Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

5.4 REGULAR EXPRESSIONS

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

http://algs4.cs.princeton.edu

5.4 REGULAR EXPRESSIONS

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

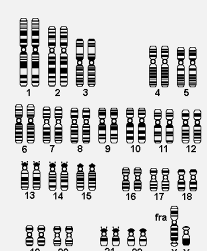
Pattern matching

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 GCG at the beginning and CTG at the end.
- Number of repeats is variable and is correlated to syndrome.



pattern GCG(CGG|AGG)*CTG

text GCGGCGTGTGTGCGAGAGAGTGGGTTTAAAGCTGGCGCGGAGGCGGCTGGCGCGGAGGCTG

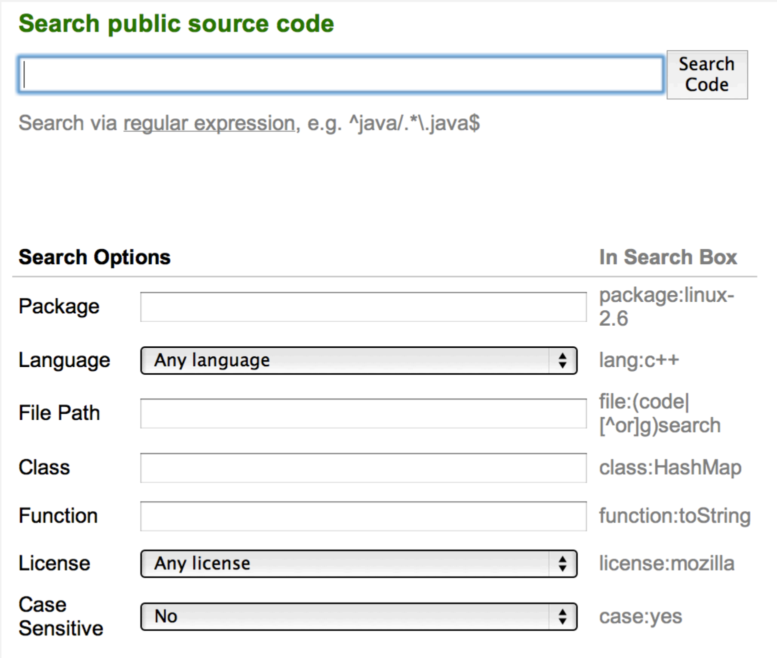
Syntax highlighting

```
/*************************************************************************
 *  Compilation:  javac NFA.java
 *  Execution:    java NFA regexp text
 *  Dependencies: Stack.java Bag.java Digraph.java DirectedDFS.java
 *
 *  % java NFA "(A*B|AC)D" AAAABD
 *  true
 *
 *  % java NFA "(A*B|AC)D" AAAAC
 *  false
 *
 *************************************************************************/

public class NFA
{
    private Digraph G;         // digraph of epsilon transitions
    private String regexp;     // regular expression
    private int M;             // 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>();
        G = new Digraph(M+1);
        ...
```

GNU source-highlight 3.1.4

input	output
Ada	HTML
Asm	XHTML
Applescript	LATEX
Awk	MediaWiki
Bat	ODF
Bib	TEXINFO
Bison	ANSI
C/C++	DocBook
C#	
Cobol	
Caml	
Changelog	
Css	
D	
Erlang	
Flex	
Fortran	
GLSL	
Haskell	
Html	
Java	
Javalog	
Javascript	
Latex	
Lisp	
Lua	
⋮	

Google code search



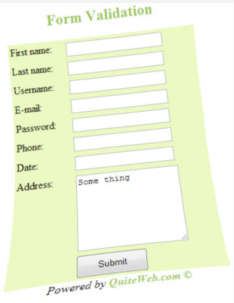
http://code.google.com/p/chromium/source/search

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).

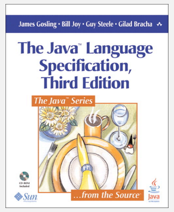
...



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 expressions

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

possibly infinite

operation	order	example RE	matches	does not match
concatenation	3	AABAAB	AABAAB	*every other string*
or	4	AA | BAAB	AA BAAB	*every other string*
closure	2	AB*A	AA ABBBBBBBBA	AB ABABA
parentheses	1	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.

