5.4 REGULAR EXPRESSIONS

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications

Review: substring search

- Knuth-Morris-Pratt (deterministic finite automaton)
- Boyer-Moore (skip-ahead heuristic)
- Rabin-Karp (modular hashing)

Deterministic Finite Automaton

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Trick question

Which search pattern does this DFA correspond to?

Either an A or a B followed by a C.

Every string corresponds to a DFA, but not every DFA corresponds to a string

Every DFA corresponds to a pattern called a regular expression (strings are a simple type of regular expression)
Finding interesting words

$ egrep '^[a-j]{8,}$' /usr/share/dict/words
acidified
beachhead
beheaded
headache

$ egrep '^[qwertyuiop]{10,}$' /usr/share/dict/words
perpetuity
proprietor
repertoire
typewriter

Subtle differences in syntax

Google allows a limited form of regular expression search

![Google search result](image)

XKCD t-shirt

![XKCD t-shirt](image)

Genomics

- Fragile X syndrome is a common cause of mental retardation.
- A human’s genome is a string.
- It contains triplet repeats of **GCC** or **AGG**, bracketed by **GCC** at the beginning and **CTG** at the end.
- Number of repeats is variable and is correlated to syndrome.

| pattern | GCC(AGG|CGG)CTG |
|---------|--------------|
| text    | GCCGCCCTGTCGCCAGAGAGGTGCTTAAGCTGCCGCCAGGCCGTGCCGCCAGGCTG |
Syntax highlighting

```java
public class Skf {
    private Digraph G; // 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;
        N = regexp.length();
        Stack<Integer> sp = new Stack<Integer>();
        G = new Digraph(N);
        ...
    }
}
```

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: https://chromium.googlesource.com/chromium.git

**Language**

- Any language

**File Path**

- file://code.google.com/p/chromium

**Class**

- class: HashMap

**Function**

- function: toSring

**License**

- Any license

**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. PROSITE is complemented by ProRule, a collection of rules based on profiles and patterns, which increases the discriminatory power of PROSITE and by providing additional information about functionality and/or structurally critical amino acids [More...]

**Release 20.13 of 26-Mar-2013 contains 1716 documentation entries, 1360 patterns, 1112 profiles and 1112 ProRule.**

Even more 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).

  ```text
  ...
  ```

- **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.

  ```text
  ...
  ```

Even more 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).

  ```text
  ...
  ```

- **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.

  ```text
  ...
  ```
Regular expressions

A regular expression is a notation to specify a set of strings.

![Regular expression: operator precedence](image)

<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 BAAB</td>
</tr>
<tr>
<td>star (aka closure)</td>
<td>AB*A</td>
<td>AA ABBBBBBBBA</td>
<td>AB ABABA</td>
</tr>
<tr>
<td>parentheses</td>
<td>A(A</td>
<td>B)AAB</td>
<td>AAAAAAB AABAAB</td>
</tr>
<tr>
<td>(AB)*A</td>
<td>A ABAABABABA A ABBA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Regular expression: quiz 1

Which one of the following strings is not matched by the regular expression \((AB | C \ast D) \ast\)?

A. A B A B A B
B. C D 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](image)

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,.U.</td>
<td>CUMULUS JUGILUM</td>
<td>SUCCUBUS TUMILUTOUS</td>
</tr>
<tr>
<td>character class</td>
<td>[A-Za-z][A-Za-z]+</td>
<td>word Capitalized</td>
<td>camelCase 414legal</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>111111111 366-54-11</td>
</tr>
</tbody>
</table>

Note. These operations are useful but not essential.

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

Simplify the following regular expression over the alphabet \( \{A, B\} \):
\[(B \mid A^+B^* \mid BAA^*)^*\]

---

Exercise

Simplify the following regular expression over the alphabet \( \{A, B\} \):
\[(B \mid A^+B^* \mid BAA^*)^* \\
\text{matches 'A'} \\
\equiv (B \mid A)^* \\
\equiv .^*\]

