5.4 REGULAR EXPRESSIONS

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
5.4 Regular Expressions

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
Pattern matching

Substring search. Find a single string in text.

Pattern matching. Find one of a specified set (pattern) of strings in text.

Ex. [genomics]

- Fragile X syndrome is a common cause of mental retardation.
- A human's genome is a string.
- In Fragile X syndrome, contains triplet repeats of CGG or AGG, bracketed by CGG at the beginning and CTG at the end.
- Number of repeats is variable and is correlated to syndrome.

\[
\text{pattern} \quad GCG(CGG|AGG)^*CTG
\]

\[
\text{text} \quad GCGGCGGTGTGTCGAGAGAGTGGTTTAAAGCTG\text{\underline{GCG}}\text{\underline{CGG}}\text{\underline{AGG}}\text{\underline{CGG}}\text{\underline{CTG}}\text{GCGGAGGCTG}
\]
Prosite (computational biochemistry)

Database of protein domains, families and functional sites

PROSITE consists of documentation entries describing protein domains, families and functional sites as well as associated patterns and profiles to identify them [More... / References / Commercial users]. PROSITE is complemented by ProRule, a collection of rules based on profiles and patterns, which increases the discriminatory power of profiles and patterns by providing additional information about functionally and/or structurally critical amino acids [More...].


http://prosite.expasy.org
/*Compilation: javac NFA.java
 * Execution: java NFA regexp text
 * Dependencies: Stack.java Bag.java Digraph.java DirectedDFS.java
 */

% java NFA "(A*B|AC)D" AAAABD
true
%
% java NFA "(A*B|AC)D" AAAAC
false

public class NFA
{
    private Digraph G; // digraph of epsilon transitions
    private String regexp; // regular expression
    private int M; // number of characters in regular expression

    // Create the NFA for the given RE
    public NFA(String regexp)
    {
        this.regexp = regexp;
        M = regexp.length();
        Stack<Integer> ops = new Stack<Integer>();
        G = new Digraph(M+1);
        ...
    
    GNU source-highlight 3.1.4
Pattern matching: applications

Test if a string matches some pattern.
- Scan for virus signatures.
- Process natural language.
- Specify a programming language.
- Access information in digital libraries.
- Search genome using PROSITE patterns.
- Filter text (spam, NetNanny, Carnivore, malware).
- Validate data-entry fields (dates, email, URL, credit card).
  ...

Parse text files.
- Compile a Java program.
- Crawl and index the Web.
- Read in data stored in ad hoc input file format.
- Create Java documentation from Javadoc comments.
  ...
## Regular expressions

A regular expression is a notation to specify a set of strings.

<table>
<thead>
<tr>
<th>operation</th>
<th>order</th>
<th>example RE</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>concatenation</td>
<td>3</td>
<td>AABAAB</td>
<td>AABAAB</td>
<td>every other string</td>
</tr>
<tr>
<td>or</td>
<td>4</td>
<td>AA</td>
<td>BAAB</td>
<td>AA BAAB</td>
</tr>
<tr>
<td>closure</td>
<td>2</td>
<td>AB*A</td>
<td>AA ABBBBBBBBBA</td>
<td>AB ABABA</td>
</tr>
<tr>
<td>parentheses</td>
<td>1</td>
<td>A(A</td>
<td>BA)AAB</td>
<td>AAAAB ABAAB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AB)*A</td>
<td>A ABABABABABABA</td>
<td>AA ABBA</td>
</tr>
</tbody>
</table>
Which one of the following strings is not matched by the regular expression $$( A B \mid C^* D )^*$$?

A. A B A B A B
B. C D C C D D D D
C. A B C C D A B
D. A B D A B C A B D
E. I don't know.
Regular expression shortcuts

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><strong>wildcard</strong></td>
<td>.U.U.U.</td>
<td>CUMULUS JUGULUM</td>
<td>SUCCUBUS TUMULTUOUS</td>
</tr>
<tr>
<td><strong>character class</strong></td>
<td>[A-Za-z][a-z]*</td>
<td>word Capitalized</td>
<td>camelCase 4illegal</td>
</tr>
<tr>
<td><strong>one or more</strong></td>
<td>A(BC)+DE</td>
<td>ABCDE</td>
<td>ADE BCDE</td>
</tr>
<tr>
<td><strong>exactly k</strong></td>
<td>[0-9]{5}-[0-9]{4}</td>
<td>08540-1321 19072-5541</td>
<td>1111111111 166-54-111</td>
</tr>
</tbody>
</table>