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	111111111 166-54-111

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

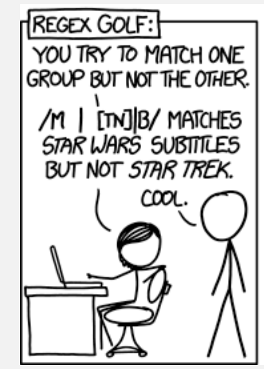
Regular expression examples

RE notation is surprisingly expressive.

regular expression	matches	does not match
`.*SPB.*` *(substring search)*	RASPBERRY CRISPBREAD	SUBSPACE SUBSPECIES
`[0-9]{3}-[0-9]{2}-[0-9]{4}` *(U. S. Social Security numbers)*	166-11-4433 166-45-1111	11-55555555 8675309
`[a-z]+@([a-z]+\.)+(edu|com)` *(simplified email addresses)*	wayne@princeton.edu rs@princeton.edu	spam@nowhere
`[$_A-Za-z][$_A-Za-z0-9]*` *(Java identifiers)*	ident3 PatternMatcher	3a ident#3

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

Regular expression golf



http://xkcd.com/1313

yes	no
obama	romney
bush	mccain
clinton	kerry
reagan	gore
…	…
washington	clinton

Ex. Match elected presidents but not opponents (unless they later won).

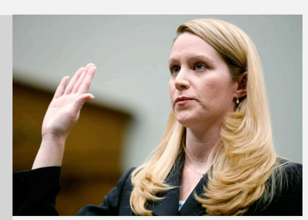
RE. `bu|[rn]t|[coy]e|[mtg]a|j|iso|n[hl]|[ae]d|lev|sh|[lnd]i|[po]o|ls`

madison harrison

Illegally screening a job candidate

" [First name]! and pre/2 [last name] w/7
bush or gore or republican! or democrat! or charg!
or accus! or criticiz! or blam! or defend! or iran contra
or clinton or spotted owl or florida recount or sex!
or controvers! or fraud! or investigat! or bankrupt!
or layoff! or downsiz! or PNTR or NAFTA or outsourc!
or indict! or enron or kerry or iraq or wmd! or arrest!
or intox! or fired or racis! or intox! or slur!
or controvers! or abortion! or gay! or homosexual!
or gun! or firearm! "

— LexisNexis search string used by Monica Goodling to illegally screen candidates for DOJ positions

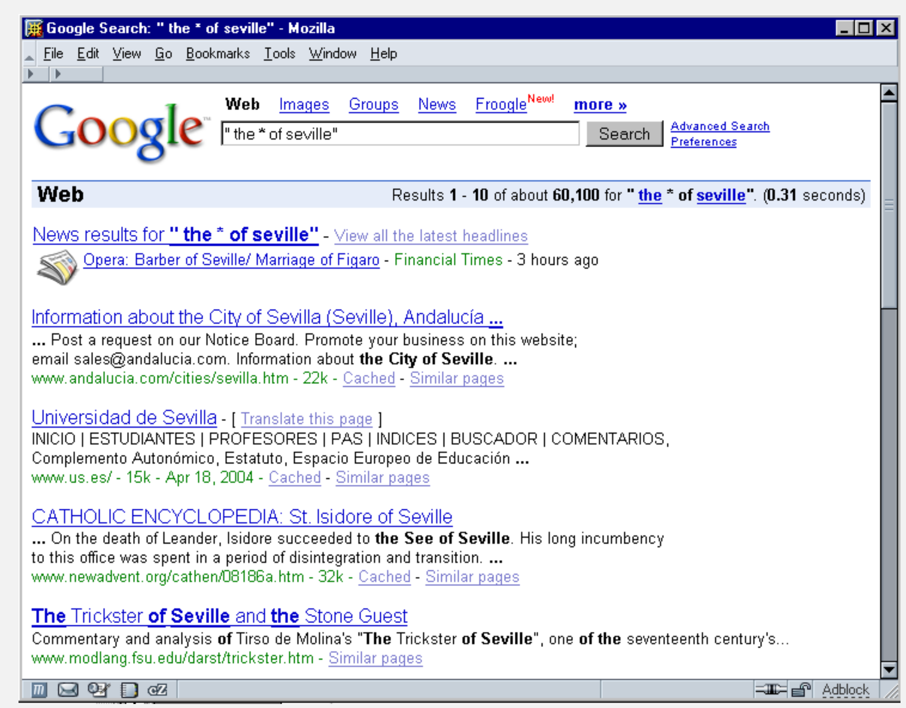


LexisNexis™

http://www.justice.gov/oig/special/s0807/final.pdf

Can the average web surfer learn to use REs?

