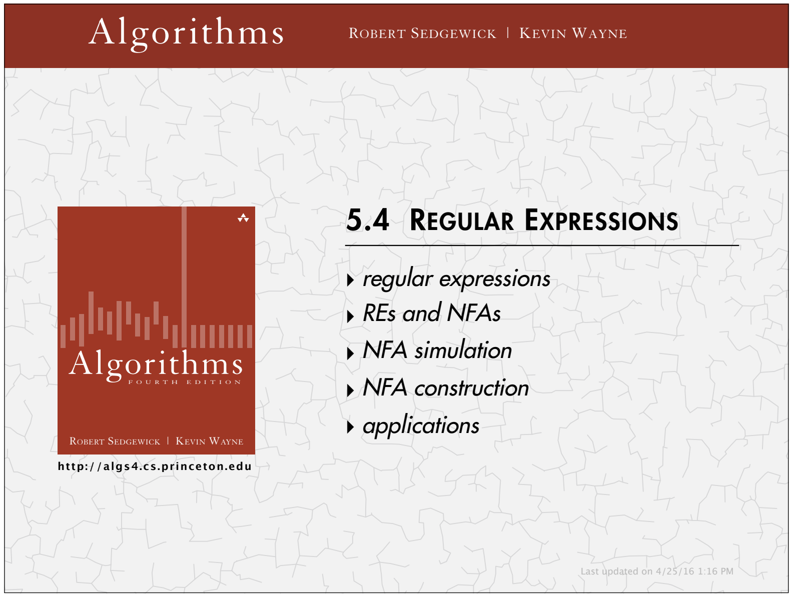


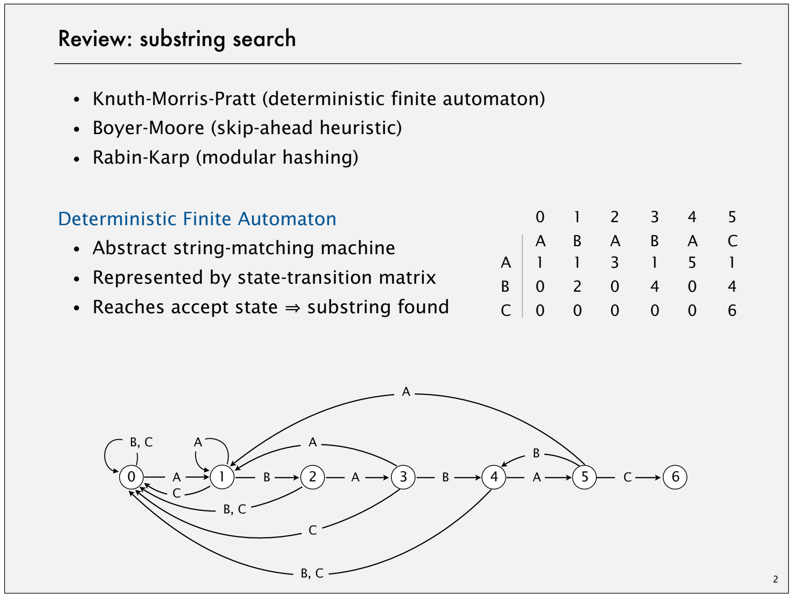
Algorithms
Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

5.4 Regular Expressions

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

Last updated on 4/25/16 1:16 PM



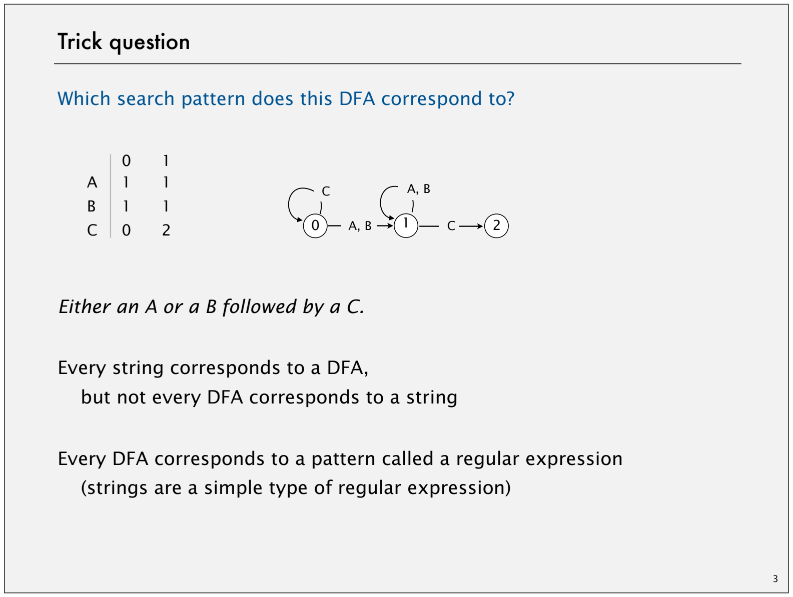
Review: substring search

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

Deterministic Finite Automaton

- Abstract string-matching machine
- Represented by state-transition matrix
- Reaches accept state ⇒ substring found

	0	1	2	3	4	5
	A	B	A	B	A	C
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6



Trick question

Which search pattern does this DFA correspond to?

	0	1
A	1	1
B	1	1
C	0	2

Either an A or a B followed by a C.

