CSE 30151
Theory of Computing
TUESDAY, 2018/03/20
READING: SIPSER 3.1
Turing machines are more powerful than
CFGs and PDAs are more powerful than
DFAs, NFAs, and regular expressions
Intuitive notion of algorithm = Turing machine algorithm
TURING MACHINES
OVERVIEW

- **Tape** that has a left end and extends infinitely to the right
- **Head** that moves across the cells of the tape
- **State** (just like finite and pushdown automata)
**TURING MACHINES**

**INITIAL CONFIGURATION**

<table>
<thead>
<tr>
<th>tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>_</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

- **Tape** initialized to input string followed by blanks (_)
- **Head** starts at first cell of state
- **State** is the start state (q₀)
TURING MACHINES

TRANSITIONS

If in state $q$ and read symbol $a$
then write $b$, move in direction $D$, and go to state $r$
where $D$ can be L (left), S (stay), or R (right)

If in state $q$ and read symbol $a_1$ or $a_2$
then move in direction $D$ and go to state $r$
If a state has no transition for a symbol, assume there is an implicit transition to the reject state.
<table>
<thead>
<tr>
<th>Outcome</th>
<th>Description</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td>and halt</td>
<td>by entering $q_{\text{accept}}$</td>
</tr>
<tr>
<td>reject</td>
<td>and halt</td>
<td>by entering $q_{\text{reject}}$</td>
</tr>
<tr>
<td>loop</td>
<td></td>
<td>otherwise</td>
</tr>
</tbody>
</table>
TURING MACHINES
RECOGNIZING AND DECIDING LANGUAGES

Turing-recognizable:
If the string is in $L$, then accept and halt
Otherwise, reject and halt, or loop

(Turing-)decidable:
If the string is in $L$, then accept and halt
Otherwise, reject and halt
TURING MACHINES
THREE WAYS OF WRITING

• Formal description: tuple and table, or state diagram

• Implementation description: pseudocode
  • Describes exact contents of tape and motion of head
  • Arithmetic, etc. not allowed
  • Should enable the reader to reimplement the machine

• High-level description:
  • Should convince the reader that the machine exists
TURING MACHINES
EXAMPLE IMPLEMENTATION DESCRIPTION

\[ A = \{0^n \mid n \text{ is a power of 2}\} \]

\[ M_2 = \text{“On input string } w: \]

1. Sweep left to right across the tape, crossing off every other 0.

2. If in stage 1 the tape contained a single 0, accept.

3. If in stage 1 the contained more than a single 0 and the number of 0s was odd, reject.

4. Return the head to the left-hand end of the tape.

5. Go to stage 1.”
TURING MACHINES
EXAMPLE FORMAL DESCRIPTION (STATE DIAGRAM)
\[
\begin{array}{|c|c|c|c|c|}
\hline
_x & x & x & x & \_ \\
\hline
\end{array}
\]

Diagram:

- \( q_1 \rightarrow_R \) \( q_{accept} \) (\( x \rightarrow R \), \( \_ \rightarrow R \))
- \( q_{reject} \rightarrow_R \) \( q_1 \) (\( \_ \rightarrow R \), \( x \rightarrow R \))
- \( q_2 \rightarrow_R \) \( q_{accept} \) (\( x \rightarrow R \), \( \_ \rightarrow R \))
- \( q_{accept} \rightarrow_R \) \( q_4 \) (\( x \rightarrow R \))
- \( q_4 \rightarrow_R \) \( q_{accept} \) (\( \_ \rightarrow R \))
- \( q_5 \rightarrow_R \) (\( 0 \rightarrow L \), \( x \rightarrow L \))
- \( q_5 \rightarrow_R \) \( q_1 \) (\( \_ \rightarrow R \), \( \_ \rightarrow L \))

Transitions:

- \( 0 \rightarrow _- \), \( R \)
- \( 0 \rightarrow x, R \)
- \( 0 \rightarrow R \)
- \( 0 \rightarrow x, R \)
\[
\begin{array}{cccccc}
\_ & \textbf{X} & \textbf{X} & \textbf{X} & \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ & \ldots
\end{array}
\]

\[
\begin{array}{c}
\text{ACCEPT} \\
\text{REJECT}
\end{array}
\]

\[
\begin{array}{c}
q_1 \\
q_{\text{reject}}
\end{array}
\]

\[
\begin{array}{c}
q_2 \\
q_{\text{accept}}
\end{array}
\]

\[
\begin{array}{c}
q_3 \\
q_4
\end{array}
\]

\[
\begin{array}{c}
q_5
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \_ \\
x \rightarrow \text{L}
\end{array}
\]

\[
\begin{array}{c}
x \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \_ \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \_ \rightarrow \text{L}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \_ \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \_ \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
x \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
x \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
x \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \text{R}
\end{array}
\]

\[
\begin{array}{c}
0 \rightarrow \text{R}
\end{array}
\]
\[ q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \]

Transitions:
- \( 0 \rightarrow L \)
- \( x \rightarrow L \)
- \( x \rightarrow R \)
- \( \_ \rightarrow R \)
- \( \_ \rightarrow L \)
- \( \_ \rightarrow R \)
- \( 0 \rightarrow \_ \), \( R \)
- \( 0 \rightarrow x \), \( R \)
- \( 0 \rightarrow x \), \( R \)
- \( 0 \rightarrow x \), \( R \)
- \( x \rightarrow R \)
- \( x \rightarrow R \)

States:
- \( q_{reject} \)
- \( q_{accept} \)
Write a state diagram for a Turing machine recognizing the language \( \{a^{2n} \mid n \geq 0\} \).
Write an implementation description, then a state diagram for a Turing machine recognizing the language \{ww^R \mid w \in \{0,1\}^*\}. 
THURSDAY, 2018/03/22
READING: SIPSER 3.2
CHURCH-TURING THESIS
IN MODERN LANGUAGE

Intuitive notion of algorithm

= Turing machine algorithm
CHURCH-TURING THESIS
WHY SHOULD WE BELIEVE IT?

- Turing’s original argument
- Convergence of several proposed models
  - Turing machines (1936)
  - Untyped lambda calculus (1936)
  - Partial recursive functions (1920, 1935, 1952)
  - Unrestricted (type 0) grammars (1956)
1+1=2 IN LAMBDA CALCULUS

\[
\lambda m.\lambda n.\lambda f.\lambda x. mf(nfx) \ (\lambda f.\lambda x. fx) \ (\lambda f.\lambda x. fx)
\]

\[
(\lambda n.\lambda f.\lambda x. (\lambda f.\lambda x. fx)f(nfx)) \ (\lambda f.\lambda x. fx)
\]

\[
\lambda f.\lambda x. (\lambda f.\lambda x. fx)f((\lambda f.\lambda x. fx)fx)
\]

\[
(\lambda f.\lambda x. (\lambda x. fx)((\lambda f.\lambda x. fx)fx)) \ (\lambda f.\lambda x. fx)
\]

\[
\lambda f.\lambda x. f((\lambda f.\lambda x. fx)fx)
\]

\[
\lambda f.\lambda x. f((\lambda x. fx)x)
\]

\[
\lambda f.\lambda x. f(fx)
\]
CHURCH-TURING THESIS

WHY SHOULD WE BELIEVE IT?

- Turing’s original argument
- Convergence of several proposed models
  - Turing machines (1936)
  - Untyped lambda calculus (1936)
  - Partial recursive functions (1920, 1935, 1952)
  - Unrestricted (type 0) grammars (1956)
- Today: Explore extensions to Turing machines
ALL OF UNDERGRADUATE COMPUTER SCIENCE
ACCORDING TO ME

- Fundamentals
- Data Structures
- Systems Programming
- Compilers
- Operating Systems
- Architecture
- Logic Design
ALL OF UNDERGRADUATE COMPUTER SCIENCE
ACCORDING TO ME

- Algorithms
- RAM model
- Turing machines
- Theory

- Fundamentals
- Data Structures
- Systems Programming

- Compilers
- Operating Systems

- Architecture

- Logic Design
## TURING MACHINES

**DISCUSS**

What do computers (or computer languages) have that Turing Machines don’t?

