Turing machines

Chanathip Namprempre

Computer Science
Reed College

Outline

Outline

1. Turing Machines
   - Introduction
   - Definitions and Examples
   - Variants of TMs
2. Enumerators
3. Church-Turing Thesis

We have learned about DFAs, NFAs, and PDAs.
Now we turn to a much more powerful type of machine. In particular, these machines have the same power as our computers!

Turing Machines

Invented by Alan Turing in 1936.

1. A TM uses infinite tape (with a left end) as its unlimited memory.
2. A TM can read/write symbols and move left/right on the tape.
3. Tape initially contains only the input string followed by blank symbols.
4. A TM can halt and accept or reject.
5. A TM can halt and produce an output by leaving it on the tape.
6. A TM can loop forever.

TMs are more powerful than FAs and PDAs

Consider languages that you know are not regular or not context-free.

- $L_1 = \{0^n 1^n \mid n \geq 0\}$
- $L_2 = \{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$
- $L_3 = \{w\#w \mid w \in \{0,1\}^*\}$

Turing machines can recognize these languages. We will construct these Turing machines later.

Defining Turing Machines

We will learn how to define Turing machines at three different levels of details.

Formal definition: State diagram. Tape manipulation. Head movement.

Implementation description: English description. Talk of tapes and head movement.

Algorithm description: No mentioning of tapes and head movement.

Formal definition of Turing Machines

Definition

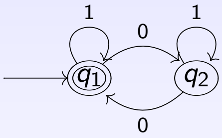
A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where

1. Q is a finite set of states,
2. Σ is a finite set of input alphabet (the blank symbol $\sqcup \notin \Sigma$),
3. Γ is a finite set of tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state,
7. $q_{\text{reject}} \in Q$ is the reject state where $q_{\text{accept}} \neq q_{\text{reject}}$.

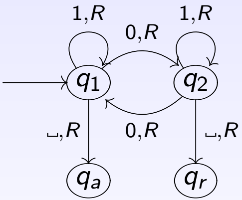
Examples of TMs

$$L_0 = \{ w \mid w \text{ contains an even number of 0s } \}$$

A DFA for L_0 looks like this:



A TM for L_0 looks like this:

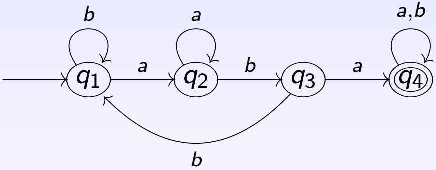


(Note: q_r is often omitted.)
(Note: Outgoing arrows from q_a, q_r are omitted.)

Examples of TMs

$$L_1 = \{w \mid w \text{ contains aba as a substring }\}$$

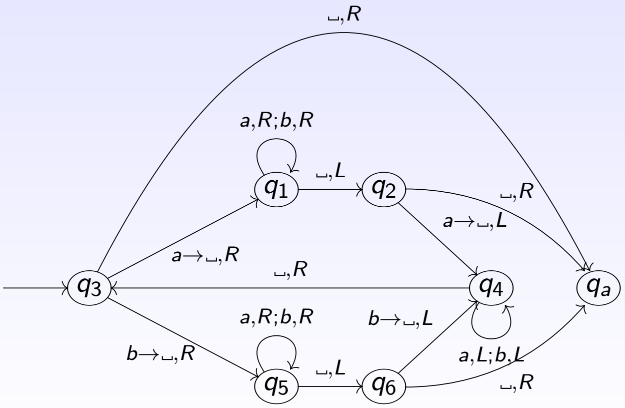
A DFA for L_1 looks like this:



Can you draw a TM for L_1?
Can you convert your TM diagram into the corresponding formal definition?

More examples

$$L_2 = \{w \mid w \text{ is a palindrome over } \{a, b\}\}$$



Terminology: configuration

Let u, v be strings and q be a state. The configuration uqv is the configuration in which

- the current state is q
- the tape contents is uv
- the tape head is under the first symbol of v

Terminology: yield

Configuration C_1 yields configuration C_2 if the TM can legally go from C_1 to C_2 in a single step. In particular, suppose that u and v are strings and that a, b, and c are symbols, then

[moving left] uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$.

[moving right] uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$.

The special cases are

- the left most end of the tape
  left moving transition: $q_ibv \Rightarrow q_jcv$
- the right most end of the input
  uq_ia becomes $uaq_i\sqcup$

To see this, try stepping through the computation of a TM on a particular input.

