6.4 Pattern Matching

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

- regular expressions
- NFAs
- NFA simulation
- NFA construction
- appplications

Pattern matching

Substring search. Find occurrences of a single string in text.
Pattern matching. Find occurrences of 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`

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.

Regular expressions

A regular expression is a notation to specify a (possibly infinite) set of strings.

a "language"

operation	example RE	in language	not in language
concatenation	`AABAAB`	`AABAAB`	every other string
or	`AA | BAAB`	`AA` `BAAB`	every other string
closure	`AB*A`	`AA` `ABBBBBBBBA`	`AB` `ABABA`
parentheses	`A(A|B)AAB`	`AAAAB` `ABAAB`	every other string
	`(AB)*A`	`A` `ABABABABABA`	`AA` `ABBA`

Regular expression shortcuts

Additional operations are often added for convenience.

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

operation	example RE	in language	not in language
wildcard	`.U.U.U.`	`CUMULUS` `JUGULUM`	`SUCCUBUS` `TUMULTUOUS`
at least 1	`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]{6}`	`RHYTHM`	`DECADE`

Regular expression examples

Notation is surprisingly expressive

regular expression	in set	not in set
`.*SPB.*` *(contains the trigraph spb)*	`RASPBERRY` `CRISPBREAD`	`SUBSPACE` `SUBSPECIES`
`[0-9]{3}-[0-9]{2}-[0-9]{4}` *(Social Security numbers)*	`166-11-4433` `166-45-1111`	`11-55555555` `8675309`
`[a-z]+@([a-z]+\.)+(edu|com)` *(valid email addresses)*	`wayne@princeton.edu` `rs@princeton.edu`	`spam@nowhere`
`[$_A-Za-z][$_A-Za-z0-9]*` *(valid Java identifiers)*	`ident3` `PatternMatcher`	`3a` `ident#3`

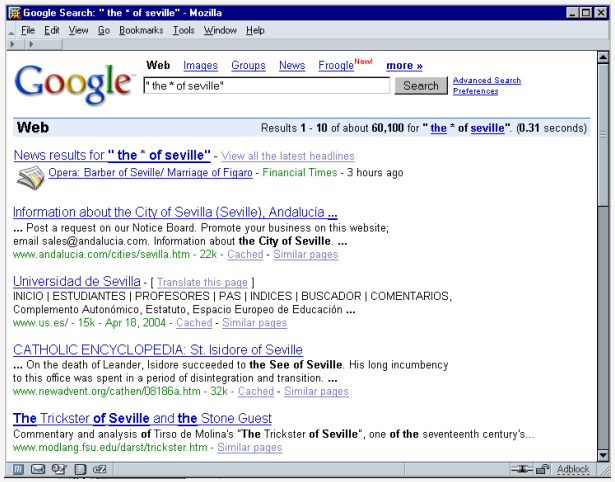
and plays a well-understood role in the theory of computation.

Regular expressions to the rescue

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.

Using * in WishList Searches. To search for similar words in Keyword and Title WishList searches, use the asterisk (*) as a special symbol that replaces the endings of words. For example, the keyword *AIRP** would find shows containing "airport," "airplane," "airplanes," as well as the movie "Airplane!" To enter an asterisk, press the SLOW ( ) button as you are spelling out your keyword or title.

The asterisk can be helpful when you're looking for a range of similar words, as in the example above, or if you're just not sure how something is spelled. Pop quiz: is it "irresistible" or "irresistable?" Use the keyword *IRRESIST** and don't worry about it! Two things to note about using the asterisk:

- It can only be used at a word's end; it cannot be used to omit letters at the beginning or in the middle of a word. (For example, *AIR*NE* or **PLANE* would not work.)

Reference: page 76, Hughes DirectTV TiVo manual

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])*)(?:\.(?:(?:\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])*)?:(?:(?:\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 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.

