Overview

Attempt to understand essential nature of computation by studying properties of simple machine models.

Goal: simplest machine that is "as powerful" as conventional computers.

Surprising Fact 1.

Surprising Fact 2.

Adding Power to FSA

FSA advantages:
- Extremely simple and cheap to build.
- Well suited to certain important tasks.
  - pattern matching, filtering, dishwashers, remote controls, traffic lights, sequential circuits

FSA disadvantages:
- Not sufficiently "powerful" to solve numerous problems of interest.

How can we make FSA’s more powerful?
- NFSA = FSA + "nondeterminism", i.e., ability to guess the right answer (!)

Nondeterministic Finite State Automata

Nondeterministic FSA (NFSA).
- Simple machine with N states.
- Start in state 0.
- Read a bit.
- Depending on current state and input bit
  - move to any of several new states
- Stop when last bit read.
- Accept if ANY choice of new states ends in state X, reject otherwise.

If in state 2, and next bit is 1:
  - can move to state 1
  - can move to state 2
  - can move to state 3
Nondeterministic Finite State Automata

Nondeterministic FSA (NFSA).
- Simple machine with N states.
- Start in state 0.
- Read a bit.
- Depending on current state and input bit
  - move to any of several new states
- Stop when last bit read.
- Accept if ANY choice of new states ends in state X, reject otherwise.

Which strings are accepted?
✓ 0010001
✓ 00
✓ 1000111001100
✓ 1000111001101

A Systematic Method for NFSA

Harder to determine whether an NFSA accepts a string than an FSA.
- For FSA, only one possible path to follow.
- For NFSA, need to consider many paths.

Systematic method for NFSA.
- Keep track of ALL possible states that the NFSA could be in for a given input.
- Accept if one of possible ending states is accept state.

Power of nondeterminism is very useful, but is it essential?

NFSA Example 2

Build an NFSA to match all strings whose 5th to last character is 'x'.
- % egrep ‘x...$’ /usr/dict/words
  - asphyxiate
  - carboxylic
  - contextual
  - inflexible

A Systematic Method for NFSA

Harder to determine whether an NFSA accepts a string than an FSA.
- For FSA, only one possible path to follow.
- For NFSA, need to consider many paths.

Systematic method for NFSA.
- Keep track of ALL possible states that the NFSA could be in for a given input.
- Accept if one of possible ending states is accept state.

Power of nondeterminism is very useful, but is it essential?

FSA - NFSA Equivalence

Theorem: FSA and NFSA are "equally powerful".
- Given any NFSA, can construct FSA that accepts same inputs.

Notation: $X \subseteq Y$.
- Y is at least as powerful as X.
- Machine class Y all the languages that X can (and maybe more).

Proof (Part 1): $FSA \subseteq NFSA$.
- A FSA is a special type of NFSA.
**FSA - NFSA Equivalence**

Theorem: FSA and NFSA are "equally powerful".
- Given any NFSA, can construct FSA that accepts same inputs.

Notation: $X \subseteq Y$.
- $Y$ is at least as powerful as $X$.
- Machine class $Y$ all the languages that $X$ can (and maybe more).

Proof (part 2): NFSA $\subseteq$ FSA.
- Given a nondeterministic FSA, we give method to construct a deterministic FSA that recognizes the same language.
- One state in FSA for every set of states in the NFSA.
- N-state NFSA $\Rightarrow 2^N$ state FSA.

**Pushdown Automata**

How can we make FSA more powerful?
- Nondeterminism didn't help.
- Instead, add "memory" to the FSA.
- A pushdown stack (amount of memory is arbitrarily large)

Pushdown Automata (PDA).
- Simple machine with N states.
- Start in state 0.
- Read a bit, check bit at top of stack.
- Depending on current state/input bit/stack bit:
  - move to new state
  - push the input onto stack, or pop topmost element from stack
- Stop when last bit is read.
- ACCEPT if stack is empty, REJECT otherwise.

**PDA for deciding whether input is of form $0^N1^N$.**

- N 0's followed by N 1's for some N.
- $\varepsilon$, 01, 0011, 000111, \ldots
- Use notation $x/y/z$
- If input is $x$ and top of stack is $y$, then do $z$.

Did it help?
- More powerful, can recognize:
  - all bit strings with an equal number of 0's and 1's
  - all bit strings of the form $0^N1^N$
  - all "balanced" strings in alphabet: $(, [, , ], ),$
- Can't recognize language of all palindromes.
  - 11 * 181 = 1991 = 181 * 11
  - a man a plan a canal al pan a
  - murder for a jar of red rum
- More powerful machines still needed.
Turing Machine

- Simple machine with N states.
- Start in state 0.
- Input on an arbitrarily large TAPE that can be read from 'and' written to.
- Read a bit from tape.
- Depending on current state and input bit
  - write a bit to tape
  - move tape right or left
  - move to new state
- Stop if enter yes or no state.
- Accept if yes, reject if no or does not terminate.

