Introduction to Theoretical Computer Science

Fundamental questions:
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.

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.

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

Pattern matching problem. Is a given string in a specified set of strings?

Ex. [genomics]
- Fragile X syndrome is a common cause of mental retardation.
- Human genome contains triplet repeats of CGG or AGG, bracketed by GCG at the beginning and CTG at the end.
- Number of repeats is variable, and correlated with syndrome.

Specified set of strings: "all strings of G, C, T, A having some occurrence of CGG followed by any number of CGG or AGG triplets, followed by CTG"

Q: "Is this string in the set?"
A: Yes

Pattern Matching Application

PROSITE. Huge database of protein families and domains.

Q. How to describe a protein motif?

Ex. [signature of the C2H2-type zinc finger domain]
1. C
2. Between 2 and 4 amino acids.
3. C
4. 3 more amino acids.
5. One of the following amino acids: LIVMFYWCX.
6. 8 more amino acids.
7. H
8. Between 3 and 5 more amino acids.
9. H

A. Use a regular expression.

Pattern Matching Applications

Test if a string matches some pattern.
- Process natural language.
- Scan for virus signatures.
- 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.
## 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 tumulous</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>aabaab</td>
</tr>
<tr>
<td></td>
<td>(ab)*a</td>
<td>a abababa</td>
<td>aa abba</td>
</tr>
</tbody>
</table>

### Generalized Regular Expressions

Regular expressions are a standard programmer’s tool.

- Built into Java, Perl, Unix, Python, ...
- Additional operations typically added for convenience.
  - Ex 1: [a-e]+ is shorthand for (a|b|c|d|e)(a|b|c|d|e)*.
  - Ex 2: \s is shorthand for “any whitespace character” (space, tab, ...).

<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(b</td>
<td>c)+de</td>
<td>abode, abodec</td>
</tr>
<tr>
<td>character class</td>
<td>[A-Za-z][a-z]*</td>
<td>lowercase, Capitalized</td>
<td>camelCase, 4illegal</td>
</tr>
<tr>
<td>exactly k</td>
<td>[0-9][5]-[0-9][4]</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>
</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>.<em>spb.</em></td>
<td>raspberry</td>
<td>crispbread</td>
</tr>
<tr>
<td>(ab)*a</td>
<td>subspecies</td>
<td></td>
</tr>
<tr>
<td>a*</td>
<td>(a<em>ba</em>ba<em>ba</em>)*</td>
<td>bbb aaa</td>
</tr>
<tr>
<td>.*0....</td>
<td>1000234 98701234</td>
<td>11111111 403982772</td>
</tr>
<tr>
<td>gcg(cgg</td>
<td>agg)*ctg</td>
<td>gcgctg</td>
</tr>
<tr>
<td></td>
<td>gcgctg</td>
<td>gcgctg</td>
</tr>
</tbody>
</table>

### Regular Expression Challenge 1

**Q.** Consider the RE

\[ a^*bb(ab|ba)^* \]

Which of the following strings match (is in the set described)?

- a. abb
- b. abba
- c. aaba
- d. bbbab
- e. cbb
- f. bbababba

<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(b</td>
<td>c)+de</td>
<td>abode, abodec</td>
</tr>
<tr>
<td>character class</td>
<td>[A-Za-z][a-z]*</td>
<td>lowercase, Capitalized</td>
<td>camelCase, 4illegal</td>
</tr>
<tr>
<td>exactly k</td>
<td>[0-9][5]-[0-9][4]</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>
</tbody>
</table>
Regular Expression Challenge 2

Q. Give an RE that describes the following set of strings:

- characters are A, C, T or G
- starts with ATG
- length is a multiple of 3
- ends with TAG, TAA, or TTG

Describing a Pattern

PROSITE. Huge database of protein families and domains.

Q. How to describe a protein motif?

Ex. [signature of the C\(_2\)H\(_2\)-type zinc finger domain]

1. C
2. Between 2 and 4 amino acids.
3. C
4. 3 more amino acids.
5. One of the following amino acids: LIVMFYWCX.
6. 8 more amino acids.
7. H
8. Between 3 and 5 more amino acids.
9. H

A. C.(2,4)C...[LIVMFYWC].(8)H.(3,5)H

REs in Java

```
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
    }
}
```

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));
    }
}
```

% java Validate "C.(2,4)C...[LIVMFYWC].(8)H.(3,5)H" CAASCGGYPACGGWAGYHAGWH
true

legal Java identifier

% java Validate "[$_A-Za-z$_A-Za-z0-9]*" ident123
true

valid email address (simplified)

% java Validate "[a-z]+@[a-z]+\.(edu|com)" doug@cs.princeton.edu
true

need quotes to "escape" the shell
REs in Java

```java
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+", " ");
  // RE that matches any sequence of whitespace characters (at least 1).
  // Extra \ distinguishes from the string \s+

  String[] words = s.split("\s+");
  // create an array of the words in StdIn

  DFAs

  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.