---

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;SP.*&quot; (substring search)</td>
<td>RASPBERRY CRISPBREAD</td>
<td>SUBSPACE SUBSPECIES</td>
</tr>
<tr>
<td>[0-9]{3}-[0-9]{2}-[0-9]{4} (U. S. Social Security numbers)</td>
<td>166-11-4413 166-45-1111</td>
<td>11-5555555 8675309</td>
</tr>
<tr>
<td>[a-z]+@[a-z]+(.)+{edu,com} (simplified email addresses)</td>
<td><a href="mailto:wayne@princeton.edu">wayne@princeton.edu</a> <a href="mailto:rsdy@princeton.edu">rsdy@princeton.edu</a></td>
<td>spam#nowhere</td>
</tr>
<tr>
<td>[$_A-Za-_Z[0-9]* (Java identifiers)</td>
<td>1dent3 PatternMatcher 3a 1dent3</td>
<td></td>
</tr>
</tbody>
</table>

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

Exercise

Write a regular expression that matches strings of even length that start with an 'A' and contain a 'B'.
Exercise

Write a regular expression that matches strings of even length that start with an 'A' and contain a 'B'.

Case 1: A and B are separated by an even number of characters
A (.)* B (.)*

Case 2: A and B are separated by an odd number of characters
A (.)* . B . (.)*

Put it together:
A (.)*B(.)* | A(.)* . B (.)*

Optionally simplify:
A (.)* (B | .B) (.)*

You can go crazy with regular expressions

Perl RE for valid RFC822 email addresses

http://algs4.cs.princeton.edu

http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html

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

Bottom line. REs are amazingly powerful and expressive, but using them in applications can be amazingly complex and error-prone.

5.4 Regular Expressions

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.

---

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.

---

Nondeterministic finite-state automata

**Regular-expression-matching NFA.**
- We assume RE enclosed in parentheses.
- One state per RE character (start = 0, accept = M).
- Match transition (change state and scan to next text char).
- Dashed ε-transition (change state, but don’t scan text).
- 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.

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

Q. Is A 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 * B | A C ) D )

Q. Is A A A C matched by NFA?
A. No, because no sequence of legal transitions ends in state 11.

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

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

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

NFA. Nondeterministic ⇒ 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 vs. quantum computers

How are nondeterministic finite automata different from quantum computers?

Quantum computers are actually, physically nondeterministic.

With NFAs, we’re just pretending. We can simulate them efficiently with regular computers (Turing machines). We can’t do that with quantum computers (as far as we know).

NFA representation

State names. Integers from 0 to $M$.

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

$\varepsilon$-transitions. Store in a digraph $G$.

0→1, 1→2, 2→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 the first $i$ text characters.

one step in simulating an NFA

Q. How to perform reachability?
A. DFS with multiple source vertices.
NFA simulation demo

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

NFA corresponding to the pattern \( ( \{ A \neq B \} \cup A \cap C \cup D) \)

Before reading any input characters:
- Find states reachable by \( \varepsilon \)-transitions from start state

**set of states reachable via \( \varepsilon \)-transitions from start**

Before reading any input characters:
- Find states reachable by \( \varepsilon \)-transitions from start state

**set of states reachable via \( \varepsilon \)-transitions from start:** \{ 0, 1, 2, 3, 4, 6 \}
Before reading any input characters:
- Find states reachable by $\varepsilon$-transitions from start state

set of states reachable via $\varepsilon$-transitions from start: \{ 0, 1, 2, 3, 4, 6 \}

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable after matching A: \{ 3, 7 \}

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable via $\varepsilon$-transitions after matching A
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by ε-transitions

set of states reachable via ε-transitions after matching A : { 2, 3, 4, 7 }

set of states reachable after matching A A

set of states reachable after matching A A : { 3 }
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by \( \varepsilon \)-transitions

set of states reachable via \( \varepsilon \)-transitions after matching A A

set of states reachable via \( \varepsilon \)-transitions after matching A

set of states reachable after matching A A B
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable after matching A A B : { 5 }

set of states reachable via $\varepsilon$-transitions after matching A A B : { 5, 8, 9 }
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by ε-transitions

set of states reachable after matching A A B D

match D transition

set of states reachable after matching A A B D : { 10 }

NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by ε-transitions

set of states reachable via ε-transitions after matching A A B D

set of states reachable via ε-transitions after matching A A B D : { 10, 11 }
When no more input characters:
- Accept if any state reachable is an accept state.
- Reject otherwise.

set of states reachable \( \{ 10, 11 \} \)

**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 \( M \) \( N \) 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 \( \epsilon \)-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 \( \epsilon \)-transition edge from parentheses to next state.

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

Single-character closure  

Closure expression

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

Building an NFA corresponding to an RE

Building an NFA corresponding to an RE
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.

