17. Introduction to Theoretical CS

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. In practice there is." — Yogi Berra

17. Introduction to Theoretical CS

- Regular expressions
- DFAs
- Applications
- Limitations
Pattern matching

**Pattern matching problem.** Is a given string a member of a given set of strings?

**Example 1 (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?

```
ncgctgtcgcagagagtggtttaagctctcggtggtggtgcgcggaggcggctggcgcggaggctg
ggcgcggaggctgtg
```

A. Yes.

---

**Example 2 (from computational biochemistry)**

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

A protein is a string of amino acids.

A C$_{2}$H$_{2}$-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 C$_{2}$H$_{2}$-type zinc finger domain?

```
CAASCGGPYACGGWAGYHAGWH
```

A. Yes.

---

**Example 3 (from commercial computing)**

An e-mail address is

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

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

- rs@cs.princeton.edu ✓
- not an e-mail address x
- wayne@cs.princeton.edu ✓
- eve@airport x
- rs123@princeton.edu x

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

---

**Regular expressions**

A regular expression (RE) is a notation for specifying sets of strings.

A RE is

- A sequence of letters or ".
- The union of two REs
- The closure of an RE (any number of occurrences)
- May be delimited by ()

<table>
<thead>
<tr>
<th>operation</th>
<th>example RE</th>
<th>matches (IN the set)</th>
<th>does not match (NOT in the set)</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.</td>
<td>cumulus jugulum</td>
<td>succubus tumultuous</td>
</tr>
<tr>
<td>union</td>
<td>aa</td>
<td>aa</td>
<td>every other string</td>
</tr>
<tr>
<td>closure</td>
<td>ab*a</td>
<td>aa</td>
<td>abba</td>
</tr>
<tr>
<td>parentheses</td>
<td>a(a</td>
<td>b)a</td>
<td>aaba</td>
</tr>
<tr>
<td></td>
<td>(ab)*a</td>
<td>a</td>
<td>abababa</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a</td>
<td>abba</td>
</tr>
</tbody>
</table>
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>.<em>spb.</em></td>
<td>raspberry crispsread</td>
<td>subspace subspecies</td>
</tr>
<tr>
<td>a^n</td>
<td>(a*ba*ba*ba*)^n</td>
<td>multiple of three b's</td>
</tr>
<tr>
<td>.{0,...}</td>
<td>fifth to last digit is 0</td>
<td>1000234 98761234</td>
</tr>
<tr>
<td>gcgcgcggctg gcgcggctg gcgcggctg gcgcggctg</td>
<td>gcgcgcggctg gcgcggctg gcgcggctg gcgcggctg</td>
<td>gcgcgcggctg gcgcggctg gcgcggctg gcgcggctg</td>
</tr>
<tr>
<td>fragile X syndrome pattern</td>
<td></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}d{e}</td>
<td>abcde</td>
<td>ade bcde</td>
</tr>
<tr>
<td>character class</td>
<td>[A-Z][a-z]^[a-z]^n</td>
<td>Towercase Capitalized</td>
<td>camelCase 4ITLegal</td>
</tr>
<tr>
<td>exactly k</td>
<td></td>
<td>08540-1321 19072-5541</td>
<td>11111111 166-54-1111</td>
</tr>
<tr>
<td>negation</td>
<td>[^aeiou]{6}</td>
<td>rhythm</td>
<td>decade</td>
</tr>
<tr>
<td>white space</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 C_{2H_2}-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. Give a generalized RE for all such signatures.

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

"Wildcard" matches any of the letters
CAVLIMCRHDESTYFWMP

Example of a real-world RE application: PROSITE

PROSITE Database of protein domains, families and functional sites

PROSITE consists of pattern-recognition entries describing protein domains, families and functional sites as well as associated patterns and profiles to identify them. PROSITE is a collection of sites based on profiles and patterns, which increases the domain miner's ability to find and use functional domains by providing additional information about functionally and structurally similar amino acids. (Brief.)

Enzyme changes information can be found here.


PROSITE entry

Access:
- by documentation entry
- by ProDom description
- by ProDom name
- by number of ProDom sub

Type an RE here
Another 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 sequence of letters, followed by
- "edu" or "com" (others omitted for brevity).

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

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

Exercise. Extend to handle rs1230princeton.edu, more suffixes such as .org, and any other extensions you can think of.

Next. Determining whether a given string matches a given RE.

Self-assessment 1 on REs

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

1. abb
2. aaba
3. abba
4. bbaaab
5. cbb
6. bbababbb

Self-assessment 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).
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 set of strings (all the strings that it recognizes).

A DFA is an abstract machine with a finite number states, each labeled Y or N, and
transitions between states, each labelled with a symbol. One state is the start state.
- Begin in the start state (denoted by an arrow from nowhere).
- Read an input symbol and move to the indicated state.
- Repeat until the last input symbol has been read.
- Turn on the "YES" or "NO" light according to the label on the current state.