**Note.** These operations are useful but not essential.

**Ex.** [A-E]+ is shorthand for (A|B|C|D|E)(A|B|C|D|E)*
Regular expression examples

RE 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 CRISPBREAD</td>
<td>SUBSPACE SUBSPECIES</td>
</tr>
<tr>
<td>(substring search)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[0-9]{3}-[0-9]{2}-[0-9]{4}</td>
<td>166-11-4433 166-45-1111</td>
<td>11-55555555 8675309</td>
</tr>
<tr>
<td>(U. S. Social Security numbers)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[a-z]+@[a-z]+.(edu</td>
<td>com)</td>
<td><a href="mailto:wayne@princeton.edu">wayne@princeton.edu</a> <a href="mailto:users@princeton.edu">users@princeton.edu</a></td>
</tr>
<tr>
<td>(simplified email addresses)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[$_A-Za-z][_A-Za-z0-9]*</td>
<td>ident3 PatternMatcher</td>
<td>3a ident#3</td>
</tr>
<tr>
<td>(Java identifiers)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

REs play a well-understood role in the theory of computation.
Which of the following REs match genes:
(1) alphabet is \{ A, G, T, C \}
(2) length is a multiple of 3
(3) starts with ATG (a start codon)
(4) ends with TAG or TAA or TTG (a stop codon)

A. \text{ATG}( (A\mid C\mid T\mid G) (A\mid C\mid T\mid G) (A\mid C\mid T\mid G) )^{*}(\text{TAG}\mid \text{TAA}\mid \text{TTG})

B. \text{ATG}( [\text{AGTC}] \{3\} )^{*}(\text{TAG}\mid \text{TAA}\mid \text{TTG})

C. Both A and B.

D. Neither A nor B.

E. \text{I don't know.}
Illegally screening a job candidate

“[First name]! and pre/2 [last name] w/7
bush or gore or republican! or democrat! or charg!
or accus! or criticiz! or blam! or defend! or iran contra
or clinton or spotted owl or florida recount or sex!
or controvers! or fraud! or investigat! or bankrupt!
or layoff! or downsiz! or PNTR or NAFTA or outsourc!
or indict! or enron or kerry or iraq or wmd! or arrest!
or intox! or fired or racis! or intox! or slur!
or controvers! or abortion! or gay! or homosexual!
or gun! or firearm! ”

— LexisNexis search pattern used by Monica Goodling
to screen candidates for DOJ positions

Regular expressions to the rescue

http://xkcd.com/208
Can the average programmer learn to use REs?

Perl RE for valid RFC822 email addresses

http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html
Regular expression caveat

Writing a RE is like writing a program.

- Need to understand programming model.
- Can be easier to write than read.
- Can be difficult to debug.

“Some people, when confronted with a problem, think ‘I know I'll use regular expressions.’ Now they have two problems.”

— Jamie Zawinski  (flame war on alt.religion.emacs)

Bottom line. REs are amazingly powerful and expressive, but using them in applications can be amazingly complex and error-prone.
5.4 Regular Expressions

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
Duality between REs and DFAs

**RE.** Concise way to describe a set of strings.

**DFA.** Machine to recognize whether a given string is in a given set.

**Kleene's theorem.**
- 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 of strings.

\[
0^* | (0^*10^*10^*10^*)^* \\
\text{number of 1's is a multiple of 3}
\]

\[
\text{number of 1's is a multiple of 3}
\]
Pattern matching implementation: basic plan (first attempt)

Overview is the same as for KMP.
- No backup in text input stream.
- Linear-time guarantee.

Underlying abstraction. Deterministic finite state automata (DFA).

Basic plan. [apply Kleene’s theorem]
- Build DFA from RE.
- Simulate DFA with text as input.

Bad news. Basic plan is infeasible (DFA may have exponential # of states).
Pattern matching implementation: basic plan (revised)

Overview is similar to KMP.
- No backup in text input stream.
- Quadratic-time guarantee (linear-time typical).

Underlying abstraction. Nondeterministic finite state automata (NFA).

Basic plan. [apply Kleene’s theorem]
- Build NFA from RE.
- Simulate NFA with text as input.

Q. What is an NFA?
Nondeterministic finite-state automata

Regular-expression-matching NFA.
- We assume RE enclosed in parentheses.
- One state per RE character (start = 0, accept = M).
- Red \( \epsilon \)-transition (change state, but don't scan text).
- Black match transition (change state and scan to next text char).
- Accept if any sequence of transitions ends in accept state.

