Harvard CS 121 and CSCI E-207
Lecture 14: The Church–Turing Thesis

Salil Vadhan

October 18, 2012

- **Reading:** Sipser, §3.2, §3.3.

"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

$$\cdots \begin{array}{|c|c|c|c|c|}\hline \sqcup & a & b & a & a \\ \hline\end{array} \cdots$$

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."

$$\cdots \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline c & a & \sqcup & b & a & \sqcup & b & a & a \\ \hline -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 \end{array} \cdots$$

Tape of 2-way infinite TM M

$$\begin{array}{|c|}\hline b \\ \hline a \\ \hline\end{array} = \langle b, a\rangle \qquad \begin{array}{c|c|c|c|c|c|} & -1 & -2 & -3 & -4 & \\ \hline \multirow{2}{*}{\$} & b & \sqcup & a & c & \cdots \\ \cline{2-6} & a & \sqcup & b & a & a \\ \hline & 0 & 1 & 2 & 3 & 4 \end{array}$$

Corresponding tape of 1-way infinite TM M'

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, <u>two</u> 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

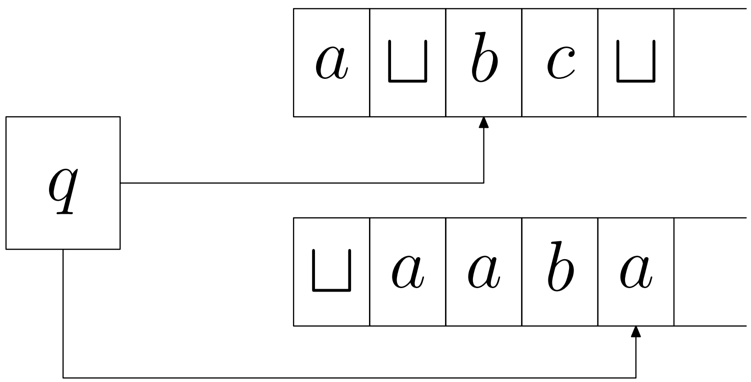
- Prose description of tape contents, head movements
- Omit details of states and transition functions (but do convince yourself that a TM can do what you're describing!)
- This lecture, next lecture, ps6

High-Level Description

- Starting in a couple of lectures...

More extensions

- Adding multiple tapes does not increase power of TMs



2-tape TM

(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)

<table>
<tr><td rowspan="4">$\$$</td><td>a</td><td>$\sqcup$</td><td>b</td><td>c</td><td>$\sqcup$</td><td></td></tr>
<tr><td></td><td></td><td>$\uparrow$</td><td></td><td></td><td></td></tr>
<tr><td>$\sqcup$</td><td>a</td><td>a</td><td>b</td><td>a</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>$\uparrow$</td><td></td></tr>
</table>

(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.

- <u>Claim:</u> $L(M') = L(M)$

- **Q:** Slowdown?