CS21
Decidability and Tractability

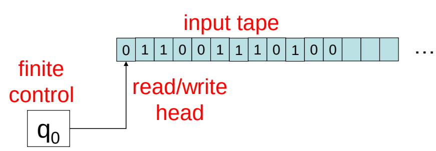
Lecture 9

January 27, 2014

Outline

- Turing Machines and variants
  - multitape TMs
  - nondeterministic TMs
- Church-Turing Thesis
- decidable, RE, co-RE languages

Turing Machines



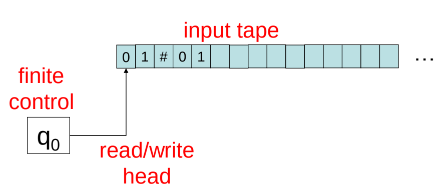
- New capabilities:
  - infinite tape
  - can read OR write to tape
  - read/write head can move left and right

Turing Machine

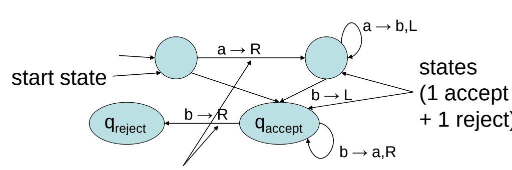
- Informal description:
  - input written on left-most squares of tape
  - rest of squares are blank
  - at each point, take a step determined by
    - current symbol being read
    - current state of finite control
  - a step consists of
    - writing new symbol
    - moving read/write head left or right
    - changing state

Example Turing Machine

language L = {w#w : w $\in$ {0,1}*}



Turing Machine diagrams

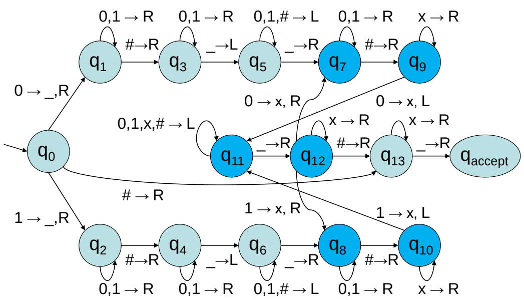


transition label: (tape symbol read → tape symbol written, direction moved)

- a → R means "read a, move right"
- a → L means "read a, move left"
- a → b, R means "read a, write b, move right

"_" means blank tape square

Example TM diagram



TM formal definition

- A TM is a 7-tuple
  (Q, Σ, Γ, δ, q_0, q_{accept}, q_{reject}) where:
  - Q is a finite set called the states
  - Σ is a finite set called the input alphabet
  - Γ is a finite set called the tape alphabet
  - $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a function called the transition function
  - q_0 is an element of Q called the start state
  - q_{accept}, q_{reject} are the accept and reject states

Example TM operation

#	0	1			start
#	0	1			start
#	0	1			start
#	0	1			start
#	0	1			t
#	0	0			t
#	1	0			accept

program
for "binary
successor"

q	σ	δ(q,σ)
start	0	(start, 0, R)
start	1	(start, 1, R)
start	_	(t, _, L)
start	#	(start, #, R)
t	0	(accept, 1, -)
t	1	(t, 0, L)
t	#	(accept, #, R)

TM configurations

- At every step in a computati
  a configuration determined
  - the contents of the tape
  - the state
  - the location of the read/write head
- next step completely determined by current configuration
- shorthand: string uqv with $u,v \in \Gamma^*$, $q \in Q$

meaning:
- tape contents: uv followed by blanks
- in state q
- reading first symbol of v

TM configurations

- configuration C_1 yields configuration C_2 if TM can legally* move from C_1 to C_2 in 1 step
  - notation: $C_1 \Rightarrow C_2$
  - also: "yields in 1 step" notation: $C_1 \Rightarrow^1 C_2$
  - "yields in k steps" notation: $C_1 \Rightarrow^k C_2$

  if there exists configurations $D_1, D_2, \ldots D_{k-1}$ for which $C_1 \Rightarrow D_1 \Rightarrow D_2 \Rightarrow \ldots \Rightarrow D_{k-1} \Rightarrow C_2$
  - also: "yields in some # of steps" ($C_1 \Rightarrow^* C_2$)