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

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

http://algs4.cs.princeton.edu

5.4 Regular Expressions

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

Finding interesting words

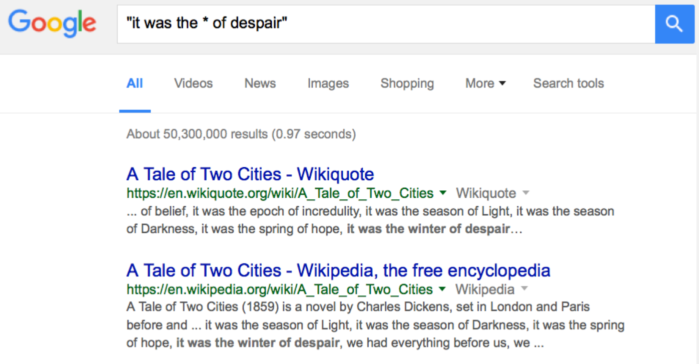
```
$ egrep '^[a-j]{8,}$' /usr/share/dict/words
acidified
beachhead
beheaded
headache

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

Subtle differences in syntax

XKCD t-shirt

Google allows a limited form of regular expression search



"it was the * of despair"

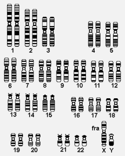
About 50,300,000 results (0.97 seconds)

A Tale of Two Cities - Wikiquote
https://en.wikiquote.org/wiki/A_Tale_of_Two_Cities ▾ Wikiquote ▾
... of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, **it was the winter of despair**...

A Tale of Two Cities - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/A_Tale_of_Two_Cities ▾ Wikipedia ▾
A Tale of Two Cities (1859) is a novel by Charles Dickens, set in London and Paris before and ... it was the season of Light, it was the season of Darkness, it was the spring of hope, **it was the winter of despair**, we had everything before us, we ...

Genomics

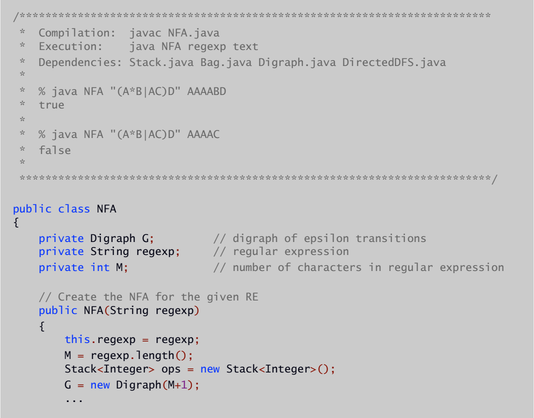
- Fragile X syndrome is a common cause of mental retardation.
- A human's genome is a string.
- It contains triplet repeats of CGG or AGG, bracketed by GCG at the beginning and CTG at the end.
- Number of repeats is variable and is correlated to syndrome.



pattern GCG(CGG|AGG)*CTG

text GCGGCGTGTGTGCGAGAGAGTGGGTTTAAAGCTGGCGCGGAGGCGGCTGGCGCGGAGGCTG

Syntax highlighting



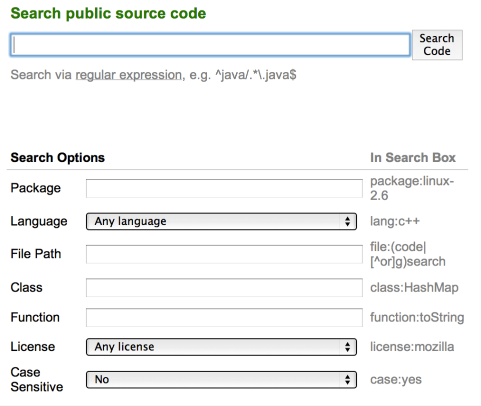
```
/*************************************************************************
 *  Compilation:  javac NFA.java
 *  Execution:    java NFA regexp text
 *  Dependencies: Stack.java Bag.java Digraph.java DirectedDFS.java
 *
 *  % java NFA "(A*B|AC)D" AAAABD
 *  true
 *
 *  % java NFA "(A*B|AC)D" AAAAC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

Google code search



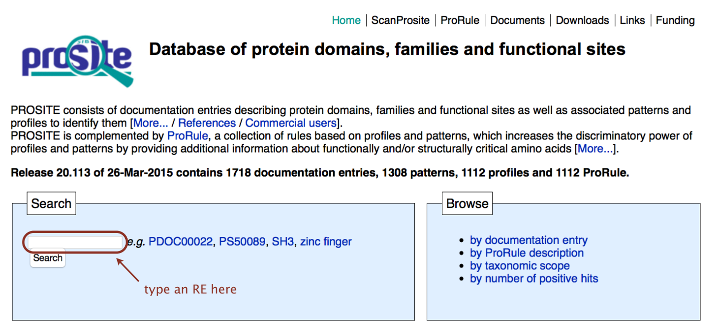
Search public source code

Search via regular expression, e.g. ^java/.*\.java$

Search Options		In Search Box
Package		package:linux-2.6
Language	Any language	lang:c++
File Path		file:(code|[^or]g)search
Class		class:HashMap
Function		function:toString
License	Any license	license:mozilla
Case Sensitive	No	case:yes

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

Prosite (computational biochemistry)



Home | ScanProsite | ProRule | Documents | Downloads | Links | Funding

Database of protein domains, families and functional sites

PROSITE consists of documentation entries describing protein domains, families and functional sites as well as associated patterns and profiles to identify them [More... / References / Commercial users].
PROSITE is complemented by ProRule, a collection of rules based on profiles and patterns, which increases the discriminatory power of profiles and patterns by providing additional information about functionally and/or structurally critical amino acids [More...].

Release 20.113 of 26-Mar-2015 contains 1718 documentation entries, 1308 patterns, 1112 profiles and 1112 ProRule.

- by documentation entry
- by ProRule description
- by taxonomic scope
- by number of positive hits

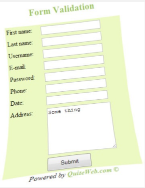
http://prosite.expasy.org

Even more applications

Test if a string matches some pattern.

- Scan for virus signatures.
- Process natural language.
- Specify a programming language.
- Access information in digital libraries.
- Search genome using PROSITE patterns.
- Filter text (spam, NetNanny, Carnivore, malware).
- Validate data-entry fields (dates, email, URL, credit card).

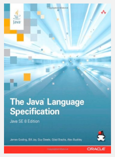
...



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

Regular expressions: operator precedence

- Star applies only to immediately preceding char or parenthetical group
  AB*A
- | has the lowest priority
  AA|BA*B(AB)*

operation	example RE	matches	does not match
concatenation	AABAAB	AABAAB	*every other string*
or	AA | BAAB	AA BAAB	*every other string*
star (aka 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: quiz 1

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

A. ABABAB

B. CDCCDDDD

C. ABCCDAB

D. ABDABCABD

E. *I don't know.*

Regular expression shortcuts

Additional operations further extend the utility of REs.

