# BBM 202 - ALGORITHMS

HACETTEPE UNIVERSITY

DEPT. OF COMPUTER ENGINEERING

ERKUT ERDEM

# REGULAR EXPRESSIONS

May. 8, 2014

**Acknowledgement:** The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.

# TODAY

- Regular Expressions
- REs and NFAs
- NFA simulation
- NFA construction
- Applications

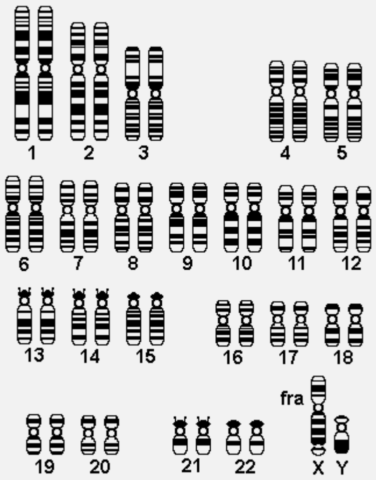
# Pattern matching

Substring search. Find a single string in text.

Pattern matching. Find one of a specified set of strings in text.

Ex. [genomics]

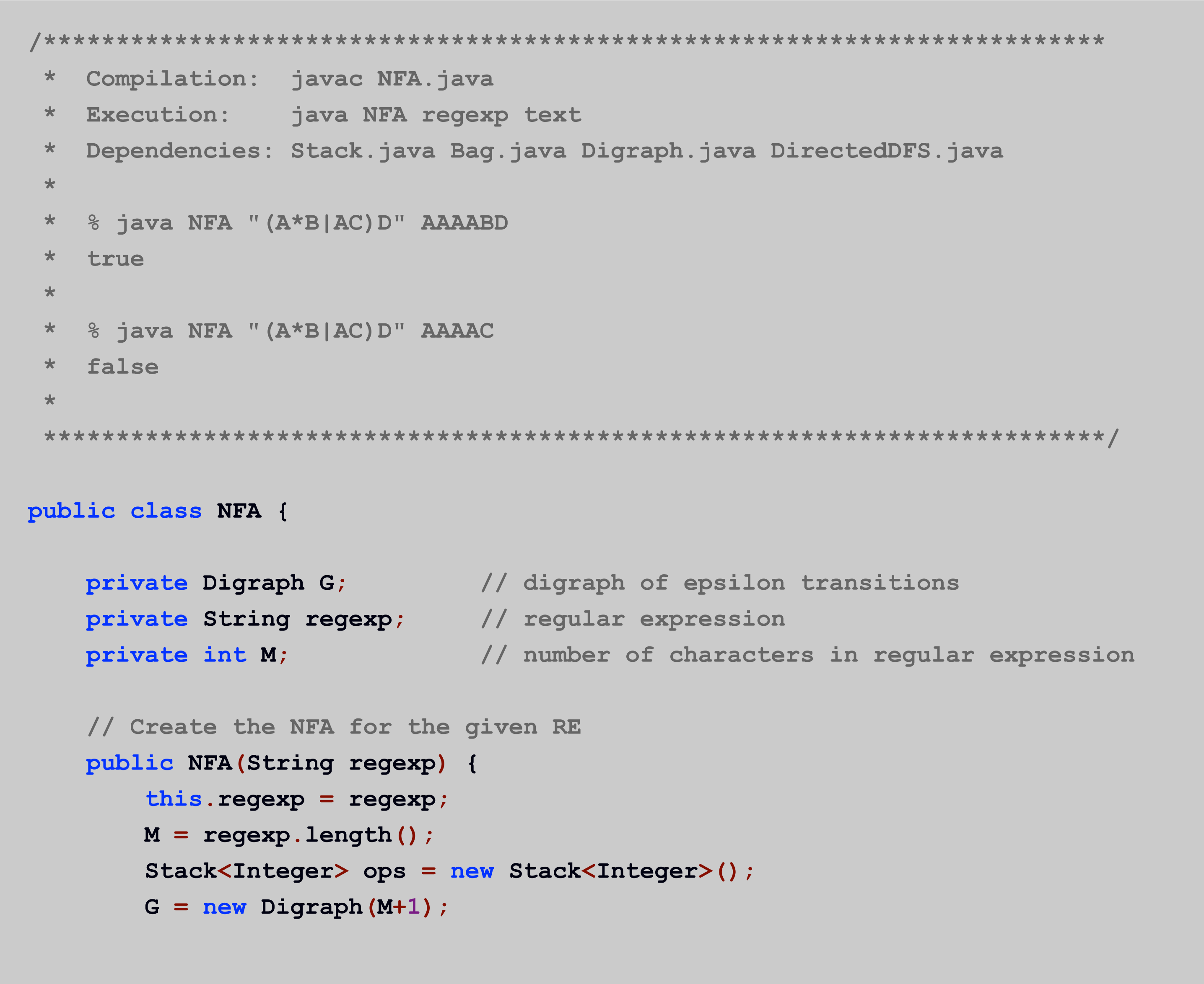
- Fragile X syndrome is a common cause of mental retardation.
- Human genome contains triplet repeats of `CGG` or `AGG`, bracketed by `GCG` at the beginning and `CTG` at the end.
- Number of repeats is variable, and correlated with syndrome.



