14. Introduction to Theoretical CS

- Overview
- Regular expressions
- DFAs
- Applications
- Limitations

Introduction to theoretical computer science

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.

Surprising outcome. Sweeping and relevant statements about all computers.

Why study theory?

In theory...
- Deeper understanding of computation.
- 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."

— Yogi Berra
Abstract machines

Abstract machine
- Mathematical model of computation.
- Each machine defined by specific rules for transforming input to output.
- This lecture: Deterministic finite automata (DFAs).

Formal language
- A set of strings.
- Each defined by specific rules that characterize it.
- This lecture: Regular expressions (REs).

Questions for this lecture
- Is a given string in the language defined by a given RE, or not?
- Can a DFA help answer this question?

Pattern matching

Pattern matching problem. Is a given string an element of a given set of strings?

Example 1 (from computational biochemistry)

An amino acid is represented by one of the characters CAVILMRKHDENQSTYWFP.

A protein is a string of amino acids.

A C5H2-type zinc finger domain signature is
- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
- L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
- H followed by 3, 4, or 5 amino acids, followed by H.

Q. Is this protein in the C5H2-type zinc finger domain?

A. Yes.
**Pattern matching**

**Example 2 (from commercial computing)**

An e-mail address is:
- A sequence of letters, followed by
- the character "@", followed by
- followed by a nonempty sequence of lowercase letters, followed by the character ".".
- [any number of occurrences of the previous pattern]
- "edu" or "com" (others omitted for brevity).

**Q.** Which of the following are e-mail addresses?

<table>
<thead>
<tr>
<th>Address</th>
<th>✔️</th>
<th>✗</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:rs@cs.princeton.edu">rs@cs.princeton.edu</a></td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>not an e-mail address</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td><a href="mailto:wayne@cs.princeton.edu">wayne@cs.princeton.edu</a></td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>eve@airport</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td><a href="mailto:rs123@princeton.edu">rs123@princeton.edu</a></td>
<td>✗</td>
<td></td>
</tr>
</tbody>
</table>

**Oops, need to fix description**

**Challenge.** Develop a precise description of the set of strings that are legal e-mail addresses.

**Pattern matching**

**Example 3 (from genomics)**

**A nucleic acid** is represented by one of the letters a, c, t, or g.

**A genome** is a string of nucleic acids.

**A Fragile X Syndrome pattern** is a genome having an occurrence of gcg, followed by any number of cgg or agg triplets, followed by ctg.

**Note.** The number of triplets correlates with Fragile X Syndrome, a common cause of mental retardation.

**Q.** Does this genome contain a such a pattern?

\[
\text{gcgcggctgctgagagtttaagctgctggcgcgcggaggctg}
\]

**A.** Yes.

**More examples of regular expressions**

The notation is surprisingly expressive.

<table>
<thead>
<tr>
<th>regular expression</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>.&quot;spb,&quot;</td>
<td>raspberry</td>
<td>subspace</td>
</tr>
<tr>
<td>contains the trigraph spb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a&quot;</td>
<td>(a&quot;ba&quot;ba&quot;ba&quot;)&quot;</td>
<td>multiple of three b's</td>
</tr>
<tr>
<td>fifth to last digit is 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.&quot;g&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcg(cgg</td>
<td>agg)ctg.</td>
<td></td>
</tr>
<tr>
<td>fragile X syndrome pattern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcgcgg...gcgcgcggctg...gcgcgcggctg...</td>
<td>gcgcgg gcgcgcggctg gcgcgcggctg</td>
<td></td>
</tr>
<tr>
<td>gcgcgg...gcgcgcggctg...gcgcgcggctg...</td>
<td>gcgcgg gcgcgcggctg gcgcgcggctg</td>
<td></td>
</tr>
</tbody>
</table>
Generalized regular expressions

Additional operations further extend the utility of REs.

<table>
<thead>
<tr>
<th>operation</th>
<th>example RE</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>one or more</td>
<td>a(bc)+de</td>
<td>abcd</td>
<td>ade</td>
</tr>
<tr>
<td>character class</td>
<td>[A-Za-z][a-z] *</td>
<td>lowercase Capitalized</td>
<td>camelCase Illegal</td>
</tr>
<tr>
<td>exactly j</td>
<td>[0-9]{5} [0-9]{4}</td>
<td>08540-1321 19072-5541</td>
<td>11111111 166-54-1111</td>
</tr>
<tr>
<td>between j and k</td>
<td>a.{2,4}b</td>
<td>abcb</td>
<td>abcbc</td>
</tr>
<tr>
<td>negation</td>
<td>[^aeiou]{6}</td>
<td>rhythm</td>
<td>decade</td>
</tr>
<tr>
<td>whitespace</td>
<td>\s</td>
<td>any whitespace char (space, tab, newline...)</td>
<td>every other character</td>
</tr>
</tbody>
</table>

Note. These operations are all **shorthand**. They are very useful but not essential.

