7. Theory of Computation

Introduction to Theoretical CS

Two fundamental questions.

- What can a computer do?
- What can a computer do with limited resources?

General approach.

e.g., Intel Atom running Linux kernel 2.6

- 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

Describing a Pattern

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**

CAASCGGPYACGGWAGYHAGWH

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, 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.

Operation	Regular Expression	Yes	No
Concatenation	`aabaab`	aabaab	*every other string*
Wildcard	`.u.u.u.`	cumulus jugulum	succubus tumultuous
Union	`aa | baab`	aa baab	*every other string*
Closure	`ab*a`	aa abbba	ab ababa
Parentheses	`a(a|b)aab`	aaaab abaab	*every other string*
	`(ab)*a`	a ababababa	aa abbba

Regular Expressions: Examples

Regular expression. Notation is surprisingly expressive.

Regular Expression	Yes	No
`.*spb.*` *contains the trigraph* spb	raspberry crispbread	subspace subspecies
`a* | (a*ba*ba*ba*)*` *multiple of three* b's	bbb aaa bbbaababbaa	b bb baabbbaa
`.*0....` *fifth to last digit is* 0	1000234 98701234	111111111 403982772
`gcg(cgg|agg)*ctg` *fragile X syndrome indicator*	gcgctg gcgcggctg gcgcggaggctg	gcgcgg cggcggcggctg gcgcaggctg

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)*`.

Operation	Regular Expression	Yes	No
One or more	`a(bc)+de`	abcde abcbcde	ade bcde
Character classes	`[A-Za-z][a-z]*`	lowercase Capitalized	camelCase 4illegal
Exactly k	`[0-9]{5}-[0-9]{4}`	08540-1321 19072-5541	111111111 166-54-1111
Negations	`[^aeiou]{6}`	rhythm	decade

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));
   }
}
```

powerful string library method

C_2H_2 type zinc finger domain

```
% java Validate "C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H" CAASCGGPYACGGAAGYHAGAH
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)" wayne@cs.princeton.edu
true
```

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 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

---

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

create array of words in document

regular expression that matches any whitespace character

DFAs

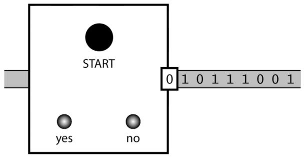
Solving the Pattern Match Problem

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.

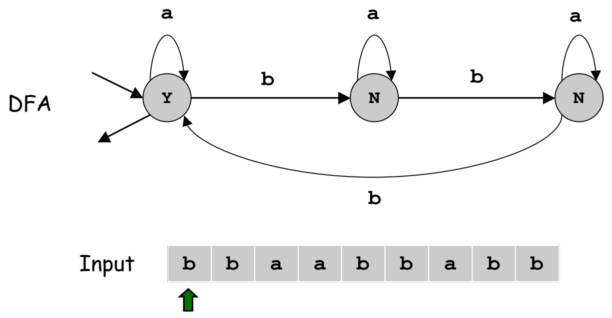
- Different machine for each pattern.
- Accepts or rejects string specified on input tape.
- Focus on `true` or `false` questions for simplicity.



Deterministic Finite State Automaton (DFA)

Simple machine with N states.

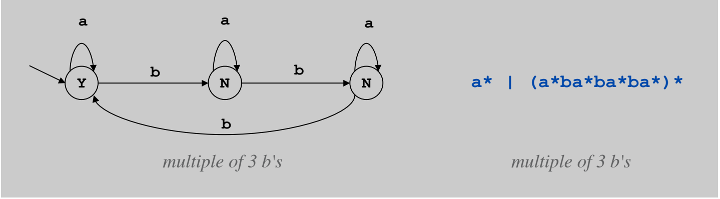
- Begin in start state.
- Read first input symbol.
- Move to new state, depending on current state and input symbol.
- Repeat until last input symbol read.
- Accept input string if last state is labeled Y.



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.



multiple of 3 b's

`a* | (a*ba*ba*ba*)*`

multiple of 3 b's

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.

```
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.

```
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgcggcggcggcggcggctg
gcgctg
gcgctg
gcgcggcggcggaggcggaggcggctg
```

- Harvest email addresses from web for spam campaign.

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

Application: Harvester

equivalent, but more efficient representation of a DFA

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());
      }
   }
}
```

create NFA from RE

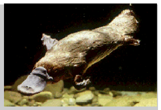
create NFA simulator

look for next match

the match most recently found

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 CLM1-393H9,
ACCESSION AC146846
VERSION AC146846.2 GI:38304214
KEYWORDS HTG; HTGS_PHASE2; HTGS_DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN
     1 tgtatttcat ttgaccgtgc tgttttttcc cggtttttca gtacggtgtt agggagccac
    61 gtgattctgt ttgttttatg ctgccgaata gctgctcgat gaatctctgc atagacagct  // a comment
   121 gccgcaggga gaaatgacca gtttgtgatg acaaaatgta ggaaagctgt ttcttcataa
   ...
128101 ggaaatgcga cccccacgct aatgtacagc ttctttagat tg
//
```

```
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
   }
}
```

extract the part of match in parens

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.