

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 `GCG (CGG | AGG) *CTG`

text `GCGGCGTGTGTGCGAGAGAGTGGGTTTAAAGCTGGCGCGGAGGCGGCTGGCGCGGAGGCTG`

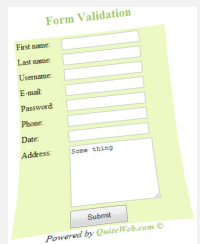
- ▶ regular expressions

- ▶ NFAs
- ▶ NFA simulation
- ▶ NFA construction
- ▶ applications

Pattern matching: applications

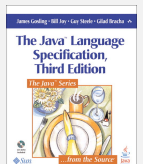
Test if a string matches some pattern.

- Process natural language.
- Scan for virus signatures.
- 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 (possibly infinite) set of strings.

↑
a "language"

operation	example RE	matches	does not match
concatenation	AABAAB	AABAAB	every other string
or	AA BAAB	AA BAAB	every other string
closure	AB*A	AA ABBBBBBBA	AB ABABA
parentheses	A (A B) AAB	AAAAB ABAAB	every other string
	(AB) *A	A ABABABABABA	AA ABBA

5

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	.U.U.U.	CUMULUS JUGULUM	SUCCUBUS TUMULTUOUS
character class	[A-Za-z] [a-z] *	word Capitalized	camelCase 4illegal
at least 1	A (BC) +DE	ABCDE ABCBCDE	ADE BCDE
exactly k	[0-9] {5} - [0-9] {4}	08540-1321 19072-5541	11111111 166-54-111
complement	[^AEIOU] {6}	RHYTHM	DECADE

6

Regular expression examples

Notation is surprisingly expressive

regular expression	matches	does not match
. *SPB. * <i>(substring search)</i>	RASPBERRY CRISPBREAD	SUBSPACE SUBSPECIES
[0-9] {3} - [0-9] {2} - [0-9] {4} <i>(Social Security numbers)</i>	166-11-4433 166-45-1111	11-5555555 8675309
[a-z] +@ ([a-z] +\.) + (edu com) <i>(email addresses)</i>	wayne@princeton.edu rs@princeton.edu	spam@nowhere
[\$_A-Za-z] [\$_A-Za-z0-9] * <i>(Java identifiers)</i>	ident3 PatternMatcher	3a ident#3

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

7

Can the average web surfer learn to use REs?

Google. Supports ***** for full word wildcard and **|** for union.



8

- › regular expressions
- › NFAs
- › NFA simulation
- › NFA construction
- › applications

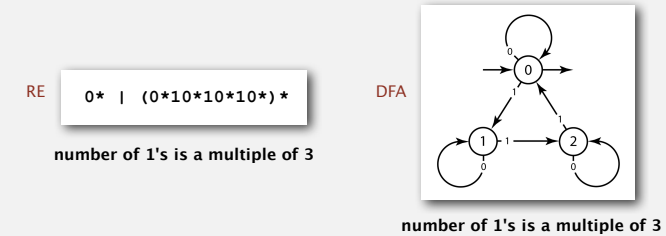
Duality

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.



Stephen Kleene
Princeton Ph.D. 1934

Pattern matching implementation: basic plan (first attempt)

Overview is the same as for KMP.

- No backup in text input stream.
- Linear-time guarantee.



Ken Thompson
Turing Award '83

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).

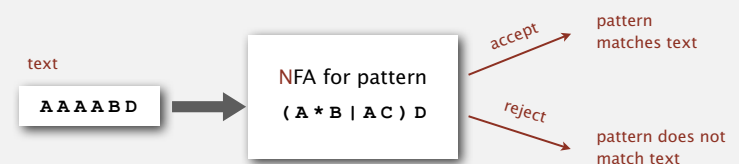


Ken Thompson
Turing Award '83

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?

Nondeterministic finite-state automata

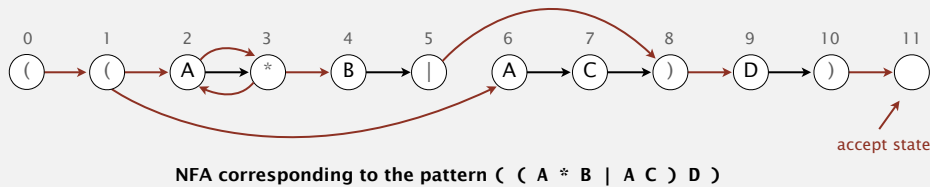
Regular-expression-matching NFA.