Nondeterminism.
- One view: machine can guess the proper sequence of state transitions.
- Another view: sequence is a proof that the machine accepts the text.

NFA corresponding to the pattern \(( ( A * B | A C ) D )\)
Q. Is $A A A A B D$ matched by NFA?
A. Yes, because some sequence of legal transitions ends in state 11.

NFA corresponding to the pattern $( ( A * B | A C ) D )$
Nondeterministic finite-state automata

Q. Is A A A A B D matched by NFA?
A. Yes, because some sequence of legal transitions ends in state 11.
   [ even though some sequences end in wrong state or get stuck ]

NFA corresponding to the pattern ( ( A * B | A C ) D )
Nondeterministic finite-state automata

Q. Is A A A C matched by NFA?
A. No, because no sequence of legal transitions ends in state 11.
   [ but need to argue about all possible sequences ]

NFA corresponding to the pattern ( ( A * B | A C ) D )
Nondeterminism

Q. How to determine whether a string is matched by an automaton?

DFA. Deterministic ⇒ easy (only one applicable transition at each step).

NFA. Nondeterministic ⇒ hard (can be several applicable transitions at each step; need to select the "right" ones!)

Q. How to simulate NFA?

A. Systematically consider all possible transition sequences. [stay tuned]

NFA corresponding to the pattern ( ( A * B | A C ) D )
5.4 Regular Expressions

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
NFA representation

**State names.** Integers from 0 to $M$.

**Match-transitions.** Keep regular expression in array $re[]$.

$$re[] = ( ( A \ast B \mid A C ) D )$$

**$\varepsilon$-transitions.** Store in a digraph $G$.

$0 \rightarrow 1, 1 \rightarrow 2, 1 \rightarrow 6, 2 \rightarrow 3, 3 \rightarrow 2, 3 \rightarrow 4, 5 \rightarrow 8, 8 \rightarrow 9, 10 \rightarrow 11$

NFA corresponding to the pattern $( ( A \ast B \mid A C ) D )$
Q. How to efficiently simulate an NFA?
A. Maintain set of all possible states that NFA could be in after reading in the first \( i \) text characters.

one step in simulating an NFA

Q. How to perform reachability?
Goal. Check whether input matches pattern.

NFA simulation demo

NFA corresponding to the pattern \(( ( A * B | A C ) D )\)
Digraph reachability review

Goal. Find all vertices reachable from a given vertex or set of vertices.

Solution. Run DFS from each source, without unmarking vertices.
Performance. Runs in time proportional to $E + V$. 

```java
public class DirectedDFS {
    DirectedDFS(Digraph G, int s) find vertices reachable from s
    DirectedDFS(Digraph G, Iterable<Integer> s) find vertices reachable from sources
    boolean marked(int v) is v reachable from source(s)?
}
```
public class NFA
{
    private char[] re; // match transitions
    private Digraph G; // epsilon transition digraph
    private int M; // number of states

    public NFA(String regexp)
    {
        M = regexp.length();
        re = regexp.toCharArray();
        G = buildEpsilonTransitionDigraph();
    }

    public boolean recognizes(String txt)
    { /* see next slide */ }

    public Digraph buildEpsilonTransitionDigraph()
    { /* stay tuned */ }
}
public boolean recognizes(String txt) {
    Bag<Integer> pc = new Bag<Integer>();
    DirectedDFS dfs = new DirectedDFS(G, 0);
    for (int v = 0; v < G.V(); v++)
        if (dfs.marked(v)) pc.add(v);

    for (int i = 0; i < txt.length(); i++)
    {
        Bag<Integer> states = new Bag<Integer>();
        for (int v : pc)
            {
                if (v == M) continue;
                if (((re[v] == txt.charAt(i)) || re[v] == '.')).states.add(v+1);
            }
        dfs = new DirectedDFS(G, states);
        pc = new Bag<Integer>();
        for (int v = 0; v < G.V(); v++)
            if (dfs.marked(v)) pc.add(v);
    }
    for (int v : pc)
        if (v == M) return true;
    return false;
}
**NFA simulation: analysis**

**Proposition.** Determining whether an $N$-character text is recognized by the NFA corresponding to an $M$-character pattern takes time proportional to $MN$ in the worst case.