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
one or more	A(BC)+DE	ABCDE ABCBCDE	ADE BCDE
exactly k	[0-9]{5}-[0-9]{4}	08540-1321 19072-5541	111111111 166-54-111

Note. These operations are useful but not essential.

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

Exercise

Simplify the following regular expression over the alphabet {A, B}:
`(B | A*B* | BAA*)*`

Exercise

Simplify the following regular expression over the alphabet {A, B}:
`(B | A*B* | BAA*)*`

matches 'A'

≡ `(B | A)*`

≡ `.*`

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.

Exercise

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

Exercise

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

Case 1: A and B are separated by an even number of characters

```
A (..)* B (..)*
```

Case 2: A and B are separated by an odd number of characters

```
A (..)* . B . (..)*
```

Put it together:

```
A(..)*B(..)* | A(..)*.B.(..)*
```

Optionally simplify:

```
A (..)* (B | .B.) (..*)
```

You can go crazy with regular expressions

Perl RE for valid RFC822 email addresses

```
(?:(?:\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*

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.

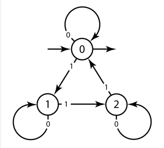
- For any DFA, there exists a RE that describes the same set of strings.
- For any RE, there exists a DFA that recognizes the same set of strings.

RE

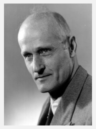
0* | (0*10*10*10*)*

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)

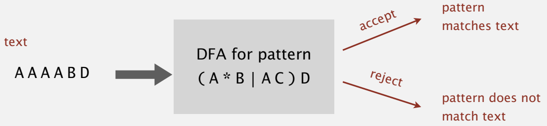
Overview is the same as for KMP.

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

Underlying abstraction. Deterministic finite state automata (DFA).

Basic plan. [apply Kleene's theorem]

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



text

AAAABD

DFA for pattern
(A * B | A C) D

accept

pattern
matches text

reject

pattern does not
match text

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

Pattern matching implementation: basic plan (revised)

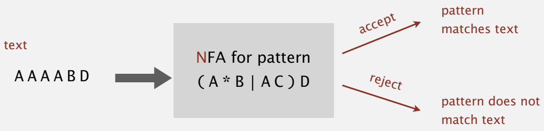
Overview is similar to KMP.

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

Underlying abstraction. Nondeterministic finite state automata (NFA).

Basic plan. [apply Kleene's theorem]

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



text

AAAABD

NFA for pattern
(A * B | A C) D

accept

pattern
matches text

reject

pattern does not
match text

Q. What is an NFA?

Nondeterministic finite-state automata