- RE enclosed in parentheses.
- One state per RE character (start = 0, accept = M).
- Red ϵ -transition (change state, but don't scan text).
- Black match transition (change state and scan to next text char).
- Accept if **any** sequence of transitions ends in accept state.

after scanning all text characters

Nondeterminism.

- One view: machine can guess the proper sequence of state transitions.
- Another view: sequence is a proof that the machine accepts the text.

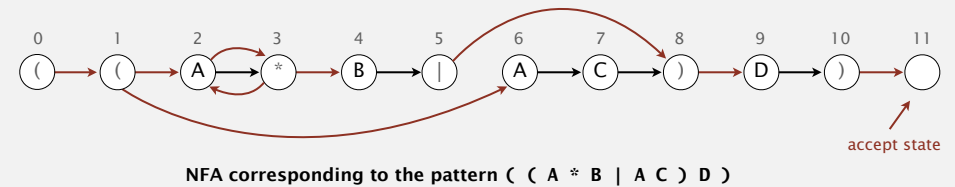
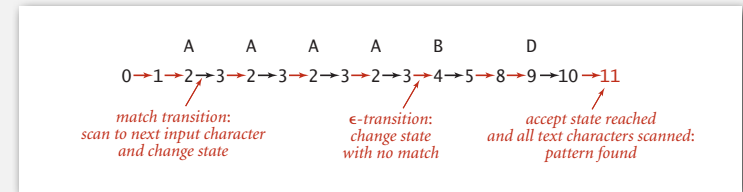


17

Nondeterministic finite-state automata

Q. Is AAAABD matched by NFA?

A. Yes, because **some** sequence of legal transitions ends in state 11.

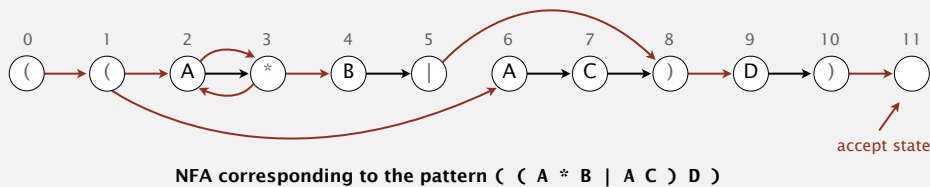
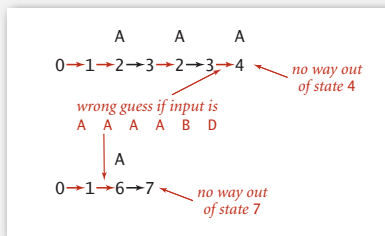


18

Nondeterministic finite-state automata

Q. Is AAAABD matched by NFA?

A. Yes, because **some** sequence of legal transitions ends in state 11.
[even though some sequences end in wrong state or stall]

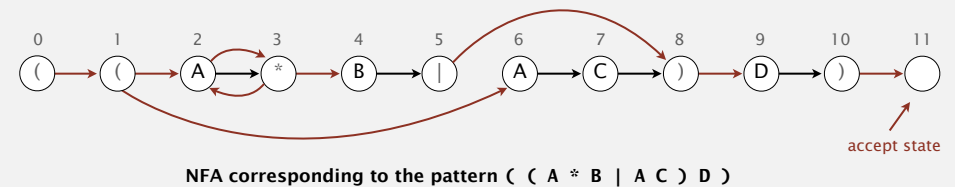
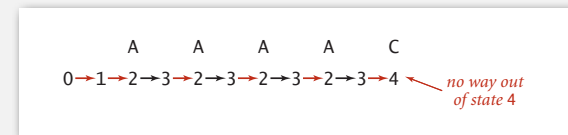


19

Nondeterministic finite-state automata

Q. Is AAAC matched by NFA?

A. No, because **no** sequence of legal transitions ends in state 11.
[but need to argue about all possible sequences]



20

Nondeterminism

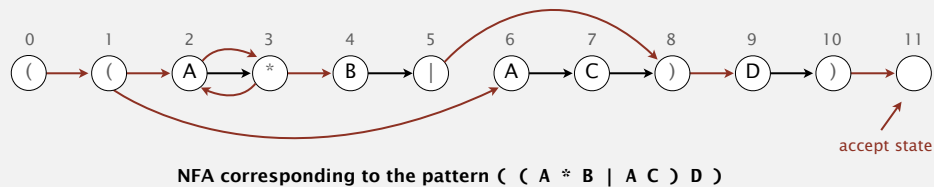
Q. How to determine whether a string is matched by an automaton?

