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

Harry Lewis

October 21, 2010

- **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

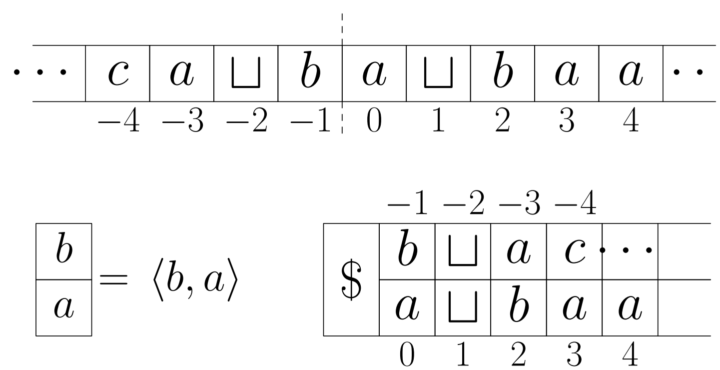
- TMs with a 2-way infinite tape, unbounded to left and right

$$\cdots \;\boxed{\sqcup}\boxed{a}\boxed{b}\boxed{a}\boxed{a}\; \cdots$$

  - Unbounded tape to left as well as right

<u>Proof</u> 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."




Tape of 2-way infinite TM $M$

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 $q_{\text{accept}}$
  - the reject state $q_{\text{reject}}$ $(\neq q_{\text{accept}})$
- $\Sigma$ is the input alphabet
- $\Gamma$ is the tape alphabet
  - Contains $\Sigma$
  - 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

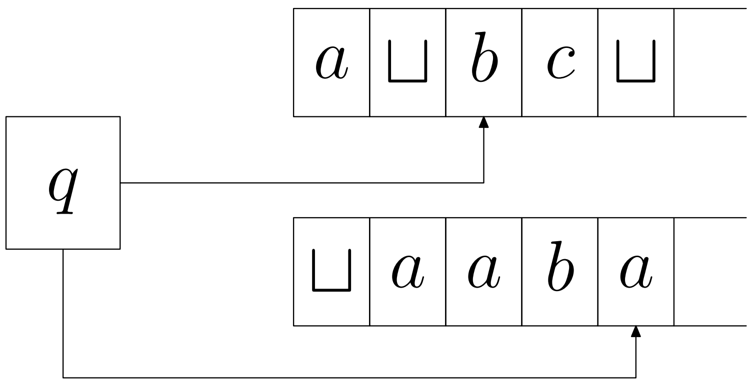
- Prose description of tape contents, head movements
- This lecture, some of next lecture, ps5

## High-Level Description

- Does not refer to specific computational model
- Starting next time!

## More extensions

- Adding <u>multiple tapes</u> 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 $2k$ <u>tracks</u>:
  - $k$ 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 $\sim |w|^2$. But there is a linear-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 $w$.

## 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 $b$.
- Number those transitions $1, \ldots, b$ (or less)
- Any computation of $k$ steps is determined by a sequence of $k$ numbers $\leq b$ (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 $\{1, \ldots, b\}^*$ 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?

# 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 $\lambda$-calculus ($\mu$-recursive functions)
- Markov algorithms
- Your favorite high-level programming language (C, Lisp, Java, ...)
- ... We'll just sketch two, general grammars and 2-counter machines

## General Grammars

- Like context-free grammars, except that if $u \to v$ is a rule, then $u$ may be any string containing a nonterminal.

- So the rule $AXY \to AYX$ where $A, X, Y \in V$, “means” that the two-symbol substring $XY$ can be replaced by $YX$ whenever it appears with an $A$ to its left.

## Example of a General Grammar

A grammar to generate $\{a^n b^n c^n : n \geq 0\}$.

$\Sigma = \{a, b, c\} \qquad V = \{A, B, C, A', B', C', S\}$

- $A, B, C$ are "aliases" for the terminal symbols $a, b, c$.
- Only a single occurrence of $A'$, $B'$, or $C'$ can be in the string being derived.
- It "crawls" from right to left, transforming nonterminal symbols into terminals.