Regular-expression-matching NFA.

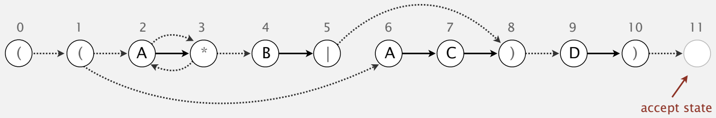
- We assume RE enclosed in parentheses.
- One state per RE character (start = 0, accept = M).
- Match transition (change state and scan to next text char).
- Dashed ε-transition (change state, but don't scan text).
- Accept if any sequence of transitions ends in accept state.

text chars in nodes, not edges

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.



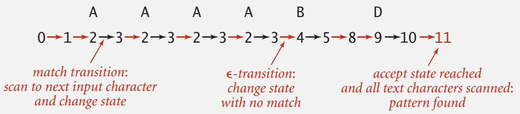
accept state

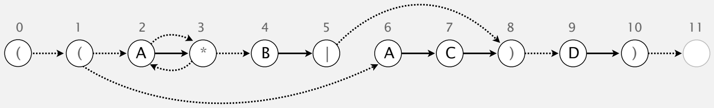
NFA corresponding to the pattern ((A * B | A C) D)

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.



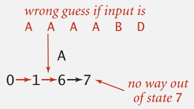


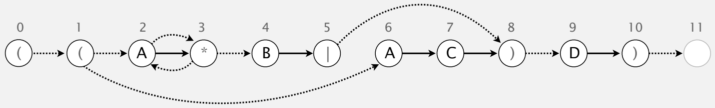
NFA corresponding to the pattern ((A * B | A C) D)

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 get stuck]



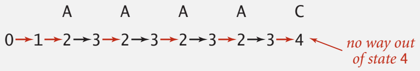


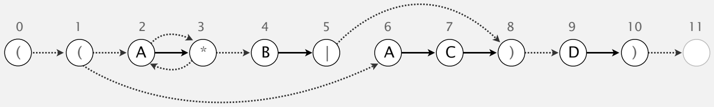
NFA corresponding to the pattern ((A * B | A C) D)

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

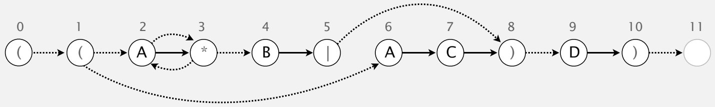
Q. How to determine whether a string is matched by an automaton?

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

NFA. Nondeterministic ⇒ hard (can be several applicable transitions at each step; need to select the "right" ones!)

Q. How to simulate NFA?

A. Systematically consider all possible transition sequences. [stay tuned]



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

NFA vs. quantum computers

How are nondeterministic finite automata different from quantum computers?

Quantum computers are *actually, physically* nondeterministic.

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

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.

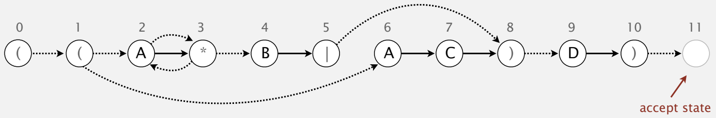
number of symbols in RE

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

	0	1	2	3	4	5	6	7	8	9	10
re[]	(	(	A	*	B	|	A	C	)	D	)

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



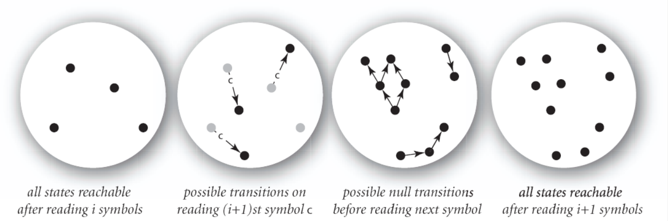
accept state

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



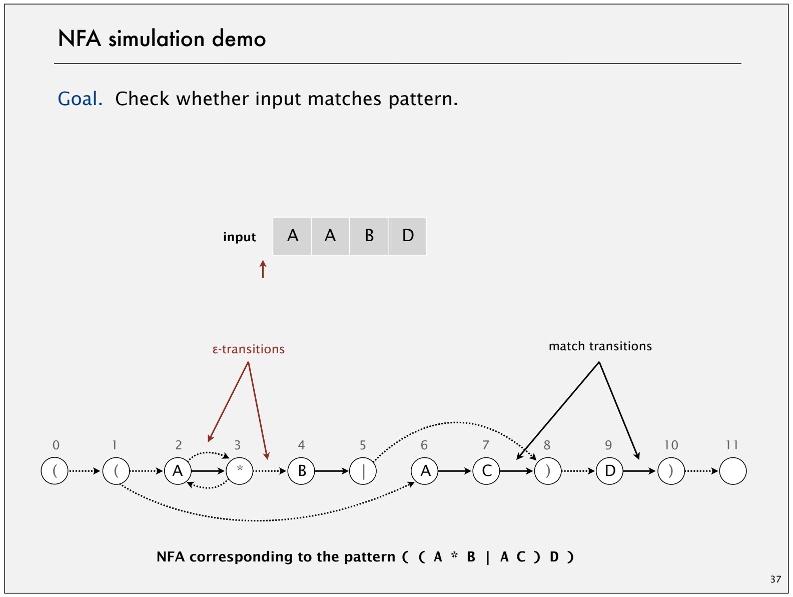
all states reachable after reading i symbols

possible transitions on reading (i+1)st symbol c

possible null transitions before reading next symbol

all states reachable after reading i+1 symbols

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

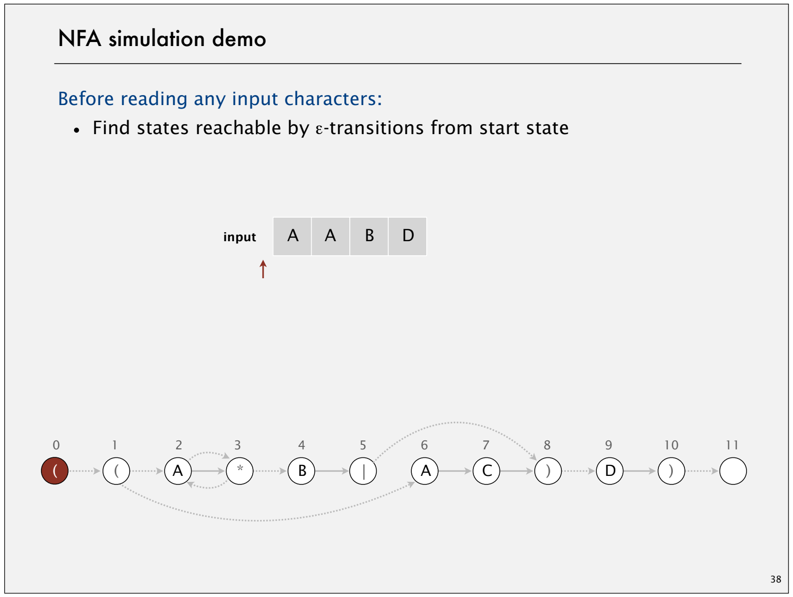


NFA simulation demo

Goal. Check whether input matches pattern.

input | A | A | B | D