**pattern** `GCG(CGG|AGG)*CTG`

**text** `GCGGCGTGTGTGCGAGAGAGTGGGTTTAAAGCTGGCGCGGAGGCGGCTGGCGCGGAGGCTG`

# 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

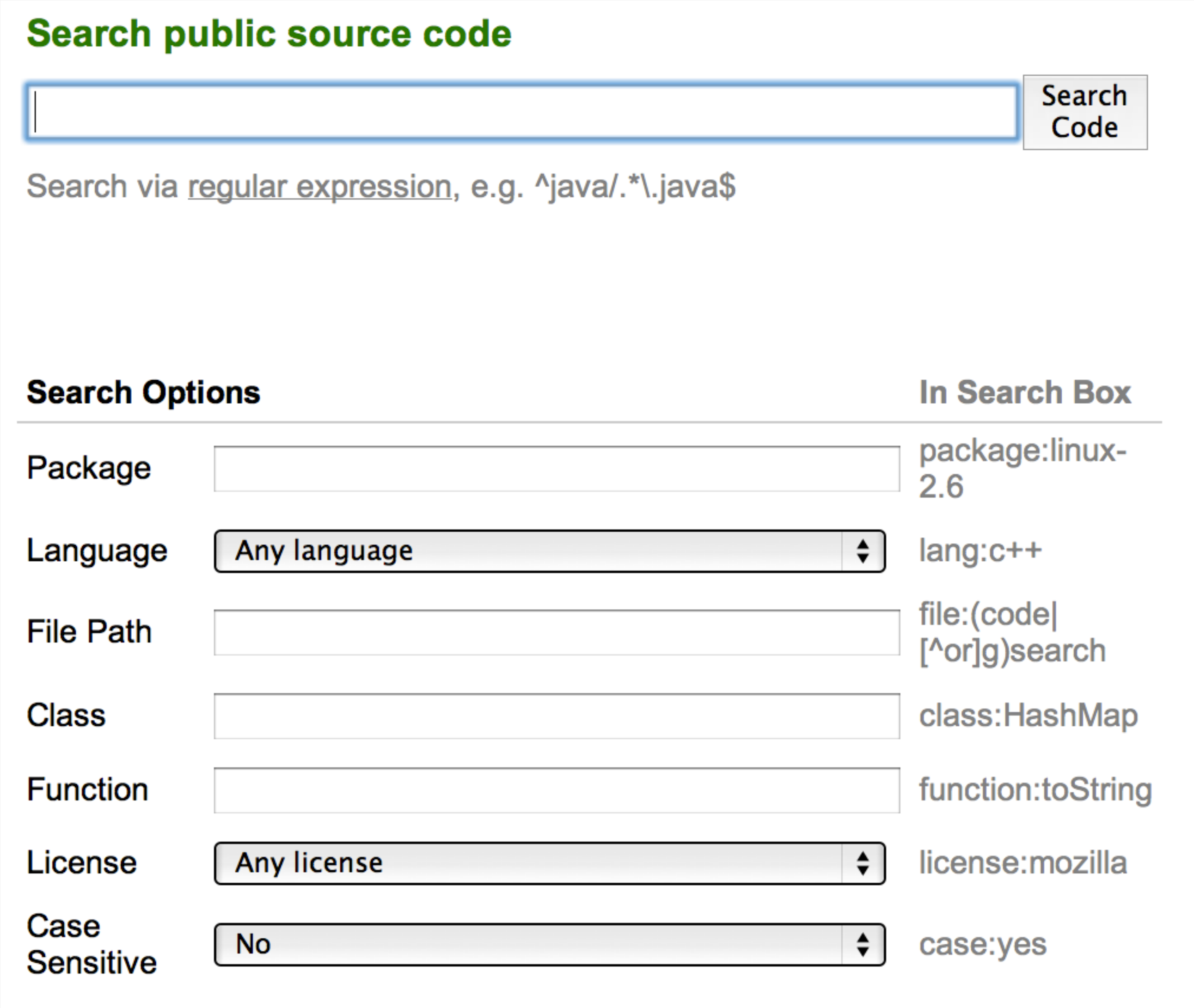
**input**

Ada
Asm
Applescript
Awk
Bat
Bib
Bison
C/C++
C#
Cobol
Caml
Changelog
Css
D
Erlang
Flex
Fortran
GLSL
Haskell
Html
Java
Javalog
Javascript
Latex
Lisp
Lua
⋮