More terminology

Definition

Let q_0 be the start state of a TM. The start configuration of M on input w is the configuration $q_0 w$.

Definition

Let q_a be the accept state of a TM. The accepting configuration is the configuration in which the state is q_a.

Definition

Let q_r be the reject state of a TM. The rejecting configuration is the configuration in which the state is q_r.

Definition

Accepting and rejecting configurations are halting configurations. They do not yield further configurations.

Acceptance of TMs

Definition

A TM M accepts input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists where

1. C_1 is the start configuration of M on iput w,
2. each C_i yields C_{i+1}, and
3. C_k is an accepting configuration

So if L is the language recognized by a TM M, then

- $w \in L$ iff M accepts w, and
- $w \notin L$ iff M does not accept w.

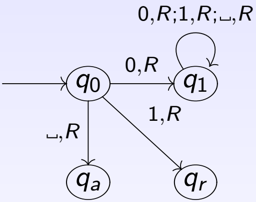
Notice: There are two possible ways in which $w \notin L$.

- M processes w and at some point lands in q_r or
- M processes w and never lands in q_r or q_a.

In the second case, we say that M loops on w.

Example

Consider the following TM M:



Can you think of strings accepted/rejected/looped on by M?
(Try ε, 0, 1, 11, etc.)

Example: Implementation description of TM

We can also describe a TM in prose rather than drawing a diagram.

$$L_3 = \{\ w\#w \mid w \in \{0,1\}^*\ \}$$

M_3 = "On input string w,

1. Scan the input to be sure that it contains a single $\#$ symbol. If not, reject.
2. Zig-zag across the tape to corresponding positions on either side of the $\#$ symbol to check on whether these positions contain the same symbol. If they do not, reject. Cross off symbols as they are checked to keep track of which symbols correspond.
3. When all symbols to either the left or the right of the $\#$ have been crossed of (whichever comes first), check for any remaining symbols on the other side of the $\#$. If any symbols remain, reject. Otherwise, accept."

Notice that we do not give details for M, but we make sure that everything we specify can actually be done on a TM. For example,

1. Scanning: This can be done by starting from the left end and reading each symbol and moving right until hitting a blank symbol.
2. Zig-zagging: This can be done by marking the symbol under our tape head, moving right until we pass the $\#$ symbol and just pass the last marked symbol, marking the symbol under our tape head, moving left until we pass the $\#$ symbol and reach a marked symbol, then moving right one position. Repeat.

When we write a TM description, staying above this level of details helps simplifying the description.

Examples of TMs

$$L_4 = \{ 0^{2^n} \mid n \geq 0 \}$$

$M_4 =$ "On input string w,

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

Examples of TMs

$$L_5 = \{\ a^i b^j c^k \ \mid\ i \times j = k \text{ and } i, j, k \geq 1\ \}$$

M_5 = "On input string w,

1. Scan the input from left to right. Make sure that it has the format $a^* b^* c^*$. Reject if it does 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. Zig-zagging 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 until all a's are crossed off. Check whether all b's and c's are also crossed off. If so, accept. Otherwise, reject."

This TM essentially multiplies.

Other examples: add, subtract, divide.

Examples of TMs

$$L_6 = \{ \#x_1\#x_2 \ldots \#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j \}$$

We design a machine M_6 that compares x_1 with $x_2, x_3, \ldots x_l$, then comparing x_2 with $x_3, x_4, \ldots, x_l$, etc. If there's a stage during which there are two equal strings, we reject. Otherwise, we accept when we are done with the last stage.

Examples of TMs

$$L_6 = \{ \#x_1\#x_2 \ldots \#x_l \mid \text{ each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j \}$$

M_6 = "On input string w,

1. If the first symbol is not $\#$, reject. Place a mark on top of the leftmost tape symbol.
2. Scan right to the next $\#$ and place a second mark on top of it. If we never see $\#$, accept.
3. Zig-zag to compare the two strings to the right of the marked $\#$. If they are equal, reject.
4. Move the rightmost of the two marks to the next $\#$ symbol to the right. If we see no $\#$ symbol before $\sqcup$, move the leftmost mark to the next $\#$ to its right and the rightmost mark to the $\#$ after that. If there's no more $\#$ for the rightmost mark, accept.
5. Go to stage 3."

Recognizing and deciding a language

Definition

A language is Turing-recogizable if some Turing Machine recognizes it.

Note: A Turing Machine M recognizes a language L iff M halts and accepts every string in L and doesn't accept any string not in L.