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



Regular expressions to the rescue

http://xkcd.com/208

Can the average programmer learn to use REs?

Perl RE for valid RFC822 email addresses

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:
\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[
 \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\
](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:
(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)
?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
 \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t]
)*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*
)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)
*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r
\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t
]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](
?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?
:\r\n)?[ \t])*))*\>(?:(?:\r\n)?[ \t])*)|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?
[ \t]))*"(?:(?:\r\n)?[ \t])*)*:(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|
\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"
(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[
\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(
?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([
^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\
]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\
r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]
|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\
.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?
:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".
\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]
]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*\>(?:(?:\r\n)?[ \t])*)(?:,\s*(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[
\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t
])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|
\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\
]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["
()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>
@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,
;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".
\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\[
"()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])
+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z
|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*\>(?:(?:\r\n)?[ \t])*))*)?;\s*)
```

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 (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.

http://algs4.cs.princeton.edu

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.

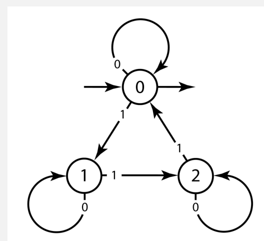
Kleene's theorem.

- 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.

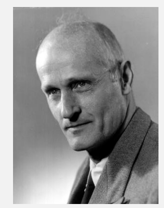
RE `0* | (0*10*10*10*)*`

number of 1's is a multiple of 3

DFA



number of 1's is a multiple of 3



Stephen Kleene
Princeton Ph.D. 1934

Pattern matching implementation: basic plan (first attempt)

Overview is the same as for KMP.

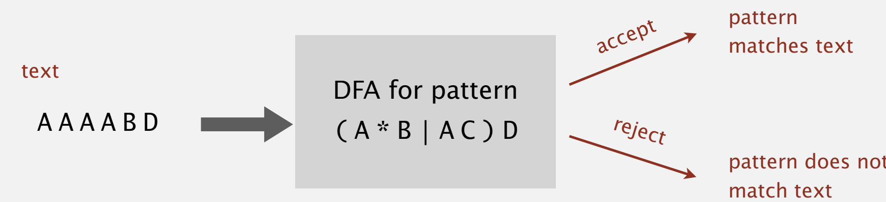
- No backup in text input stream.
- Linear-time guarantee.

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.

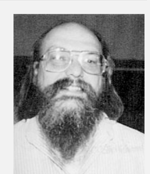


Bad news. Basic plan is infeasible (DFA may have exponential # of states).

Pattern matching implementation: basic plan (revised)

Overview is similar to KMP.

- No backup in text input stream.
- Quadratic-time guarantee (linear-time typical).

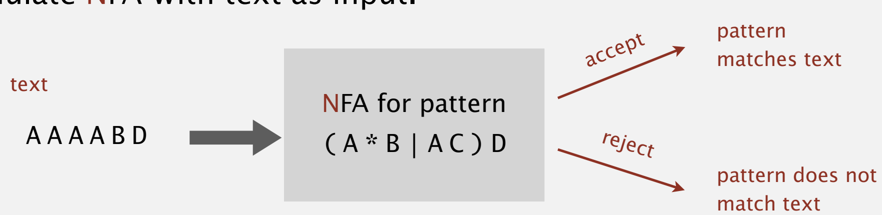


Ken Thompson
Turing Award '83

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 is an NFA?

Nondeterministic finite-state automata

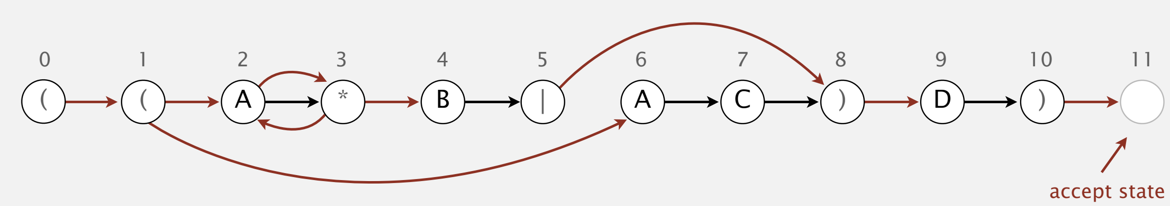
Regular-expression-matching NFA.

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

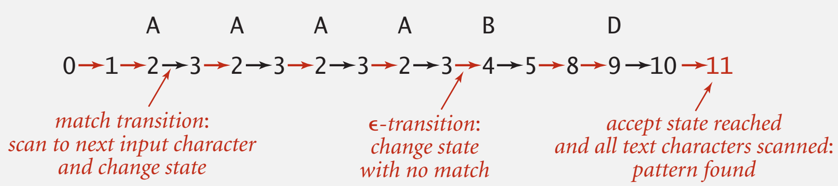


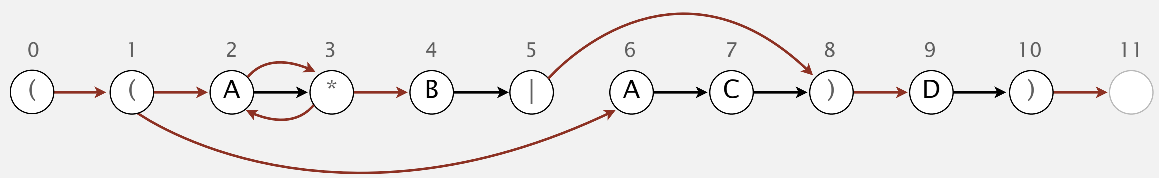
NFA corresponding to the pattern `( ( A * B | A C ) D )`

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.





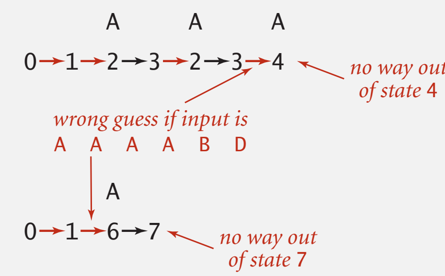
NFA corresponding to the pattern ((A * B | A C) D)