**output**

HTML
XHTML
LATEX
MediaWiki
ODF
TEXINFO
ANSI
DocBook

# 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 |

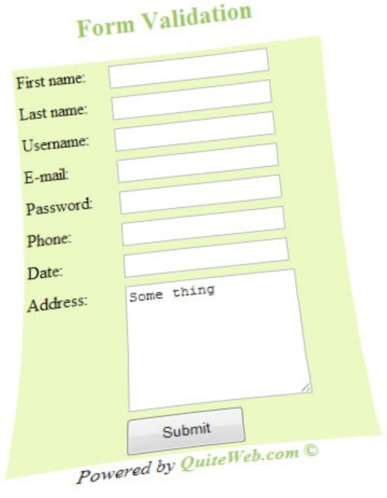
http://code.google.com/p/chromium/source/search

# Pattern matching: applications

## Test if a string matches some pattern.

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

...



Form Validation

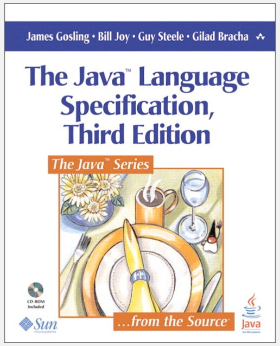
First name:
Last name:
Username:
E-mail:
Password:
Phone:
Date:
Address: Some thing

Powered by QuiteWeb.com ©

## 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.

...



James Gosling • Bill Joy • Guy Steele • Gilad Bracha

The Java™ Language Specification, Third Edition

# Regular expressions

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

a “language”

| operation | order | example RE | matches | does not match |
|---|---|---|---|---|
| concatenation | 3 | `AABAAB` | `AABAAB` | every other string |
| or | 4 | `AA \| BAAB` | `AA`<br>`BAAB` | every other string |
| closure | 2 | `AB*A` | `AA`<br>`ABBBBBBBBA` | `AB`<br>`ABABA` |
| parentheses | 1 | `A(A\|B)AAB` | `AAAAB`<br>`ABAAB` | every other string |
| | | `(AB)*A` | `A`<br>`ABABABABABA` | `AA`<br>`ABBA` |

# Regular expression shortcuts

Additional operations are often added for convenience.

| operation | example RE | matches | does not match |
|---|---|---|---|
| wildcard | `.U.U.U.` | `CUMULUS`<br>`JUGULUM` | `SUCCUBUS`<br>`TUMULTUOUS` |
| character class | `[A-Za-z][a-z]*` | `word`<br>`Capitalized` | `camelCase`<br>`4illegal` |
| at least 1 | `A(BC)+DE` | `ABCDE`<br>`ABCBCDE` | `ADE`<br>`BCDE` |
| exactly k | `[0-9]{5}-[0-9]{4}` | `08540-1321`<br>`19072-5541` | `111111111`<br>`166-54-111` |
| complement | `[^AEIOU]{6}` | `RHYTHM` | `DECADE` |