> *" Sometimes you have a programming problem and it seems like the best solution is to use regular expressions; now you have two problems. "*

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

- regular expressions
- **NFAs**
- NFA simulation
- NFA construction
- appplications

Pattern matching implementation: basic plan (first attempt)

Overview is the same as for KMP !

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

Ken Thompson

Underlying abstraction:

Deterministic finite state automata (DFA)

Basic plan.

- Build DFA from RE.
- Simulate DFA with text as input.

Bad news: Basic plan is infeasible (DFA may have exponential number of states)

Pattern matching implementation: basic plan (revised)

Overview is similar to KMP

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

Ken Thompson

Underlying abstraction:

Nondeterministic finite state automata (NFA)

Basic plan.

- Build NFA from RE.
- Simulate NFA with text as input.

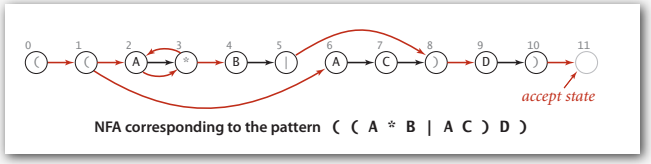
Nondeterministic finite-state automata

Pattern matching NFA.

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

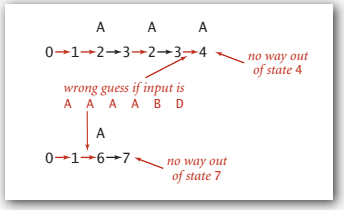
Nondeterminism.

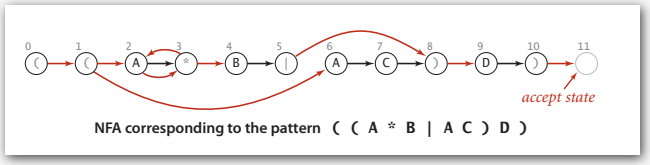
- one view: machine can guess the proper sequence of state transitions
- another view: sequence is a proof that the machine accepts the text



Nondeterministic finite-state automata

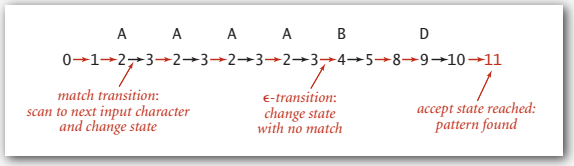
Ex. Is AAAABD recognized by NFA?





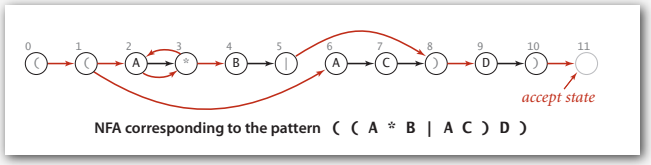
Nondeterministic finite-state automata

Ex. Is AAAABD recognized by NFA?



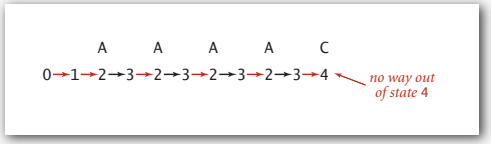
YES!

Note: any sequence of legal transitions that ends in 11 is a proof.



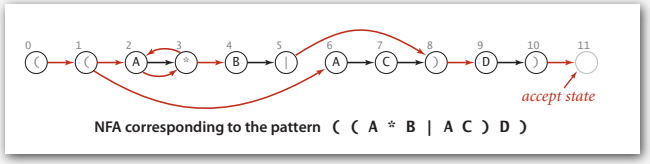
Nondeterministic finite-state automata

Ex. Is AAAAC recognized by NFA?



NO

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

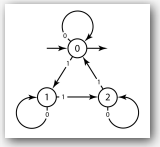


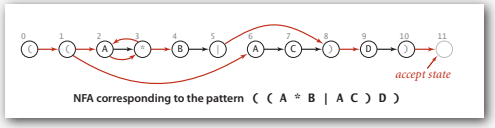
Nondeterminism

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

DFA. Deterministic ⇒ exactly one applicable transition.

