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 of strings in text.

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

\[
\text{pattern } \text{GCCCGGGTGTGCGAGAGAGTGGTTTAAAGCTTGGCCGGAGGGCCTTGCCCGGGAGGCTG}
\]

\[
\text{text } GCGCGGTGTGCGAGAGAGTGGTTTAAAGCTTGGCCGGAGGGCCTTGCCCGGGAGGCTG
\]
Google code search

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 every other string</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>4</td>
<td>AA</td>
<td>BAAB</td>
<td>AA BAAB every other string</td>
</tr>
<tr>
<td>closure</td>
<td>2</td>
<td>AB*A</td>
<td>AB ABAB every other string</td>
<td></td>
</tr>
<tr>
<td>parentheses</td>
<td>1</td>
<td>A(A</td>
<td>B)AAB</td>
<td>AAAAAAAAAAA every other string</td>
</tr>
</tbody>
</table>

Regular expression shortcuts

Additional operations are often added for convenience.

<table>
<thead>
<tr>
<th>operation</th>
<th>example RE</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>wildcard</td>
<td>.U.U.U.</td>
<td>CUMULUS JUGULUM</td>
<td>SUCCUBUS TUMULTUOUS</td>
</tr>
<tr>
<td>character class</td>
<td>[A-Za-z][A-z]*</td>
<td>word Capitalized</td>
<td>camelCase illegal</td>
</tr>
<tr>
<td>at least k</td>
<td>A(BC)+DE</td>
<td>ABCDE ABCBCDE</td>
<td>ADE BCDE</td>
</tr>
<tr>
<td>exactly k</td>
<td>[0-9]{5}–[0-9]{4}</td>
<td>08540–1321 19072–5541</td>
<td>111111111 166–54–111</td>
</tr>
</tbody>
</table>

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 CRISPREAD</td>
<td>SUBSPACE SUBSPECIES</td>
</tr>
<tr>
<td>[0-9]{3}-[0-9]{2}-[0-9]{4}</td>
<td>166-11-4433 166-45-1111</td>
<td>11-5555555 8675309</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:rs@princeton.edu">rs@princeton.edu</a></td>
</tr>
<tr>
<td>$A-Za-z]*</td>
<td>PatternMatcher</td>
<td>3a ident#3</td>
</tr>
</tbody>
</table>

REs play a well-understood role in the theory of computation.

Can the average web surfer learn to use REs?

Google. Supports * for full word wildcard and | for union.

Illegally screening a job candidate

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


Regular expression golf

<table>
<thead>
<tr>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>obama</td>
<td>romney</td>
</tr>
<tr>
<td>bush</td>
<td>mccain</td>
</tr>
<tr>
<td>clinton</td>
<td>kerry</td>
</tr>
<tr>
<td>reagan</td>
<td>gore</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>washington</td>
<td>clinton</td>
</tr>
</tbody>
</table>

Ex. Match elected presidents but not opponents (unless they later won).

RE. bu[rn]t[coy][mtg][a][iso][hl][ae][lev][sh][nd][i][po][ls

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

Regular expressions to the rescue

Oh no, the killer must have followed me on vacation!

But to find them we’d have to search through 200 Mb of emails looking for something formatted like an address!

It’s hopeless!

Everybody stand back.

I know regular expressions.

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

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

5.4 Regular Expressions

‣ Regular expressions
‣ REs and NFAs
‣ NFA simulation
‣ NFA construction
‣ Applications

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.
**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.

---

**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 ε-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.
Nondeterministic finite-state automata

Q. Is A A A B D matched by NFA?
A. Yes, because some sequence of legal transitions ends in state 11.

Q. Is A 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 ]

Nondeterminism

Q. How to determine whether a string is matched by an automaton?
DFA. Deterministic ⇒ easy because exactly one applicable transition.

NFA. Nondeterministic ⇒ can be several applicable transitions; 
need to select the right one!

Q. How to simulate NFA?
A. Systematically consider all possible transition sequences. [stay tuned]
5.4 Regular Expressions

- Regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- Applications

NFA representation

- State names. Integers from 0 to $M$.
- Number of symbols in RE

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

- $\varepsilon$-transitions. Store in a digraph $G$.
  0→1, 1→2, 1→6, 2→3, 3→2, 3→4, 5→8, 8→9, 10→11

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

NFA simulation

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.

Q. How to perform reachability?
NFA simulation demo

When no more input characters:
- Accept if any state reachable is an accept state.
- Reject otherwise.

set of states reachable : \{ 10, 11 \}

Digraph reachability

Digraph reachability. Find all vertices reachable from a given source or set of vertices. [recall Section 4.2]

Solution. Run DFS from each source, without unmarking vertices.

