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 CGG at the beginning and CTG at the end.
- Number of repeats is variable and is correlated to syndrome.

pattern \texttt{GCG(CGG|AGG)\*CTG}

text \texttt{GCCGCCGTCGTCGACAGAGTCGCTTTAAAGCTGCGCGGAGGCTGCTGCGCGGAGGCTG}

Syntax highlighting

```java
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>();
        C = new Digraph(M+1);
    }
}
```

GNU source-highlight 3.1.4
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 AA ABBBBBBBBBBBA</td>
<td>AB ABABA</td>
<td></td>
</tr>
<tr>
<td>parentheses</td>
<td>1</td>
<td>A(A</td>
<td>B)AAB AABAB AABAB</td>
<td>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 4illegal</td>
</tr>
<tr>
<td>at least 1</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>1111111111 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>-“SPB.*”</td>
<td>RASPBERRY CRISPBREAD</td>
<td>SUBSPACE SUBSPECIES</td>
</tr>
<tr>
<td>[0-9]{1,2}-0-9{2}-0-9{4}</td>
<td>166-11-4433 166-45-1111</td>
<td>11-5555555 8675309</td>
</tr>
<tr>
<td>[a-zA-Z]+@[a-zA-Z]+.edu</td>
<td><a href="mailto:wayne@princeton.edu">wayne@princeton.edu</a> <a href="mailto:rs@princeton.edu">rs@princeton.edu</a></td>
<td>spam@nowhere</td>
</tr>
<tr>
<td>[$A-Za-z]{2,}$</td>
<td>ident3 PatternMatcher</td>
<td>3a ident3</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.

Regular expressions to the rescue

http://xkcd.com/208

Illegal 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 criticize! or blame! or defend! or iran contra or clinton or spotted owl or florida recount or sex!
or controversies! or fraud! or investigate! or bankrupt!
or layoff! or downsiz! or PNTR or NAFTA or outsource!
or indict! or enron or kerry or iraq or wmd or arrest!
or intox! or fired or racist! or intox! or slur!
or controversies! or abortion! or gay! or homosexual!
or gun! or firearm! "

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

Can the average programmer learn to use REs?

Perl RE for valid RFC822 email addresses

http://www.ex-parrot.com/~pjd/Max1-RFC822-Address.html

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.

Stephen Kleene
Princeton Ph.D. 1934

"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)
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

Q. Is 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 stall ]

Stalling sequences for ( ( A * B | A C ) D )
NFA no way out of state 4
no way out of state 7
wrong guess if input is A A A A B D
A
0→1→2→3→2→3→4

A     A
0  1  2  3  2  3  4

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 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.

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

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

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

NFA simulation

**Q.** How to efficiently simulate an NFA?
**A.** Maintain set of all possible states that NFA could be in after reading the first $i$ text characters.

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

NFA simulation demo

**Goal.** Check whether input matches pattern.

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.

- **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)? }
}
```

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 = buildEpsilonTransitionsDigraph();
    }
    public boolean recognizes(String txt) {
        /* see next slide */
    }
    public Digraph buildEpsilonTransitionDigraph() {
        /* stay tuned */
    }
}
```

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
- 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 ε-transition edge from parentheses to next state.

Building an NFA corresponding to an RE

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 showing single-character closure and closure expression with \( \varepsilon \)-transition edges]

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

**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: pop corresponding ( and any intervening |; add \( \varepsilon \)-transition edges for closure/or.

![Diagram with stack and NFA corresponding to the pattern \( ( ( A * B | A C ) D ) \)]

Building an NFA corresponding to an RE

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

![Diagram showing or expression with \( \varepsilon \)-transition edges]

- **NFA corresponding to the pattern** \( ( ( A * B | A C ) D ) \)
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] == '|')) 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 $\epsilon$-transitions and execute at most two stack operations.

5.4  **Regular Expressions**

- regular expressions
- 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.

```java
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.

Industrial-strength grep implementation

To complete the implementation:
- Add character classes.
- Handle metacharacters.
- 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>`?

Typical grep application: crossword puzzles

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

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

Typical grep application: crossword puzzles

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

Dictionary (standard in Unix) also on booksite

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, ...

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

% egrep '^[qwertyuiop]*[zxcvbnm]*$' words.txt | egrep '..........'
```

Typewriter

```perl
% perl -p -i -e 's|from|to|g' input.txt
```

Replace all occurrences of from with to in the file input.txt

```perl
% perl -n -e 'print if /^[A-Z][A-Za-z]*$/' words.txt
```

Print all words that start with uppercase letter

Do for each line
**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-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
```

**Harvesting information**

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

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

```
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgctg
gcgctg
gcgctg
gcgctg
harvest patterns from DNA
harvest links from website
% java Harvester "http://(\w+\.)+(\w+)" http://www.cs.princeton.edu
http://www.princeton.edu
http://www.google.com
http://www.cs.princeton.edu/news
```

**Algorithmic complexity attacks**

**Warning.** Typical implementations do **not** guarantee performance!

```java
% java Validate "(a|a)b"aaaaaaaaaaaaaaaaaaaaaaaaaac 1.6 seconds
% java Validate "(a|a)b"aaaaaaaaaaaaaaaaaaaaaaaaaac 3.7 seconds
% java Validate "(a|a)b"aaaaaaaaaaaaaaaaaaaaaaaaaac 9.7 seconds
% java Validate "(a|a)b"aaaaaaaaaaaaaaaaaaaaaaaaaac 23.2 seconds
% java Validate "(a|a)b"aaaaaaaaaaaaaaaaaaaaaaaaaac 62.2 seconds
% java Validate "(a|a)b"aaaaaaaaaaaaaaaaaaaaaaaaaac 161.6 seconds
```

**SpamAssassin regular expression.**

```java
% java RE "[a-z]+@[a-z]+\.(a-z\.|.)+[a-z]+" spamer0x..............
```

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

Back-references.
- \l notation matches subexpression that was matched earlier.
- Supported by typical RE implementations.

```
(\+\l) \l // beriberi couscus
1?\l(11+\l)\l // 1111 1111111 111111111
```

Some non-regular languages.
- Strings of the form $w^2$ 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: `attnctctaat`.

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.

Context

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.

<table>
<thead>
<tr>
<th></th>
<th>KMP</th>
<th>grep</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>string</td>
<td>RE</td>
<td>program</td>
</tr>
<tr>
<td>parser</td>
<td>unnecessary</td>
<td>check if legal</td>
<td>check if legal</td>
</tr>
<tr>
<td>compiler output</td>
<td>DFA</td>
<td>NFA</td>
<td>byte code</td>
</tr>
<tr>
<td>simulator</td>
<td>DFA simulator</td>
<td>NFA simulator</td>
<td>JVM</td>
</tr>
</tbody>
</table>