Deterministic finite state automata (DFA)

Deterministic finite state automata details and example

Simulating the operation of a DFA
Self-assessment 1 on DFAs

Q. Which of the following strings does this DFA accept?

1. Bitstrings that end in 1
2. Bitstrings with an equal number of occurrences of 01 and 10
3. Bitstrings with more 1s than 0s
4. Bitstrings with an equal number of occurrences of 0 and 1
5. Bitstrings with at least one 1

Self-assessment 2 on DFAs

Q. Which of the following strings does this DFA accept?

1. Bitstrings with at least one 1
2. Bitstrings with an equal number of occurrences of 01 and 10
3. Bitstrings with more 1s than 0s
4. Bitstrings with an equal number of occurrences of 0 and 1
5. Bitstrings that end in 1

Kleene’s theorem

Two ways to define a set of strings
• 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.
GREP: a solution to the RE pattern matching problem

An algorithm for the RE pattern matching problem?
• Build the DFA corresponding to the given RE.
• Simulate the operation of the DFA.

Practical difficulty: The DFA might have exponentially many states.

A more efficient algorithm: use Nondeterministic Finite Automata (NFA)
• Build the NFA corresponding to the given RE.
• Simulate the operation of the NFA.

"GREP" (Generalized Regular Expression Pattern matcher).
• Developed by Ken Thompson, who designed and implemented Unix.
• Indispensable programming tool for decades.
• Found in most development environments, including Java.

Java RE client example: Validation

```java
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.
• ...
Way beyond matching

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

<table>
<thead>
<tr>
<th>Java’s Pattern and Matcher classes give fine control over the GREP implementation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>public class Pattern</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>static Pattern compile(String re)</td>
</tr>
<tr>
<td>create the Pattern object</td>
</tr>
<tr>
<td>Matcher matcher(String input)</td>
</tr>
<tr>
<td>return the Matcher object</td>
</tr>
<tr>
<td>public class Matcher</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>boolean find()</td>
</tr>
<tr>
<td>set internal variable to next substring that matches the RE in input</td>
</tr>
<tr>
<td>Set null if non match or not complete match</td>
</tr>
<tr>
<td>String group()</td>
</tr>
<tr>
<td>return the first group that includes parentheses in the RE match</td>
</tr>
<tr>
<td>String group(int k)</td>
</tr>
<tr>
<td>return the kth group that includes parentheses in the RE match</td>
</tr>
</tbody>
</table>

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

Java pattern matcher client example: Harvester

<table>
<thead>
<tr>
<th>Java pattern matcher client example: Harvester</th>
</tr>
</thead>
<tbody>
<tr>
<td>import java.util.regex.Pattern;</td>
</tr>
<tr>
<td>import java.util.regex.Matcher;</td>
</tr>
<tr>
<td>public class Harvester {</td>
</tr>
<tr>
<td>public static void main(String[] args)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>String re = args[0];</td>
</tr>
<tr>
<td>In in = new In(args[1]);</td>
</tr>
<tr>
<td>String input = in.readAll();</td>
</tr>
<tr>
<td>Pattern pattern = Pattern.compile(re);</td>
</tr>
<tr>
<td>Matcher matcher = pattern.matcher(input);</td>
</tr>
<tr>
<td>while (matcher.find())</td>
</tr>
<tr>
<td>StdOut.println(matcher.group(3));</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Harvest information from input stream

- Take RE from command line.
- Take input from file or web page.
- Print all substrings matching RE.

Java pattern matcher real-world example: Parsing a data file

A typical situation

- An institution publishes data on the web to be shared by all.
- The data is published in human-readable form.
- You want to strip out everything but the raw data in order to process it.

Example: National Center for Biotechnology Information genome data

[Image of genome data]

Key challenge: Develop an appropriate RE.

- Slight glitch: Need to remove spaces afterwards.
- Ignore everything else.

Extract data after spaces followed by a line number.

LOCUS AC146846 128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhyncus anatinus clone CLM1-39H9,
VERSION AC146846.1 GI:38304214
KEYWORDS HTG, HTGS, PHASE2; HTGS, DRAFT;
SOURCE Ornithorhyncus anatinus (platyplus)

1st match

61 gcgcggcgggaggcggaggcggctg // a comment
don’t want “a”

128101 gcgcggcgggaggcggaggcggctg // a comment
don’t want this “a”

LOCUS AC146846 128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhyncus anatinus clone CLM1-39H9,
VERSION AC146846.1 GI:38304214
KEYWORDS HTG, HTGS, PHASE2; HTGS, DRAFT;
SOURCE Ornithorhyncus anatinus (platyplus)

ORIGIN
Java pattern matcher real-world example: Parsing a data file

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

class ParseNCBI {
    public static void main(String[] args) {
        String re = "[^\[0-9]*\{(actg \}\)}"];
        Pattern pattern = Pattern.compile(re);
        while (in.hasNextLine()) {
            String line = in.readLine();
            Matcher matcher = pattern.matcher(line);
            if (matcher.find()) {
                StdOut.print(matcher.group(1).replaceAll(" ", "").trim());
            }
        }
        StdOut.println();
    }
}
```

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 ...
Summary

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

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.
- 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.
- Strings that represent legal REs.
- Decimal strings that represent prime numbers.
- DNA strings that are Watson-Crick complemented palindromes.
- ...

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]