*Convention: TM halts upon entering q_{accept}, q_{reject}

TM configurations

$u,v \in \Gamma^*$
$a,b,c \in \Gamma$
$q_i, q_j \in Q$

- Formal definition of "yields":

  $uaq_ibv \Rightarrow uq_jacv$

  if $\delta(q_i, b) = (q_j, c, L)$, and

  $uaq_ibv \Rightarrow uacq_jv$

  if $\delta(q_i, b) = (q_j, c, R)$

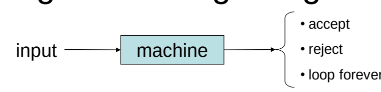
  ($q_i \neq q_{accept}$, q_{reject})
- two special cases:
  - left end: $q_ibv \Rightarrow q_jcv$ if $\delta(q_i, b) = (q_j, c, L)$
  - right end: uaq_i same as $uaq_i\_$

TM acceptance

- start configuration: $q_0 w$ (w is input)
- accepting config.: any config.with state q_{accept}
- rejecting config.: any config. with state q_{reject}

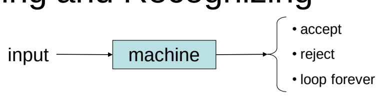
TM M accepts input w if there exist configurations $C_1, C_2, \ldots, C_k$
- C_1 is start configuration of M on input w
- $C_i \Rightarrow C_{i+1}$ for i = 1, 2, 3, …, k-1
- C_k is an accepting configuration

Deciding and Recognizing



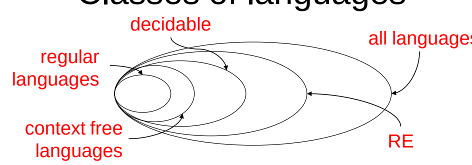
- TM M:
  - L(M) is the language it recognizes
  - if M rejects every $x \notin L(M)$ it decides L
  - set of languages recognized by some TM is called Turing-recognizable or recursively enumerable (RE)
  - set of languages decided by some TM is called Turing-decidable or decidable or recursive

Deciding and Recognizing



- TM M:
  - L(M) is the language it recognizes
  - if M rejects every $x \notin L(M)$ it decides L
  - set of languages recognized by some TM is called Turing-recognizable or recursively enumerable (RE)
  - set of languages decided by some TM is called Turing-decidable or decidable or recursive

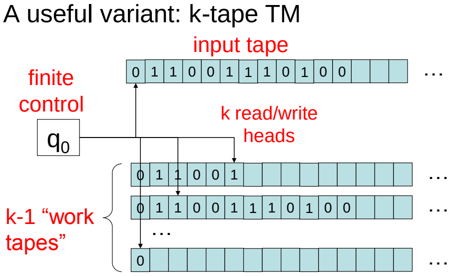
Classes of languages



- We know: regular $\subset$ CFL (proper containment)
- CFL $\subset$ decidable
  - proof?
  - decidable $\subset$ RE $\subset$ all languages
  - proof?

Multitape TMs

- A useful variant: k-tape TM



Multitape TMs

- Informal description of k-tape TM:
  - input written on left-most squares of tape #1
  - rest of squares are blank on all tapes
  - at each point, take a step determined by
    - current k symbols being read on k tapes
    - current state of finite control
  - a step consists of
    - writing k new symbols on k tapes
    - moving each of k read/write heads left or right
    - changing state

Multitape TM formal definition

- A TM is a 7-tuple
  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:
  - everything is the same as a TM except the transition function:

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

$\delta(q_i, a_1, a_2, \ldots, a_k) = (q_j, b_1, b_2, \ldots, b_k, L, R, \ldots, L) =$
"in state q_i, reading $a_1, a_2, \ldots, a_k$ on k tapes, move to state q_j, write $b_1, b_2, \ldots, b_k$ on k tapes, move L, R on k tapes as specified."