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

# Regular expression examples

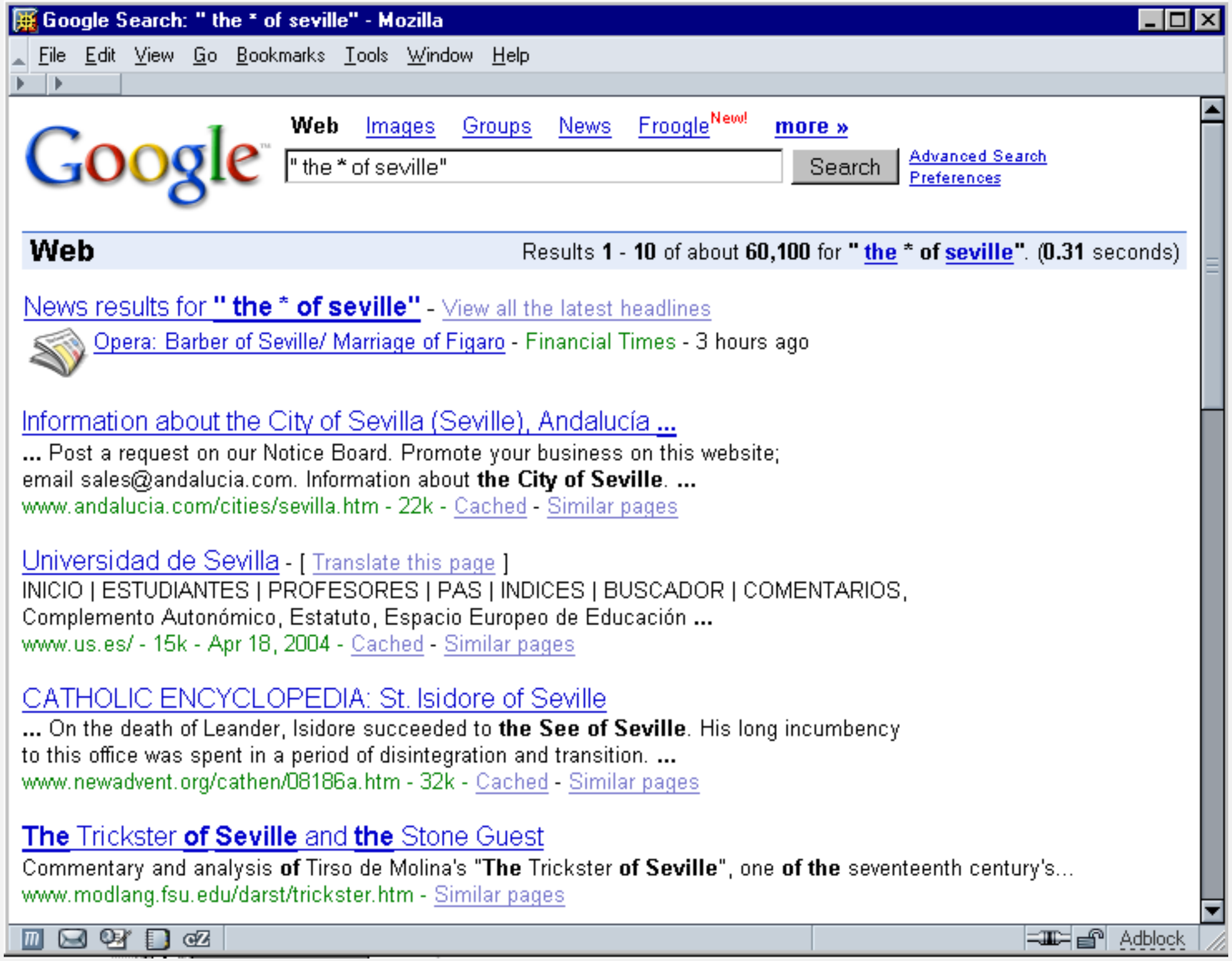
RE notation is surprisingly expressive

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

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

# Can the average web surfer learn to use REs?

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



" the * of seville"

**Web** Results **1** - **10** of about **60,100** for " **the** * **of seville**". (**0.31** seconds)

News results for " **the** * **of seville**" - View all the latest headlines

Opera: Barber of Seville/ Marriage of Figaro - Financial Times - 3 hours ago

Information about the City of Sevilla (Seville), Andalucía ...

... Post a request on our Notice Board. Promote your business on this website; email sales@andalucia.com. Information about **the City of Seville**. ...

www.andalucia.com/cities/sevilla.htm - 22k - Cached - Similar pages

Universidad de Sevilla - [ Translate this page ]

INICIO | ESTUDIANTES | PROFESORES | PAS | INDICES | BUSCADOR | COMENTARIOS, Complemento Autonómico, Estatuto, Espacio Europeo de Educación ...

www.us.es/ - 15k - Apr 18, 2004 - Cached - Similar pages

CATHOLIC ENCYCLOPEDIA: St. Isidore of Seville

... On the death of Leander, Isidore succeeded to **the See of Seville**. His long incumbency to this office was spent in a period of disintegration and transition. ...

www.newadvent.org/cathen/08186a.htm - 32k - Cached - Similar pages

**The** Trickster **of Seville** and **the** Stone Guest

Commentary and analysis **of** Tirso de Molina's "**The** Trickster **of Seville**", one **of the** seventeenth century's...

www.modlang.fsu.edu/darst/trickster.htm - Similar pages

# 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** **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.

# 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.

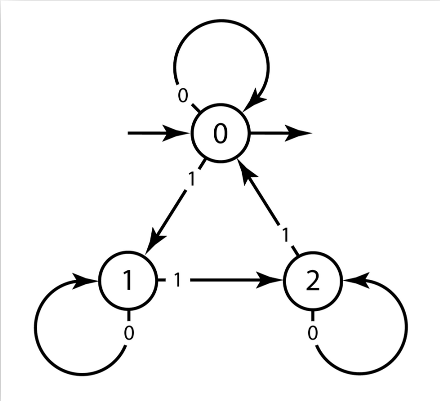
- For any DFA, there exists a RE that describes the same set of strings.
- For any RE, there exists a DFA that recognizes the same set of strings.