Example of describing a pattern with a generalized RE

A C3H2-type zinc finger domain signature is

- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
  - L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
  - H followed by 3, 4, or 5 amino acids, followed by

Q. Give a generalized RE for all such signatures.

A. `C\{2,4\}C...[LIVMFYWCTX].{8}H.{3,5}H`

Example of a real-world RE application: PROSITE

![Prosite logo]

Type an RE here

Example of describing a pattern with a generalized RE

An e-mail address is

- A sequence of letters, followed by
- the character "@", followed by
  - the character ".", followed by a nonempty sequence of lowercase letters, followed by
  - [any number of occurrences of the previous pattern]
  - "edu" or "com" (others omitted for brevity).

Q. Give a generalized RE for e-mail addresses.

A. `([a-z]+@[a-z]+\.\+)+\{edu | com\}`

Exercise. Extend to handle rs1230@princeton.edu, more suffixes such as .org, and any other extensions you can think of.

Next. Determining whether a given string matches a given RE.
Pop quiz 1 on REs

Q. Which of the following strings match the RE \( a^*b(b|ba)^* \)?

1. abb
2. aaba
3. abba
4. bbbbaab
5. cbb
6. bbababbab

is in the set it describes

Pop quiz 2 on REs

Q. Give an RE for genes
- Characters are a, c, t or g.
- Starts with atg (a start codon).
- Length is a multiple of 3.
- Ends with tag, taa, or ttg (a stop codon).

COMPUTER SCIENCE
SEDEWICK/WAYNE
PART I: PROGRAMMING IN JAVA

14. Introduction to Theoretical CS
- Overview
- Regular expressions
- DFAs
- Applications
- Limitations

Image sources
http://en.wikipedia.org/wiki/Homology_modeling/media/File:DHR578_homology_model.png
Deterministic finite automata (DFA)

A DFA is an abstract machine that solves a pattern matching problem.
• A string is specified on an input tape (no limit on its length).
• The DFA reads each character on input tape once, moving left to right.
• The DFA lights “YES” if it recognizes the string, “NO” otherwise.
Each DFA defines a language (the set of strings that it recognizes).

Simulating the operation of a DFA

public class DFA {
  private int start;
  private boolean[] action;
  private int start; alphabet
  for (int i = 0; i < input.length(); ++i) {
    int state = next[state].action.getCharAt(i);
    return (state);[
  }
  public static void main(String[] args) {
    DFA dfa = new DFA(args[0]);
    if (dfa.recognize(input)) StdOut.println("Yes");
    else StdOut.println("No");
  }
}
Kleene’s theorem

Two ways to define a set of strings (language)
• Regular expressions (REs).
• Deterministic finite automata (DFAs).

Remarkable fact. DFAs and REs are equivalent.

Equivalence theorem (Kleene)
Given any RE, there exists a DFA that accepts the same set of strings.
Given any DFA, there exists an RE that matches the same set of strings.

Consequence: A way to solve the RE pattern matching problem
• Build the DFA corresponding to the given RE.
• Simulate the operation of the DFA.
14. Introduction to Theoretical CS

- Overview
- Regular expressions
- DFAs
- Applications
- Limitations

**REs in Java**

Java’s String class implements GREP.

```java
public class String {
    ...  
    boolean matches(String re)  
    ...  

    String re = "C.(2,4)C...[LIVMFYW].(8)H.(3,5)H";
    String zincFinger = "CAASCGGPYACCGWAGYHAGWH";
    boolean test = zincFinger.matches(re);
}
```

**Java RE client example: Validation**

```java
public class Validate {
    public static void main(String[] args) {
        String re = args[0];
        while (!StdIn.isEmpty()) {
            String input = StdIn.readString();
            StdOut.println(input.matches(re));
        }
    }
}
```

**Applications**

- Scientific research.
- Compilers and interpreters.
- Internet commerce.
- ...
Beyond matching

Java’s String class contains other useful RE-related methods.
• RE search and replace
• RE delimited parsing

public class String

... String replaceAll(String re, String to) replace all occurrences of substrings matching RE with to
String[] split(String re) split the string around matches of the given RE ...