<table>
<thead>
<tr>
<th>variables</th>
<th>output of strings, numbers, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>numbers, arithmetic</td>
<td>output, e.g., graphics, sound, music</td>
</tr>
<tr>
<td>process one character at a time</td>
<td>input, e.g., mouse, keyboard</td>
</tr>
<tr>
<td>loops, if/then/else</td>
<td>network</td>
</tr>
<tr>
<td>functions</td>
<td></td>
</tr>
<tr>
<td>data structures</td>
<td></td>
</tr>
<tr>
<td><strong>random access memory</strong></td>
<td></td>
</tr>
<tr>
<td><strong>concurrency</strong></td>
<td></td>
</tr>
<tr>
<td>classes</td>
<td></td>
</tr>
</tbody>
</table>
MULTITAPE TURING MACHINES

IDEA

- Fixed (usually small) number of tapes
- One head per tape, each moving independently
- Single global state
MULTITAPE TURING MACHINES
EQUIVALENCE WITH SINGLE-TAPE

How do you convert a multitape Turing machine into an equivalent single-tape Turing machine?
MULTITAPE TURING MACHINES
EQUIVALENCE WITH SINGLE-TAPE

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>...</th>
</tr>
</thead>
</table>

becomes

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>_</th>
<th>...</th>
</tr>
</thead>
</table>

| # | 0 | 0 | 0 | 0 | # | 1 | 1 | 0 | 1 | 1 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
NONDETERMINISTIC TURING MACHINES

IDEA

Machine will follow both transitions in two computation branches.
<table>
<thead>
<tr>
<th>accept</th>
<th>when any branch enters $q_{\text{accept}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>and halt</td>
</tr>
<tr>
<td>reject</td>
<td>when all branches enter $q_{\text{reject}}$</td>
</tr>
<tr>
<td>loop</td>
<td>otherwise</td>
</tr>
<tr>
<td></td>
<td>and halt</td>
</tr>
</tbody>
</table>
How do you convert a nondeterministic Turing machine into an equivalent deterministic Turing machine?
Each string here selects a branch: choose #1, then #2, etc.
Enumerate all branches in BFS order: 1, 2, 3, ..., 11, 12, 13, ..., 21, 22, 23, ...
### NONDETERMINISTIC TURING MACHINES

**EQUIVALENCE WITH DETERMINISTIC SINGLE-TAPE**

<table>
<thead>
<tr>
<th></th>
<th>q₁</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>#</th>
<th>1</th>
<th>q₂</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>#</th>
<th>...</th>
</tr>
</thead>
</table>

- Tape holds a queue of simulated configurations
  - State on tape means head is on next square
- While front configuration is not accepting:
  - For each possible move in front configuration:
    - Push new configuration to back of queue
  - Pop front configuration
## RANDOM ACCESS MACHINES

### IDEA

|   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 123 | 5 | 0 | -6 | 7 | 1 | -88 | 1 | ... |

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X[i] \leftarrow C$</td>
<td>write constant</td>
</tr>
<tr>
<td>$X[i] \leftarrow X[j] + X[k]$</td>
<td>add (also subtract) cells</td>
</tr>
<tr>
<td>$X[i] \leftarrow X[X[j]]$</td>
<td>copy from dereferenced cell</td>
</tr>
<tr>
<td>$X[X[i]] \leftarrow X[j]$</td>
<td>copy to dereferenced cell</td>
</tr>
<tr>
<td>$\text{TRA } m \text{ if } X[j] &gt; 0$</td>
<td>conditional branch</td>
</tr>
</tbody>
</table>
How do you convert a random access machine into an equivalent multitape Turing machine?
Cook-Reckhow targeted a multitape TM: additional tapes for an address register, a value register, and scratch space.

I don’t know how they represented negative numbers.
Demo:
C code → ELVM assembly → Turing machine
https://github.com/shinh/elvm
TURING MACHINES
WHAT’S NEXT?

• The most powerful Turing machine
• Is there a language that is undecidable?
• What other languages are undecidable?
• Is there really nothing beyond Turing machines?