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.

pattern: GCG(CGG|AGG)*CTG

text: GCGCGGTGTGGCGAGAGATGGTTTAAAGCTGGCAGCGCGCGGCTGGCAGCGGCTG

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>();
        G = 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.

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>AABAB</td>
<td>AABAB</td>
<td>every other string</td>
</tr>
<tr>
<td>or</td>
<td>4</td>
<td>AA</td>
<td>AA</td>
<td>every other string</td>
</tr>
<tr>
<td>closure</td>
<td>2</td>
<td>AB*A</td>
<td>AB</td>
<td>ABBBAAAA</td>
</tr>
<tr>
<td>parentheses</td>
<td>1</td>
<td>A(A</td>
<td>B)B</td>
<td>AAAAB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AB)*A</td>
<td>AB</td>
<td>ABBBBB</td>
</tr>
</tbody>
</table>

Regular expression: quiz 1

Which one of the following strings is not matched by the regular expression \((A B | C * D) *\) ?

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 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>.&quot;SPB.&quot;</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>[[S-A-Za-z][S-A-Za-z]-[S-A-Za-z]]+</td>
<td>ident3 PatternMatcher</td>
<td>3a</td>
</tr>
</tbody>
</table>

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

Regular expression golf

http://xkcd.com/1313

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

RE. bu[rn|e|t|mtg|[a]j|s|o|n|[h]|l|[ae]d|lev|sh|[lnd]i|[po]o|ls

Illegally screening a job candidate

" [First name]| and pre/[Last name] w/7
bush or gore or republican | or democrat | or champ
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 investigate | or bankrupt
or layoff| or downsiz| or PNT| 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 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 e-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.

NFA corresponding to the pattern \((A \ast B \mid A C)D\)

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

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

NFA. Nondeterministic \(\Rightarrow\) 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]
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 $re[]$.

ε-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.

one step in simulating an NFA

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}

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 */
        }
    }
}
```

Digraph reachability

**Digraph reachability.** 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 $E + V$.

NFA simulation: Java implementation

```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 v = 0; v < txt.length(); v++)
        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;
}
```

accept if can end in state M

not necessarily a match
(RE needs to match full text)

states reachable from start by $\varepsilon$-transitions

set of states reachable after scanning past txt.charAt(i)
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.

**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.

---

Building an NFA corresponding to an RE

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

---

Building an NFA corresponding to an RE

**2-way or.** Add two \( \varepsilon \)-transition edges for each \( | \) operator.

---

**Regular Expressions**

Q. How to implement the + and multiway or operators?
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.

```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] == 'r')
            G.addEdge(i, i+1);
    }
    return G;
}
```

closure
left parentheses and |
2-way or
needs |-character lookahead
metasymbols
accept state
**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.

**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>`?

```
\text{reluctant} \quad \text{reluctant} \quad \text{greedy}
```

```
<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, ...
  ```
  % grep 'NEWLINE' */*.java
  print all lines containing NEWLINE which occurs in any file with a .java extension
  ```
  ```
  % egrep '^[qwertyuiop]*[zxcvbnm]*$' words.txt | egrep '.........'
  typewritten
  ```
- Built in to many languages: awk, Perl, PHP, Python, JavaScript, ...
  ```
  % perl -p -i -e 's|from|to|g' input.txt
  replace all occurrences of from with to in the file input.txt
  ```
  ```
  % perl -n -e 'print if /^(A-Z)[A-Za-z]*$/' words.txt
  print all words that start with uppercase letter
  ```
  do for each line

**5.4 Regular Expressions**

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
### 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));
    }
}
```

### Harvesting information

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

```java
% java Harvester "g cg[cd]agggctg" chromosome.txt
```

### Algorithmic complexity attacks

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

```java
% java Validate "[a-z]+\([a-z]\)+" ident123
```

**SpamAssassin regular expression.**

```java
% 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.

```plaintext
(\+)?\1 // beriberi couscous
17\$1^+(11+?)\1+ // 1111 111111 1111111111
```

Some non-regular languages.
- Strings of the form \(ww\) 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: atttcggaaat.

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