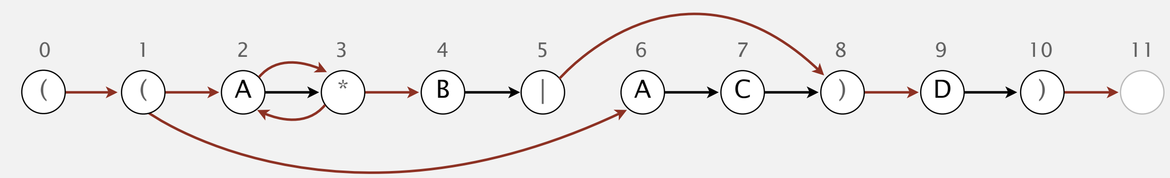
Nondeterministic finite-state automata

Q. Is A A A A B D matched by NFA?

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

[even though some sequences end in wrong state or stall]





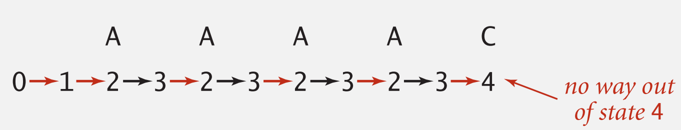
NFA corresponding to the pattern ((A * B | A C) D)

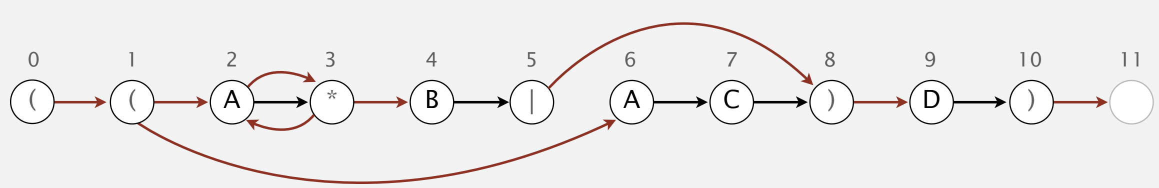
Nondeterministic finite-state automata

Q. Is A A A C matched by NFA?

A. No, because no sequence of legal transitions ends in state 11.

[but need to argue about all possible sequences]





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

Nondeterminism

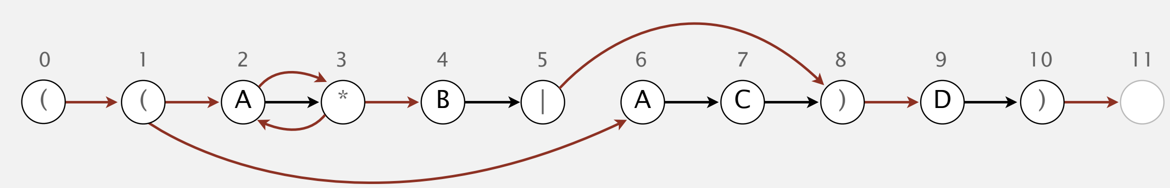
Q. How to determine whether a string is matched by an automaton?

DFA. Deterministic ⇒ easy because 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. [stay tuned]



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

http://algs4.cs.princeton.edu

5.4 Regular Expressions

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

NFA representation

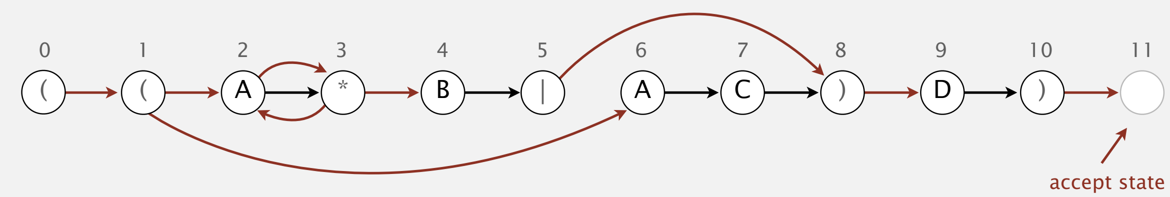
State names. Integers from 0 to M.

number of symbols in RE

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

ε-transitions. Store in a digraph G.

0→1, 1→2, 1→6, 2→3, 3→2, 3→4, 5→8, 8→9, 10→11

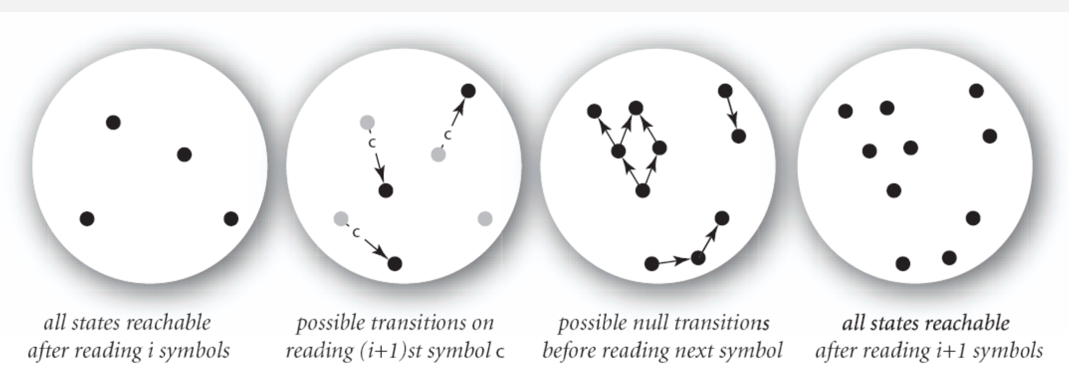


NFA corresponding to the pattern ((A * B | A C) D)

NFA simulation

Q. How to efficiently simulate an NFA?

A. Maintain set of all possible states that NFA could be in after reading in the first i text characters.



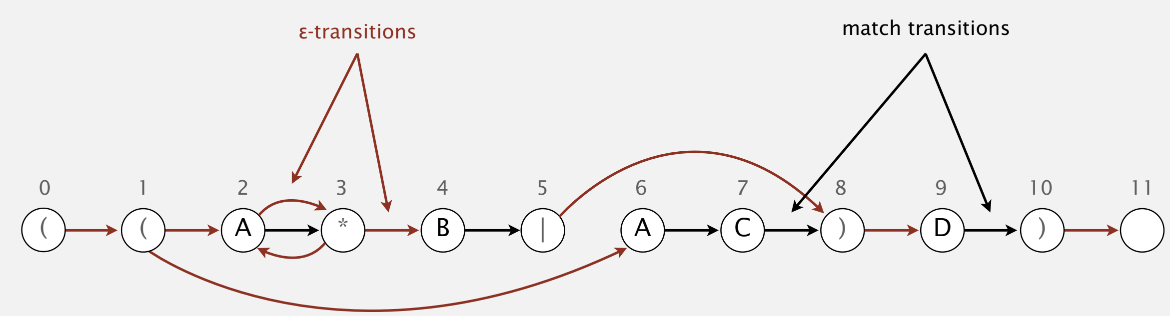
One step in simulating an NFA

Q. How to perform reachability?

NFA simulation demo

Goal. Check whether input matches pattern.

input A A B D



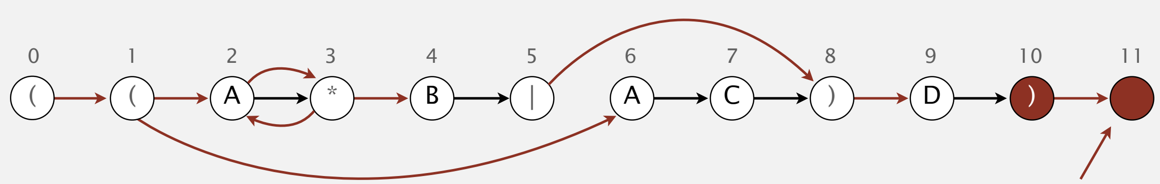
NFA corresponding to the pattern ((A * B | A C) D)

NFA simulation demo

When no more input characters:

- Accept if any state reachable is an accept state.
- Reject otherwise.

input A A B D



set of states reachable : { 10, 11 }

Digraph reachability

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 $E + V$.

NFA simulation: Java implementation