Performance. Runs in time proportional to $E + V$.

NFA simulation: Java implementation

```java
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 */
    }
}
```

```java
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: Java implementation
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 $\leq 3M$.]

---

### 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.

![Diagram of NFA corresponding to the pattern \(( ( A \ast B | A C ) \ D )\)](image)

Building an NFA corresponding to an RE

**Or.** Add two \( \varepsilon \)-transition edges for each \(|\) operator.

![Diagram of NFA corresponding to the pattern \(( ( A \ast B | A C ) \ D )\)](image)

NFA construction: implementation

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

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

![Diagram of NFA construction rules](image)
NFA construction demo

NFA construction: Java implementation

```java
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] == ')') {  // left parenthases and |
            ops.push(i);
        } else if (re[i] == '|') {  // 2-way or
            int or = ops.pop();
            if (re[i] == ')') {
                lp = ops.pop();
                G.addEdge(lp, or+1);
                G.addEdge(or, i);
            } else lp = or;
        }
        if (i < M-1 && re[i+1] == '*') {  // closure (needs 1-character lookahead)
            G.addEdge(lp, i+1);
        } else if (re[i] == '(' || re[i] == ')' || re[i] == '|') {  // metasymbols
            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.
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);
        }
    }
}
```

Contains RE as a substring

Bottom line. Worst-case for grep (proportional to $MN$) is the same as for brute-force substring search.

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>`? One possibility is `text <blink>more text</blink>`.
Regular expressions in other languages

Broadly applicable programmer’s tool.
- Originated in Unix in the 1970s.
- Many languages support extended regular expressions.
- Built into grep, awk, emacs, Perl, PHP, Python, JavaScript, ...

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

```
% egrep '![qwertuiop]*[zcxbnm]*$' words.txt | egrep '.........'
```
typed written

**PERL.** Practical Extraction and Report Language.

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

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

Harvesting information

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

```
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcggcgcggcgccggtg
gcgcggcgccgctg
gcggcgcggcgaggcggcggcggcggctg
```
harvest patterns from DNA

```
% java Harvester "http://(\w+)\*((\w+)\)" http://www.cs.princeton.edu
```
harvest links from website

```
% java Harvester "http://(\w+)\*((\w+)\)" http://www.google.com
```
http://www.cs.princeton.edu/news

Regular expressions in Java

**Validity checking.** Does the input match the re?

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

```
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-][0-9-][2][0-9-][4]" 166-11-4433
true
```
legal Java identifier
valid email address (simplified)
Social Security number

```
% java Validate "([A-Za-z])\"+ident123"
true
```

Harvesting information

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

```
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());
        }
    }
}
```

```
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());
        }
    }
}
```

```
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());
        }
    }
}
```

```
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());
        }
    }
}
```

```
Algorithmic complexity attacks

Warning. Typical implementations do not guarantee performance!

Unix grep, Java, Perl, Python

\% java Validate "(a|aa)b" aaaaaaaaaaaaaaaaaaaaaaaaaaaac 1.6 seconds
\% java Validate "(a|aa)b" aaaaaaaaaaaaaaaaaaaaaaaaaaaac 3.7 seconds
\% java Validate "(a|aa)b" aaaaaaaaaaaaaaaaaaaaaaaaaaaac 9.7 seconds
\% java Validate "(a|aa)b" aaaaaaaaaaaaaaaaaaaaaaaaaaaac 23.2 seconds
\% java Validate "(a|aa)b" aaaaaaaaaaaaaaaaaaaaaaaaaaaac 62.2 seconds
\% java Validate "(a|aa)b" aaaaaaaaaaaaaaaaaaaaaaaaaaaac 161.6 seconds

SpamAssassin regular expression.

\% java RE "[a-z]+@[a-z]+([a-z]+\.)+[a-z]^" spammer9x....................

- Takes exponential time on pathological email addresses.
- Troublemaker can use such addresses to DOS a mail server.

Not-so-regular expressions

Back-references.
- \(\textbackslash 1\) notation matches subexpression that was matched earlier.
- Supported by typical RE implementations.

\(\textbackslash (\textbackslash 1:\textbackslash 1)\textbackslash 1\) // beriberi couscous
175$\textbackslash (\textbackslash 1:7)\textbackslash 1$ // 1111 11111 1111111

Some non-regular languages.
- Strings of the form \(w^2\) for some string \(w\): beriberi.
- Unary strings with a composite number of 1s: 1111111.
- Bitstrings with an equal number of 0s and 1s: 01110100.
- Watson-Crick complemented palindromes: atttcggaat.

Remark. Pattern matching with back-references is intractable.

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. Practical application of core computer science principles.

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