5.4 Pattern Matching

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
• regular expressions
• 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.
• Human genome contains triplet repeats of \texttt{CGG} or \texttt{AGG}, bracketed by \texttt{GCG} at the beginning and \texttt{CTG} at the end.
• Number of repeats is variable, and correlated with syndrome.

\begin{align*}
\text{pattern} &:\quad \texttt{GCG (CGG | AGG) *CTG} \\
\text{text} &:\quad \texttt{GCGCGGTGTGTGCAGAGAGTGTTTTAAGCTG} \texttt{GCGCGAGGCGGCTG} \texttt{GCGCGAGGCTG}
\end{align*}
Pattern matching: applications

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

Parse text files.
• Compile a Java program.
• Crawl and index the Web.
• Read in data stored in ad hoc input file format.
• Automatically create Java documentation from Javadoc comments.
A *regular expression* is a notation to specify a (possibly infinite) set of strings.

<table>
<thead>
<tr>
<th>operation</th>
<th>example RE</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>concatenation</td>
<td>AABAAB</td>
<td>AABAAB</td>
<td>every other string</td>
</tr>
<tr>
<td>or</td>
<td>AA</td>
<td>BAAB</td>
<td>AA</td>
</tr>
</tbody>
</table>
| closure        | AB*A           | AA
               | ABBBBBBBBBA | AB
               | ABABA | |
| parentheses    | A (A | B) AAB | AAAAB
               | ABAAB | every other string |
|                | (AB) *A        | A
               | ABABABABABA | AA
               | ABBA |
Regular expression shortcuts

Additional operations are often added for convenience.

**Ex.** \([A-E]+\) is shorthand for \((A|B|C|D|E)(A|B|C|D|E)^*\)

<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>at least 1</td>
<td>A(BC)+DE</td>
<td>ABCDE, ABCBCDE</td>
<td>ADE, BCDE</td>
</tr>
<tr>
<td>character classes</td>
<td>[A-Za-z][a-z]*</td>
<td>word, Capitalized</td>
<td>camelCase, 4illegal</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>
<tr>
<td>complement</td>
<td>[^AEIOU]{6}</td>
<td>RHYTHM</td>
<td>DECADE</td>
</tr>
</tbody>
</table>
Regular expression examples

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><em>(contains the trigraph spb)</em></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><em>(Social Security numbers)</em></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:rs@princeton.edu">rs@princeton.edu</a></td>
</tr>
<tr>
<td><em>(valid email addresses)</em></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><em>(valid Java identifiers)</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and plays a well-understood role in the theory of computation.
Regular expressions to the rescue

Whenever I learn a new skill I concoct elaborate fantasy scenarios where it lets me save the day.

Oh no! The killer must have followed her 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.

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

Google. Supports * for full word wildcard and | for union.
Can the average programmer learn to use REs?

Perl RE for valid RFC822 email addresses
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.
- regular expressions
- NFAs
- NFA simulation
- NFA construction
- applications
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.
• Build DFA from RE.
• Simulate DFA with text as input.

Bad news. Basic plan is infeasible (DFA may have exponential number 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. Non-deterministic finite state automata (NFA).

Basic plan.
• Build NFA from RE.
• Simulate NFA with text as input.
Nondeterministic finite-state automata

Pattern matching NFA.
- Pattern enclosed in parentheses.
- One state per pattern character (start = 0, accept = M).
- Red \( \varepsilon \)-transition (change state, but don't scan input).
- Black match transition (change state and scan to next 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

**Ex.** Is $\text{aaaabd}$ matched by NFA?

\[
\begin{array}{ccccccccc}
\text{A} & \text{A} & \text{A} & \text{A} & \text{A} & \text{A} & \text{A} & \text{A} & \text{A} \\
0 & 1 & 2 & 3 & 2 & 3 & 4 & & \\
\end{array}
\]

- **Accept state:** 11
- **Stalling sequences for NFA:** 0, 1, 2, 3, 2, 3, 4
- **Wrong guess if input is:** $\text{aaaabd}$

\[
\begin{array}{ccccccccc}
0 & 1 & 2 & 3 & 2 & 3 & 4 & & \\
\end{array}
\]

- **No way out of state:** 4
- **No way out of state:** 7

NFA corresponding to the pattern $\left( ( \text{A} \ast \text{B} | \text{A} \text{C} ) \text{D} \right)$
Ex. Is **aaaabD** matched by NFA?

Nondeterministic finite-state automata

Note: any sequence of legal transitions that ends in state 11 is a proof.
Nondeterministic finite-state automata

Ex. Is $\text{AAAAC}$ matched by NFA?

Note: this is not a complete proof!
(need to mention the infinite number of sequences involving $\varepsilon$-transitions between 2 and 3)
Nondeterminism

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

DFA. Deterministic ⇒ 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.
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. **Non**deterministic finite state automata (**NFA**).

Basic plan.
- Build **NFA** from RE.
- Simulate **NFA** with text as input.
• regular expressions
• NFAs
• **NFA simulation**
• NFA construction
• applications
NFA representation

State names. Integers from 0 to \( m \).

