Lecture 18: Theory of Computation

Two fundamental questions.

- What can a computer do?
- What can a computer do with limited resources?

General approach.

- Don’t talk about specific machines or problems.
- Consider minimal abstract machines.
- Consider general classes of problems.

Why Learn Theory

In theory...
- Deeper understanding of what is a computer and computing.
- Foundation of all modern computers.
- Pure science.
- Philosophical implications.

In practice...
- Web search: theory of pattern matching.
- Sequential circuits: theory of finite state automata.
- Compilers: theory of context free grammars.
- Cryptography: theory of computational complexity.
- Data compression: theory of information.

"In theory there is no difference between theory and practice. In practice there is." - Yogi Berra

Regular Expressions and DFAs

\[ a^* \mid (a^*ba^*ba^*)^* \]
Pattern Matching Applications

- Test if a string matches some pattern.
  - Process natural language.
  - Scan for virus signatures.
  - Search for information using Google.
  - Access information in digital libraries.
  - Retrieve information from Lexis/Nexis.
  - Search-and-replace in a word processors.
  - Filter text (spam, NetNanny, Carnivore, malware).
  - Validate data-entry fields (dates, email, URL, credit card).
  - Search for markers in human genome using PROSITE patterns.

Parse text files.
- Compile a Java program.
- Crawl and index the Web.
- Read in data stored in TOY input file format.
- Automatically create Java documentation from Javadoc comments.

Regular Expressions: Examples

Regular expression. Notation is surprisingly expressive.

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>.* spb .* contains the trigraph spb</td>
<td>raspberry crisbread</td>
<td>subspace subspecies</td>
</tr>
<tr>
<td>a*</td>
<td>(a<em>ba</em>ba*)* multiple of three a's</td>
<td>bbb aababbaa</td>
</tr>
<tr>
<td>.+0.... fifth to last digit is 0</td>
<td>10000 98701234</td>
<td>11111111 403982772</td>
</tr>
<tr>
<td>ggc (gg</td>
<td>gagg)* ctg fragile X syndrome indicator</td>
<td>ggcgtg ggcggggtg ggcggagctg ggcgggctg</td>
</tr>
</tbody>
</table>

Regular Expressions: Basic Operations

Regular expression. Notation to specify a set of strings.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Regular Expression</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenation</td>
<td>aabaab</td>
<td>aabaab</td>
<td>every other string</td>
</tr>
<tr>
<td>Wildcard</td>
<td>.u.</td>
<td>cumulus jugulum</td>
<td>succubus tumulous</td>
</tr>
<tr>
<td>Union</td>
<td>aa</td>
<td>baab</td>
<td>aa</td>
</tr>
<tr>
<td>Closure</td>
<td>ab*a</td>
<td>aa</td>
<td>abhaba</td>
</tr>
<tr>
<td>Parentheses</td>
<td>a(a</td>
<td>b)a</td>
<td>aababab</td>
</tr>
</tbody>
</table>

Generalized Regular Expressions

Regular expressions are a standard programmer's tool.
- Built in to Java, Perl, Unix, Python, ...
- Additional operations typically added for convenience.
- Ex: [a-e]+ is shorthand for (a|b|c|d|e)(a|b|c|d|e)*.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Regular Expression</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>One or more</td>
<td>a(bc)+de</td>
<td>abcd</td>
<td>abcdode</td>
</tr>
<tr>
<td>Character classes</td>
<td>[A-Za-z][a-z]*</td>
<td>capitalized Word</td>
<td>camelCase 4illegal</td>
</tr>
<tr>
<td>Exactly k</td>
<td>[0-9]{5}-[0-9]{4}</td>
<td>08540-1321 19072-5541</td>
<td>11111111 16-54-111</td>
</tr>
<tr>
<td>Negations</td>
<td>![aeiou]6</td>
<td>rhythm</td>
<td>decade</td>
</tr>
</tbody>
</table>
Deterministic Finite State Automaton (DFA)

Simple machine with $N$ states.
- Begin in start state.
- Read first input symbol.
- Move to new state, depending on current state and input symbol.
- Repeat until last input symbol read.
- Accept or reject string depending on last state.

Input: $b$ $b$ $a$ $a$ $b$ $b$ $a$ $b$ $b$

Theory of DFAs and REs

**RE.** Concise way to describe a set of strings.

**DFA.** Machine to recognize whether a given string is in a given set.

**Duality:** for any DFA, there exists a regular expression to describe the same set of strings; for any regular expression, there exists a DFA that recognizes the same set.

Practical consequence of duality proof: to match regular expression patterns, (i) build DFA and (ii) simulate DFA on input string.
Implementing a Pattern Matcher

Problem: given a regular expression, create program that tests whether given input is in set of strings described.

Step 1: build the DFA.
  - A compiler!
  - See COS 226 or COS 320.

Step 2: simulate it with given input. Easy.

```
State state = start;
while (!CharStdIn.isEmpty()) {
    char c = CharStdIn.readChar();
    state = state.next(c);
}
System.out.println(state.accept());
```

Application: Harvester

Harvest information from input stream.