NFA. Nondeterministic ⇒ can be several applicable transitions; need to select the right one!





Q. How to simulate NFA?

A. Systematically consider all possible transition sequences.

Pattern matching implementation: basic plan (revised)

Overview is similar to KMP

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

Ken Thompson

Underlying abstraction:

Nondeterministic finite state automata (NFA)

Basic plan.

- Build NFA from RE.
- Simulate NFA with text as input.

- regular expressions
- NFAs
- **NFA simulation**
- NFA construction
- appplications

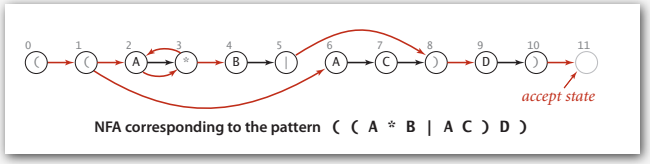
NFA representation

State names. Integers from `0` to `M`.

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

ε-transitions. Store in a digraph `G`.

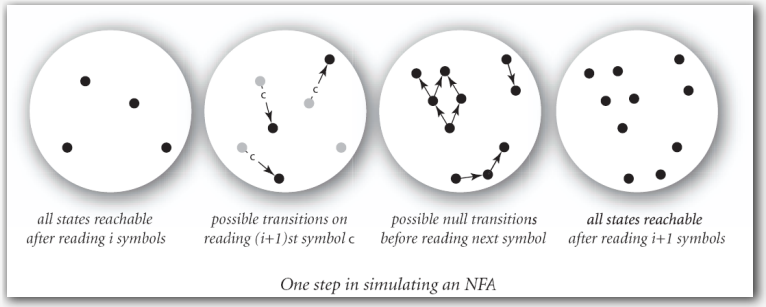
- `0→1, 1→2, 1→6, 2→3, 3→2, 3→4, 5→8, 8→9, 10→11`



NFA 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.



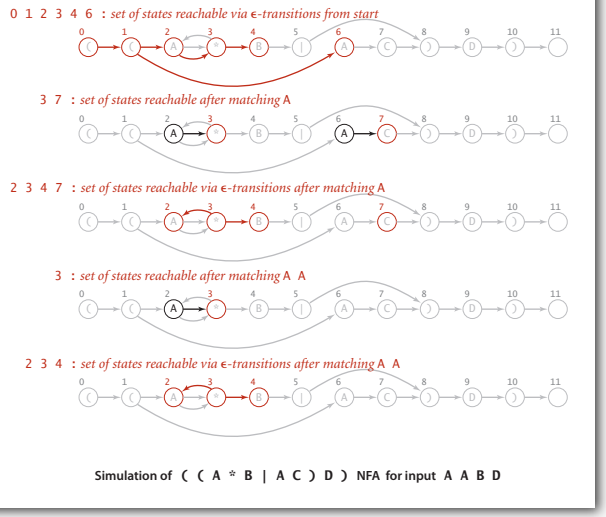
Q. How to perform reachability?

Digraph reachability

Find all vertices reachable from a given set of vertices.

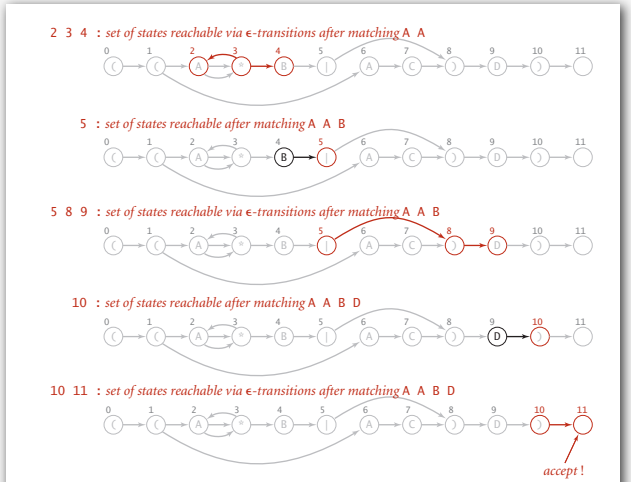
```
public class DFS
{
   private SET<Integer> marked;
   private Digraph G;

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

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

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


NFA simulation example



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

NFA simulation example



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

NFA simulation: Java implementation

