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.

```
public class NFA {
    private Digraph C;       // digraph of epsilon transitions
    private String regexp;   // regular expression
    private int N;           // 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);
        ...
    }

    // Pattern matching
    public boolean isPattern(String text) {
        // Implementation details...
    }

    // Syntax highlighting
    public boolean isHighlight(String code) {
        // Implementation details...
    }
}
```

GNU source-highlight 3.1.4
Google code search

**Search public source code**

Search via regular expression, e.g. `*java/*.java$

**Search Options**

- **Package**: package:linux-2.6
- **Language**: Any language
- **File Path**: file (code) [~org].search
- **Class**: class:HashMap
- **Function**: function.toString
- **License**: Any license
- **Case Sensitive**: No

http://code.google.com/p/chromium/source/search

---

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 structurally critical amino acids [More...].


http://prosite.expasy.org

---

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 expression: quiz 1

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 D D D
C. A B 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>wildcard</td>
<td>.U.U.</td>
<td>Cumulus Juggulum</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>one or more</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>11111111 166–54–111</td>
</tr>
</tbody>
</table>

Note. These operations are useful but not essential.
Ex. [A-E]+ is shorthand for \((A\mid B\mid C\mid D\mid E)(A\mid B\mid C\mid D\mid E)^*\)

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>Rasperry Crispbread</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-zA-Z0-9]+(.</td>
<td>-)(edu</td>
<td>com)\</td>
</tr>
<tr>
<td>[S,A-Za-z] [S,A-Za-z-0-9]+\</td>
<td>ident3 PatternMatcher</td>
<td>3a ident#3</td>
</tr>
</tbody>
</table>

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

Regular expression: quiz 2

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. ATG\((A\mid C\mid T\mid G)(A\mid C\mid T\mid G)(A\mid C\mid T\mid G)^*\)(TAG\mid TAA\mid TTG)
B. ATG\((AGTC\{3\})^*\)(TAG\mid TAA\mid TTG)
C. Both A and B.
D. Neither A nor B.
E. I don't know.
Illegally screening a job candidate

" [First name] and pre/2 [last name] w/?
    bush or gore or republican or democrat or chargi
    or accusi or criticiz or blame or defend or iran contra
    or clinton or spotted owl or florida recount or sex
    or controversi or fraud or investigati or bankrupt
    or layoff or downsi or pnt or NAFTA or outsourc
    or indict or eron or kerry or iraq or wmd or arrest
    or intox or fired or racist or intox or slur
    or controversi 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

Can the average programmer learn to use REs?

Perl RE for valid RFC822 email addresses

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.

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 $\varepsilon$-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 $A A A A B D$ matched by NFA?
A. Yes, because some sequence of legal transitions ends in state 11.

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.
Nondeterminism

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]

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**

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

![NFA diagram](image)

**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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DirectedDFS(G, int s)</code></td>
<td>Find vertices reachable from <code>s</code></td>
</tr>
<tr>
<td><code>DirectedDFS(G, Iterable&lt;Integer&gt; s)</code></td>
<td>Find vertices reachable from sources</td>
</tr>
<tr>
<td><code>boolean marked(int v)</code></td>
<td>Is <code>v</code> reachable from source(s)?</td>
</tr>
</tbody>
</table>

**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 */
    }
}
```

**NFA simulation: Java implementation**

```java
public boolean recognizes(String txt) {
    Bag<Integer> pc = new Bag<Integer>();
    DirectedDFS dfs = new DirectedDFS(G, M);
    for (int v = 0; v < G.V(); v++)
        if (dfs.marked(v)) pc.add(v);

    for (int i = 0; i < txt.length(); i++)
        if (re[i] == txt.charAt(i)) pc.add(i);

    Bag<Integer> states = new Bag<Integer>();
    for (int v : pc)
        if (v == M) continue;
        if (re[v] == '.' || 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**

```java
public boolean recognizes(String txt) {
    Bag<Integer> pc = new Bag<Integer>();
    DirectedDFS dfs = new DirectedDFS(G, M);
    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 $\leq 3M$.]

---

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.

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

Building an NFA corresponding to an RE

2-way or. Add two ε-transition edges for each | operator.

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

Building an NFA corresponding to an RE

Closure. Add three ε-transition edges for each * operator.

Regular expression: quiz 4

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

A. Remove ε-transition edge 1→7.
B. Remove ε-transition edge 7→1.
C. Remove ε-transition edges 1→7 and 7→1.
D. I don't know.
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: 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] == ')$\) symbol: pop corresponding ( and any intervening $|$; add $\varepsilon$-transition edges for closure/or.

NFA construction demo

**NFA construction demo**

```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] == ')$\) symbol: pop corresponding ( and any intervening $|$; add $\varepsilon$-transition edges for closure/or.
            if (i < M-1 && re[i+1] == '*') {
                G.addEdge(lp, i+1);
            }  
        }  
    }  
    return G;
}  ```

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] == ')$\) symbol: pop corresponding ( and any intervening $|$; add $\varepsilon$-transition edges for closure/or.
            if (i < M-1 && re[i+1] == '*') {
                G.addEdge(lp, i+1);
            }  
        }  
    }  
    return G;
}  ```

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] == ')$\) symbol: pop corresponding ( and any intervening $|$; add $\varepsilon$-transition edges for closure/or.
            if (i < M-1 && re[i+1] == '*') {
                G.addEdge(lp, i+1);
            }  
        }  
    }  
    return G;
}  ```

NFA construction demo

**NFA construction demo**

---

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

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

---

NFA construction demo: 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] == ')$\) symbol: pop corresponding ( and any intervening $|$; add $\varepsilon$-transition edges for closure/or.
            if (i < M-1 && re[i+1] == '*') {
                G.addEdge(lp, i+1);
            }  
        }  
    }  
    return G;
}  ```

---

NFA construction demo: 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] == ')$\) symbol: pop corresponding ( and any intervening $|$; add $\varepsilon$-transition edges for closure/or.
            if (i < M-1 && re[i+1] == '*') {
                G.addEdge(lp, i+1);
            }  
        }  
    }  
    return G;
}  ```

---

NFA construction demo: 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] == ')$\) symbol: pop corresponding ( and any intervening $|$; add $\varepsilon$-transition edges for closure/or.
            if (i < M-1 && re[i+1] == '*') {
                G.addEdge(lp, 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.

\[ \text{NFA corresponding to the pattern } ( ( A \ast B | A C ) D ) \]

---

**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>`?

- Reluctant
- Greedy

---

**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  
do for each line  
print all words that start with uppercase letter
## Regular expressions in Java

### Validity checking
Does the input match the \texttt{re}?

### Java string library
Use \texttt{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

---

## Harvesting information

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

% java Harvester "gcg|agg"*ctg" chromosome.txt gcgcctg gcgcctg gcgcctg gcgcctg
harvest patterns from DNA

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

---

## Algorithmic complexity attacks

### Warning
Typical implementations do not guarantee performance!

Unix \texttt{grep} Java, Perl, Python

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

### SpamAssassin regular expression

% java RE "[a-z]+@[a-zA-Z]+\([a-z]\.\)+\[a-z]" spammer@x...
Not-so-regular expressions

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

```
(.+)?\1 // beriberi couscous
173.\1(11+?)\1 // 1111 1111111111111111
```

Some non-regular languages.
- Strings of the form \w \w 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: attcggaat.

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.
- 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.

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>