CS 121: Lecture 13
Turing Equivalence & Universality

Madhu Sudan

https://madhu.seas.Harvard.edu/courses/Fall2020

Book: https://introtcs.org

How to contact us

The whole staff (faster response): CS 121 Piazza
Only the course heads (slower): cs121.fall2020.course.heads@gmail.com
Announcements:

• Advanced Sections: Josh Alman on Matrix Multiplication
• Midterms yet to be graded. Will post details on Piazza when ready
• Homework 4 out today. Due in two weeks.
• Participation Survey done?
  • Sign up for active participation here!
• Midterm Feedback Survey coming soon!
  • Mandatory (5 points on homework 4.). Anonymous!
  • Staff takes it seriously! (Be open – call out specific people, actions).
• Section 6 cycle starts today. Material in usual place!
Where we are:

Part I: Circuits: Finite computation, quantitative study

Part II: Automata: Infinite restricted computation, quantitative study

Part III: Turing Machines: Infinite computation, qualitative study

Part IV: Efficient Computation: Infinite computation, quantitative study

Part V: Randomized computation: Extending studies to non-classical algorithms
Today:

- Two results to be aware of, and to use (heavily)?
- No proofs to know/remember.
  - Proofs/sketches available in book.
  - We will discuss. But suffices to know they exist!
- Result 1: Turing-Church Thesis
  - Provable part: TMs as powerful as any high-level programming language.
  - Usable part: To prove computability, suffices to give program in high-level lang.
- Result 2: ∃ a Universal Turing Machine
  - Takes as input description $E(M) \in \{0,1\}^*$ of any Turing Machine, and $x \in \{0,1\}^*$
  - Outputs $M(x)$, the result computed by $M$ on $x$ (if $M$ halts) – no output otherwise.
Recall Turing Machines

• (Barak, Definition 7.1):
• TM on $k$ states and alphabet $\Sigma \supseteq \{0,1,\triangleright,\phi\}$ is given by $\delta: [k] \times \Sigma \rightarrow [k] \times \Sigma \times \text{Action}$, where $\text{Action} = \{L,R,S,H\}$
  • $L=$Left, $R=$Right, $S=$Stay (don’t move), $H=$Halt (done!!)
• Operation:
  • Start in state 0, Tape $T = \phi x_0 \ldots x_{n-1} \phi \phi \phi \ldots$, Head ($i$) at $x_0$
  • General step: current state $q$ ; input symbol $\sigma$:
    Let $\delta(q, \sigma) = (r, \tau, X) \Rightarrow \text{Write } \tau \text{ on tape (overwriting } \sigma) \text{ ; Move to state } r$;
    Move Head left ($i \leftarrow i - 1$) if $X = L$; right if $X = R$; don’t move if $X = S$.
  • Repeat General step until $X = H$
Exercise Break 1

• Pick a high-level language
• Identify features that are very different from Turing Machines.
• Discuss differences after the break.
My list of differences:

• General programming languages allow multiple, multidimensional arrays!
  • TMs have one array: Tape[0, ∞]

• Allow "random" (arbitrary) access into arrays/memory.
  • Can look at $A[i]$ in one step and then $A[i^2 + 10i + 5]$ or even $A[A[i]]$ in next step
  • TMs: If this step involves Tape[i]
     then next can only involve {Tape[$i - 1$], Tape[$i$], Tape[$i + 1$]}

• Rest? Syntactic Sugar
  • Sophisticated constructs: loops, cases, recursion
  • Data structures: Lists, Queues, Stacks ...
Dealing with the differences - 1

- Random access:
  - Deal with by brute force.
  - Store index on Tape. Compute new index and overwrite on tape.
  - Make a linear pass of tape to recover $A[i]$
  - (Quadratic slowdown in run time immediately)
Dealing with the differences - 2

- Multiple Arrays+Indices
  - Same solution.
- Multi-dimensional Arrays
  - (Draw this out)

- Consequence: If algorithm A runs in time $T$ with high-level program, can be implemented to run in time $O(T^2)$ on Turing Machine.
- Details in Barak: Chapter 8
Road Map of details

• TMs

• Define NAND-TMs. Show equivalent to TMs.
  • Just a program version of TMs. Like NAND circuits vs. NAND-CIRC programs.

• Define NAND-RAMs. Show equivalent to NAND-TMs.
  • Allows loops and general indices.
  • This is the crucial step.

