Outline

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

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 \subseteq CFL (proper containment)
- CFL \subseteq decidable
  - proof?
  - decidable \subseteq RE \subseteq 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 
  \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\) 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."

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

Nondeterministic TMs

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

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

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 \( i = 1, 2, 3, \ldots, k-1 \)
- \( C_k \) 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 \( \{1,2,\ldots,b\}^* \))
- 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 +, -, *, /

Universal TMs and encoding

• the input to a TM is always a string in Σ*
• often we want to interpret the input as representing another object
• examples:
  – tuple of strings (x, y, z)
  – 0/1 matrix
  – graph in adjacency-list format
  – Context-Free Grammar

Universal TMs and encoding

• the input to a TM is always a string in Σ*
• we must encode our input as such a string
• examples:
  – tuples separated by #: #x#y#z
  – 0/1 matrix given by: #n#x# where x ∈ {0,1}^n
• any reasonable encoding is OK
• emphasize “encoding of X” by writing <X>

Universal TMs and encoding

• some strings not valid encodings and these are not in the language

Church-Turing Thesis

• many other models of computation
  – we saw multitape TM, nondeterministic TM
  – others don’t resemble TM at all
  – common features:
    • unrestricted access to unlimited memory
    • finite amount of work in a single step
  – every single one can be simulated by TM
  – many are equivalent to a TM
  – problems that can be solved by computer does not depend on details of model!
Church-Turing Thesis

- the belief that TMs formalize our intuitive notion of an algorithm is:

  The Church-Turing Thesis
  everything we can compute on a physical computer
  can be computed on a Turing Machine

- Note: this is a belief, not a theorem.