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.



21

› regular expressions

› NFAs

› NFA simulation

› NFA construction

› applications

22

NFA representation

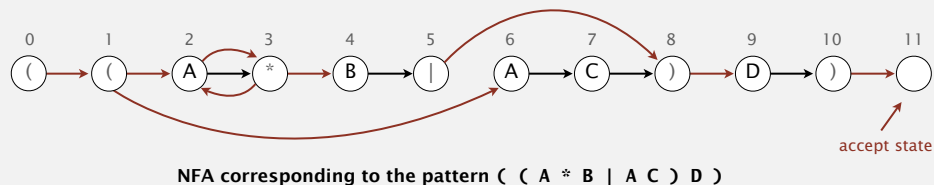
State names. Integers from 0 to M .

number of symbols in RE

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

ϵ -transitions. Store in a digraph G .

- $0 \rightarrow 1, 1 \rightarrow 2, 1 \rightarrow 6, 2 \rightarrow 3, 3 \rightarrow 2, 3 \rightarrow 4, 5 \rightarrow 8, 8 \rightarrow 9, 10 \rightarrow 11$

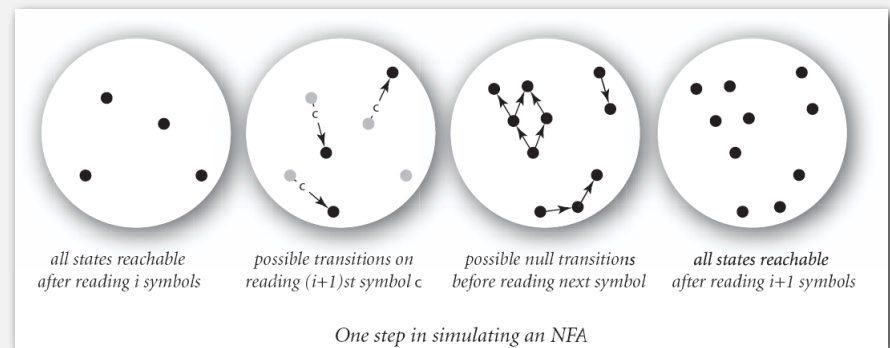


23

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.



Q. How to perform reachability?

24

Digraph reachability. Find all vertices reachable from a given source or set of vertices.

recall Section 4.2

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

Solution. Run DFS from each source, without unmarking vertices.

Performance. Runs in time proportional to $E + V$.

```
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 */ }
}
```

stay tuned

```
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 i = 0; i < txt.length(); i++)
    {
        Bag<Integer> match = new Bag<Integer>();
        for (int v : pc)
        {
            if (v == M) continue;
            if ((re[v] == txt.charAt(i)) || re[v] == '.')
                match.add(v+1);
        }

        dfs = new DirectedDFS(G, match);
        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;
}
```

states reachable from start by ϵ -transitions

states reachable after scanning past `txt.charAt(i)`

follow ϵ -transitions

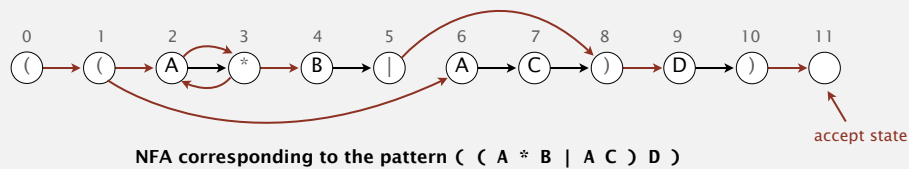
accept if can end in state M

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 ϵ -transitions.

(The NFA construction we consider ensures the number of edges $\leq 3M$.)



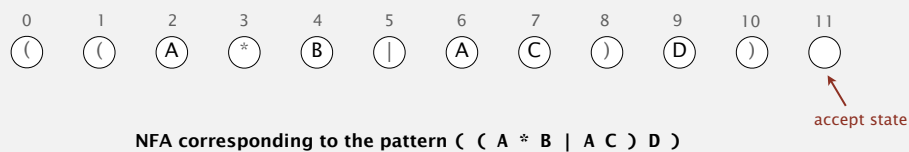
29

- › regular expressions
- › NFAs
- › NFA simulation
- › **NFA construction**
- › applications

30

Building an NFA corresponding to an RE

States. Include a state for each symbol in the RE, plus an accept state.