RE

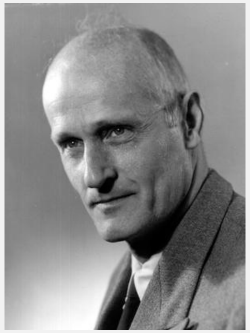
`0* | (0*10*10*10*)*`

**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.

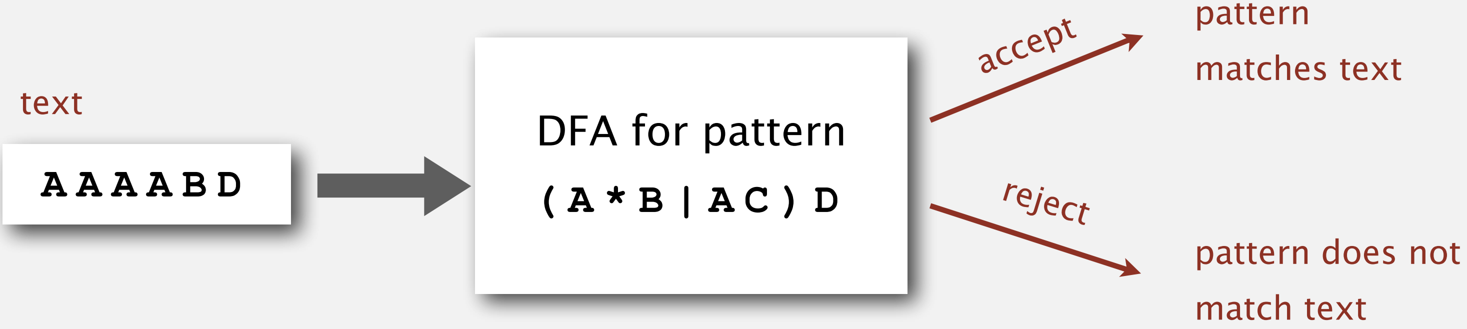
- No backup in text input stream.
- Linear-time guarantee.

**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)

**Ken Thompson**
**Turing Award '83**

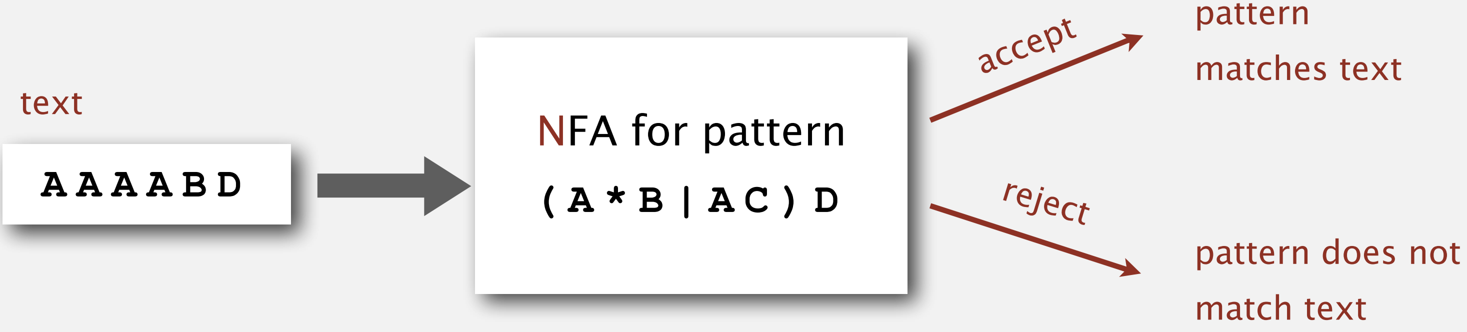
Overview is similar to KMP.

