COMS3261: Computer Science Theory

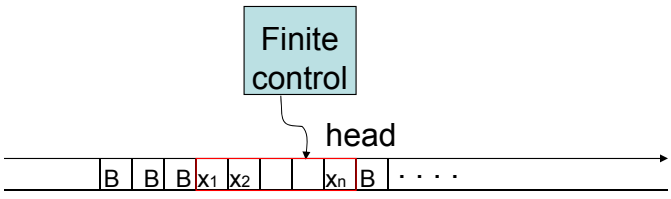
Fall 2013

Mihalis Yannakakis

Lecture 17, 11/5/13

Turing Machine [Turing 1936]

- Formal model of a computer running a particular program
- Comparison with real computer: TM is very primitive, but memory is unlimited
  - Difficult to program : too low level
  - But simple, so more feasible to analyze



Tape (2-way infinite), made of cells
Initially has input $x_1 \ldots x_n$ and blanks (B) to left and right
Head at left end of input (x_1)

Turing machine

- M= $(Q,\Sigma,\Gamma,\delta,q_0,B,F)$
- Q= finite set of states (finite control holds state)
- Σ= finite input alphabet
- Γ= finite tape alphabet = $\Sigma \cup \{B\} \cup$ other symbols
- B = blank symbol
- q_0= initial state
- F = set of accepting states
- δ = transition function (deterministic for now): depends on state, symbol scanned in cell; action =change state, write new tape symbol, move head 1 cell L or R
- $\delta(q,X)$ = (p,Y,L/R) (or nothing); can allow also S for stay (same power)
- δ: $Q\times\Gamma \rightarrow Q\times \Gamma \times \{L,R\}$ (or nothing)

Instantaneous Description (Configuration)

- State, tape content, head position
- At any point in time, only finite amount of tape has been visited – rest is B.
- ID can be represented by string $X_1X_2\ldots X_{i-1}qX_i\ldots X_n$:
  - non-B portion $\subseteq X_1X_2\ldots X_{i-1}X_i\ldots X_n$
  - state=q
  - head scans cell i
- Represents interval [k,l] of tape where
  k=min{head position, min nonB position}
  l=max{head position, max nonB position}
- Adding B's at the beginning or end of the string represents the same ID

Moves of TM

- |-- move of TM from one ID to the next
- |--* sequence of any number of moves (0 or more)
- If $\delta(q,X_i) = (p,Y,L)$ then

  $X_1X_2\ldots X_{i-1}qX_i\ldots X_n$ |-- $X_1X_2\ldots pX_{i-1}Y\ldots X_n$ if $1<i<n$

  - if i=1 then $qX_1X_2\ldots X_n$ |-- $pBYX_2\ldots X_n$
  - if i=n and Y=B then $X_1X_2\ldots X_{n-1}qX_n$ |-- $X_1X_2\ldots pX_{n-1}$
- Move Right is symmetric

Language of TM

- Language accepted (recognized) by M:

  $L(M)= \{w \in \Sigma^* \mid q_0w$ |--* $\alpha p \beta$ for some $p \in F, \alpha,\beta \in \Gamma^*\}$

Note: input is accepted if some accepting state is reached at *any* step of the computation (not only at end of reading the input – unlike FA, PDA)

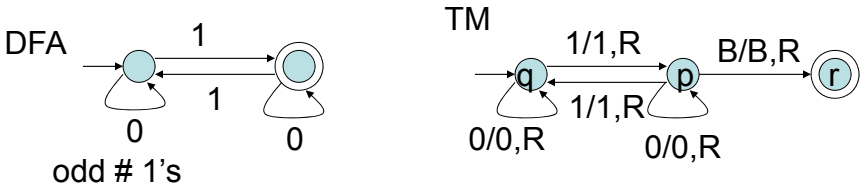
Once an accepting state has been reached, there is no reason to continue the computation. Can eliminate all transitions from accepting states (i.e. make TM halt at all accepting states), and will accept the same language.

If $w \notin L(M)$ then w is rejected: either TM halts at some point or runs forever without reaching any accepting state

TM Representation - Examples

- Can represent δ by transition table ($Q \times \Gamma$) or diagram

1. TM for a DFA



Computation:

q01011 |-- 0q1011 |-- 01p011 |-- 010p11 |-- 0101q1 |--

01011pB |-- 01011BrB

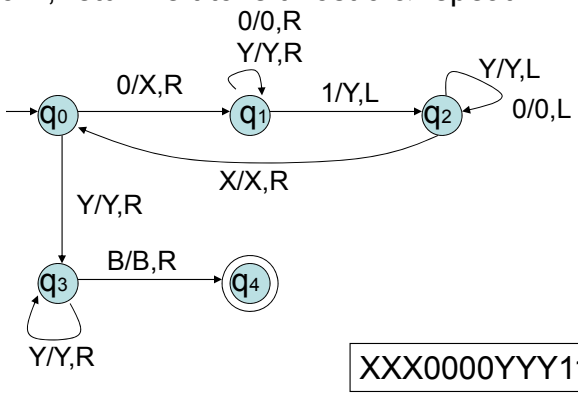
Every regular language can be recognized similarly by a TM

Head moves left to right over input, accepts/rejects at end

Examples ctd.