```
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 */  }

}
```

stay tuned (next segment)

NFA simulation: Java implementation

```
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> states = new Bag<Integer>();
      for (int v : pc)
      {
         if (v == M) continue;
         if ((re[v] == txt.charAt(i)) || 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;
}
```

states reachable from start by ε-transitions

set of states reachable after scanning past txt.charAt(i)

not necessarily a match (RE needs to match full text)

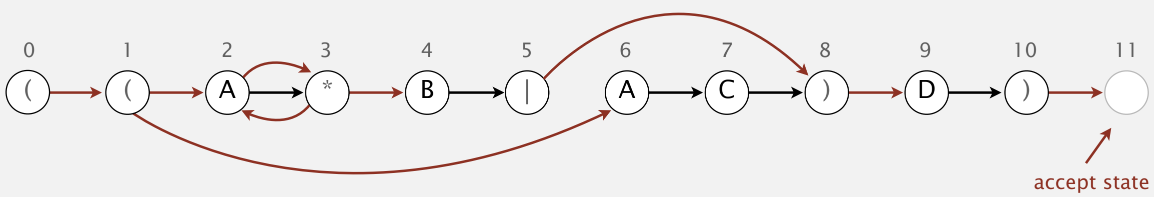
follow ε-transitions

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 $M N$ 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 ε-transitions.
[The NFA construction we will consider ensures the number of edges ≤ $3M$.]



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

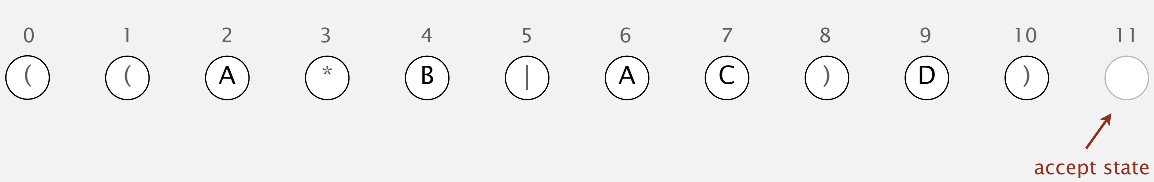
http://algs4.cs.princeton.edu

5.4 Regular Expressions

- regular expressions
- REs and 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.

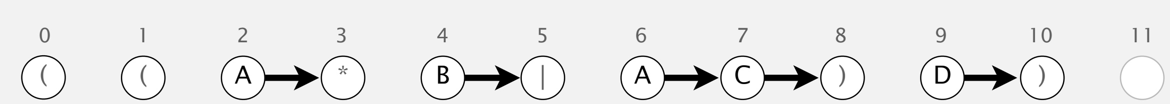


NFA corresponding to the pattern ((A * B | A C) D)

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. ( ) . * |