- 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|AC)D`

accept

pattern matches text

reject

pattern does not match text

Q. What is an NFA?

# Nondeterministic finite-state automata

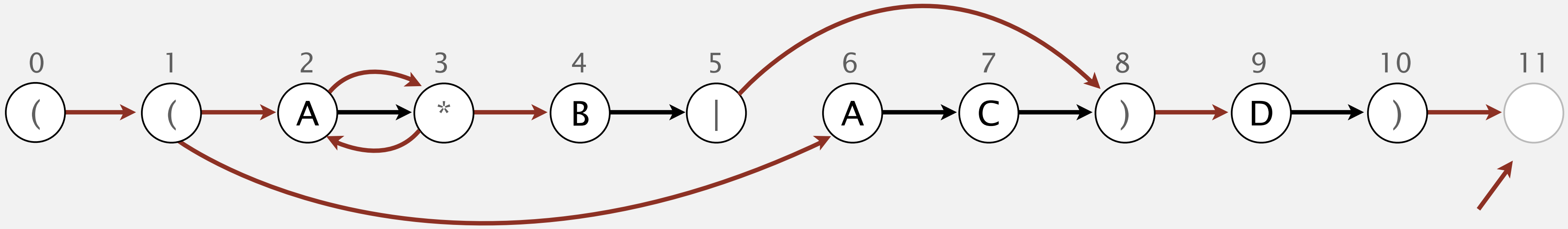
## Regular-expression-matching NFA.

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

after scanning all text characters

## Nondeterminism.

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



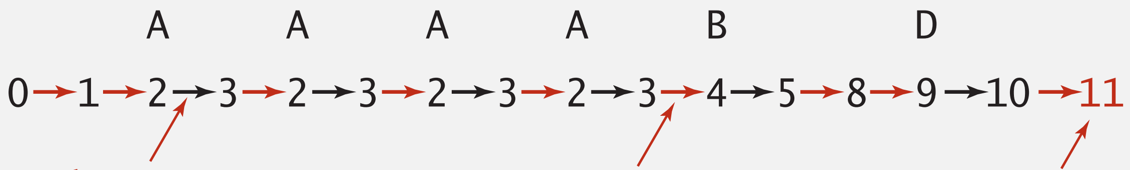
accept state

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

# Nondeterministic finite-state automata

Q. Is `AAAABD` matched by NFA?

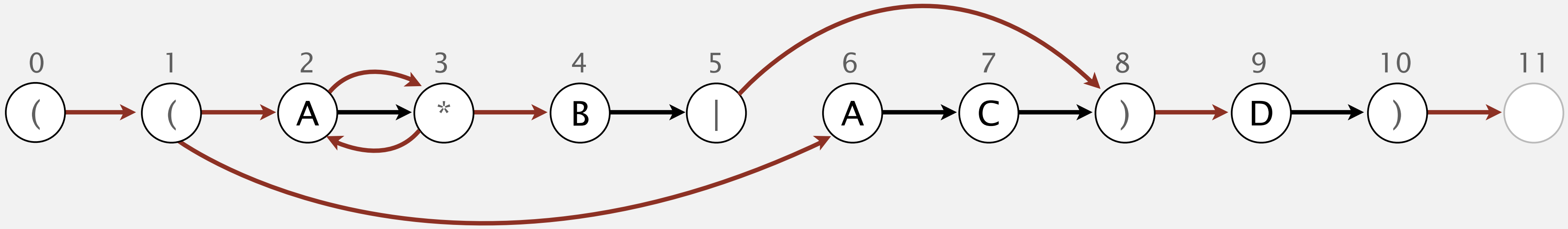
A. Yes, because some sequence of legal transitions ends in state `11`.



*match transition: scan to next input character and change state*

*ϵ-transition: change state with no match*

*accept state reached and all text characters scanned: pattern found*



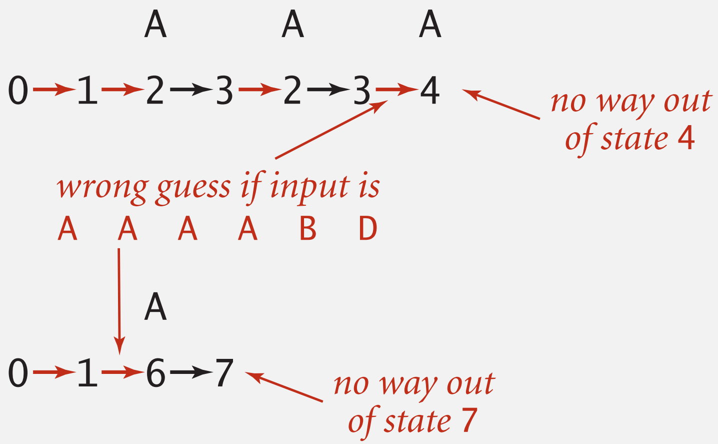
**NFA corresponding to the pattern ( ( A * B | A C ) D )**

