7. 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
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, Nethanny, ads, 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.

Pattern Matching in TiVo

TiVo. WishList has very limited pattern matching.

Using * In WishList Searches. To search for similar words in Keyword and Title.
WishList searches, use the asterisk (*) as a special symbol that replaces the endings of words. For example, the keyword drrf** would find shows containing “airport,” “airplane,” “airplanes,” as well as the movie “Airplane!” To enter an asterisk, press the SLOW (>) button as you are spelling out your keyword or title.

The asterisk can be helpful when you're looking for a range of similar words, as in the example above, or if you're just not sure how something is spelled. For quick, it is “irresistible” or “irresistible?” Use the keyword driesst** and don't worry about it!

Two things to note about using the asterisk:
- It can only be used at a word's end; it cannot be used to omit letters at the beginning or in the middle of a word. (For example, dft*ne or *plane would not work.)

Q. How to describe a protein motif?

Ex. [signature of the C4H2-type zinc finger domain]
- C
- Between 2 and 4 amino acids.
- C
- 3 more amino acids.
- One of the following amino acids: LIVMFYWCX.
- 8 more amino acids.
- H
- Between 3 and 5 more amino acids.
- H

PROSITE. Huge database of protein families and domains.

Describing a Pattern

Google. Supports * for full word wildcard and | for union.
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><em>every other string</em></td>
</tr>
<tr>
<td>Wildcard</td>
<td>.u.uu.</td>
<td>cumulus jugulum</td>
<td>succubus tumultuous</td>
</tr>
<tr>
<td>Union</td>
<td>aa</td>
<td>baab</td>
<td>aa baab</td>
</tr>
<tr>
<td>Closure</td>
<td>ab+a</td>
<td>aa abba</td>
<td>ab ababa</td>
</tr>
<tr>
<td>Parentheses</td>
<td>a(a</td>
<td>b)aab</td>
<td>aab abab</td>
</tr>
<tr>
<td></td>
<td>(ab)*a</td>
<td>a ababababa</td>
<td>aa abba</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</td>
<td>d)e</td>
<td>abcd abcde</td>
</tr>
<tr>
<td>Character classes</td>
<td>[A-Za-z] [a-z]*</td>
<td>lowercase Capitalized camelCase 4illegal</td>
<td></td>
</tr>
<tr>
<td>Exactly k</td>
<td><a href="5">0-9</a>-<a href="4">0-9</a></td>
<td>08540-1321  19072-5541   11111111  166-54-1111</td>
<td></td>
</tr>
<tr>
<td>Negations</td>
<td>[^aeiou] [6]</td>
<td>rhythm</td>
<td>decade</td>
</tr>
</tbody>
</table>
More String Library Functions

String searching methods.

public class String (Java's String library)

boolean matches(String re)

does this string match the given regular expression

String replaceAll(String re, String str)

replace all occurrences of regular expression with the replacement string

int indexOf(String r, int from)

return the index of the first occurrence of the string r after the index from

String[] split(String re)

split the string around matches of the given regular expression

String s = StdIn.readAll();
s = s.replaceAll("\s+", " ");

replace all sequences of whitespace characters with a single space

More String Library Functions

Solving the Pattern Match Problem

Regular expressions are a concise way to describe patterns.

- How would you implement the method matches()?
- Hardware: build a deterministic finite state automaton (DFA).
- Software: simulate a DFA.

DFA: simple machine that solves a pattern match problem.

- Different machine for each pattern.
- Accepts or rejects string specified on input tape.
- Focus on true or false questions for simplicity.

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 input string if last state is labeled Y.

Input: b b a a b a b a b

DFA

Y

N

N

a

b

a

b

START
DFA and RE Duality

**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 RE that describes the same set of strings; for any RE, there exists a DFA that recognizes the same set.

DFA

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>
```

**Machine to recognize a string.**

```
a* | (ab*a*b*a*)*
```

![DFA Diagram](image)

**Multiple of 3 b's**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

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

For any DFA, there exists a RE that describes the same set of strings; for any RE, there exists a DFA that recognizes the same set.

Practical consequence of duality proof: to match RE, (i) build DFA and (ii) simulate DFA on input string.

--

**Application: Harvester**

Harvest information from input stream.

- Harvest patterns from DNA.

```
% java Harvester "gog(ogg)agg" chromosomeX.txt
```

- Harvest email addresses from web for spam campaign.

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

**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.

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

--

**Application: Harvester**

**Harvest information from input stream.**

- Use Pattern data type to compile regular expression to NFA.
- Use Matcher data type to simulate NFA.

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class Harvester {
    public static void main(String[] args) {
        String re = args[0];
        In in = new In(args[1]);
        String input = in.readString();
        Pattern pattern = Pattern.compile(re);
        Matcher matcher = pattern.matcher(input);
        find next substring matching pattern
        while (matcher.find()) {
            Stdout.println(matcher.group());
        }
    }
}
```

The substring that matches the pattern.
Summary

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

Theoretician.
- RE is a compact description of a set of strings.
- DFA is an abstract machine that solves RE pattern match problem.

You. Practical application of core CS principles.

Variations.
- Terminology: DFA, FSA, FSM.
- DFAs with output: Moore machines, Mealy machines.

Application: Parsing a Data File

Ex: parsing an NCBI genome data file.

```java
String re = "[^0-9]+([actg ])*";
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(1).replaceAll(" ", ");
        // do something with s
    }
}
```

Regular Expressions

Q. Are there patterns that cannot be described by any RE/DFA?
A. Yes.
- Bit strings with equal number of 0s and 1s.
- Decimal strings that represent prime numbers.
- DNA strings that are Watson-Crick complemented palindromes.

Q. Can we extend RE/DFA to describe richer patterns?
A. Yes.
- Context free grammar (e.g., Java).
- Turing machines.