CS 121: Lecture 11
More on Turing Machines

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: Christina Ilvento on Differential Privacy!
• Homework 3 due today.
• Sample midterm available for tech/TeX/rules.
• Actual Midterm:
  • Pick up on Canvas;
  • TeX your answers ;
  • Submit on Gradescope-submit your answers like a problem set.
• Section: no video this week; review for midterm.
  • Section on Turing Machines: next week.
• Midterm review materials:
  • Diego/Joanna’s handout
  • Past midterms: two on finite automata without solutions; several from Boaz with solutions.
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:

• Part 1: More examples of Turing Machines
  • TM to compute $PAL: \{0,1\}^* \rightarrow \{0,1\}$ where $PAL(x) = 1 \iff x = x^R$
  • TM to compute $h: \{0,1\}^* \rightarrow \{0,1\}^*$, where $h(x) = y$ where $x = yz$ and $|y| \in \{|z|, |z| + 1\}$

• Part 2: (Discussion) Looking to the future:
  • Computable functions.
    • Def (7.2 in Barak): Function computable $\iff$ computable by TM
  • Equivalence with other computing & non-computing models: Multiple tapes, RAM, $\lambda$-calculus, polynomials ...
Recall Turing Machines

• (Barak, Definition 7.1):
  • TM on \( k \) states and alphabet \( \Sigma \supseteq \{0, 1, \triangleright, \phi\} \)
  
is given by \( \delta: \mathbb{N} \times \Sigma \rightarrow \mathbb{N} \times \Sigma \times \text{Action}, \)
  where \( \text{Action} = \{L, R, S, H\} \)
  • \( L = \text{Left}, R = \text{Right}, S = \text{Stay (don’t move)}, H = \text{Halt (done!!)} \)

• Operation:
  • Start in state 0, Tape \( T = \#x_0 \ldots x_{n-1} \phi \phi \ldots \), 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 \( X = H \)
Recognizing Palindromes

- **PAL:** \( \{0,1\}^* \rightarrow \{0,1\} \) where \( PAL(x) = 1 \iff x = x^R \)

- **Overview/Idea:**
  - Scan left to right between #s.
  - Replace extreme symbols by # if they match, Reject if they don’t
  - Till middle region is empty.

\[ \text{LEVEL} \rightarrow 0110110 \rightarrow 0110 \rightarrow \text{ou} \]

```
\[
0110010
\]
```
More details:

- **Alphabet:** $\Sigma = \{0, 1, \triangleright, \phi, \#\}$
- **States:**
  - 0: Start
  - 1: Scan Right 0
  - 2: Scan Right 1
  - 3: Check 0
  - 4: Check 1
  - 5: Move Left
  - 6: Accept and Halt
  - 7: Reject and Clean Left
### Alphabet: \( \Sigma = \{0, 1, \triangleright, \phi, \#\} \)

#### States:
1. **Start**
2. **Scan Right 0**
3. **Scan Right 1**
4. **Check 0**
5. **Check 1**
6. **Move Left**
7. **Accept and Halt**
8. **Reject and Clean Left**

<table>
<thead>
<tr>
<th>State/Input</th>
<th>( \triangleright )</th>
<th>0</th>
<th>1</th>
<th>( \phi )</th>
<th>( # )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invalid</td>
<td>(1,#,R)</td>
<td>(2,#,R)</td>
<td>(1,1,H)</td>
<td>(1,1,H)</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
<td>(1,0,R)</td>
<td>(1,1,R)</td>
<td>(3,#,L)</td>
<td>(3,#,L)</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>invalid</td>
<td>(5,#,L)</td>
<td>(7,0,L)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>invalid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>invalid</td>
<td></td>
<td></td>
<td>invalid</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>invalid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>invalid</td>
<td></td>
<td></td>
<td>invalid</td>
<td></td>
</tr>
</tbody>
</table>
Alphabet: $\Sigma = \{0, 1, \triangleright, \phi, \#\}$