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

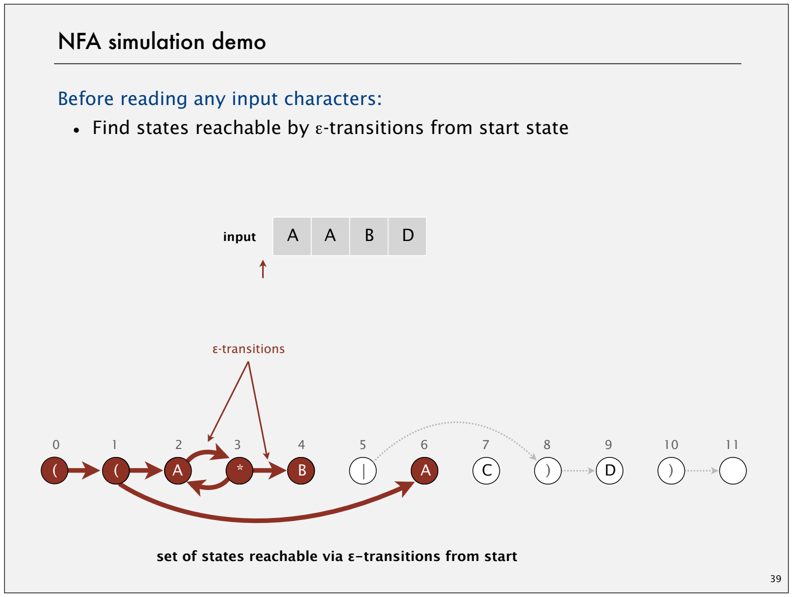


NFA simulation demo

Before reading any input characters:

- Find states reachable by ε-transitions from start state

input | A | A | B | D



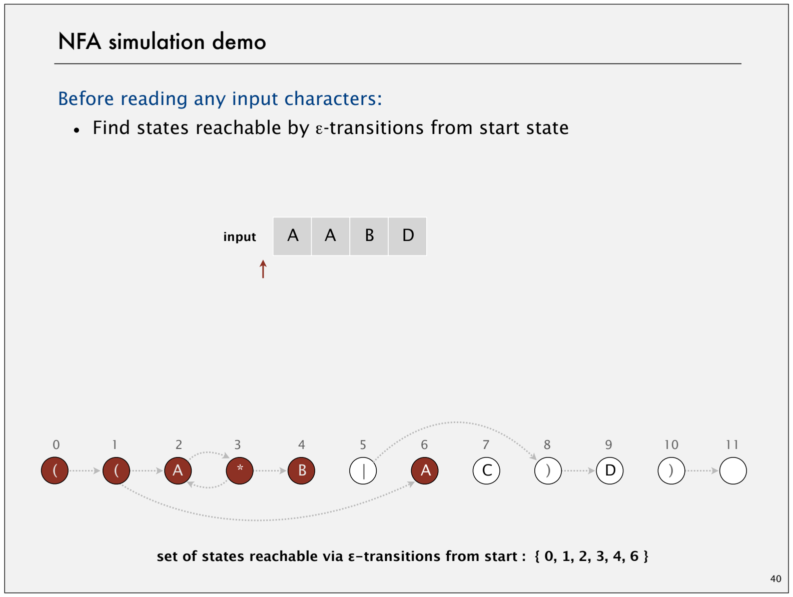
NFA simulation demo

Before reading any input characters:

- Find states reachable by ε-transitions from start state

input | A | A | B | D

set of states reachable via ε-transitions from start



NFA simulation demo

Before reading any input characters:

- Find states reachable by ε-transitions from start state

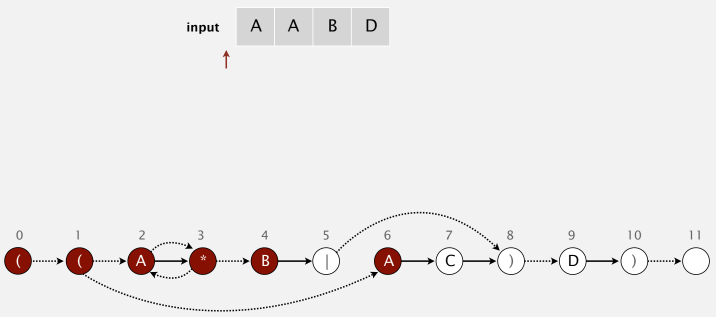
input | A | A | B | D

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

NFA simulation demo

Before reading any input characters:

- Find states reachable by ε-transitions from start state

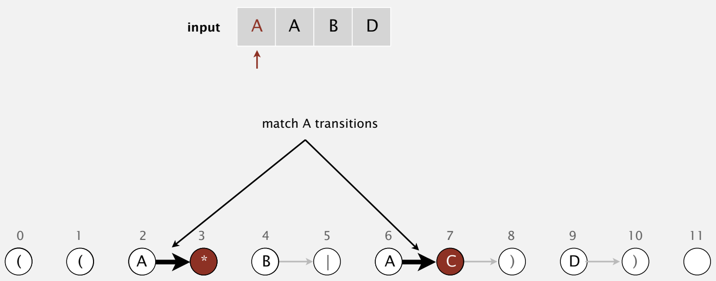


set of states reachable via ε–transitions from start : { 0, 1, 2, 3, 4, 6 }

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

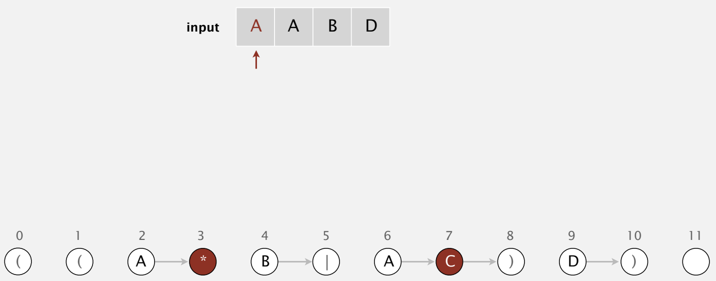


set of states reachable after matching A

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

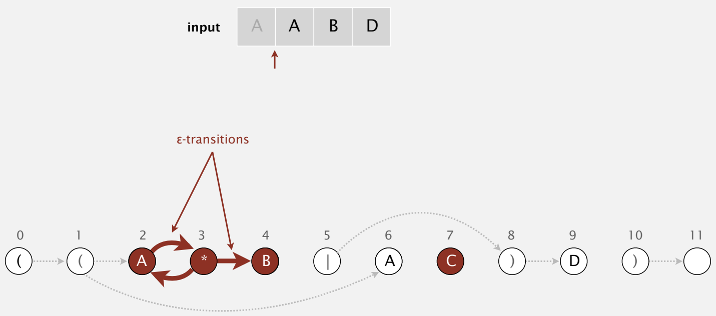


set of states reachable after matching A : { 3, 7 }

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions



set of states reachable via ε–transitions after matching A

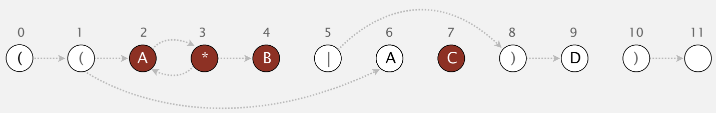
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

input | A | A | B | D

↑



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

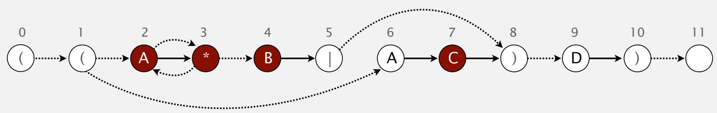
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

input | A | A | B | D

↑



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

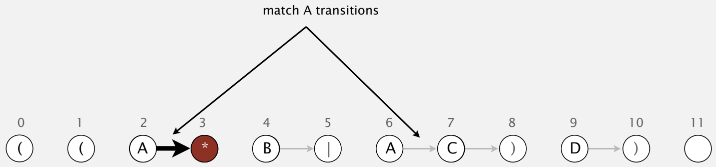
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

input | A | A | B | D

↑



set of states reachable after matching A A

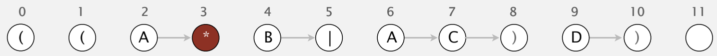
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

input | A | A | B | D

↑



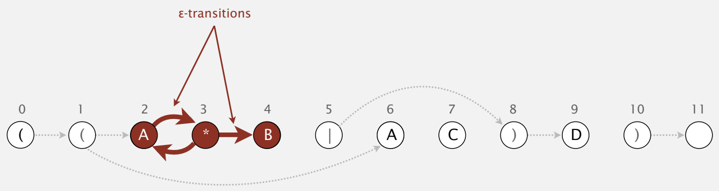
set of states reachable after matching A A : { 3 }

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

input A A B D



ε-transitions

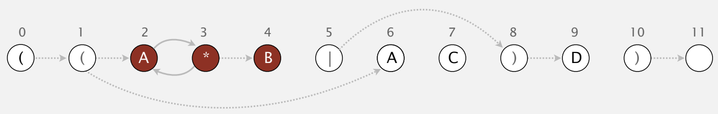
set of states reachable via ε-transitions after matching A A

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

input A A B D



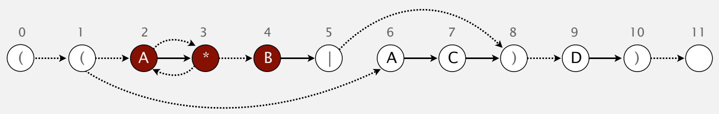
set of states reachable via ε-transitions after matching A A : { 2, 3, 4 }