Tricky notation (typical in string processing): \ signals "special character" so "\\" means "\" and \"\\\" means "\\".

Examples using the RE "\\s" (matches one or more whitespace characters).

Replace each sequence of at least one whitespace character with a single space.
String s = StdIn.readString();
s = s.replaceAll("\\s"," ");

Create an array of the words in StdIn (basis for StdIn.readString() method).
String s = StdIn.readString();
String[] words = s.split("\\s");

Way beyond matching

Java’s Pattern and Matcher classes give fine control over the CREP implementation.

public class Pattern

... static Pattern compile(String re) parse the re to construct a Pattern
Matcher matcher(String input) create a Matcher that can find substrings matching the pattern in the given input string ...

public class Matcher

... boolean find() set internal variable match to the next substring that matches the RE in the input. If none, return false, else return true
String group() return match
String group(int k) return the kth group (identified by parens within RE) in match ...

[A sophisticated interface designed for pros, but very useful for everyone.]

Java pattern matcher client example: Harvester

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

Harvest information from input stream
• Take RE from command line.
• Take input from file or web page.
• Print all substrings matching RE.

Applications of REs

Pattern matching and beyond.
• Compile a Java program.
• Scan for virus signatures.
• Crawl and index the Web.
• Process natural language.
• Access information in digital libraries.
• Search-and-replace in a word processors.
• Process NCBI and other scientific data files.
• 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.
• Automatically create Java documentation from Javadoc comments.

GREP and related facilities are built in to Java, Unix shell, PERL, Python ...

virtually every computing environment
14. Introduction to Theoretical CS

- Overview
- Regular expressions
- DFAs
- Applications
- Limitations

Summary

Programmers
- Regular expressions are a powerful pattern matching tool.
- Equivalent DFA/NFA paradigm facilitates implementation.
- Combination greatly facilitates real-world string data

Theoreticians
- REs provide compact descriptions of sets of strings.
- DFAs are abstract machines with equivalent descriptive power.
- Are there languages and machines with more descriptive power?

You
- CS core principles provide useful tools that you can exploit now.
- REs and DFAs provide an introduction to theoretical CS.

Basic questions

Q. Are there sets of strings that cannot be described by any RE?
A. Yes.
- Bitstrings with equal number of 0s and 1s (stay tuned).
- Strings that represent legal REs.
- Decimal strings that represent prime numbers.
- DNA strings that are Watson-Crick complemented palindromes.
- ...

Q. Are there sets of strings that cannot be described by any DFA?
A. Yes.
- Bit strings with equal number of 0s and 1s (see next slide).
- Strings that represent legal REs.
- Decimal strings that represent prime numbers.
- DNA strings that are Watson-Crick complemented palindromes.
- ...

The same question, by Kleene’s theorem
A limit on the power of REs and DFAs

**Proposition.** There exists a set of strings that cannot be described by any RE or DFA.

**Proof sketch.** No DFA can recognize the set of bitstrings with equal number of 0s and 1s.
- **Assume that you have such a DFA, with** \( N \) **states.**
- It recognizes the string with \( N + 1 \) 0s followed by \( N + 1 \) 1s.
- Some state is **revisited** when scanning the 0s in that string.
- Delete the substring of 0s between visits of that state.
- DFA recognizes that string, too.
- It does not have equal number of 0s and 1s.
- **Proof by contradiction:** the assumption that such a DFA exists must be false.

---

Another basic question

**Q.** Are there abstract machines that are more powerful than DFAs?  
**A.** Yes, a 1-stack DFA can recognize
- Bitstrings with equal number of 0s and 1s.
- Strings that represent legal REs.

**Proof.** [details omitted]

---

Yet another basic question

**Q.** Are there abstract machines that are more powerful than a 1-stack DFA?  
**A.** Yes, a 2-stack DFA can recognize
- Decimal strings that represent prime numbers.
- Strings that represent legal Java programs.
- ...

[stay tuned for next lecture]

---

One last basic question

**Q.** Are there machines that are more powerful than a 2-stack DFA?  
**A.** No! Not even a roomful of supercomputers (!!!)

[stay tuned for next lecture]
14. Introduction to Theoretical CS