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
• build DFA
• simulate DFA on input string.

DFA Challenge 1
Q. Consider this DFA:
Which of the following sets of strings does it recognize?
a. Bitstrings with at least one 1
b. Bitstrings with an equal number of occurrences of 01 and 10
c. Bitstrings with more 1s than 0s
d. Bitstrings with an equal number of occurrences of 0 and 1
e. Bitstrings that end in 1
DFA Challenge 2

Q. Consider this DFA:

Which of the following sets of strings does it recognize?

a. Bitstrings with at least one 1
b. Bitstrings with an equal number of occurrences of 01 and 10
b. Bitstrings with more 1s than 0s
d. Bitstrings with an equal number of occurrences of 0 and 1
e. Bitstrings that end in 1

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());

It is actually better to use an NFA, an equivalent (but more efficient) representation of a DFA. We ignore that distinction in this lecture.

Direct Application: Harvester

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

% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgccggccggccggcggcggctg
gcgctg
gcgctg
gcgccggccggccggcggcggcggctg
% java Harvester "([a-z]+@[a-z]+\.|)(edu|com)" http://www.princeton.edu/~cos126
rs@cs.princeton.edu
dgabai@cs.princeton.edu
doug@cs.princeton.edu
wayne@cs.princeton.edu
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgccggccggccggcggcggcggctg
gcgctg
gcgctg
gcgccggccggcggcggcggcggctg
% java Harvester "([a-z]+@[a-z]+\.|)(edu|com)" http://www.princeton.edu/~cos126
rs@cs.princeton.edu
dgabai@cs.princeton.edu
doug@cs.princeton.edu
wayne@cs.princeton.edu
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgccggccggccggcggcggcggctg
gcgctg
gcgctg
gcgccggccggcggcggcggcggcggctg

Direct 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.readAll();
        Pattern pattern = Pattern.compile(re);
        Matcher matcher = pattern.matcher(input);
        while (matcher.find())
            StdOut.println(matcher.group());
    }
}
Real-World Application: Parsing a Data File

Java's Pattern and Matcher classes
- use REs for pattern matching (previous slide)
- extend REs to facilitate processing string-based data

Ex: parsing an NCBI genome data file.

Goal. Extract the data as a single actg string.

Real-World Application: Parsing a Data File

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

public class ParseNCBI
{
    public static void main(String[] args)
    {
        String re = "\[ \*\]" + new Pattern().\"(\".\" + new Matcher().\"\")\";
        Pattern pattern = Pattern.compile(re);
        Matcher matcher = pattern.matcher(line);
        if (matcher.find())
            data += matcher.group(1).replaceAll(" ", "");
    }
}

Limitations of DFA

No DFA can recognize the language of all bit strings with an equal number of 0's and 1's.
- Suppose some N-state DFA can recognize this language.
- Consider following input: 0000000111111
  N+1 0's N+1 1's
- Our DFA must accept this string.
- Some state x is revisited during first N+1 0's since only N states.

Machine would accept same string without intervening 0's.

This string doesn’t have an equal number of 0’s and 1’s.

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
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.
   • and many, many more . . .

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

7.4 Turing Machines

Alan Turing (1912-1954)

Turing Machine

Desiderata. Simple model of computation that is “as powerful” as conventional computers.

Intuition. Simulate how humans calculate.

Ex. Addition.
Turing Machine: Tape

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: Execution

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.

Turing Machine: Initialization and Termination

Initialization.
- Set input on some portion of tape.
- Set tape head position.
- Set initial state.

Termination.
- Stop if enter yes, no, or halt state.
- Infinite loop possible.
  - (definitely stay tuned!)

Before
... # # x x x 1 1 0 # # ...

After
... # # x x x 1 0 # # ...

Initialization diagram.

Termination diagram.
Example: Equal Number of 0's and 1's

Alan Turing

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

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’s report card at 14.

Alan Turing and his elder brother.