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.
- Human genome contains triplet repeats of CGG or AGG, bracketed by GCG at the beginning and CTG at the end.
- Number of repeats is variable, and correlated with syndrome.

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.

---

### Regular expressions to the rescue

To illustrate the power of regular expressions, consider the following 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</td>
<td>SUBSPACE</td>
</tr>
<tr>
<td></td>
<td>CRISPBREAD</td>
<td>SUBSPECIES</td>
</tr>
<tr>
<td>[0-9]{3}-[0-9]{2}-[0-9]{4}</td>
<td>166-11-4433</td>
<td>11-5555555</td>
</tr>
<tr>
<td></td>
<td>166-45-1111</td>
<td>8675309</td>
</tr>
<tr>
<td>[a-z]+@[a-z]+.+[edu]+</td>
<td><a href="mailto:wayne@princeton.edu">wayne@princeton.edu</a></td>
<td>spam@nowhere</td>
</tr>
<tr>
<td></td>
<td><a href="mailto:rs@princeton.edu">rs@princeton.edu</a></td>
<td></td>
</tr>
<tr>
<td>[6-A-Za-z][6-A-Za-z0-9]*</td>
<td>ident3</td>
<td>3a</td>
</tr>
<tr>
<td>[valid Java identifiers]</td>
<td>PatternMatcher</td>
<td>ident#3</td>
</tr>
</tbody>
</table>

and plays a well-understood role in the theory of computation.

---

### Regular expression shortcuts

Additional operations are often added for convenience.

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

| operation                   | example RE               | matches                  | does not match           |
|-----------------------------|--------------------------|--------------------------|
| wildcard                    | .\.\.\.                   | CUMULUS                  |
|                            | JUGULUM                  | SUCCUBUS                 |
|                            | TUMULTUOUS               |                          |
| at least \(k\)             | \((A(BC))^+\)DE           | ABCDE                    |
|                            | ABCBCDE                  | ADE                      |
|                            | BCDE                     |                          |
| character classes \([A-Za-z][a-z]+\)\* | word            | Capitalized             |
|                            | camelCase                | 4illegal                 |
| exactly \(k\)              | \([0-9]{5}-[0-9]{4}\)    | 08540-1321               |
|                            | 19072-5541               | 111111111               |
|                            | 166-54-111               |                          |
| complement \([^AEIOU]+\)\*  | RHYTHM                   | DECADE                   |

---

### Regular expressions to the rescue

On the eve of a murder trial, the defense attorney presented a smoking gun.

**But to find them we’d have to search through 200 MB of emails looking for something formatted like an address!**

**It’s hopeless!**

---

[http://xkcd.com/208](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 TV viewer learn to use REs?

TiVo. WishList has very limited pattern matching.

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.
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 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. Nondeterministic finite state automata (NFA).

Basic plan. [apply Kleene's theorem]
- Build NFA from RE.
- Simulate NFA with text as input.

Q. What exactly is an NFA?

Duality

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.

Good news. Basic plan works in theory.
Bad news. Basic plan fails in practice.
Nondeterministic finite-state automata

Regular-expression-matching NFA.

- RE enclosed in parentheses.
- One state per RE character (start = 0, accept = M).
- Red ε-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.

Q. Is AAAABD matched by NFA?
A. Yes, because some sequence of legal transitions ends in state 11.

Q. Is AAAAC 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 \(\Rightarrow\) exactly one applicable transition.

NFA. Nondeterministic \(\Rightarrow\) can be several applicable transitions; need to select the right one!

Q. How to simulate NFA?
A. Systematically consider all possible transition sequences.

Partial proof of Kleene’s theorem (RE \(\Rightarrow\) DFA)

For any RE, there exists a DFA that recognizes the same set of strings.
• Given an RE, construct an NFA (stay tuned)
• Given an NFA, construct a DFA (see construction below)

To construct a DFA that recognizes the same language as a given NFA:
• create a DFA state for every set of NFA states
• systematically infer transitions

Problem: \(2^N\) states in DFA

Insight: Need to consider all possible transitions to simulate NFA

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. How to construct NFA and how to efficiently simulate NFA?
NFA representation

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

Number of symbols in RE

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

\( \varepsilon \)-transitions. Store in a digraph \( G \).

- \( 0 \to 1, 1 \to 2, 1 \to 6, 2 \to 3, 3 \to 2, 3 \to 4, 5 \to 8, 8 \to 9, 10 \to 11 \)

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

NFA simulation example

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?

NFA simulation example (continued)

Simulation of \( ( ( A \ast B \mid A C ) D ) \) NFA for input \( A \ A \ B \ D \)
Digraph reachability

Recall Section 4.2. Find all vertices reachable from a given set of vertices.

public class DirectedDFS
DirectedDFS(Digraph G, int s)
    find vertices reachable from s
DirectedDFS(Digraph G, 
    Iterable<Integer> sources)
    find vertices reachable from sources
boolean marked(int v)
    is v reachable from source(s)?

NFA simulation: Java implementation

public class NFA
    private char[] re; // match transitions
    private Digraph G; // epsilon transitions
    private int M; // number of states

    public NFA(String regexp)
    { 
        M  = regexp.length();
        re = regexp.toCharArray();
        G = buildEpsilonTransitionsGraph();
    }

    public boolean recognizes(String txt)
    {  /* see next slide */  }
}

NFA simulation: Java implementation

public boolean recognises(String txt)
{   Bag<Integer> pc = new Bag<Integer>();   DirectedDFS dfs = new DirectedDFS(G, 0); ... 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 string 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 \( \epsilon \)-transitions.
(The NFA construction we consider ensures the number of edges in \( G \approx 3M \).)
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.** ( ) . * |

- **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 \( \ast \) operator.

![Diagram showing single-character closure and closure expression](image)

\[ \begin{align*}
G & . addEdge(i, i+1); \\
G & . addEdge(i+1, i);
\end{align*} \]

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

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

**Goal.** Write a program to build the \( \varepsilon \)-transition digraph.

**Challenges.** Need to remember left parentheses to implement closure and or; also need to remember | to implement or.

**Solution.** Maintain a stack.
- ( symbol: push ( onto stack.
- ) symbol: push ) onto stack.
- ) symbol: pop corresponding ( and possibly intervening |;
  - add \( \varepsilon \)-transition edges for closure/or.

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

Building the NFA corresponding to \((A \ast B | A C) D\)
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.
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 regexp = "(.*) + args[0] + ".*";
        NFA nfa = new NFA(regexp);
        while (StdIn.hasNextLine()) {
            String line = StdIn.readLine();
            if (nfa.recognizes(line))
                StdOut.println(line);
        }
    }
}
```

**Bottom line.** Worst-case for grep (proportional to $M N$) is the same as for elementary exact substring match.

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
```

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

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

Typical grep application: crossword puzzles

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constrictor
stricter
stricture
```

```
% grep 's..ict..' words.txt
constric...
Regular expressions in Java

Validity checking. Does the input match the regexp?

Java string library. Use input.matches(regexp) 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(g{0,1}g{1})*ctg" chromosomeX.txt
     gcgcggcggcggcggcggctg
ggcgtg
ggcgtg
ggcgcggcggcggaggcggaggcggctg
% 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|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 1.6 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 3.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 9.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 23.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 62.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 161.6 seconds
```

SpamAssassin regular expression.

- 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 $w^2$ 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.

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.

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 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.