Harvard CS 121 and CSCI E-121

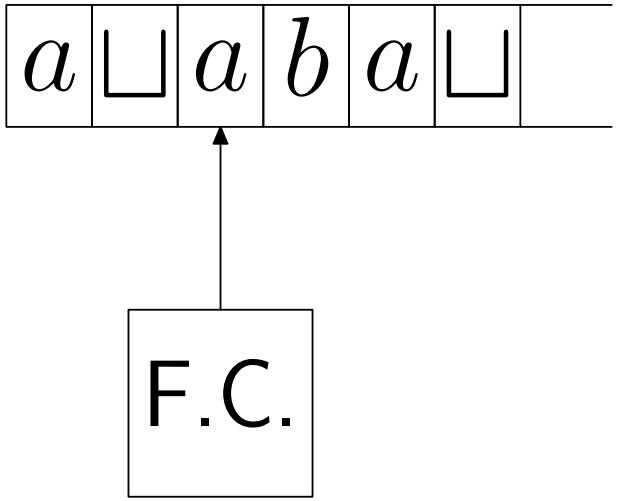
Lecture 14: Turing Machines and the Church–Turing Thesis

Harry Lewis

October 22, 2013

- **Reading:** Sipser, §3.2, §3.3.

The Basic Turing Machine



- Head can both read and write, and move in both directions
- Tape has unbounded length
- ␣ is the blank symbol. All but a finite number of tape squares are blank.

Formal Definition of a TM

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

- Q is a finite set of states, containing
  - the start state q_0
  - the accept state q_{accept}
  - the reject state q_{reject} ($\neq q_{\text{accept}}$)
- Σ is the input alphabet
- Γ is the tape alphabet
  - Contains Σ
  - Contains "blank" symbol $\sqcup \in \Gamma - \Sigma$

The transition function

$$Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

- L and R are "move left" and "move right"
- $\delta(q, \sigma) = (q', \sigma', R)$
  - Rewrite σ as σ' in current cell
  - Switch from state q to state q'
  - And move right
- $\delta(q, \sigma) = (q', \sigma', L)$
  - Same, but move left
  - *Unless* at left end of tape, in which case stay put

Computation of TMs

- A configuration is uqv, where $q \in Q$, $u, v \in \Gamma^*$.
  - Tape contents = uv followed by all blanks
  - State = q
  - Head on first symbol of v.
  - Equivalent to uqv', where $v' = v\sqcup$.
- Start configuration = $q_0 w$, where w is input.
- One step of computation:
  - $uq\sigma v$ yields $u\sigma' q' v$ if $\delta(q, \sigma) = (q', \sigma', R)$.
  - $u\tau q\sigma v$ yields $uq'\tau\sigma' v$ if $\delta(q, \sigma) = (q', \sigma', L)$.
  - $q\sigma v$ yields $q'\sigma' v$ if $\delta(q, \sigma) = (q', \sigma', L)$.
- If $q \in \{q_{\text{accept}}, q_{\text{reject}}\}$, computation halts.

TMs and Language Membership

- M accepts w if there is a sequence of configurations $C_1, \ldots, C_k$ such that
  1. $C_1 = q_0 w$.
  2. C_i yields C_{i+1} for each i.
  3. C_k is an accepting configuration (i.e. state of M is q_{accept}).
- $L(M) = \{w : M \text{ accepts } w\}$.
- L is Turing-recognizable if $L = L(M)$ for some TM M, i.e.
  - $w \in L \Rightarrow M$ halts on w in state q_{accept}.
  - $w \notin L \Rightarrow$ M halts on w in state q_{reject} OR M never halts (it "loops").

Decidability, a.k.a. Recursiveness

- L is (Turing-)decidable if there is a TM M s.t.
  - $w \in L \Rightarrow M$ halts on w in state q_{accept}.
  - $w \notin L \Rightarrow M$ halts on w in state q_{reject}.
- Other common terminology
  - Recursive = decidable
  - Recursively enumerable (r.e.) = Turing-recognizable
  - Because of alternate characterizations as sets that can be defined via certain systems of recursive (self-referential) equations.

Example

- **Claim:** $L = \{a^n b^n c^n : n \geq 0\}$ is decidable.

Questions

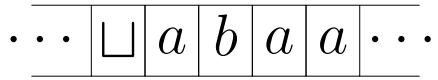
- Does every TM recognize some language?
- Does every TM decide some language?
- How many Turing-recognizable languages are there?
- How many decidable languages are there?