2. $\{0^n1^n \mid n \geq 1\}$:

Loop: change leftmost 0 to X, move right, change leftmost 1 to Y, return left to leftmost 0 & repeat



XXX0000YYY1111

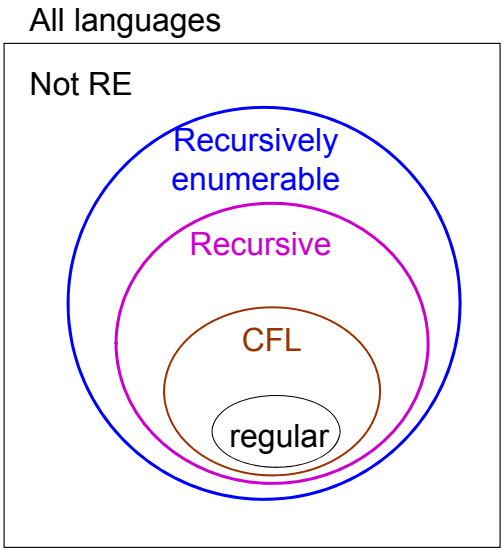
Halting and Recursive languages

- A TM halts if it is in a state q, reads tape symbol X and there is no move, i.e. $\delta(q,X)$ undefined
- Can assume wlog that TM halts when it accepts: If we delete transitions out of accepting states, the language does not change (different than FA, PDA)
- But TM may reject either by running forever or by halting at a nonaccepting (rejecting) state.

Recursive and R. E. languages

- Recursively enumerable (r.e.) or Recognizable language L:
  There is a Turing Machine M such that L=L(M)
  - M may not halt on some inputs not in L
- Recursive (Decidable) language L:
  $\exists$ TM M that *halts on every input* and L=L(M)
  M = algorithm that solves the membership problem for L:
  Given input string w, M accepts w $\Leftrightarrow$ w $\in$L
- Decidable problem: $\exists$ TM that always halts (an algorithm) that solves the problem $\Leftrightarrow$ recursive (decidable) language

Relationship



Acceptance by Halting

- Halting language of a TM: H(M) = {w | M halts on input w}

In this case we do not need to specify a set F of accepting states.

Theorem: L= H(M) for some TM M $\Leftrightarrow$ L=L(N) for some TM N .

Proof: 1. Suppose L = H(M). Let N = M +accepting state r. If $\delta_M(q,X)$ undefined, then $\delta_N(q,X)=(r,X,R)$, i.e. if M halts then N goes to accepting state r

2. Suppose L=L(N). Assume wlog N halts when it accepts. If there are other undefined transitions $\delta_N(q,X)$, where $q \notin F$, then add new state r, and $\delta_M(q,X)=(r,X,R)$; $\delta_M(r,Y) =(r,Y,R)$, $\forall Y \in \Gamma$. Then M halts only when it accepts, i.e. H(M)=L(N).

Turing machines with output

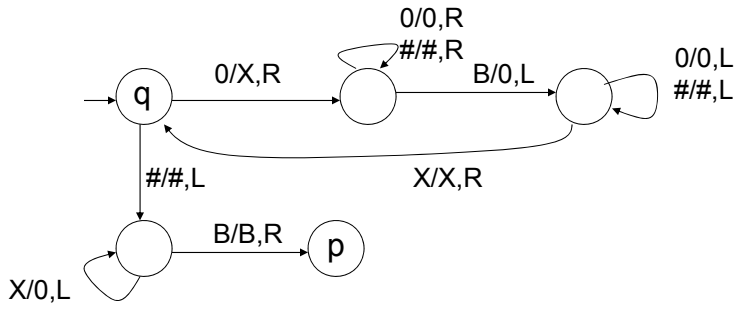
- Besides just accepting/rejecting the input, TMs can compute also functions of the input
- TM starts with input on the tape, ends with output on the tape (and halts)

- Examples:
- shift input one place right
- copy the input: w $\rightarrow$ ww
- Reverse the input: w $\rightarrow$ w^R
- Add numbers in unary notation: $0^i\#0^j \rightarrow 0^{i+j}$
- Add numbers in binary notation: bin(i)#bin(j) $\rightarrow$ bin(i+j)
- Multiplication etc.

Example: Copy

Copy one block of 0's at the end of another block

Start ID $q0^n\#0^j \rightarrow$ End ID $p0^n\#0^{j+n}$



HW: Copy one binary string x at the end of another string y

Input x#y $\rightarrow$ Output x#yx

TM for Unary Addition

- input $0^m\#0^n \rightarrow$ output 0^{m+n}
- Copy 1st block to end of the 2nd block
- At end: $0^m\#0^{m+n} \rightarrow$ erase prefix $0^m\#$

TM for Unary Multiplication

- input $0^m\#0^n\# \rightarrow$ output 0^{mn}
- Loop: $0^i\#0^n\#0^j \rightarrow 0^{i-1}\#0^n\#0^{j+n}$ for i=m down to 1
  i.e. Repeatedly erase a 0 from 1st block
  & copy 2nd block to end of the 3rd block
  (Use the Copy subroutine)
- At end: $\#0^n\#0^{mn} \rightarrow$ erase prefix $\#0^n\#$