```
public boolean recognizes(char[] txt)
{
   DFS dfs = new DFS(G);

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

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

      pc = dfs.reachable(match);
   }

   return pc.contains(M);
}
```

states reachable from start by ε-transitions

all possible states after scanning past `txt[i]`

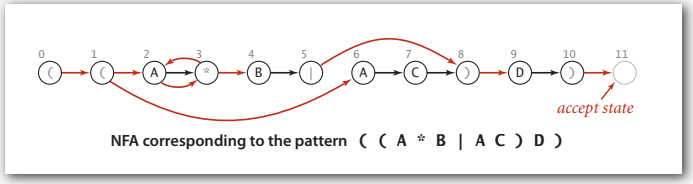
follow ε-transitions

accept if you can end in state M

NFA simulation: analysis

Proposition 1. Determining whether an N-character text string is recognized by the NFA corresponding to an M-character pattern takes time proportional to NM in the worst case.

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 construction we consider ensures the number of edges is at most M.)

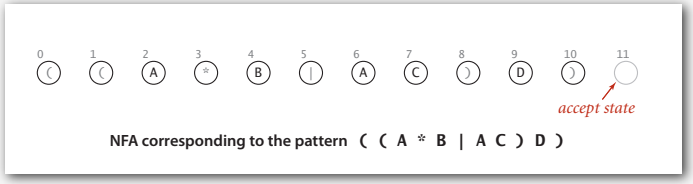


NFA corresponding to the pattern ((A * B | A C) D)

- regular expressions
- NFAs
- NFA simulation
- **NFA construction**
- appplications

Building an NFA corresponding to an RE

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

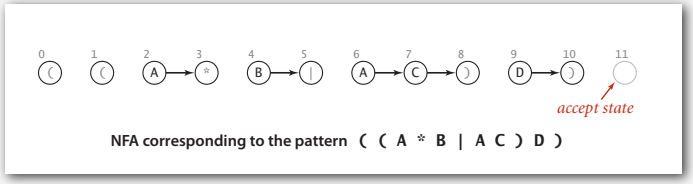


NFA corresponding to the pattern ((A * B | A C) D)

Building an NFA corresponding to an RE

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

