BROWN
Computer Science

CS1010: Theory of Computation

Lecture 8: Turning Machines

Lorenzo De Stefani

Fall 2020

Outline

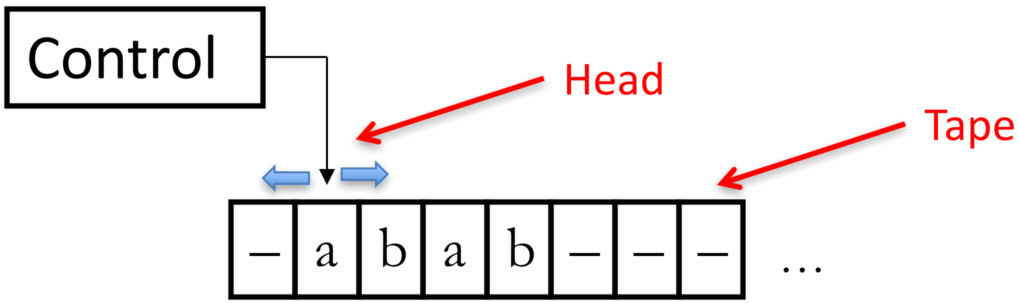
- What are Turing Machines
- Turing Machine Scheme
- Formal Definition
- Languages of a TM
- Decidability

From Sipser Chapter 3.1

FA, PDA and Turing Machines

- Finite Automata:
  - Models for devices with finite memory
- Pushdown Automata:
  - Models for devices with unlimited memory (stack) that is accessible only in Last-In-First-Out order
- Turing Machines (Turing 1936)
  - Uses unlimited memory as an infinite tape which can be read/written and moved to left or right
  - Only model thus far that can model general purpose computers – Church-Turing thesis
  - Still, TM cannot solve all problems

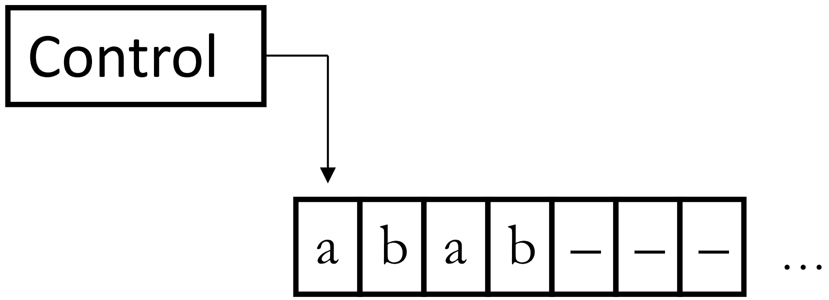
Turing Machine Scheme



Turing machines include an infinite tape

- Tape uses its own alphabet Γ, with $\Sigma \subset \Gamma$
- Initially contains the input string and *blanks* everywhere else
- Machine can read and write from tape and move left and right after each action
- Much more powerful than FIFO stack of PDAs

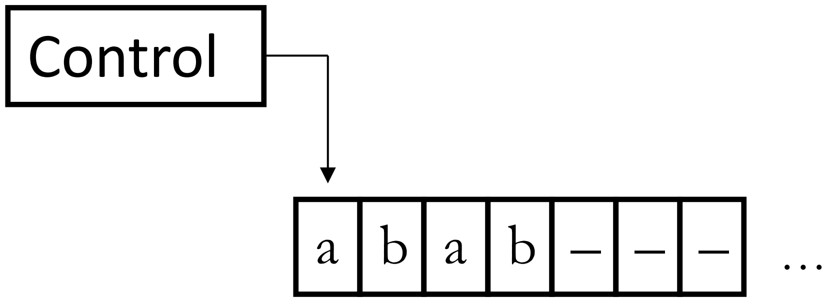
Turing Machine Scheme



The control operates as a state machine

- Starts in an initial state
- Proceeds in a series of transitions triggered by the symbols on the tape (one at a time)
- The machine continues until it enters an accept or reject state at which point it ***immediately halts*** and outputs "*accept*" or "*reject*"
- Note this is very different from FAs and PDAs