Definition

A language is Turing-decidable if some Turing Machine decides it.

Note: A Turing Machine M decides a language L iff M halts and accepts every string in L and <u>halts and rejects</u> every string not in L.

Practice problems

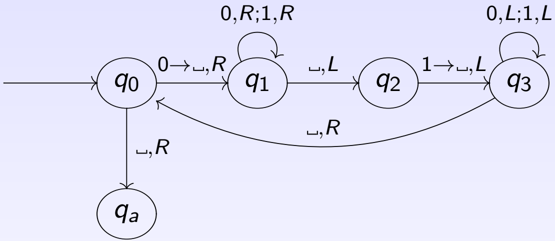
Draw a TM recognizing the language

$$\{w \mid w \text{ contains } 111 \text{ as a substring}\} .$$

(Do not omit transitions leading to q_r.)

Practice problems

Consider the following Turing Machine M. (q_r is omitted.)



1. Write the sequence of *configurations* that M goes through on 01.
2. Does M accept 01?
3. Write the sequence of *configurations* that M goes through on 01101.
4. Does M accept 01101?

Practice problems

Let the alphabet Σ be $\{1\}$.

1. Draw a Turing Machine to compute the predecessor function. Some examples.
   - If the input is 11, then output is 1.
   - If the input is 1, then the output is ε.
   - If the input is ε, then the output is ε.
2. Draw a Turing Machine that duplicates its input. In particular, if the input is w, then the output is ww.

Variants of Turing Machines

It turns out that tinkering with the definition of a TM doesn't change its power. Consider these variants:

1. Multitape Turing machines
   Try designing one for the language $\{w\#w \mid w \in \{0,1\}^*\}$.
2. Nondeterministic Turing machines
   Try designing one for the language $\{a^i b^j c^k \mid i = j \text{ or } j = k\}$.
   Try designing one for the language $\{ww \mid w \in \{0,1\}^*\}$.

Multitape Turing Machines

Multitape Turing Machines

A multitape Turing machine is like an ordinary Turing machine except that it has multiple tapes. Formally, the main difference lies in the signature of the transition function δ:

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

where k is the number of tapes.

The expression

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

means that, if the machine was in state q_i with tape heads reading $a_1, \ldots, a_k$, then it will transition to state q_j, writing $b_1, \ldots, b_k$, and moving the tape heads as indicated.

Multitape TMs and Single-Tape TMs are equivalent

Theorem

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

Proof idea: Use a single tape to store the contents of all the tapes. Use special symbols to mark the end of each tape and the position of each tape head.

Nondeterministic Turing Machines

Nondeterministic Turing Machines

A nondeterministic Turing machine is like an ordinary Turing machine except that it can make nondeterministic moves. Formally, the main difference lies in the signature of the transition function δ:

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

Nondeterministic TMs and Deterministic TMs are equivalent

Theorem

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea: Use a 3-tape Turing machine, one for storing the input (aka input tape), one for scratch work in the simulation (aka simulation tape), and one for remembering our current location in the tree (aka address tape).

Outline

Enumerators

An enumerator E works as follows. It starts with a blank input tape, then prints out strings in $L(E)$ one at a time. It can do so with or without repetitions and in any order.

Theorem

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

Proof idea:
[$\Leftarrow$] Given an enumerator E enumerating a language L, we can construct a TM M recognizing L.
[$\Rightarrow$] Given a TM M recognizing a language L, we can construct an enumerator enumerating L.

Enumerator $\equiv$ TM: Proof I

Theorem

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

Proof:
[$\Leftarrow$] Given an enumerator E enumerating a language L, we can construct a TM M recognizing L.

$M =$ On input w,

1. Run E
2. If w is printed, accept w."

Enumerator $\equiv$ TM: Proof II

Theorem

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

Proof:
[$\Rightarrow$] Given a TM M recognizing a language L, we can construct an enumerator enumerating L.

$E =$

"For $i = 1, 2, 3, \ldots$
1. Run M on $s_1, s_2, \ldots, s_i$ for i steps each
2. If any computations accept, print the corresponding s_j."

Outline

Church-Turing Thesis

Anything you can write an algorithm for, you can do with a TM.

So we figure out the limits of what we can do with real computers by studying the limits of what we can do with TMs.

In particular, there are problems that take a long time to solve. These are called intractable problems.

Worse, there are problems that cannot be solved at all!! These are called undecidable problems.