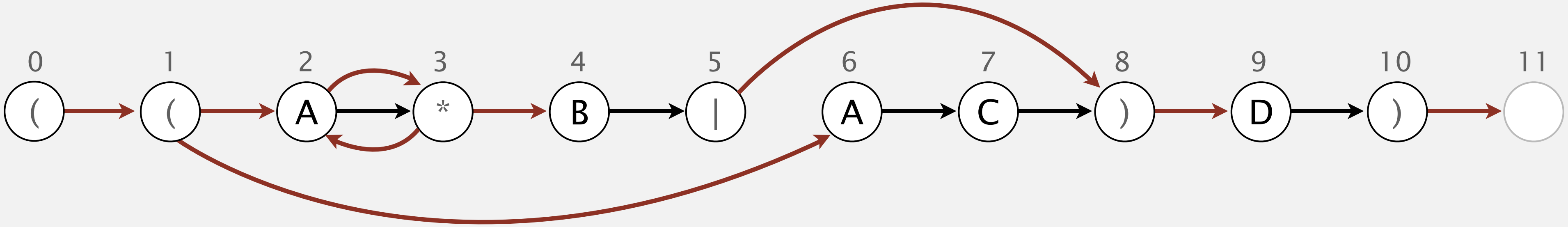
# Nondeterministic finite-state automata

Q. Is `AAAABD` matched by NFA?

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

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





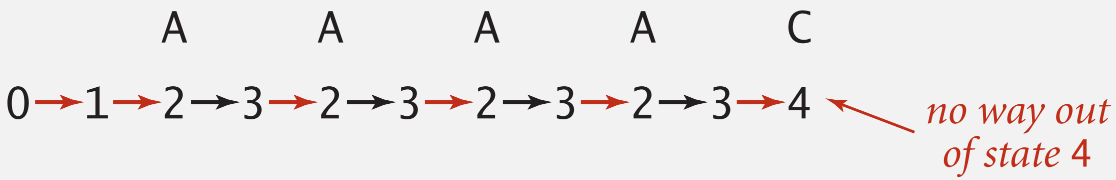
**NFA corresponding to the pattern ( ( A * B | A C ) D )**

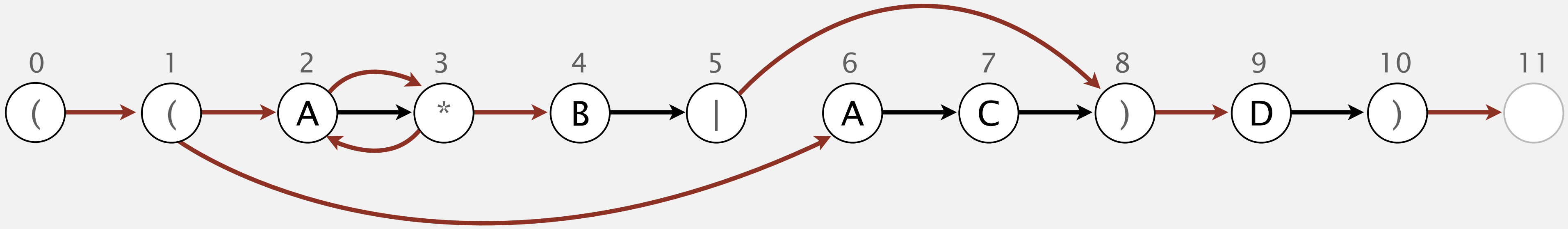
# Nondeterministic finite-state automata

Q. Is `AAAC` matched by NFA?

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

[ but need to argue about all possible sequences ]





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

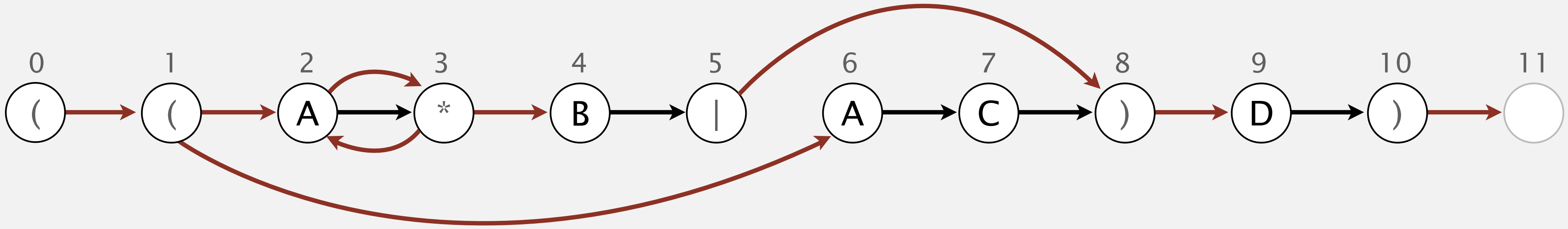
# Nondeterminism

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

DFA. Deterministic $\Rightarrow$ exactly one applicable transition.