Turing Machine Scheme



- The machine can loop forever!
  - In this case we say that the TM does not halt for a given input
- Can a FA or a PDA loop forever?
  - NO! it will terminate when input string is fully processed and will only take one “action” for each input symbol

Designing Turing Machines

Design a TM to recognize the language:

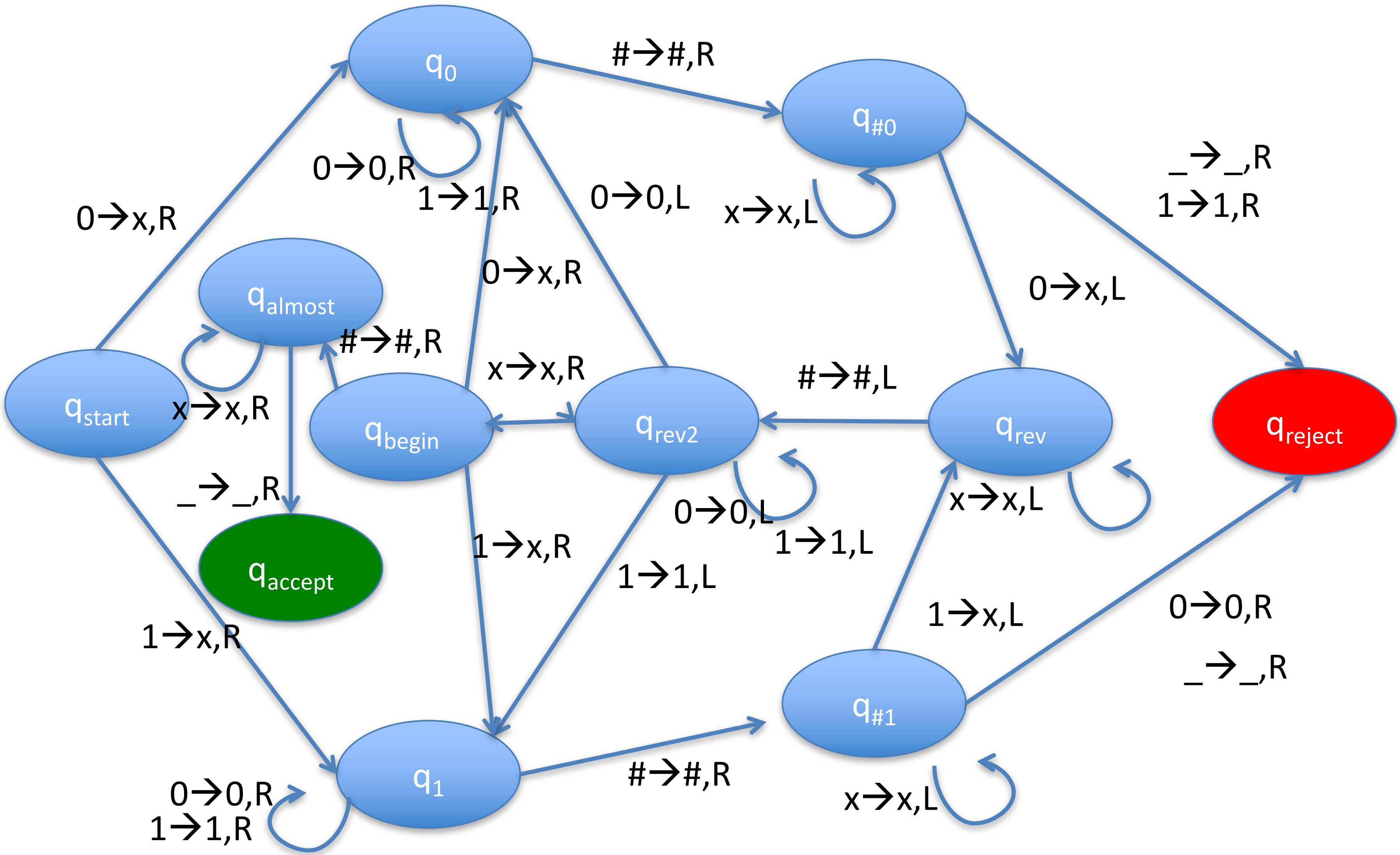
$B = \{w\#w \mid w \in \{0,1\}^*\}$

- Think of an informal description – NOT PALINDROMES
- Imagine that you are standing on an infinite tape with symbols on it and want to check to see if the string belongs to B?
  - What procedure would you use given that you can read/write and move on the tape in both directions?
  - You have a finite control so cannot remember much and thus must rely on the information on the tape

