7. Theory of Computation

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

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.

Introduction to Theoretical CS

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.

Regular Expressions and DFAs

```
a* | (a*ba*ba*)*
```
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 .*</td>
<td>raspberry</td>
<td>subspace</td>
</tr>
<tr>
<td>contains the trigraph spb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a*</td>
<td>(a<em>ba</em>ba*)*</td>
<td>bbb</td>
</tr>
<tr>
<td>multiple of three s's</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.*0...   fifth to last digit is 0</td>
<td>100011</td>
<td>11111111</td>
</tr>
<tr>
<td>98701234</td>
<td>403982772</td>
<td></td>
</tr>
<tr>
<td>gcg gcgagg* ctg</td>
<td>gcgctg</td>
<td>gcgaggctg</td>
</tr>
<tr>
<td>fragile X syndrome indicator</td>
<td></td>
<td></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.s.u.</td>
<td>cumulus</td>
<td>succubus</td>
</tr>
<tr>
<td></td>
<td>jugulorum</td>
<td></td>
<td>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>ababa</td>
<td>ababa</td>
</tr>
<tr>
<td>Parentheses</td>
<td>a(a</td>
<td>b)aab</td>
<td>aababa</td>
</tr>
<tr>
<td></td>
<td>(ab)*a</td>
<td>abbababa</td>
<td>abbabaa</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>ade</td>
</tr>
<tr>
<td></td>
<td>abc</td>
<td>bcd</td>
<td>ade</td>
</tr>
<tr>
<td>Character classes</td>
<td>[A-Za-z][a-z]*</td>
<td>capitalized</td>
<td>camelCase</td>
</tr>
<tr>
<td></td>
<td>Word</td>
<td>4illegal</td>
<td></td>
</tr>
<tr>
<td>Exactly k</td>
<td>[0-9][5]-<a href="4">0-9</a></td>
<td>08540-1321</td>
<td>11111111</td>
</tr>
<tr>
<td></td>
<td>19072-5541</td>
<td>165-54-111</td>
<td></td>
</tr>
<tr>
<td>Negations</td>
<td><a href="6">^aeiou</a></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.

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 RE, 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.

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

Harvest information from input stream.
- Use Pattern data type to compile regular expression to NFA.
- Use Matcher data type to simulate NFA.
- (NFA is fancier but equivalent variety of DFA)

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;
public static void main(String[] args) {
    String re = args[0];
    In in = new In(args[1]);
    String input = in.readAll();
    Pattern pattern = Pattern.compile(re);
    Matcher matcher = pattern.matcher(input);
    while (matcher.find()) {
        System.out.println(matcher.group());
    }
}
```

Application: Harvester

Harvest information from input stream.
- Harvest patterns from DNA.
- Harvest email addresses from web for spam campaign.

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: \[00000001111111\]

\[N+1\] 0's \[N+1\] 1's

DFA must accept this string.
- Some state \(x\) is revisited during first \(N+1\) 0's since only \(N\) states.

\[00000001111111\]
\[x\] \[x\]

Machine would accept same string without intervening 0’s.
\[00011111111\]

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 0s and 1s.
- 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

Application: Parsing a Data File

Ex: parsing an NCBI genome data file.

String re = "[^*0-9]+(tactg \*).*";
PATTERN pattern = Pattern.compile(re); In in = new In(filename);
while (!in.isEmpty()) {
  String line = in.readLine();
  Matcher matcher = pattern.matcher(line);
  if (matcher.find()) {
    String s = matcher.group().replaceAll(" \*", "");
    // do something with s
  } else {
    // extract the RE part in parentheses
    // replace this RE with this string
  }
}

Turing Machines

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

Alan Turing (1912-1954)
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.

Before

After

### 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

Alan Turing

Alan Turing (1912-1954).
- Father of computer science.
- Computer Science's "Nobel Prize" is called the Turing Award.

Alan's report card at 14.