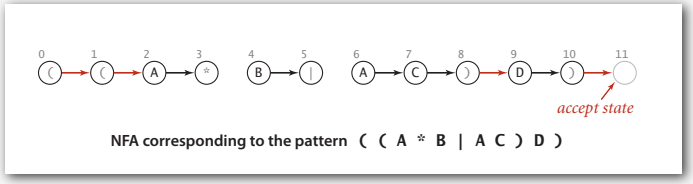
Alphabet. `A B C D`
Metacharacters. `( ) . * |`



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

Building an NFA corresponding to an RE

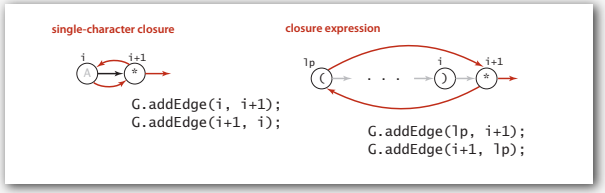
Parentheses. Add ε-transition edge from parentheses to next state.

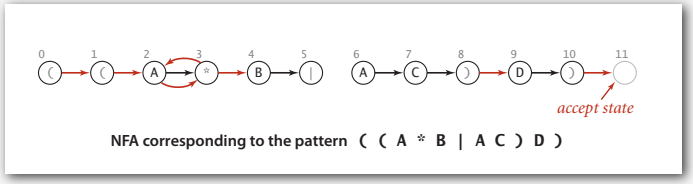


NFA corresponding to the pattern ((A * B | A C) D)

Building an NFA corresponding to an RE

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

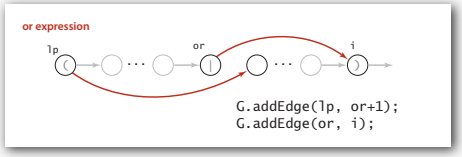


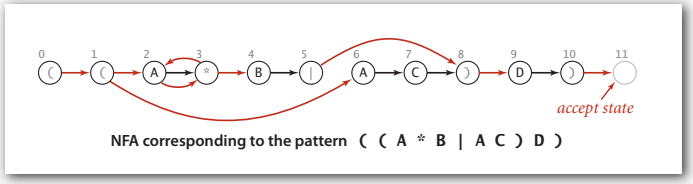


NFA corresponding to the pattern ((A * B | A C) D)

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)

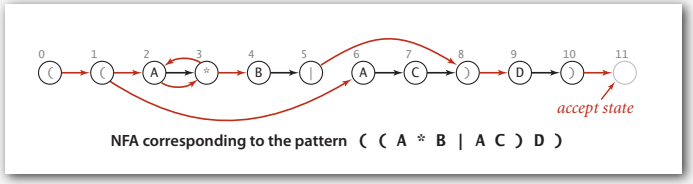
NFA construction: implementation

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

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

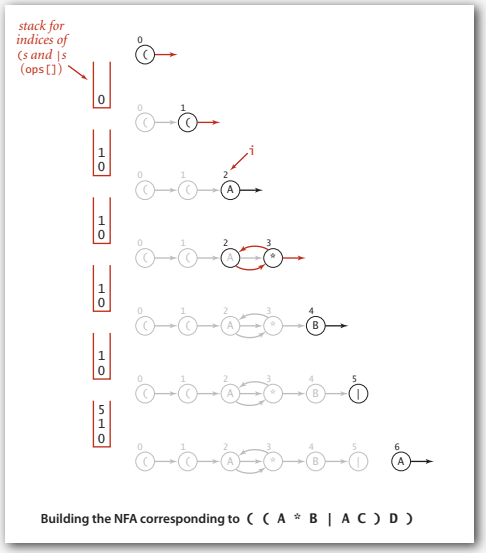
Solution. Maintain a stack.

- Left parenthesis: push onto stack.
- | symbol: push onto stack.
- Right parenthesis: add edges for closure and or.

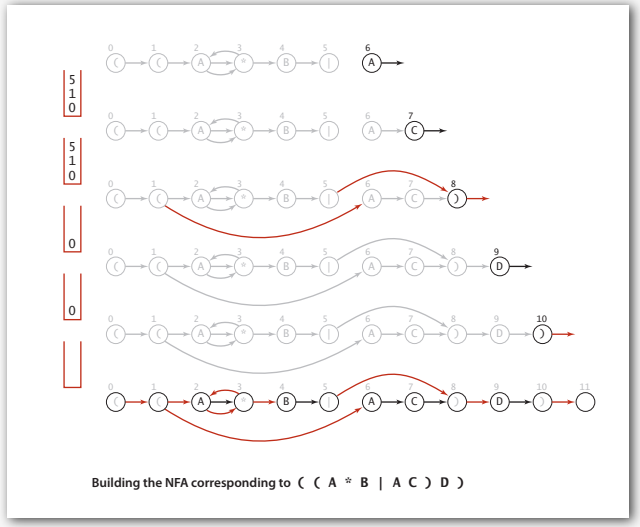


NFA corresponding to the pattern ((A * B | A C) D)

NFA construction: example



NFA construction: example



NFA construction: Java implementation

