collection of decidable languages is closed under the operation of concatenation.

For any two decidable languages L_1 and L_2, let M_1 and M_2 be the TMs that decide them. We construct a NTM M' that decides the concatenation of L_1 and L_2:
"On input w:

1. For each way to cut w into two parts $w = w_1 w_2$:
2. Run M_1 on w_1.
3. Run M_2 on w_2.
4. If both accept, accept. Otherwise, continue with next w_1, w_2.
5. All cuts have been tried without success, so reject."