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

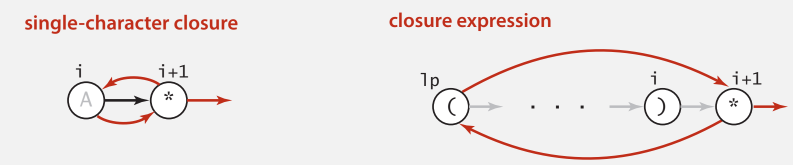
Building an NFA corresponding to an RE

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

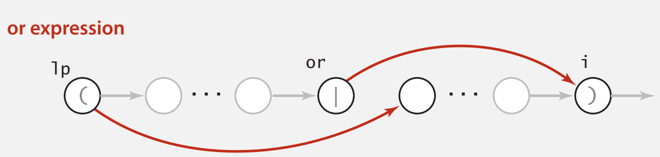
Closure. Add three ε-transition edges for each * operator.

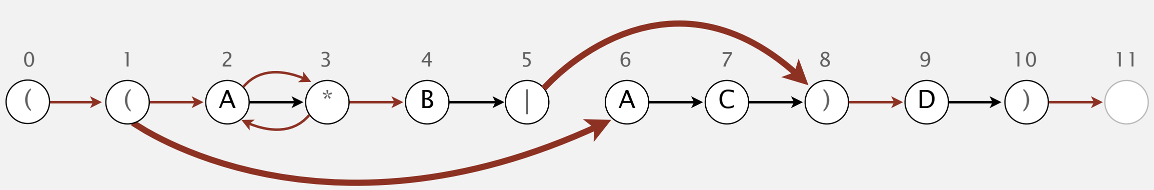


NFA corresponding to the pattern ((A * B | A C) D)

Building an NFA corresponding to an RE

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





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

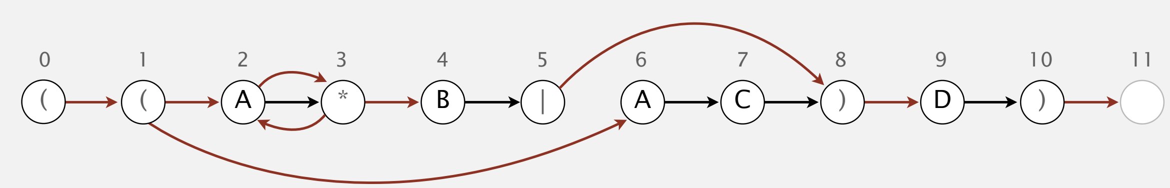
NFA construction: implementation

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

Challenges. Remember left parentheses to implement closure and or; remember | to implement or.

Solution. Maintain a stack.

- (symbol: push (onto stack.
- | symbol: push | onto stack.
-) symbol: pop corresponding (and any intervening |; add ε-transition edges for closure/or.



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

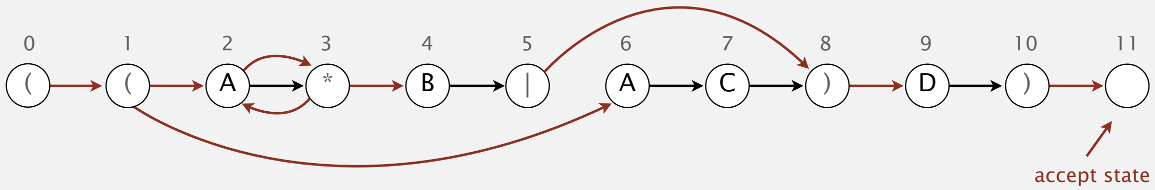
NFA construction demo

stack

((A * B | A C) D)

NFA construction demo

stack



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

NFA construction: Java implementation