```
public NFA(int R, char[] re)
{
   Stack<Integer> ops = new Stack<Integer>();
   this.re = re;
   M = re.length;
   G = new Digraph(M+1);
   for (int i = 0; i < M; i++) {
      int lp = i;

      if (re[i] == '(' || re[i] == '|') ops.push(i);

      else if (re[i] == ')') {
         int or = ops.pop();
         if (re[or] == '|') {
            lp = ops.pop();
            G.addEdge(lp, or+1);
            G.addEdge(or, i);
         }
      }

      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);
   }
}
```

left parentheses and |

or

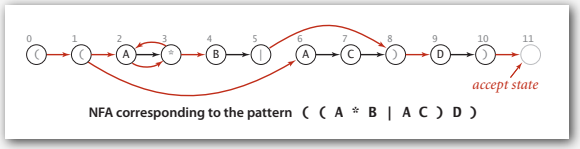
closure (needs lookahead)

metasymbols

NFA construction: analysis

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

Pf. For each of the M characters in the pattern, we add one or two ε-transitions and perhaps execute one or two stack operations.



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

Generalized regular expression print

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

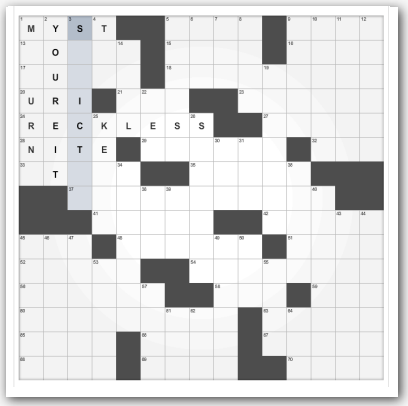
```
public class GREP
{
   public static void main(String[] args)
   {
      String regexp = "(.*" + args[0] + ".*)";
      char[] re = regexp.toCharArray();
      while (!StdIn.isEmpty())
      {
         String line = StdIn.readLine();
         char[] txt = line.toCharArray();
         NFA nfa = new NFA(256, re);
         if (nfa.recognizes(txt))
            StdOut.println(line);
      }
   }
}
```

find lines containing RE as a substring

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

Typical grep application

Crossword puzzle



dictionary (standard in UNIX) also on booksite

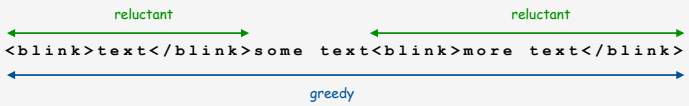
```
% more words.txt
a
aback
abacus
abalone
abandon
...

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


Industrial-strength grep implementation

To complete the implementation:

- Add character classes.
- Handling metacharacters.
- Add capturing capabilities.
- Extend the closure operator.
- Error checking and recovery.
- Greedy vs. reluctant matching.

Ex. Which substring(s) should be matched by the RE `<blink>.*</blink>` ?



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]*$' dict.txt | egrep '..........'
```

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-Za-z][a-z]*$/' dict.txt
```

print all uppercase words

do for each line

Regular expressions in Java

Validity checking. Is `input` in the set described by the `re`?

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

```
public class Validate
{
   public static void main(String[] args)
   {
      String re    = 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
gcgcggcggcggcggctg
gcgctg
gcgctg
gcgcggcggcggaggcggaggcggctg

% 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 patterns from DNA

harvest links from website

Harvesting information

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 re       = args[0];
      In in           = new In(args[1]);
      String input    = in.readAll();
      Pattern pattern = Pattern.compile(re);
      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!

grep, Java, Perl

```
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaac              1.6 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaac             3.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac            9.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac          23.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac         62.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac   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"  dictionary.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: `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 exact pattern matching via DFA simulation.
- Implement RE pattern matching via NFA simulation.

Theoretician.

- RE is a compact description of a set of strings.
- NFA is an abstract machine equivalent in power to RE.
- DFAs and REs have limitations.

You. Practical application of core CS principles.

Example of essential paradigm in computer science.

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