"Computability"

- Defined in terms of Turing machines
- Computable = recursive/decidable (sets, functions, etc.)
- In fact an abstract, universal notion
- Many other computational models yield exactly the same classes of computable sets and functions
- Power of a model = what is computable using the model (extensional equivalence)
- Not programming convenience, speed (for now...), etc.
- All translations between models are **constructive**

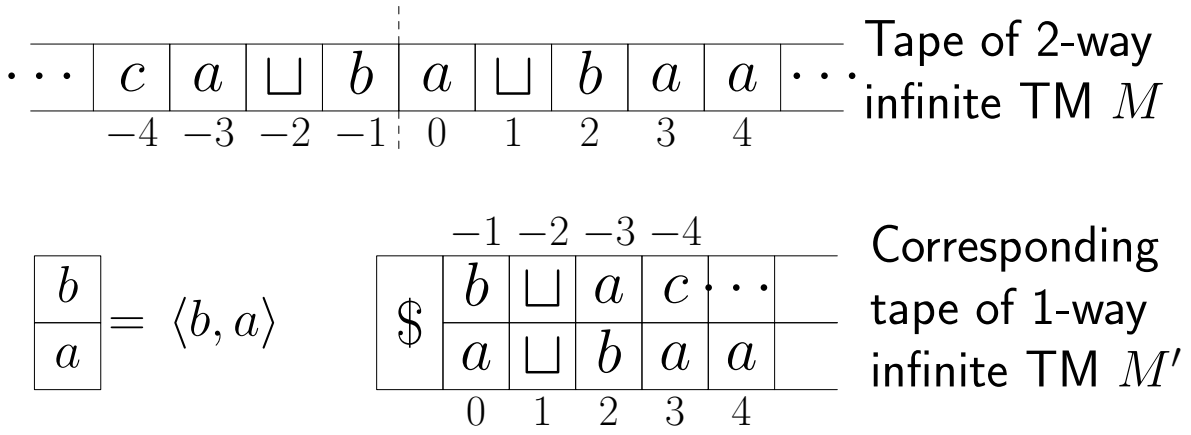
TM Extensions That Do Not Increase Its Power

- TMs with a 2-way infinite tape, unbounded to left and right



Proof that TMs with 2-way infinite tapes are no more powerful than the 1-way infinite tape variety:

"Simulation." Convert any 2-way infinite TM into an equivalent 1-way infinite TM "with a two-track tape."



Recall the Formal Definition of a TM:

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

- Q is a finite set of states, containing
  - the start state q_0
  - the accept state q_{accept}
  - the reject state q_{reject} ($\neq q_{\text{accept}}$)
- Σ is the input alphabet
- Γ is the tape alphabet
  - Contains Σ
  - Contains "blank" symbol $\sqcup \in \Gamma - \Sigma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.

Formalization of the Simulation of 2-way infinite tape TM

Formally, $\Gamma' = (\Gamma \times \Gamma) \cup \{\$\}$.

M' includes, for every state q of M, two states:

$\langle q, 1 \rangle \sim$ "q, but we are working on upper track"

$\langle q, 2 \rangle \sim$ "q, but we are working on lower track"

e.g. If $\delta_M(q, a_1) = (p, b, L)$ then
$\delta_{M'}(\langle q, 1 \rangle, \langle a_1, a_2 \rangle) = (\langle p, 1 \rangle, \langle b, a_2 \rangle, R)$.

Also need transitions for:

- Lower track
- U-turn on hitting endmarker
- Formatting input into "2-tracks"

Describing Turing Machines

Formal Description

- 7-tuple or state diagram
- Most of the course so far

Implementation Description

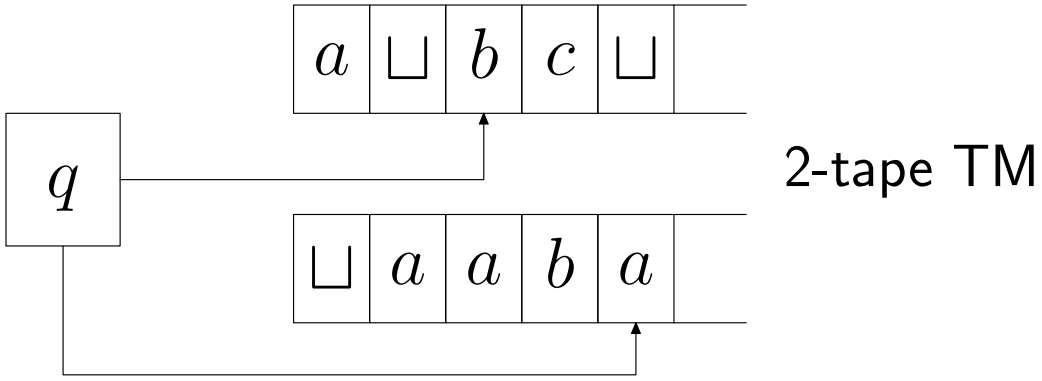
- Prose description of tape contents, head movements
- Omit details of states and transition functions (but do convince yourself that a TM can do what you're describing!)
- This lecture, next lecture, ps6

High-Level Description

- Starting in a couple of lectures...