# Rules for $a^n b^n c^n$

$$S \to ABCS \qquad S \to C' \qquad S \to \varepsilon$$

(Thus $S \Rightarrow^* (ABC)^n C'$ for any $n \geq 0$.)

$$CA \to AC \qquad BA \to AB \qquad CB \to BC$$

(Any inversions of the proper order can be repaired)

$$CC' \to C'c \qquad CC' \to B'c$$

(The $c$-transformer can crawl to the left, and turn into a $b$-transformer)

$$BB' \to B'b \qquad BB' \to A'b$$

$$AA' \to A'a \qquad A' \to \varepsilon$$

The only way to get a string of <u>terminals</u> yields one of the form $a^n b^n c^n$.

# Grammars and Turing Machines are Equivalent

Theorem: A language is generated by a grammar if and only if it is Turing-recognizable.

Proof:

(1) $L$ is generated by a grammar $\Rightarrow L$ is Turing-Recognizable

Pf: Let $L = L(G)$, $G$ a grammar. To construct an NTM $M$ such that $L(M) = L$, construct $M$ so that

$M$ nondeterministically carries out a derivation $S = w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \ldots$, checking each step to see if $w_i = w$.

# $L$ Turing-recognizable $\Rightarrow$ $L$ is generated by a grammar.

(2) $L$ is recognized by a TM $M \Rightarrow L$ is generated by a grammar $G$

Pf: Without loss of generality, we assume that if $M$ halts having started on input $w$, right before halting it erases its tape.

$G$ will simulate a backwards computation by $M$. The intermediate strings will be configurations $\$uqav\$$.

## Rules of the Grammar

- $S \rightarrow \$q_{\text{accept}}\$$
- If $\delta(q, a) = (p, b, R)$, then $G$ has

  $bp \rightarrow qa$

  $bp\$ \rightarrow q\$$, if $a = \sqcup$
- If $\delta(q, a) = (p, b, L)$, then $G$ has

  $pcb \rightarrow cqa$ for each $c \in \Sigma$

  $p\$ \rightarrow qa\$$, if $b = \sqcup$
- Finally, $\$ \rightarrow \varepsilon$ and, if $s$ is the start state of the TM, $s \rightarrow \varepsilon$

## Reduction of TMs to 2-CMs

A 2-counter machine (2-CM) has:

- A finite-state control
- Two counters, i.e. $C1$ and $C2$, which are registers containing integers $\geq 0$ with only $3$ operations:
  - Add $1$ to $C1/C2$
  - Subtract $1$ from $C1/C2$
  - Is $C1/C2 = 0$?

Theorem: For any TM, there is an equivalent 2-CM, in the sense that if you start the 2-CM with an encoding of the TM tape in its counters it will eventually halt with an encoding of what the TM computes.

# Simulating a TM tape with 2 pushdown stores: Split the tape at the head position into two stacks

| | | |
|---|---|---|
| Moving TM head to left | $\equiv$ | Pop from stack # 1<br>Push onto stack # 2 |
| Moving TM head to right | $\equiv$ | Pop from stack # 2<br>Push onto stack # 1 |
| Change scanned symbol | $\equiv$ | Change top of stack # 1 |

(So 2-PDAs are as powerful as TMs)

# Simulating One Stack with Two Counters: Think of the stack as number in a base = $|\Sigma| + 1$

[Assume $\leq 9$ stack symbols]

| | | |
|---|---|---|
| Pop the stack | $\equiv$ | Divide by $10$ and discard the remainder |
| Push $a_9$ | $\equiv$ | Multiply by $10$ and add $9$ |
| Is stack top $= a_3$? | $\equiv$ | Is counter mod $10 = 3$? |

$\rightarrow$ All of these can be calculated using a second counter.

# Simulating Four Counters With Two: $(p, q, r, s) \rightarrow 2^p 3^q 5^r 7^s$

| | | |
|---|---|---|
| Add $1$ to $C1$ | $\equiv$ | $p \leftarrow p + 1$ |
| | $\equiv$ | Double $C1'$ |
| Is $C3 \neq 0$? | $\equiv$ | $r \neq 0$? |
| | $\equiv$ | Does $5$ divide $C1'$ evenly? |
| Subtract $1$ from $s$ | $\equiv$ | Divide $C1'$ by $7$ |

# 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

# Is Church-Turing Thesis Provable?