7. Theory of Computation

Q. What can a computer do?
Q. 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.

Pioneering work in the 1930s.
- Princeton == center of universe.
- Automata, languages, computability, universality, complexity, logic.

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
**PROSITE.** Huge database of protein families and domains.

Q. How to describe a protein motif?

Ex. [signature of the $C_2H_2$-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

\[ \text{CAASC} \text{CGFYACGKWAGYHAGWH} \]

---

### Regular Expressions: Basic Operations

**Regular expression.** Notation to specify a set of strings.

<table>
<thead>
<tr>
<th>operation</th>
<th>regular expression</th>
<th>matches</th>
<th>does not match</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.u.u.</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>aaaaab abaaab</td>
</tr>
<tr>
<td></td>
<td>(ab)*a</td>
<td>a abababa</td>
<td>aa abbba</td>
</tr>
</tbody>
</table>

---

### Regular Expressions: Examples

**Regular expression.** Notation is surprisingly expressive.

<table>
<thead>
<tr>
<th>regular expression</th>
<th>matches</th>
<th>does not match</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* | (a<em>ba</em>ba<em>ba</em>)* *</td>
<td>multiple of three b’s</td>
<td>bbb addaabababaa</td>
</tr>
<tr>
<td>fifth to last digit is 0</td>
<td>1000234 98701234</td>
<td>11111111 403982772</td>
</tr>
<tr>
<td>gcgcggaggctg</td>
<td>gcgcggaggctg</td>
<td>gcgcggaggctg</td>
</tr>
<tr>
<td>fragile X syndrome indicator</td>
<td>gcgcggaggctg</td>
<td>gcgcggaggctg</td>
</tr>
</tbody>
</table>

---

**Test if a string matches some pattern.**
- Process natural language.
- Scan for virus signatures.
- Check style of Java variable names.
- Access information in digital libraries.
- Search-and-replace in a word processors.
- Filter text (spam, NetNanny, 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.
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>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>one or more</td>
<td>a(bc)+de</td>
<td>abode</td>
<td>ade</td>
</tr>
<tr>
<td>character class</td>
<td>[A-Za-z][a-z]*</td>
<td>lowerCase</td>
<td>capitalized</td>
</tr>
<tr>
<td>exactly k</td>
<td>[0-9]{5}-[0-9]{4}</td>
<td>08540-1321</td>
<td>111111111</td>
</tr>
<tr>
<td>negation</td>
<td>[^aeiou]{6}</td>
<td>rhythm</td>
<td>decade</td>
</tr>
</tbody>
</table>

Regular Expressions in Java

Validity checking. Is input in the set described by the re?

```
public class Validate {
    public static void main(String[] args) {
        String re = args[0];
        String input = args[1];
        StdOut.println(input.matches(re));
    }
}
```

- legal Java identifier
- valid email address (simplified)
- need quotes to "escape" the shell

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 t)  
        replace all occurrences of regular expression with the replacement string
    int indexOf(String t, int from)  
        return the index of the first occurrence of the string t 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

```
public class String { (Java's String library)
    boolean matches(String re)  
        does this string match the given regular expression
    String replaceAll(String re, String t)  
        replace all occurrences of regular expression with the replacement string
    int indexOf(String t, int from)  
        return the index of the first occurrence of the string t after the index from
    String[] split(String re)  
        split the string around matches of the given regular expression
}
```

```
String s = StdIn.readAll();
String[] words = s.split("\s+");
```

create array of words in document

regular expression that matches any whitespace character
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 (entering arrow).
- Read first input symbol.
- Move to next state, depending on current state and input symbol.
- Repeat until last input symbol read.
- Accept if last state is an accept state (leaving arrow).

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.

Practical consequence of duality proof: to match RE, (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);
}
StdOut.println(state.accept());
```

Application: Harvester

Harvest information from input stream.

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

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

Application: Parsing a Data File

**Ex:** parsing an NCBI genome data file.

```
LOCUS AC146846 128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhynchus anatinus clone CLM1-393H.
ACCESSION AC146846
VERSION AC146846.2
KEYWORDS HTG; HTGS_PHASE2; HTGS_DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN
1 tgtatttcat ttgaccgtgc tgttttttcc cggtttttca gtacggtgtt agggagccac
61 tgtattctgt ttgtttatatg ctgccgaata gctgctcgat gaatctctgc atagacagct
121 gccgcaggga gaaatgacca gtttgtgatg acaaaatgta ggaaagctgt ttcttcataa
... 128101 ggaaatgcga cccccacgct aatgtacagc ttctttagat tg
```

```java
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);

        while (matcher.find()) {
            StdOut.println(matcher.group());
        }
    }
}
```
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()) {
        extract the part of match in parens
        String s = matcher.group(1).replaceAll(" ", "");
        // do something with s
    }
}
```

LODUS AC146846.128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhynchus anatinus clone CLM-393893,
ACCESSION AC148846
VERSION AC148846.2 GI:38304214
KEYWORDS HTG; HTGS_PHASE2; HTGS_DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN
tgtatttcat ttgaccgtgc tgttttttcc cggtttttca gtacggtgtt agggagccac
gtgattctgt ttgttttatg ctgccgaata gctgctcgat gaatctctgc atagacagct // a comment
gccgcaggga gaaatgacca gtttgtgatg acaaaatgta ggaaagctgt ttcttcataa
...
128101 ggaaatgcga cccccacgct aatgtacagc ttctttagat tg
```

Summary

Programmer.
- Regular expressions 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.

Fundamental Questions

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.