States:
0: Start
1: Scan Right 0
2: Scan Right 1
3: Check 0
4: Check 1
5: Move Left
6: Accept and Halt
7: Reject and Clean Left
<table>
<thead>
<tr>
<th>State/Input</th>
<th>$\triangleright$</th>
<th>0</th>
<th>1</th>
<th>$\phi$</th>
<th>$#$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invalid</td>
<td>(1,#,R)</td>
<td>(2,#,R)</td>
<td>(6,1,H)</td>
<td>(6,1,H)</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
<td>(1,0,R)</td>
<td>(1,1,R)</td>
<td>(3,#,L)</td>
<td>(3,#,L)</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
<td>(2,0,R)</td>
<td>(2,1,R)</td>
<td>(4,#,L)</td>
<td>(4,#,L)</td>
</tr>
<tr>
<td>3</td>
<td>invalid</td>
<td>(5,#,L)</td>
<td>(7,0,L)</td>
<td>invalid</td>
<td>(6,1,H)</td>
</tr>
<tr>
<td>4</td>
<td>invalid</td>
<td>(7,0,L)</td>
<td>(5,#,L)</td>
<td>invalid</td>
<td>(6,1,H)</td>
</tr>
<tr>
<td>5</td>
<td>invalid</td>
<td>(5,0,L)</td>
<td>(5,1,L)</td>
<td>invalid</td>
<td>(0,#,R)</td>
</tr>
<tr>
<td>6</td>
<td>invalid</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>(-, \triangleright, #)</td>
<td>(7,#,L)</td>
<td>(7,#,L)</td>
<td>invalid</td>
<td>(-,-,#)</td>
</tr>
</tbody>
</table>

Alphabet: $\Sigma = \{0, 1, \triangleright, \phi, \#$\}

States:
0: Start
1: Scan Right 0
2: Scan Right 1
3: Check 0
4: Check 1
5: Move Left
6: Accept and Halt
7: Reject and Clean Left

\[ \rightarrow \quad \# \quad \rightarrow \]
Exercise Break 1

• Design TM to compute \( h: \{0,1\}^* \to \{0,1\}^* \), where \( h(x) = y \) where \( x = yz \)
  and \( |y| \in \{|z|, |z| + 1\} \)

1. Formulate your plan

2. Break from Break (Return from Break + Discuss Plan)

3. Choose your alphabet \( \{D, 0, 1, \phi, \#, \#, \#, \#\} \)

4. Set up the states

5. Start thinking about key transitions

\[
\begin{align*}
X &= 0110101 & y &= 0110 & z = 101 \\
X &= 0001111 & y &= 000 & z = 111
\end{align*}
\]
One solution:

• Alphabet: $\Sigma = \{0,1, \triangleright, \phi, \#, \#_0, \#_1\}$

• States:
  • 0: Start: Replace $b$ by $\#_b$
  • 1: Zig Right
  • 2. Erase last symbol
  • 3. Zag Left
Alphabet: $\Sigma = \{0, 1, \downarrow, \phi, \#, \#_0, \#_1\}$
States:
0. Start: Replace $b$ by $\#_b$
1. Zig Right
2. Erase last symbol
3. Zag Left, Replace $\#_b$ by $b$, go to start

<table>
<thead>
<tr>
<th>State/Input</th>
<th>$\downarrow$</th>
<th>0</th>
<th>1</th>
<th>$\phi$</th>
<th>$#$</th>
<th>$#_0$</th>
<th>$#_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invalid</td>
<td>(1,$#_0$,R)</td>
<td>(1,$#_0$,R)</td>
<td>(-,$\phi$,H)</td>
<td>(-,$#$,H)</td>
<td>invalid</td>
<td>invalid</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
<td>(1,0,R)</td>
<td>(1,1,R)</td>
<td>(2,$\phi$,L)</td>
<td>(2,$#$,L)</td>
<td>invalid</td>
<td>invalid</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
<td>(3,$#$,L)</td>
<td>(3,$#$,L)</td>
<td>invalid</td>
<td>invalid</td>
<td>(-,0,H)</td>
<td>(-,1,H)</td>
</tr>
<tr>
<td>3</td>
<td>invalid</td>
<td>(3,0,L)</td>
<td>(3,1,L)</td>
<td>invalid</td>
<td>invalid</td>
<td>(0,0,R)</td>
<td>(0,1,H)</td>
</tr>
</tbody>
</table>
Computable Functions

- **Definition (7.1 in Barak):** A function $f: \{0,1\}^* \to \{0,1\}^*$ is computable if and only if it is computable by a Turing Machine.

- **Warning:** Definition, not a Theorem!

- **Definition:** $R = \{ f: \{0,1\}^* \to \{0,1\} \mid f \text{ is computable} \}$
  - Why $R$? ("Recursive")

- **Turing-Church Thesis:** $f$ is computable by a physical process if and only if it is computable (by a Turing Machine).
In following lectures

• Turing Equivalence
  • Turing machines can simulate other Turing Machines
    • With multiple tapes
    • With accept/reject states
    • With 1 tape and multiple heads
  • RAM programs: (Main diff: Can read Tape[i] and then Tape[3i+25] in O(1) steps.
  • High-level programs – C++, Python ...
  • Rewrite systems; \( \Lambda \)-Calculus ; Hilbert Problem

• **Universal** TMs: TM that takes other TMs as input and runs them!

• **Uncomputability** ... the bane of computing.