Match-transitions. Keep regular expression in array \( re[] \).

\( \epsilon \)-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 \( ( ( A * B | A C ) D ) \)
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?
Digraph reachability

Find all vertices reachable from a given set of vertices.

```java
public class DFS {
    private SET<Integer> marked;
    private Digraph G;

    public DFS(Digraph G) {
        this.G = G;
    }

    private void search(int v) {
        marked.add(v);
        for (int w : G.adj(v))
            if (!marked.contains(w)) search(w);
    }

    public SET<Integer> reachable(SET<Integer> s) {
        marked = new SET<Integer>();
        for (int v : s) search(v);
        return marked;
    }
}
```
NFA simulation example

Simulation of \((A * B | A C)D\) NFA for input A A B D
NFA simulation example

Simulation of \( ( ( A \ast B \mid A \ C ) \ D ) \) NFA for input \( A\ A\ B\ D \)
public boolean recognizes(String txt) {
    DFS dfs = new DFS(G);

    SET<Integer> pc = new dfs.reachable(0);

    for (int i = 0; i < txt.length(); i++) {
        SET<Integer> match = new SET<Integer>();
        for (int v : pc) {
            if (v == M) continue;
            if ((re[v] == txt.charAt(i)) || re[v] == '.')
                match.add(v+1);
        }
        pc = dfs.reachable(match);
    }
    return pc.contains(M);
}
Proposition 1. Determining whether an \( N \)-character text string is recognized by the NFA corresponding to an \( M \)-character pattern takes time proportional to \( NM \) in the worst case.

\[ \text{Pf. } \] 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 construction we consider ensures the number of edges is at most \( M \).)
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.

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

**Concatenation.** Add match-transition edge from state corresponding to letters 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 \(( ( A \ast B | A C ) D )\)
Building an NFA corresponding to an RE

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

```
G.addEdge(i, i+1);
G.addEdge(i+1, i);
G.addEdge(lp, i+1);
G.addEdge(i+1, lp);
```

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 of NFA construction](image)

NFA corresponding to the pattern \(( ( A * B | A C ) D )\)
**Goal.** Write a program to build the $\varepsilon$-transition digraph.

**Challenge.** Need to remember left parentheses to implement closure and or; need to remember $|$ to implement or.

**Solution.** Maintain a stack.
- Left parenthesis: push onto stack.
- $|$ symbol: push onto stack.
- Right parenthesis: add edges for closure and or.

NFA corresponding to the pattern $\left( ( A \ast B \mid A \ C ) \ D \right)$
NFA construction: example

Building the NFA corresponding to \(( ( A * B | A C ) D )\)
NFA construction: example

Building the NFA corresponding to \(( ( A \ast B \mid A C ) D )\)
public NFA(String regexp) {
    Stack<Integer> ops = new Stack<Integer>();
    this.re = re.toCharArray();
    M = re.length;
    G = new Digraph(M+1);
    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);
    }
}
NFA construction: analysis

**Proposition 2.** Building the NFA corresponding to an $M$-character pattern takes time and space proportional to $M$ in the worst case.

**Pf.** For each of the $M$ characters in the pattern, we add one or two $\varepsilon$-transitions and perhaps execute one or two stack operations.
- regular expressions
- NFAs
- NFA simulation
- NFA construction
- applications
**Generalized regular expression print**

**Grep.** Takes a pattern as a command-line argument and prints the lines from standard input having some substring that is matched by the pattern.

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

**Bottom line.** Worst-case for grep (proportional to MN) is the same as for elementary exact substring match.
Typical grep application

Crossword puzzle

% grep s..ict.. words.txt
constrictor
strict
strict

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

dictionary (standard in UNIX) also on booksite
Industrial-strength grep implementation

To complete the implementation:

- Add character classes.
- Handling 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>`?

```plaintext
<blink>text</blink> some text <blink>more text</blink>
```

reluctant  reluctant  reluctant

greedy
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 '^[qwertuyiop]*[zxcvbnm]*$' dict.txt | egrep '.............'
```

PERL. Practical Extraction and Report Language.

```bash
% 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-Za-z][a-z]*$/' dict.txt      print all uppercase words
                                 do for each line
Regular expressions in Java

Validity checking. Does the input match the regexp?
Java string library. Use `input.matches(regexp)` for basic RE matching.

```java
class Validate {
    public static void main(String[] args) {
        String regexp = args[0];
        String input  = args[1];
        System.out.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(cgg|agg)*ctg" chromosomeX.txt
  gcgcggcggcggcggcggctg
  gcgcggcggcggcggcggctg
  gcgcggcggcggcggcggcggctg

harvest patterns from DNA

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

harvest links from website
Harvesting information

RE pattern matching is implemented in Java's `Pattern` and `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

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

SpamAssassin regular expression.

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

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

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

Some non-regular languages.
• Set of strings of the form \(ww\) for some string \(w\): beriberi.
• Set of bitstrings with an equal number of 0s and 1s: 01110100.
• Set of Watson-Crick complemented palindromes: atttcggaaat.

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 exact pattern matching 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 and REs have limitations.

You. Practical application of core CS principles.

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