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

input A A B D



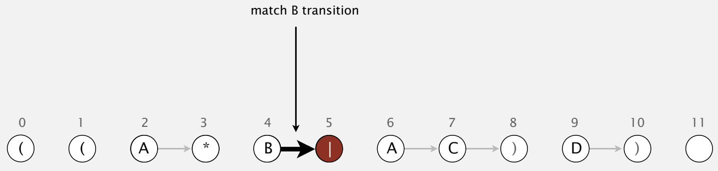
set of states reachable via ε-transitions after matching A A : { 2, 3, 4 }

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

input A A B D



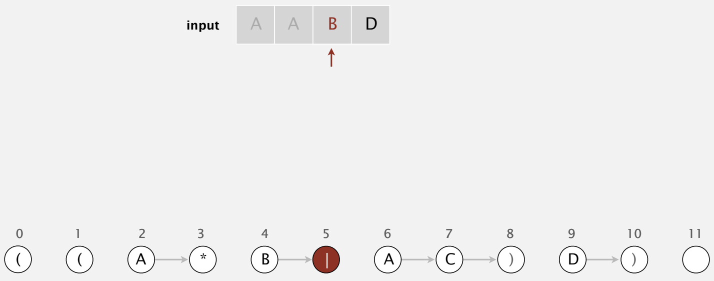
match B transition

set of states reachable after matching A A B

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

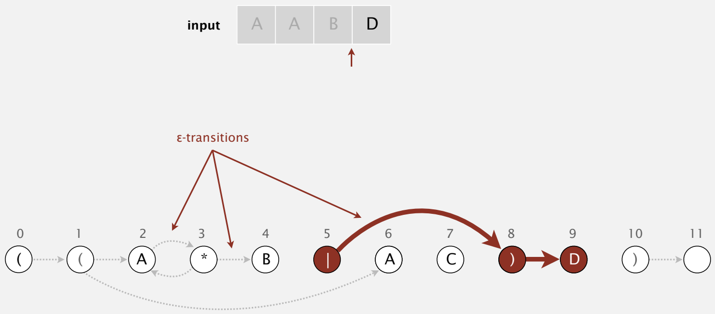


set of states reachable after matching A A B : { 5 }

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

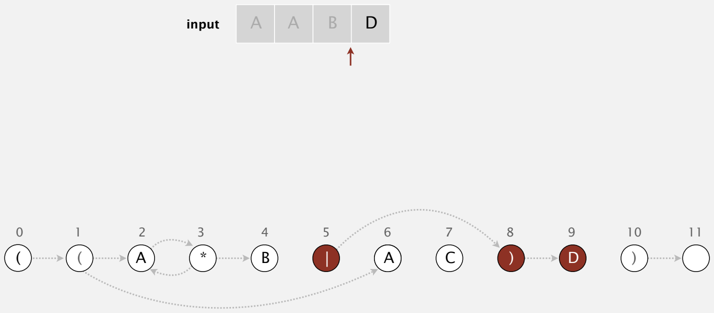


set of states reachable via ε-transitions after matching A A B

NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

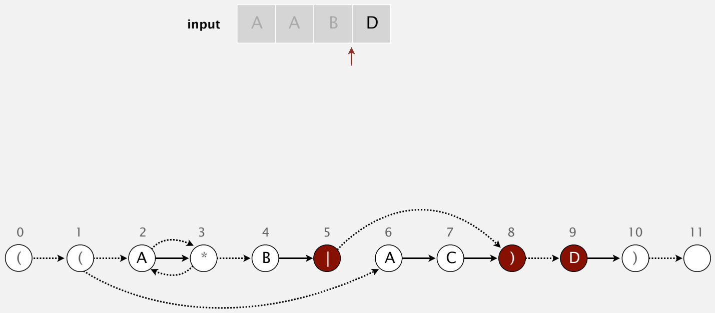


set of states reachable via ε-transitions after matching A A B : { 5, 8, 9 }

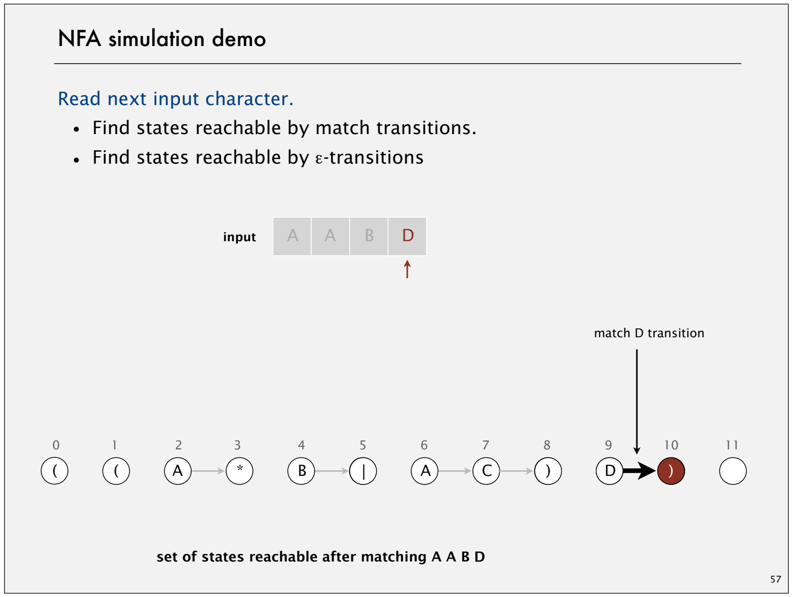
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions



set of states reachable via ε-transitions after matching A A B : { 5, 8, 9 }

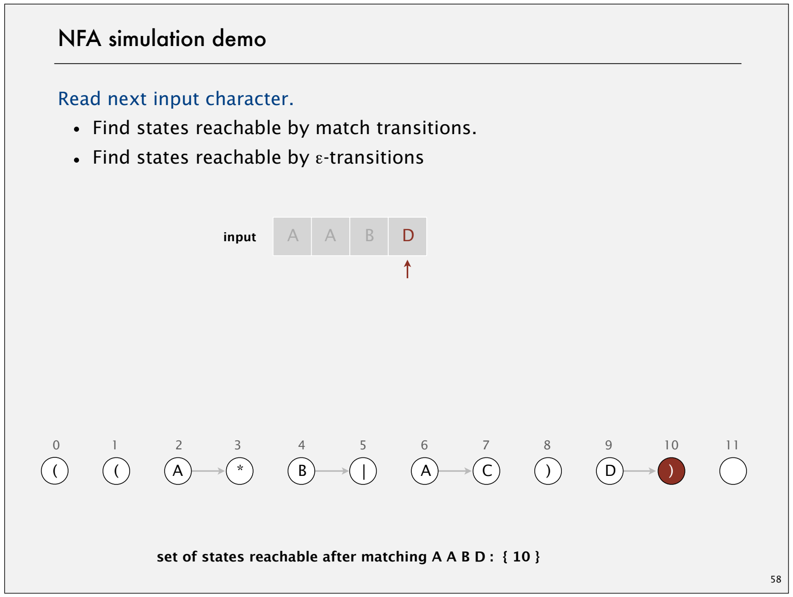


NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

set of states reachable after matching A A B D

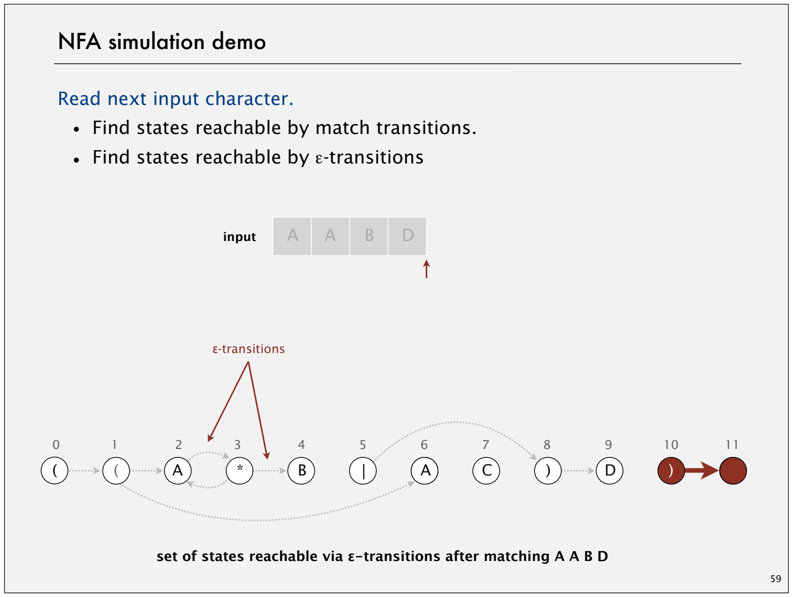


NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

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

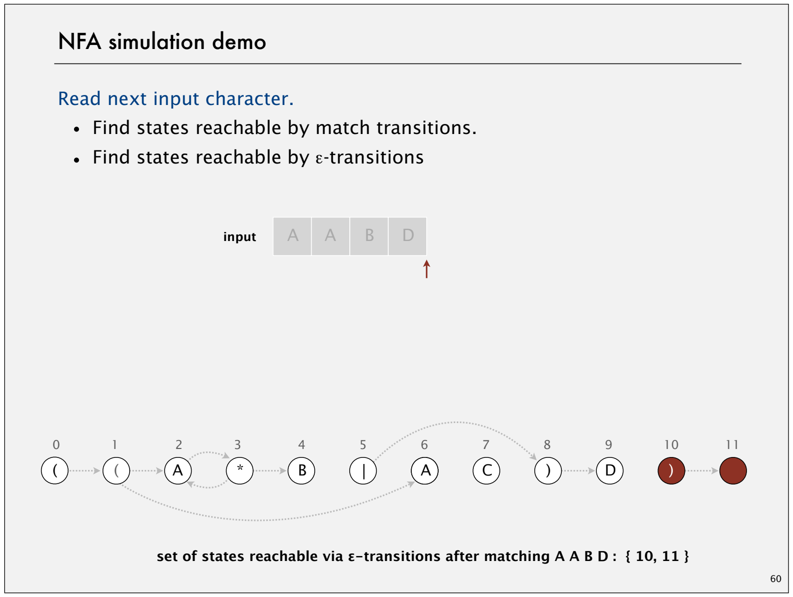


NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by ε-transitions

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



NFA simulation demo

Read next input character.

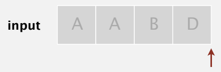
- Find states reachable by match transitions.
- Find states reachable by ε-transitions

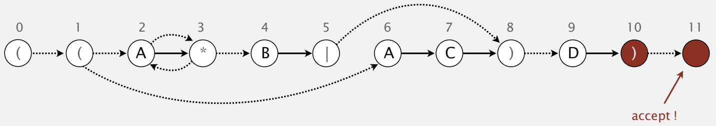
set of states reachable via ε-transitions after matching A A B D : { 10, 11 }

NFA simulation demo

When no more input characters:

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



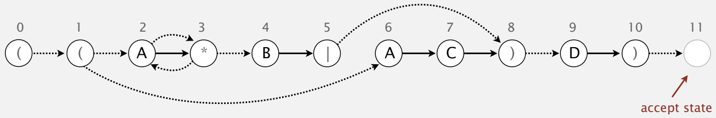


set of states reachable : { 10, 11 }

NFA simulation: analysis

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

Pf. For each of the N text characters, we iterate through a set of states of size no more than M and run DFS on the graph of ε-transitions.
[The NFA construction we will consider ensures the number of edges ≤ $3M$.]



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

Algorithms

Robert Sedgewick | Kevin Wayne

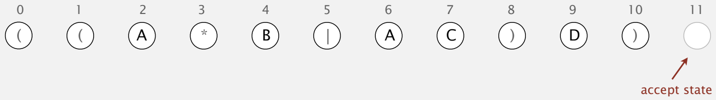
http://algs4.cs.princeton.edu

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.