**Pf.** For each of the $N$ text characters, we iterate through a set of states of size no more than $M$ and run DFS on the graph of $\varepsilon$-transitions. [The NFA construction we will consider ensures the number of edges $\leq 3M$.]
5.4 REGULAR EXPRESSIONS

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
Building an NFA corresponding to an RE

**States.** Include a state for each symbol in the RE, plus an accept state.
Building an NFA corresponding to an RE

**Concatenation.** Add match-transition edge from state corresponding to characters in the alphabet to next state.

**Alphabet.** A B C D

**Metacharacters.** ( ) . * |
Building an NFA corresponding to an RE

**Parentheses.** Add $\varepsilon$-transition edge from parentheses to next state.

NFA corresponding to the pattern ( ( A * B | A C ) D )
Building an NFA corresponding to an RE

**Closure.** Add three $\varepsilon$-transition edges for each $*$ operator.

![Diagram of NFA corresponding to the pattern ( ( A * B | A C ) D )](image-url)
Building an NFA corresponding to an RE

2-way or. Add two $\varepsilon$-transition edges for each \texttt{|} operator.

NFA corresponding to the pattern \texttt{( ( A * B | A C ) D )}
Regular expression: quiz 4

How would you modify the NFA below to match \(((ABC^*)^+)\) ?

A. Remove $\varepsilon$-transition edge 1→7.
B. Remove $\varepsilon$-transition edge 7→1.
C. Remove $\varepsilon$-transition edges 1→7 and 7→1.
D. *I don't know.*

NFA corresponding to the pattern \(((A B C^*)^+)\)
NFA construction: implementation

**Goal.** Write a program to build the $\varepsilon$-transition digraph.

**Challenges.** Remember left parentheses to implement closure and or; remember $|$ symbols to implement or.

**Solution.** Maintain a stack.
- ( symbol: push ( onto stack.
- | symbol: push $|$ onto stack.
- ) symbol: pop corresponding ( and any intervening $|$; add $\varepsilon$-transition edges for closure/or.

NFA corresponding to the pattern ( ( A * B | A C ) D )
NFA construction demo

( ( A * B | A C ) D )
NFA construction demo

NFA corresponding to the pattern \(( ( A \ast B \mid A \cdot C ) D )\)
private Digraph buildEpsilonTransitionDigraph() {
    Digraph G = new Digraph(M+1);
    Stack<Integer> ops = new Stack<Integer>();
    for (int i = 0; i < M; i++) {
        int lp = i;

        if (re[i] == ' ' || re[i] == '.')
            ops.push(i);

        else if (re[i] == ')') {
            int or = ops.pop();
            if (re[or] == ')') {
                lp = ops.pop();
                G.addEdge(lp, or+1);
                G.addEdge(or, i);
            } else lp = or;
        }

        if (i < M-1 & re[i+1] == '*') {
            G.addEdge(lp, i+1);
            G.addEdge(i+1, lp);
        }

        if (re[i] == '(' || re[i] == '*' || re[i] == ')')
            G.addEdge(i, i+1);
    }
    return G;
}
**NFA construction: analysis**

**Proposition.** Building the NFA corresponding to an $M$-character RE takes time and space proportional to $M$.

**Pf.** For each of the $M$ characters in the RE, we add at most three $\varepsilon$-transitions and execute at most two stack operations.
5.4 Regular Expressions

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
Generalized regular expression print

**Grep.** Take a RE as a command-line argument and print the lines from standard input having some substring that is matched by the RE.

```
public class GREP
{
    public static void main(String[] args)
    {
        String re = "(.* + args[0] + ".")";
        NFA nfa = new NFA(re);
        while (StdIn.hasNextLine())
        {
            String line = StdIn.readLine();
            if (nfa.recognizes(line))
                StdOut.println(line);
        }
    }
}
```

**Bottom line.** Worst-case for grep (proportional to $MN$) is the same as for brute-force substring search.
Typical grep application: crossword puzzles

% more words.txt
a
aback
abacus
abalone
abandon
...

% grep "s..ict.." words.txt
constrictor
stricter
stricture

(dictionary (standard in Unix))
Industrial-strength grep implementation

To complete the implementation:

- Add multiway or.
- Handle metacharacters.
- Support character classes.
- Add capturing capabilities.
- Extend the closure operator.
- Error checking and recovery.
- Greedy vs. reluctant matching.

**Ex.** Which substring(s) should be matched by the RE `<blink>.*</blink>`?

```plaintext
<blink>text</blink> some text <blink>more text</blink>
```
Regular expressions in the wild

Broadly applicable programmer's tool.