A Turing Machine for $B = \{w\#w \mid w \in \{0,1\}^*\}$

- M1 loops and in each iteration it matches symbols on each side of the #
  - It reads the leftmost symbol remaining and replaces it with "x"
  - Scans to the right until the "#" and proceeds to the first non-x symbol
  - Is it the same?
    - Yes! We have a match! We go back to the leftmost remaining symbol and repeat
    - No! We have a mismatch =(the TM transition to a "reject state" and halts
  - If both the symbols to the right and to the left of the "#" are x then we have a complete match! Also check same length!!!
  - The TM halts and the string is accepted
- Is looping possible?
  - NO! Guaranteed to terminate/halt since makes progress each iteration.

A Turing Machine for B $=\{w\#w \mid w \in \{0,1\}^*\}$



Conventions of representation

- Read “a→b,R” as: if symbol “a” is read on the tape then replace it with “b” and move to the right cell on the tape
- For moving to the left would be “a→b,L”
- We assume missing transitions lead to reject state, and the TM halts

Execution Example

Input string 011000#011000

The tape head is at right of the current state

q_{start}0 1 1 0 0 0 # 0 1 1 0 0 0 - -

xq_1 1 1 0 0 0 # 0 1 1 0 0 0 - -

…

X 1 1 0 0 0 # q_{rev}X 1 1 0 0 0 - -

q_{rev2}X 1 1 0 0 0 # X 1 1 0 0 0 - -

xxq_1 1 0 0 0 # X 1 1 0 0 0 - -

…

X X X X X X # X X X X X X q_{accept} - -

Formal Definition of a Turing Machine

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

- Q set of states
- Σ is the input alphabet not containing the blank
- Γ is the tape alphabet, where blank $\_ \in \Gamma$ and $\Sigma \subseteq \Gamma$
- δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
- $q_0, q_{accept}, and\ q_{reject}$ are the start, accept, and reject states
  - Do we need more than one reject or accept state?
  - No: since once enter either such a state the TM halts

Transitions in the TM

The transition function δ is key:

- $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
  - A machine is in a state $q \in Q$ and the head is over the tape at symbol $a \in \Gamma$, then after the move we are in a state $r \in Q$ with $b \in \Gamma$ replacing the a on the tape and the head has moved either left (L) or right (R)

Configurations and transitions

- At any step a TM is in some configuration
- A configuration is specified by:
  - the state
  - the position (i.e., current location reader head)
  - the symbol at the current head location
- We say that a configuration C1 yields C2 if there if the transition function δ allows to go from C1 to C2

Types of configurations

- Starting configuration: in starting state, head position at the beginning of the input
  - Leftmost position of the tape occupied by the input
- Accepting configuration: in accepting state
- Rejecting configuration: in rejecting state
- Halting configuration: either accepting or rejecting configurations
- There can be multiple accepting/rejecting configs. but a single accept/reject state

Turing Recognizable & Decidable Languages

The set of strings that a Turing Machine M accepts is the language of M, denoted as $L(M)$, or the language recognized by M

- A language L is <u>Turing-recognizable</u> if **some** Turing machine recognizes it
  - I.e., There exists a TM M such that M halts in the accept state for all and only the strings $s \in L$
  - Halting is *not required* for $s \notin L$, just non-acceptance
  - <u>Turing-recognizable languages</u> are sometimes referred as "recursively enumerable languages"
  - A TM which recognizes a language L is called a recognizer for L
- A Turing machine that halts on all inputs is a decider.
- A decider that recognizes a language decides it.
- A language is Turing-decidable, or simply decidable, if some Turing machine decides it.
  - Sometimes referred as "recursive language"

Turing Recognizable & Decidable Languages

Remarks:

- A language is decidable if it is Turing-recognizable and there exists a TM decider for it (i.e., a TM thatalways halts)
- Every decidable language is Turing-recognizable
  - A TM M which decides a language also recognizes it
  - Decidability is strictly stronger property than recognizability
- It is possible for a TM to halt only on those strings it accepts!
  - It is only a recognizer not a decider

Limits of Turing Machines

- Church-Turing thesis: Anything that can be programmed can be programmed on a TM
- Not all languages are Turing Decidable!
  - A_{TM} = {<M,w>, M is a description of a Turing Machine T_M, w is a description of an input and T_M accepts w}
    - We shall see this in Chapter 4
    - A_{TM} is not even Turing-recognizable!

Turing Machine Example

Design a TM M2 that decides A = $\{0^{2^n} | n \geq 0\}$, the language of all strings of 0s with length 2^n.

- Without designing it, do you think this can be done? Why?
  - Yes: we could write a program to do it and therefore we know a TM could do it since we said a TM can do anything a computer can do
- How would you design it?
- Solution:

  Idea: divide by 2 each time and see if result is a one

  1. Sweep left to right across the tape, crossing off every other 0.
  2. If in step 1:
     - the tape contains exactly one 0, then accept
     - the tape contains an odd number of 0's, reject immediately
     - Only alternative is even 0's. In this case return head to start and loop back to step 1.

Sample Execution of TM M2

0 0 0 0 - -	Number is 4, which is 2^2
x 0 0 0 - -	
x 0 x 0 - -	Now we have 2, or 2^1
x 0 x 0 - -	
x 0 x 0 - -	
x x x 0 - -	
x x x 0 - -	Now we have 1, or 2^0
x x x 0 - -	Seek back to start
x x x 0 - -	Scan right; one 0, so accept

Turing Machine Example II

Design TM M3 to decide the language:

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

- What is this testing about the capability of a TM?
  - That it can do (or at least check) multiplication
  - As we have seen before, we often use unary
- How would you approach this?
  - Imagine that we were trying 2 x 3 = 6

Turing Machine Example II

Solution:

1. First scan the string from left to right to verify that it is of form $a^+b^+c^+$; if it is, scan to start of tape and if not, reject.
2. Cross off the first a and scan until the first b occurs. Shuttle between b’s and c’s crossing off one of each until all b’s are gone. If all c’s have been crossed off and some b’s remain, reject.
3. Restore the crossed off b’s and repeat step 2 if there are a’s remaining. If all a’s gone, check if all c’s are crossed off; if so, accept; else reject.

* Use different symbols for crossing out the b’s so that is easier to restore them =)

Transducers

- So far we have always talked about recognizing a language, not generating a language. This is common in language theory.
- When talking about computation this seems strange and limiting.
  - Computers typically transform input into output
  - We are more likely to have a computer perform multiplication than check that the equation is correct.
  - Turing Machines can also generate/transduce
  - How would you compute c^k given $a^i b^j$ and $i x j = k$
    - In a similar manner. For every a, you scan through the b's and for each you go to the end of the string and add a c. Thus by zig-zagging a times, you can generate the appropriate number of c's.

Turing Machine Example III

The element distinctness problem:

- Given a list of strings over alphabet {0, 1} each separated by a #, accept if all strings are different.
- $E = \{\#x_1\#x_2\# \ldots \# x_n | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$

Turing Machine Example IV

- Solution:
  1. Place a mark symbol on top of the left-most symbol. If it was a blank, accept. If it was a # continue; else reject
  2. Scan right to next # and place a mark symbol on it. If no # is encountered, we only had x1 so accept.
  3. By zig-zagging, compare the two string to the right of the two 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, 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 back to step 3

Decidability

- All of these examples have been decidable, and hence Turing-recognizable.
- How do we know that these examples are decidable?
  - At each iteration progress is made toward the goal, so the goal itself is reachable
  - Not hard to prove formally. For example, if the the input is composed by *n* symbols and a symbol is erased at each iteration, the algorithm will finish after *n* iterations
- Showing that a language is Turing recognizable but not decidable is challenging