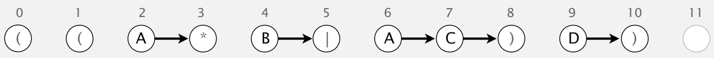
NFA corresponding to the pattern ((A * B | A C) D)

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

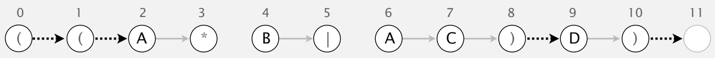
Metacharacters. ( ) . * |



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.

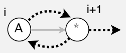


NFA corresponding to the pattern ((A * B | A C) D)

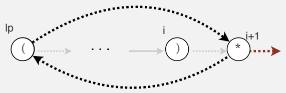
Building an NFA corresponding to an RE

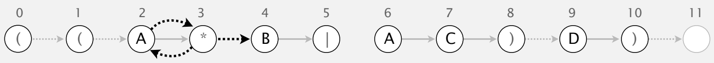
Closure. Add three ε-transition edges for each * operator.

singe-character closure



closure expression



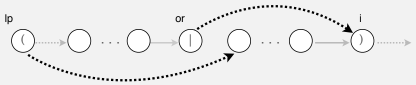


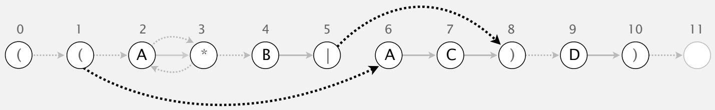
NFA corresponding to the pattern ((A * B | A C) D)

Building an NFA corresponding to an RE

2-way or. Add two ε-transition edges for each | operator.

2-way or expression





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

Building an NFA corresponding to an RE

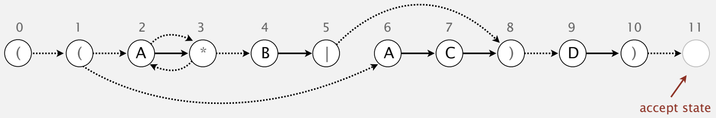
States. Include a state for each symbol in the RE, plus an accept state.

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

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

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

2-way or. Add two ε-transition edges for each | operator.



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

Regular expression: quiz 4

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

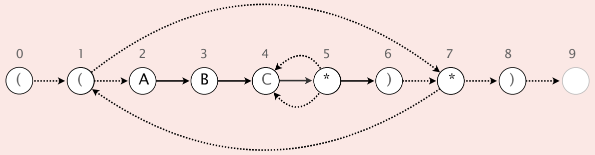
one or more occurrence

A. Remove ε-transition edge 1→7.

B. Remove ε-transition edge 7→1.

C. Remove ε-transition edges 1→7 and 7→1.

D. *I don't know.*



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

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

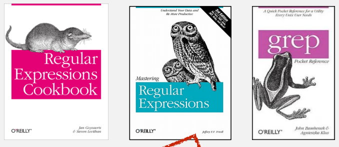
5.4 Regular Expressions

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

Industrial-strength grep implementation

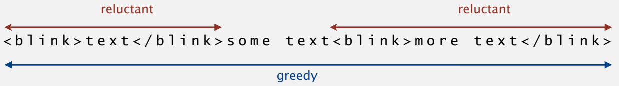
To complete the implementation:

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



Subtle differences in syntax

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



Regular expressions in the wild

Broadly applicable programmer's tool.

- Originated in Unix in the 1970s.
- Built in to many tools: grep, egrep, emacs, ….

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

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

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

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

