Lecture T2: Turing Machines
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 FSAs more powerful?
- NFSA = FSA + "nondeterminism."
  (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.
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.

If in state 2, and next bit is 0: can’t move to any state
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
✔ 10000111001100
✔ 10000111001101
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

```
0 1 2 3 4 5
```

```
x a-z a-z a-z a-z
```

```
a-z
```

```
a-z
```
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?
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 \( Y \) can be "programmed" to accept all the languages that \( X \) can (and maybe more).

Proof (Part 1): FSA \( \subseteq \) NFSA.
- A FSA is a special type of NFSA.
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 $Y$ can be "programmed" to accept all the languages that $X$ can (and maybe more).

Proof (Part 2): NFSA $\subseteq$ FSA.
- Given a nondeterministic FSA, we give recipe 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.
RE – FSA Equivalence

**Theorem:** FSA and RE are "equally powerful".

- We’ll spare you the details. 😊
- Interested students: see supplemental lecture slides.
Pushdown Automata

How can we make FSA’s 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.
Pushdown Automata

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, 00001111, ...,
- Use notation $x/y/z$
- If input is $x$ and top of stack is $y$, then do $z$.

Diagram:

- Initial state: 0
- Transition: 0/0/push
- Transition: 1/0/pop
- Final state: 1
- Transition: 1/0/pop
- Transition: 0/\varepsilon/push
Pushdown Automata

How can we make FSA more powerful?

- PDA = FSA + stack.

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: (, {, [ , ]}, }
- Still can’t recognize language of all palindromes.
  - amanaplanacanalpanama
  - 11*181=1991=181*11
  - murderforajarofredrum

- 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.
Some Examples

Build Turing machines that accepts following languages:

- Equal number of 0’s and 1’s.
  - #1100#, #0011#, #011101110000#

- Even length palindromes of 0’s and 1’s.
  - #0110#, #110011#, #10111000011101#

- Power of two 1’s.
  - #1#, #11#, #1111#, #11111111#

Notation.
- \(x/y/z\): if TM head contains character \(x\), then change it to \(y\), and move head in direction \(z\).
- # special character.
C Program to Simulate Turing Machine

Three character alphabet (0 is ‘blank’).

Position on tape.
- head

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

Details missing:
- Might run off end of tape.
C Program to Simulate Turing Machine

```c
#define MAX_TAPE_SIZE   2000
#define STATES   100
#define ACCEPT_STATE      99

int next[3][STATES], out[3][STATES], move[3][STATES];
char tape[MAX_TAPE_SIZE];
int in, d, state = 0, head = MAX_TAPE_SIZE / 2;

... /* read in machine from file */

while (scanf("%1d", &d) != EOF)
    tape[head++] = d;

while (state != ACCEPT_STATE) {
    in = tape[cursor];
    state   = next[in][state];
    tape[head] = out[in][state];
    head      += move[in][state];
}
```

- **read in tape (consists of 0, 1, 2)**
- **simulate Turing machine until accept state reached**
Nondeterministic Turing Machine

**TM with extra ability:**

- Choose one of several possible transition states given current tape contents and state.
- No more powerful than deterministic TM.
- Faster than TM? (Stay tuned for NP-Completeness).

**Exercise:**

- Nondeterministic TM to recognize language of all bit strings of the form $ww$ for some $w$.
  - 110110
  - 100011110001111
  - 001100011100001111001100011100001111

110110
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 "as powerful" as conventional computers.

Abstract machines.
- FSA: simplest machine that is still interesting.
  - pattern matching, sequential circuits (Lecture T1)
  - can’t recognize: equal number of 0’s and 1’s
- PDA: add read/write memory in the form of a stack.
  - compiler design (Lecture T3)
  - can’t recognize: palindromes
- TM: add memory in the form of an arbitrarily large array.
  - general purpose computers (Lecture T4)
  - can’t recognize: stay tuned
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 | 01
  - directed cycles: $(1^*) (00 | 01) (11 | 10)^*$
Theorem: FSA, NFSA, and RE are "equally powerful".

- NFSA $\subseteq$ FSA $\subseteq$ RE

Proof sketch (part 3): RE $\subseteq$ NFSA

- Goal: given a RE, construct a NFSA that accepts all strings matched by the RE, and rejects all others.
- Use the following rules to construct NFSA:
FSA, NFSA, and RE Are Equivalent

Example.
- RE: 01(00 | 101)*
FSA, NFSA, and RE Are Equivalent

Example.
- RE: 01(00 | 101)*
FSA, NFSA, and RE Are Equivalent

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

$\epsilon$ - transition: jump states without reading a character to next state

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

Example.

- RE: $01(00 \mid 101)^*$
Theorem: FSA, NFSA, and RE are "equally powerful".

- NFSA ⊆ FSA ⊆ RE ⊆ NFSA

Equivalence of languages and machine models is essential in the theory of computation.
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.
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.

Nondeterministic PDA more powerful than deterministic PDA.
- PDA ⊆ NPDA trivially.
- PDA cannot recognize language of all (even length) palindromes, but NPDA can.
- Therefore PDA ⊂ 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).
- A single tape TM that can only write on the portion of the tape containing the input.
- Note: allowed to increase alphabet size if desired.

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.