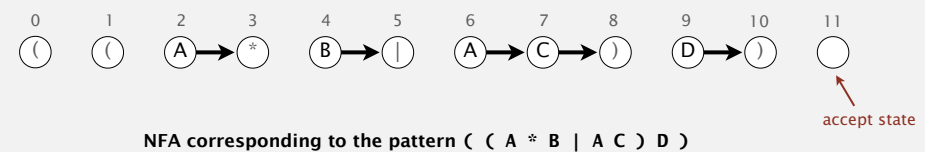
31

Building an NFA corresponding to an RE

Concatenation. Add match-transition edge from state corresponding to characters in the alphabet to next state.

Alphabet. $A B C D$

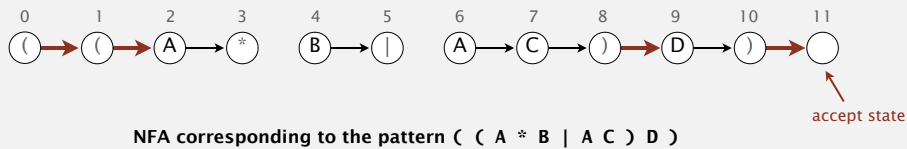
Metacharacters. $() . * |$



32

Building an NFA corresponding to an RE

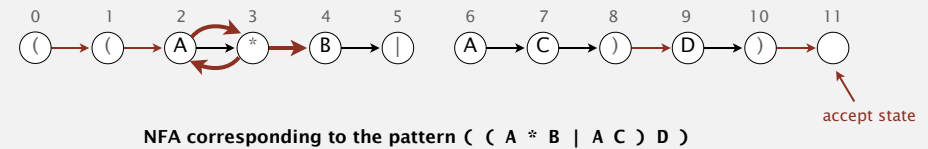
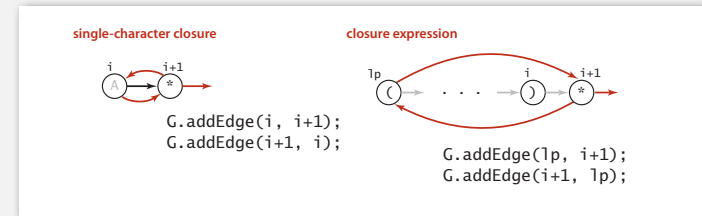
Parentheses. Add ϵ -transition edge from parentheses to next state.



33

Building an NFA corresponding to an RE

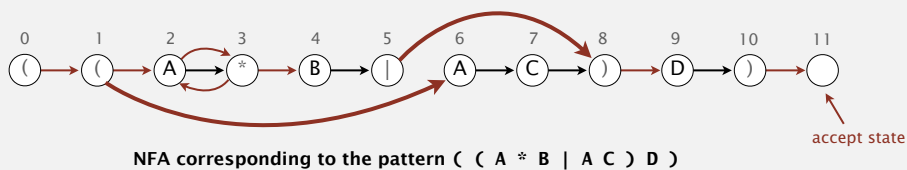
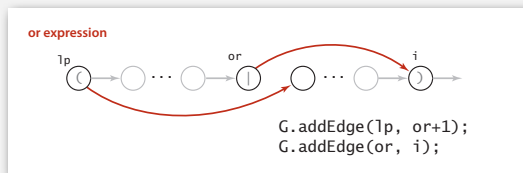
Closure. Add three ϵ -transition edges for each * operator.



34

Building an NFA corresponding to an RE

Or. Add two ϵ -transition edges for each | operator.



35

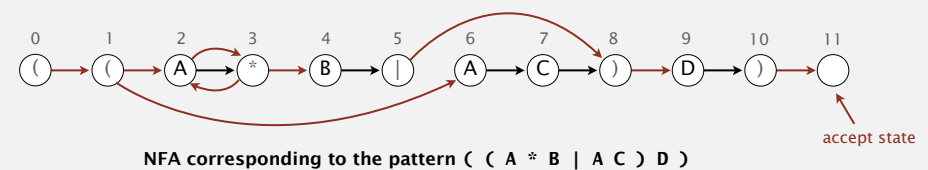
NFA construction: implementation

Goal. Write a program to build the ϵ -transition digraph.