```
% 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.w3.org
http://www.cs.princeton.edu
http://drupal.org
http://csguide.cs.princeton.edu
http://www.cs.princeton.edu
http://www.princeton.edu
```

Harvesting information

RE pattern matching is implemented in Java's `java.util.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.
- Attacker 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
```

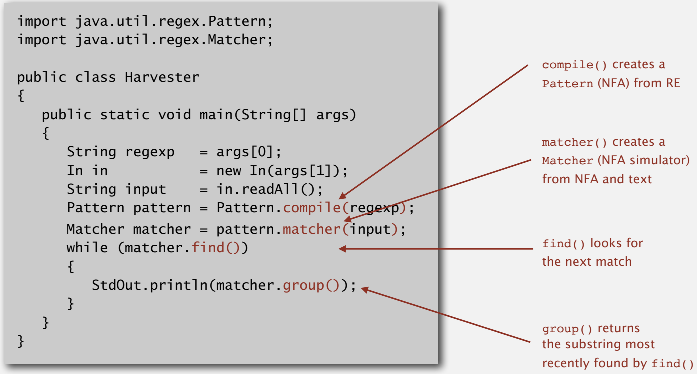
Some non-regular languages.

- 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.

Harvesting information in Java

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()

Regular expressions in context

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

Compiler. A program that translates a program to machine code.

- KMP string ⇒ DFA.
- grep RE ⇒ NFA.
- javac Java language ⇒ Java byte code.

	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

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.
- Attacker can use such addresses to DOS a mail server.

Summary of pattern-matching algorithms

Programmer.

- Implement substring search via DFA simulation.
- Implement RE pattern matching via NFA simulation.

Theoretician.

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

You.

- Core CS principles provide useful tools that you can exploit now.
- REs and NFAs provide introduction to theoretical CS.

Example of essential paradigm in computer science.

- Build the right intermediate abstractions.
- Solve important practical problems.