**Parentheses.** Add $\epsilon$-transition edge from parentheses to next state.

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

**2-way or.** Add two $\epsilon$-transition edges for each $|$ operator.

---

Regular expression: quiz 4

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

A. Remove $\epsilon$-transition edge $1\rightarrow7$.

B. Remove $\epsilon$-transition edge $7\rightarrow1$.

C. Remove $\epsilon$-transition edges $1\rightarrow7$ and $7\rightarrow1$.

D. I don’t know.

---

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>`?

---
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 "^[a-z|qnsa]*$" words.txt | egrep "............."
typewritten

- Built in to many languages: awk, Perl, PHP, Python, JavaScript, ...

% 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
print all words that start with uppercase letter
do for each line

Harvesting information

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

% java Harvester "ggc(gcggagcgpqcgqctg" chromosomeX.txt
ggcctg
ggcctg
ggcgcgcgcgcggppqcgqctg
harvest patterns from DNA
harvest links from website

% java Harvester "http://\w+/\w+/\w+" http://www.cs.princeton.edu
http://www.w3.org
http://www.cs.princeton.edu
http://dnupal.org
http://csguide.cs.princeton.edu
http://www.cs.princeton.edu
http://www.princeton.edu

Harvesting information

RE pattern matching is implemented in Java’s java.util.regex.Pattern and java.util.regex.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];
        String input = args[1];

        Pattern pattern = Pattern.compile(regexp);
        Matcher matcher = pattern.matcher(input);
        while (matcher.find()) {
            StdOut.println(matcher.group());
        }
    }
}
```

% java Harvester "[a-zA-z]+[a-zA-z-]*" (diet123)
legal java identifier
typewritten

% java Harvester "^[a-zA-z][a-zA-z-]*\"" rulcs.princeton.edu
valid email address (simplified)

% java Harvester "[0-9]+[0-9]+[0-9]+[0-9]+[0-9]+" 166-11-4431
Social Security number

% java Harvester "([a-z]+(\.[a-z]+)\.)+(edu|com)" rdulcs.princeton.edu
true
Algorithmic complexity attacks

Warning. Typical implementations do not guarantee performance!

Unix grep, Java, Perl, Python

% java Validate "(a{4})b" aaaaaaaaaaaaaaaaaaaaaaaaa 4.6 seconds
% java Validate "(a{4})b" aaaaaaaaaaaaaaaaaaaaaaaaa 3.7 seconds
% java Validate "(a{4})b" aaaaaaaaaaaaaaaaaaaaaaaaa 9.7 seconds
% java Validate "(a{4})b" aaaaaaaaaaaaaaaaaaaaaaaaa 23.2 seconds
% java Validate "(a{4})b" aaaaaaaaaaaaaaaaaaaaaaaaa 62.2 seconds
% java Validate "(a{4})b" aaaaaaaaaaaaaaaaaaaaaaaaa 161.6 seconds

SpamAssassin regular expression.

% java RE "[a-z][0-9][a-z]-([a-z].|\.)+[a-z]" spammer0e

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

Harvesting information in Java

RE pattern matching is implemented in Java’s java.util.regex.Pattern and java.util.regex.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.readLine();
        Pattern pattern = Pattern.compile(regexp);
        Matcher matcher = pattern.matcher(input);
        while (matcher.find()) {
            StdOut.println(matcher.group());
        }
    }
}
```

Not-so-regular expressions

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

```
(\.)\1 // berberi couscous
17|/\1(11-7)\1+ // 1111 11111111111111111111
```

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: atttcggaat.

Remark. Pattern matching with back-references is intractable.

Regular expressions in context

Regexes are powerful, but far less powerful than Java programs.

Compiler. A program that translates a program to machine code.
- KMP string ⇒ DFA.
- grep RE ⇒ NFA.
- javac Java language ⇒ Java byte code.
Algorithmic complexity attacks

Warning. Typical implementations do not guarantee performance!

```
unix grep, java, perl, python
```

<table>
<thead>
<tr>
<th>Command</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaac</td>
</tr>
<tr>
<td>% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaac</td>
</tr>
<tr>
<td>% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaac</td>
</tr>
<tr>
<td>% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaac</td>
</tr>
<tr>
<td>% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaac</td>
</tr>
<tr>
<td>% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaac</td>
</tr>
</tbody>
</table>

SpamAssassin regular expression.

```
% 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.

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.