Challenges. Need to remember left parentheses to implement closure and or; 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 ϵ -transition edges for closure/or.



36

```
private Digraph buildEpsilonTransitionGraph() {
    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] == ')')
            G.addEdge(i, i+1);
    }
    return G;
}
```

left parentheses and |

or

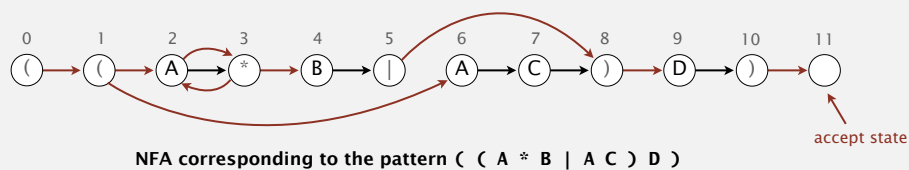
closure
(needs 1-character lookahead)

metasymbols

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 ϵ -transitions and execute at most two stack operations.



- ▶ regular expressions
- ▶ NFAs
- ▶ NFA simulation
- ▶ NFA construction
- ▶ applications

Generalized regular expression print

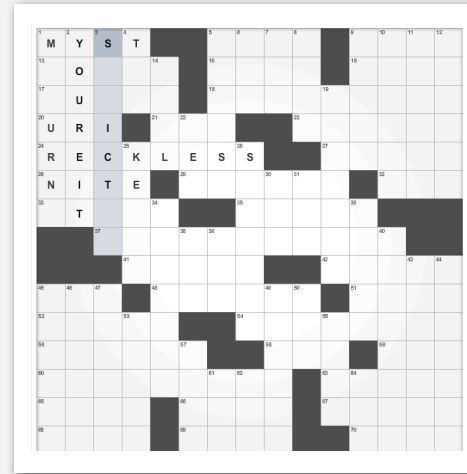
Grep. Take a RE as a command-line argument and print the lines from standard input having some substring that is matched by the RE.

```
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);
        }
    }
}
```

contains RE
as a substring

Bottom line. Worst-case for grep (proportional to MN) is the same as for brute-force substring search.

Typical grep application: crossword puzzles



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

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

dictionary
(standard in Unix)
also on booksite

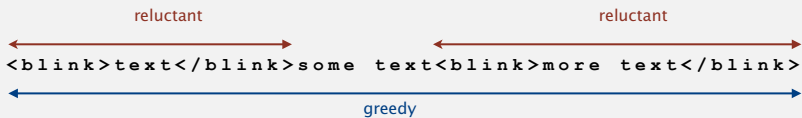
Industrial-strength grep implementation

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>?`



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.

```
% grep 'NEWLINE' */*.java
```

print all lines containing NEWLINE which occurs in any file with a .java extension

```
% egrep '^[qwertyuiop]*[zxcvbnm]*$' words.txt | egrep '.....'
typewritten
```

PERL. Practical Extraction and Report Language.

```
% 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

Validity checking. Does the input match the `regexp`?

Java string library. Use `input.matches(regexp)` for basic RE matching.

```
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
```

← legal Java identifier
← valid email address (simplified)
← Social Security number

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

```
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgcgcgcgcgcgcgcgcgctg
gcgctg
gcgctg
gcgcgcgcgcgcgaggcgaggcgctg
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
```

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

```
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()`

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

← Unix `grep`, Java, Perl

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

SpamAssassin regular expression.

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

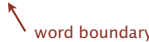
- Takes exponential time on pathological email addresses.
- Troublemaker can use such addresses to DOS a mail server.

Not-so-regular expressions

Back-references.

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

```
% java Harvester "\b(.+)\1\b" words.txt
beriberi
couscous
```

 word boundary

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: `atttcggaat`.

Remark. Pattern matching with back-references is intractable.

49

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.

	KMP	grep	Java
pattern	string	RE	program
parser	unnecessary	check if legal	check if legal
compiler output	DFA	NFA	byte code
simulator	DFA simulator	NFA simulator	JVM

50

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 computer science principles.

Example of essential paradigm in computer science.

- Build intermediate abstractions.
- Pick the right ones!
- Solve important practical problems.

51