More extensions

- Adding multiple tapes does not increase power of TMs



(Convention: First tape used for I/O, like standard TM; Second tape is available for scratch work)

Simulation of multiple tapes

- Simulate a k-tape TM by a one-tape TM whose tape is split (conceptually) into $2k$ tracks:
  - k tracks for tape symbols
  - k tracks for head position markers (one in each track)

	a	$\sqcup$	b	c	$\sqcup$	
			$\uparrow$			
$\$$	$\sqcup$	a	a	b	a	
					$\uparrow$	

(Sipser does different simulation.)

Simulation steps

- To simulate one move of the k-tape TM:

Speed of the Simulation

- Note that the "equivalence" in ability to compute functions or decide languages does not mean comparable speed.

  e.g. A standard TM can decide $L = \{w\#w : w \in \Sigma^*\}$ in time $O(|w|^2)$. But there is an $O(|w|)$-time 2-tape decider.

- Let $T_M : \Sigma^* \to \mathcal{N}$ measure the amount of time a decider M uses on an input. That is, $T_M(w)$ is the number of steps TM M takes to halt on input w.

- General fact about multitape to single-tape slowdown:

  Theorem: If M is a multitape TM that takes time $T(w)$ when run on input w, then there is a 1-tape machine M' and a constant c such that M' simulates M and takes at most $cT(w)^2$ steps on input w.

Equivalent Formalisms

Many other formalisms for computation are equivalent in power to the TM formalism:

- TMs with 2-dimensional tapes
- Random-access TMs
- General Grammars
- 2-stack PDAs, 2-counter machines
- Church's λ-calculus (μ-recursive functions)
- Markov algorithms
- Your favorite programming language (C, Python, OCaml, . . .)
- In any formalism, each formalized algorithm is expressible as a bit string, number, . . .

The Church-Turing Thesis

The equivalence of each to the others is a mathematical theorem.

That these formal models of algorithms capture our intuitive notion of algorithms is the **Church–Turing Thesis**.

- Church's thesis $=$ partial recursive functions,
  Turing's thesis = Turing machines

- The Church–Turing Thesis is an extramathematical proposition, not subject to formal proof.

Nondeterministic TMs

- Like TMs, but $\delta : Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$
- It mainly makes sense to think of NTMs as recognizers

  $L(M) = \{w : M$ has some accepting computation on input $w\}$

  Example: NTM to recognize
  $\{w : w$ is the binary notation for a product of two integers $\geq 2\}$

NTMs recognize the same languages as TMs

- Given a NTM M, we must construct a TM M' that determines, on input w, whether M has an accepting computation on input w.

- M' systematically tries

  $\rightarrow$ all one-step computations

  $\rightarrow$ all two-step computations

  $\rightarrow$ all three-step computations

  $\vdots$

Enumerating Computations by Dovetailing

- There is a bounded number of k-step computations, for each k.

  (because for each configuration there is only a constant number of "next" configurations in one step)

- Ultimately M' either:
  - discovers an accepting computation of M, and accepts itself,
  - or searches forever, and does not halt.

Dovetailing Details

- Suppose that the maximum number of different transitions for a given (q, a) is C.
- Number those transitions $1, \ldots, C$ (or less)
- Any computation of k steps is determined by a sequence of k numbers $\leq C$ (the "nondeterministic choices").
- How M' works: 3 tapes

$\#1$ | Original input to M $\sqcup$

$\#2$ | Simulated tape of M

$\#3$ | $1213\sqcup\cdots$ Nondeterministc choices for M'

Simulating one step of M

- Each major phase of the simulation by M' is to simulate one finite computation by M, using tape #3 to resolve nondeterministic ambiguities.
- Between major phases, M'
  - erases tape #2 and copies tape #1 to tape #2
  - Replaces string in $\{1, \ldots, C\}^*$ on tape #3 with the lexicographically next string to generate the next set of nondeterministic choices to follow.
- Claim: $L(M') = L(M)$
- **Q:** Slowdown?