Multitape TMs

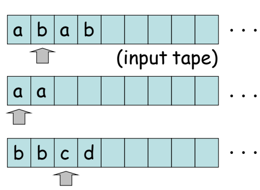
Theorem: every k-tape TM has an equivalent single-tape TM.

Proof:

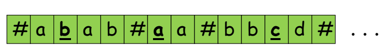
- Idea: simulate k-tape TM on a 1-tape TM.

Multitape TMs

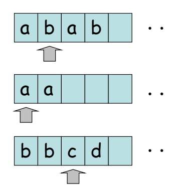
simulation of k-tape TM by single-tape TM:



- add new symbol **x** for each old x
- marks location of "virtual heads"

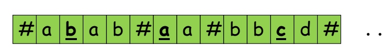


Multitape TMs



Repeat:
- scan tape, remembering the symbols under each virtual head in the state (how many new states needed?)
- make changes to reflect 1 step of M
- if hit #, shift to right to make room

if M halts, erase all but 1st string



Nondeterministic TMs

- A important variant: nondeterministic TM
- informally, several possible next configurations at each step
- formally, a NTM is a 7-tuple
  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:
  - everything is the same as a TM except the transition function:

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

NTM acceptance

- start configuration: q_0w (w is input)
- accepting config.: any config.with state q_{accept}
- rejecting config.: any config. with state q_{reject}

NTM M accepts input w if there exist configurations $C_1, C_2, \ldots, C_k$
- C_1 is start configuration of M on input w
- $C_i \Rightarrow C_{i+1}$ for i = 1, 2, 3, …, k-1
- C_k is an accepting configuration

Nondeterministic TMs

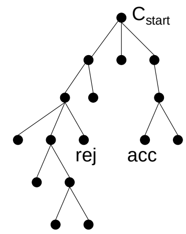
Theorem: every NTM has an equivalent (deterministic) TM.

Proof:

– Idea: simulate NTM with a deterministic TM

Nondeterministic TMs

Simulating NTM M with a deterministic TM:



- computations of M are a tree
- nodes are configs
- fanout is b = maximum number of choices in transition function
- leaves are accept/reject configs.

Nondeterministic TMs

Simulating NTM M with a deterministic TM:

- idea: breadth-first search of tree
- if M accepts: we will encounter accepting leaf and accept
- if M rejects: we will encounter all rejecting leaves, finish traversal of tree, and reject
- if M does not halt on some branch: we will not halt…

Nondeterministic TMs

Simulating NTM M with a deterministic TM:

– use a 3 tape TM:
  - tape 1: input tape (read-only)
  - tape 2: simulation tape (copy of M's tape at point corresponding to some node in the tree)
  - tape 3: which node of the tree we are exploring (string in {1,2,…b}*)

– Initially, tape 1 has input, others blank

– STEP 1: copy tape 1 to tape 2

Nondeterministic TMs

Simulating NTM M with a deterministic TM:

– STEP 2: simulate M using string on tape 3 to determine which choice to take at each step
  - if encounter blank, or a # larger than the number of choices available at this step, abort, go to STEP 3
  - if get to a rejecting configuration: DONE = 0, go to STEP 3
  - if get to an accepting configuration, ACCEPT

– STEP 3: replace tape 3 with lexicographically next string and go to STEP 2
  - if string lengthened and DONE = 1 REJECT; else DONE = 1

Examples of basic operations

- Convince yourself that the following types of operations are easy to implement as part of TM "program"

  (but perhaps tedious to write out…)

  – copying
  – moving
  – incrementing/decrementing
  – arithmetic operations +, -, *, /