- Harvest patterns from DNA.

```
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgccgcggcgcgcgcgtg
gcgctg
gcgccgcggcgcgggctg
gcgccgcggcgcgggctg
gcgccgcggcgcgggctg
gcgccgcggcgcgggctg
gcgccgcggcgcgggctg
gcgccgcggcgcgggctg
gcgccgcggcgcgggctg
```

- Harvest email addresses from web for spam campaign.

```
% java Harvester "[a-z]+@[a-z]+\.(edu|com|net|tv)"
http://www.princeton.edu/~cos126
doug@cs.princeton.edu
dsgabi@cs.princeton.edu
mona@cs.princeton.edu
```

Application: Parsing a Data File

Ex: parsing an NCBI genome data file.

```
LOCUS AC146484 128142 bp DNA linear STS 13-NOV-2003
DEFINITION Oryctolagus cuniculus clone CMA-3 DNA,
ACCESSION AC146484
VERSION AC146484.2 G13B3D04214
KEYWORDS STS, NP, PARSE, STS DRAFT.
SOURCE Oryctolagus cuniculus (rabbit)
ORIGIN 1 gpatctcat tggttttttta cggaggt gggtggtt agagatgc
        4l gpatctcat tggttttttta cggaggt gggtggtt agagatgc
// a comment
111 gpatctcat tggttttttta cggaggt gggtggtt agagatgc
/128121 gpatctcat tggttttttta cggaggt gggtggtt agagatgc
/131460 gpatctcat tggttttttta cggaggt gggtggtt agagatgc
```

```
String re = "[ ]*[0-9]+(actg )*.";
Pattern pattern = Pattern.compile(re);
In in = new In(filename);
String line;
while ((line = in.readLine()) != null) {
    Matcher matcher = pattern.matcher(line);
    if (matcher.find()) {
        String s = matcher.group().replaceAll("\[", "\""); // do something with s
        replace this RE with this string

    }
}
```
Limitations of DFA

No DFA can recognize the language of all bit strings with an equal number of 0’s and 1’s.

- Suppose an N-state DFA can recognize this language.
- Consider following input: $0^{N+1}1^{N+1}$
- DFA must accept this string.
- Some state $x$ is revisited during first $N+1$ 0’s since only $N$ states.
- Machine would accept same string without intervening 0’s.
- This string doesn’t have an equal number of 0’s and 1’s.

Fundamental Questions

Which languages CANNOT be described by any RE?
- Bit strings with equal number of 0’s and 1’s.
- Decimal strings that represent prime numbers.
- Genomic strings that are Watson-Crick complemented palindromes.
- Many more...

How can we extend REs to describe richer sets of strings?
- Context free grammar (e.g., Java).

Q. How can we make simple machines more powerful?

Q. Are there any limits on what kinds of problems machines can solve?

Summary

Programmer.
- Regular expressions are a powerful pattern matching tool.
- Implement regular expressions with finite state machines.

Theoretician.
- Regular expression is a compact description of a set of strings.
- DFA is an abstract machine that solves pattern match problem for regular expressions.
- DFAs and regular expressions have limitations.

Variations
- Yes (accept) and No (reject) states sometimes drawn differently
- Terminology: Deterministic Finite State Automaton (DFA), Finite State Machine (FSM), Finite State Automaton (FSA) are the same
- DFA’s can have output, specified on the arcs or in the states
  - These may not have explicit Yes and No states

Turing Machines

Challenge: Design simplest machine that is "as powerful" as conventional computers.
Turing Machine: Components

Alan Turing sought the most primitive model of a computing device.

**Tape.**
- Stores input, output, and intermediate results.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

**Tape head.**
- Points to one cell of tape.
- Reads a symbol from active cell.
- Writes a symbol to active cell.
- Moves left or right one cell at a time.

Turing Machine: Fetch, Execute

**States.**
- Finite number of possible machine configurations.
- Determines what machine does and which way tape head moves.

**State transition diagram.**
- Ex. if in state 2 and input symbol is 1 then: overwrite the 1 with x, move to state 0, move tape head to left.

Turing Machine: Initialization and Termination

**Initialization.**
- Set input on some portion of tape.
- Set tape head.
- Set initial state.

**Termination.**
- Stop if enter yes, no, or halt state.
- Infinite loop possible.
Example: Equal Number of 0's and 1's

Turing Machine Summary

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

**Surprising Fact 1.** Such machines are very simple: TM is enough!
**Surprising Fact 2.** Some problems cannot be solved by ANY computer.

**Consequences.**
- Precursor to general purpose programmable machines.
- Exposes fundamental limitations of all computers.
- Enables us to study the physics and universality of computation.
- No need to seek more powerful machines!

**Variations**
- Instead of just recognizing strings, TM's can produce output: the contents of the tape
- Instead of Y and N states, TM's can have a plain Halt state