- Originated in Unix in the 1970s.
- Built in to many tools: grep, egrep, emacs, ....

```bash
% grep 'NEWLINE' */*.java
```
print all lines containing NEWLINE which occurs in any file with a .java extension

```bash
% egrep '^[qwertuyiop]*[zxcvbnm]*$' words.txt | egrep '............'
```
typewritten

- Built in to many languages: awk, Perl, PHP, Python, JavaScript, ....

```bash
% perl -p -i -e 's|from|to|g' input.txt
```
replace all occurrences of from with to in the file input.txt

```bash
% perl -n -e 'print if /^[A-Z][A-Za-z]*/' words.txt
```
print all words that start with uppercase letter

```bash
# do for each line
```

Regular expressions in Java

Validity checking. Does the input match the re?

Java string library. Use input.matches(re) for basic RE matching.

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

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

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

% java Validate "[0-9]{3}-[0-9]{2}-[0-9]{4}" 166-11-4433
true

legal Java identifier
valid email address (simplified)
Social Security number
Harvesting information

**Goal.** Print all substrings of input that match a RE.

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

- harvest patterns from DNA

```java
% java Harvester "http://(\w+.)*(\w+)" http://www.cs.princeton.edu
http://www.w3.org
http://www.cs.princeton.edu
http://drupal.org
http://csguide.cs.princeton.edu
http://www.cs.princeton.edu
http://www.princeton.edu
```

- harvest links from website
Harvesting information

RE pattern matching is implemented in Java's `java.util.regex.Pattern` and `java.util.regex.Matcher` classes.

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

public class Harvester {
    public static void main(String[] args) {
        String regexp = args[0];
        In in = new In(args[1]);
        String input = in.readAll();
        Pattern pattern = Pattern.compile(regexp);
        Matcher matcher = pattern.matcher(input);
        while (matcher.find()) {
            StdOut.println(matcher.group());
        }
    }
}
```

- `compile()` creates a `Pattern` (NFA) from RE
- `matcher()` creates a `Matcher` (NFA simulator) from NFA and text
- `find()` looks for the next match
- `group()` returns the substring most recently found by `find()`
Algorithmic complexity attacks

Warning. Typical implementations do not guarantee performance!

Unix grep, Java, Perl, Python

<table>
<thead>
<tr>
<th>Expression</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a</td>
<td>aa)*b</td>
</tr>
<tr>
<td>(a</td>
<td>aa)*b</td>
</tr>
<tr>
<td>(a</td>
<td>aa)*b</td>
</tr>
<tr>
<td>(a</td>
<td>aa)*b</td>
</tr>
<tr>
<td>(a</td>
<td>aa)*b</td>
</tr>
<tr>
<td>(a</td>
<td>aa)*b</td>
</tr>
</tbody>
</table>

SpamAssassin regular expression.

% java RE "[a-z]+@[a-z]+([a-z\.]+)\+[a-z]+" spammer@x......................

- Takes exponential time on pathological email addresses.
- Attacker can use such addresses to DOS a mail server.
Not-so-regular expressions

Back-references.

• \1 notation matches subexpression that was matched earlier.
• Supported by typical RE implementations.

```
(\.+)\1       // beriberi couscous
1??|^\(11+?\)\1+  // 1111 111111 11111111
```

Some non-regular languages.

• Strings of the form \textit{ww} for some string w: beriberi.
• Unary strings with a composite number of 1s: 111111.
• Bitstrings with an equal number of 0s and 1s: 01110100.
• Watson-Crick complemented palindromes: atttcggaat.

Remark. Pattern matching with back-references is intractable.
Abstract machines, languages, and nondeterminism.

- Basis of the theory of computation.
- Intensively studied since the 1930s.
- Basis of programming languages.

Compiler. A program that translates a program to machine code.

- KMP string $\Rightarrow$ DFA.
- grep RE $\Rightarrow$ NFA.
- javac Java language $\Rightarrow$ Java byte code.
Summary of pattern-matching algorithms

Programmer.
- Implement substring search via DFA simulation.
- Implement RE pattern matching via NFA simulation.

Theoretician.
- RE is a compact description of a set of strings.
- NFA is an abstract machine equivalent in power to RE.
- DFAs, NFAs, and REs have limitations.

You.
- Core CS principles provide useful tools that you can exploit now.
- REs and NFAs provide introduction to theoretical CS.

Example of essential paradigm in computer science.
- Build the right intermediate abstractions.
- Solve important practical problems.