New accept / reject mechanism

Some Examples

Build Turing machines that accepts inputs that:

- have an equal number of 0's and 1's.
  #1100#, #0011#, #01101110000#

- are even length palindromes of 0's and 1's.
  #0110#, #110011#, #10111000011101#

- have a power of two 1's.
  #1#, #11#, #1111#, #11111111#

C Program to Simulate Turing Machine

Three character alphabet (0 is ‘blank’).

Input: description of machine (9 integers per state s).
- next[i][s] = t means if currently in state s and input character read in is i, then transition to state t.
- out[i][s] = w means if currently in state s and input character read in is i, then write w to current tape position.
- move[i][s] = ±1 means if currently in state s and input character read in is i, then move tape cursor one position to left or right.
- tape[i] is i\textsuperscript{th} character on tape initially.

Details missing:
- Might run off end of tape.
Nondeterministic Turing Machine

TM with extra ability:
- Choose one of several possible transition states given current tape contents and state.

Exercise:
- Nondeterministic TM to recognize language of all bit strings of the form \(ww\) for some \(w\).
  - 110110
  - 100011110001111
  - 001100011100011110001111

Abstract Machine Hierarchy

Each machine is strictly more powerful than the previous.
- Power = can recognize more languages.

Are there limits to machine power?

Corresponding hierarchy exists for languages.
- Essential connection between machines and languages.
  (See Lecture T3.)

<table>
<thead>
<tr>
<th>Machine</th>
<th>Nondeterminism adds power?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite state automata</td>
<td>No</td>
</tr>
<tr>
<td>Pushdown automata</td>
<td>Yes</td>
</tr>
<tr>
<td>Linear bounded automata</td>
<td>Unknown</td>
</tr>
<tr>
<td>Turing machine</td>
<td>No</td>
</tr>
</tbody>
</table>

Summary

Abstract machines are foundation of all modern computers.
- Simple computational models are easier to understand.
- Leads to deeper understanding of computation.

Goal: simplest machine that is "as powerful" as conventional computers.

Abstract machines.
- FSA: simplest machine that is still interesting.
  - pattern matching, sequential circuits (Lecture T1)
- PDA: add read/write memory in the form of a stack.
  - compiler design (Lecture T3)
- TM: add memory in the form of an arbitrarily large array.
  - general purpose computers (Lecture T4)

Lecture T2: Extra Slides
Theorem: FSA, NFSA, and RE are "equally powerful".

- NFSA $\subseteq$ FSA

Proof sketch (part 2): FSA $\subseteq$ RE

- Goal: given an FSA, find a RE that matches all strings accepted by the FSA and no other strings.
- Main idea: consider
  - paths from start state(s) to accept state(s): $00 \mid 01$
  - directed cycles: $(1^*) (00 \mid 01) (11 \mid 10)^*$

Example.

RE: $01(00 \mid 101)^*$

Example.

RE: $01(00 \mid 101)^*$
FSA, NFSA, and RE Are Equivalent

Example.
- RE: 01(00 | 101)*

\[ (00 | 101)^* \]

\( \varepsilon \) - transition: jump states without reading a character to next state

Nondeterminism Does Help PDA’s

Nondeterministic pushdown automata (NPDA).
- Same as PDA, except depending on current state/input bit/stack bit
  - move to ANY OF SEVERAL new states
  - push the input onto stack, or pop top-most element from stack

NPDA to recognize all (even length) palindromes.
- Bit string is the same forwards and backwards.

\[
\begin{align*}
0/0/\text{push} \\
0/1/\text{push} \\
0/\varepsilon/\text{push} \\
0/0/\text{push} \\
0/1/\text{push} \\
1/\varepsilon/\text{push} \\
1/0/\text{push} \\
1/1/\text{push} \\
1/\varepsilon/\text{push}
\end{align*}
\]

PDA \( \subseteq \) NPDA trivially.
- PDA cannot recognize language of all (even length) palindromes, but NPDA can.
- Therefore PDA \( \subseteq \) NPDA.
Pushdown Automata

How can we make FSA more powerful?
- NPDA = FSA + stack + nondeterminism.

Did it help?
- Can recognize language of all palindromes.
- Can’t recognize some languages:
  - equal number of 0’s 1’s and 2’s
  - $0^N 1^N 2^N$
  - bit strings with a power of two 1’s
- Need still more powerful machines.

Linear Bounded Automata

Turing machine.
- No limit on length of tape.

Linear bounded automata (LBA).
- Same as TM except length of tape = $K \cdot (\text{size of input})$.

LBA is strictly less powerful than TM.
- There are languages that can be recognized by TM but not a LBA.
- We won’t dwell on LBA in this course.