```
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);      // left parentheses and |

      else if (re[i] == ')') {                             // 2-way or
         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] == '*') {                     // closure (needs 1-character lookahead)
         G.addEdge(lp, i+1);
         G.addEdge(i+1, lp);
      }

      if (re[i] == '(' || re[i] == '*' || re[i] == ')')    // metasymbols
         G.addEdge(i, i+1);
   }
   return G;
}
```

left parentheses and |

2-way or

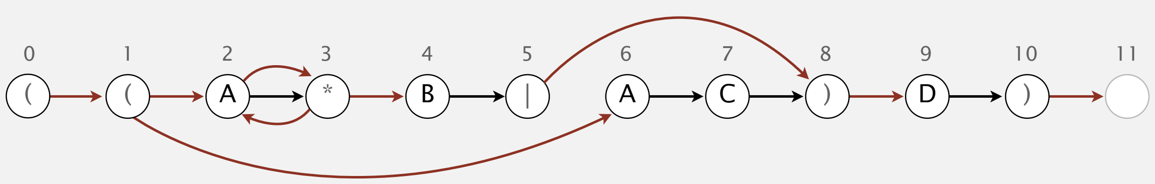
closure
(needs 1-character lookahead)

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.



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

5.4 Regular Expressions

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

Generalized regular expression print

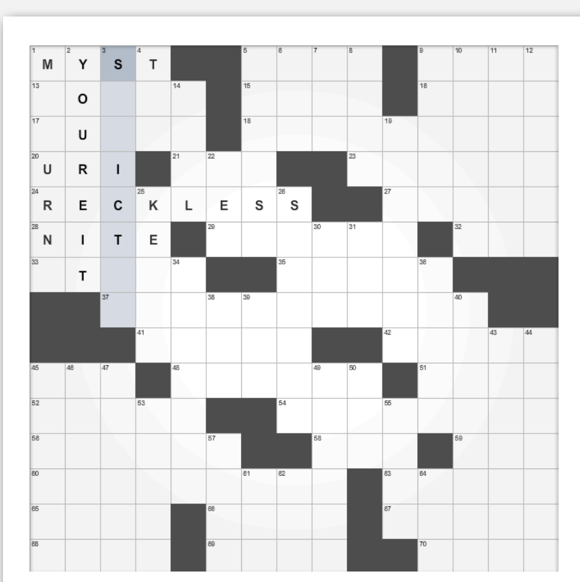
Grep. Take a RE as a command-line argument and print the lines from standard input having some substring that is matched by the RE.

```
public class GREP
{
   public static void main(String[] args)
   {
      String re = "(.*" + args[0] + ".*)";
      NFA nfa = new NFA(re);
      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 $M N$) is the same as for brute-force substring search.

Typical grep application: crossword puzzles



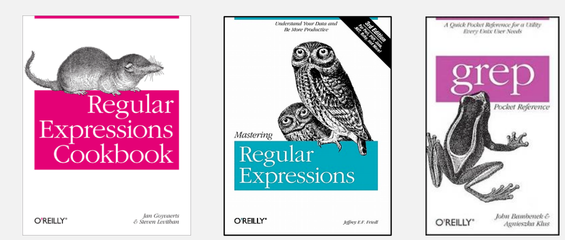
```
% more words.txt
a
aback
abacus
abalone
abandon
…

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

dictionary (standard in Unix)

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 reluctant

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

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, ...

```
% 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

Regular expressions in Java

Validity checking. Does the `input` match the `re`?
Java string library. Use `input.matches(re)` 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(re));
   }
}
```

```
% 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

Harvesting information

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

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

harvest patterns from DNA

harvest links from website

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


Harvesting information

RE pattern matching is implemented in Java's `java.util.regexp.Pattern` and `java.util.regexp.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()`

Algorithmic complexity attacks

Warning. Typical implementations do not guarantee performance!

Unix grep, Java, Perl, Python

```
% 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.

```
% 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 subexpression that was matched earlier.
- Supported by typical RE implementations.

```
(.+)\1          // beriberi couscous
1?$|^(11+?)\1+  // 1111 111111 111111111
```

Some non-regular languages.

- Strings of the form ww 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: `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 ⇒ DFA.
- `grep` RE ⇒ NFA.
- `javac` Java language ⇒ Java byte code.

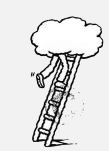
	KMP	grep	Java
pattern	string	RE	program
parser	unnecessary	check if legal	check if legal
compiler output	DFA	NFA	byte code
simulator	DFA simulator	NFA simulator	JVM

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. 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.