• Define RAM machines. Show equivalent to NAND-RAMs
  • This what most compilers use to compile “down” from the high-level spec.
  • Equivalence straightforward.
“HOCAEIT” Theorem
Have Our Cake And Eat It Too

• Recall definition of **Computable**.
  • \( F: \{0,1\}^* \to \{0,1\}^* \) is computable iff it is computable by TM.

• **Equivalence (HOCAEIT) Theorem:** TMs are equivalent to High-Level Languages.

• Having our cake: To prove \( F \) is computable only need to exhibit algorithm in high-level language.

• Eating it: To prove \( F \) is not computable only need to rule out TMs.
Church-Turing Thesis

“Every function that is computable by physical means is (Turing Machine) computable.”

Some (made-up?) history:
- Church defined computability with $\lambda$-calculus
- Turing + Church compared notes and agreed their models were equivalent.
- Many other models were shown to be equivalent.
- Turing went on to do a postdoc under von Neumann.
- Von Neumann later introduced the “stored program architecture” of computer to the computer architects of the time. Led to the first physical computers.
- Conway invented Game of Life ... simplest Turing Equivalent model?
Universality

• “One machine to rule them all”

• “There exists a single program/algorithm/TM that can run all other programs/algorithms/TMs.”

• Formally:
  1. There exists a way to encode Turing Machines so that they can be (part of) input to other Turing Machines.
  2. The exists a universal machine $U$ that takes as input a pair $(M, x)$ and outputs $U(M, x) = M(x)$ (if $M$ halts on $x$)
Part 1: Encoding Turing Machines

- Should be familiar to us:

- Recall $M$ specified by $\Sigma \supseteq \{>, 0, 1, \phi\}$, $k$, $\delta: [k] \times \Sigma \to [k] \times \Sigma \times \{L, R, S, H\}$
  
  - First encode $E_\Sigma: \Sigma \to \{0,1\}^c$; $E_A: \{L, R, S, H\} \to \{0,1\}^2$, $E_k: [k] \to \{0,1\}^{\log k}$
  
  - So $\delta: \{0,1\}^{\log k+c} \to \{0,1\}^{\log k+c+2}$

- Encoding of $M = \text{Enc}(c, k, \delta(0,000), \delta(0,001) ... \delta(k-1,111))$

- Where $\text{Enc}: \mathbb{N} \times \mathbb{N} \times (\{0,1\}^{\log k+c+2})^{k2^c} \to \{0,1\}^*$ is some 1-1 function.

- Encoding of $M = \text{Enc}(c, k, \delta)$
Part 2: Interpreting the Encoding

• Definition: Configuration of a machine $M$ on input $x$ after $t$ steps of computation, denoted $C_t$, is the “full state of the computation”:
  • Current state of Turing Machine
  • Current contents of the Tape
  • Current location $i$ of Tape head

• Core of Universal TM $U$
  • “Universal-Stepper”: $(M, C_t) \mapsto (M, C_{t+1})$
Exercise Break 2

• Discuss how to organize the information \((M, C_t)\) on \(U\)'s tape:

• Describe (in English) steps needed to compute \((M, C_t) \leftrightarrow (M, C_{t+1})\)

Definition: Configuration of a machine \(M\) on input \(x\) after \(t\) steps of computation, denoted \(C_t\), is the “full state of the computation”:

- Current state of Turing Machine
- Current contents of the Tape
- Current location \(i\) of Tape head
Computing \((M, C_t) \mapsto (M, C_{t+1})\)

- Initially: Make space for (current state, head location, current symbol)
- In each round:
  - fetch contents of Tape[head location] and update
  - Look at the code of the TM to determine next state, next location, symbol to write.
  - Write the “symbol to write” at current location.
  - Update “head location”
- Conclusion: Lots of string manipulation (string copy), adjust ... nothing profound.
Summary of Lecture:

- Turing Equivalence and Turing-Church Thesis:
  - No proofs to remember. But encouraged to read the text (Chapter 8)
  - Do remember the HOCAEIT theorem! “Do not leave home without it.”
    1. To prove computability, give algorithm in high-level language.
    2. To prove non-computability, rule out TMs.

- Universal Turing machines:
  - Single machine to simulate all others:
    1. Similar to circuits.
    2. Big difference: Simulates larger machines over larger alphabets!!!!
Next Lecture

• Uncomputability.
  • Some functions are not computable no matter how much time we are willing to take!