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

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



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

# REGULAR EXPRESSIONS

- REs and NFAs
- **NFA simulation**
- NFA construction
- Applications

# NFA representation

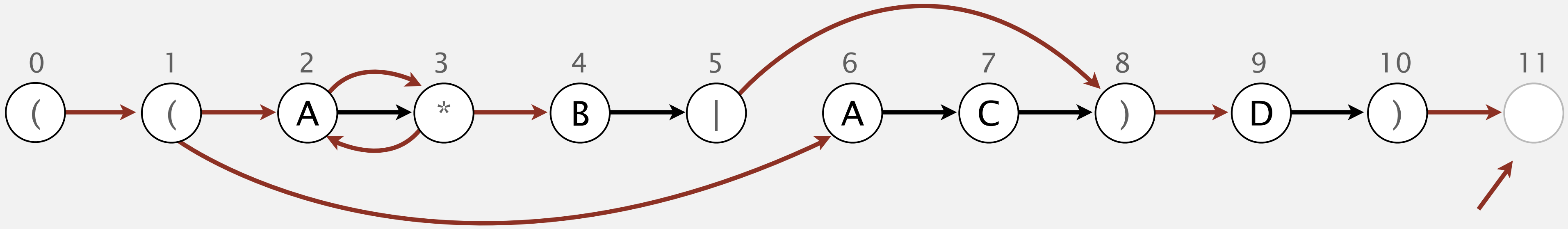
**State names.** Integers from $0$ to $M$.

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`



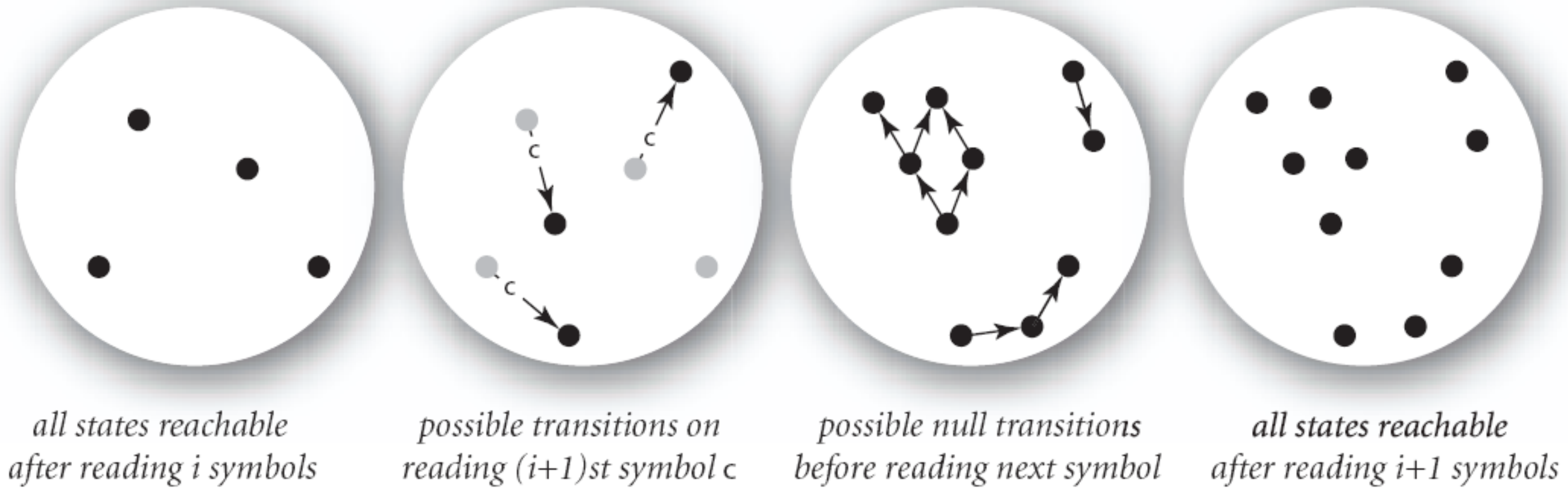
accept state

**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.

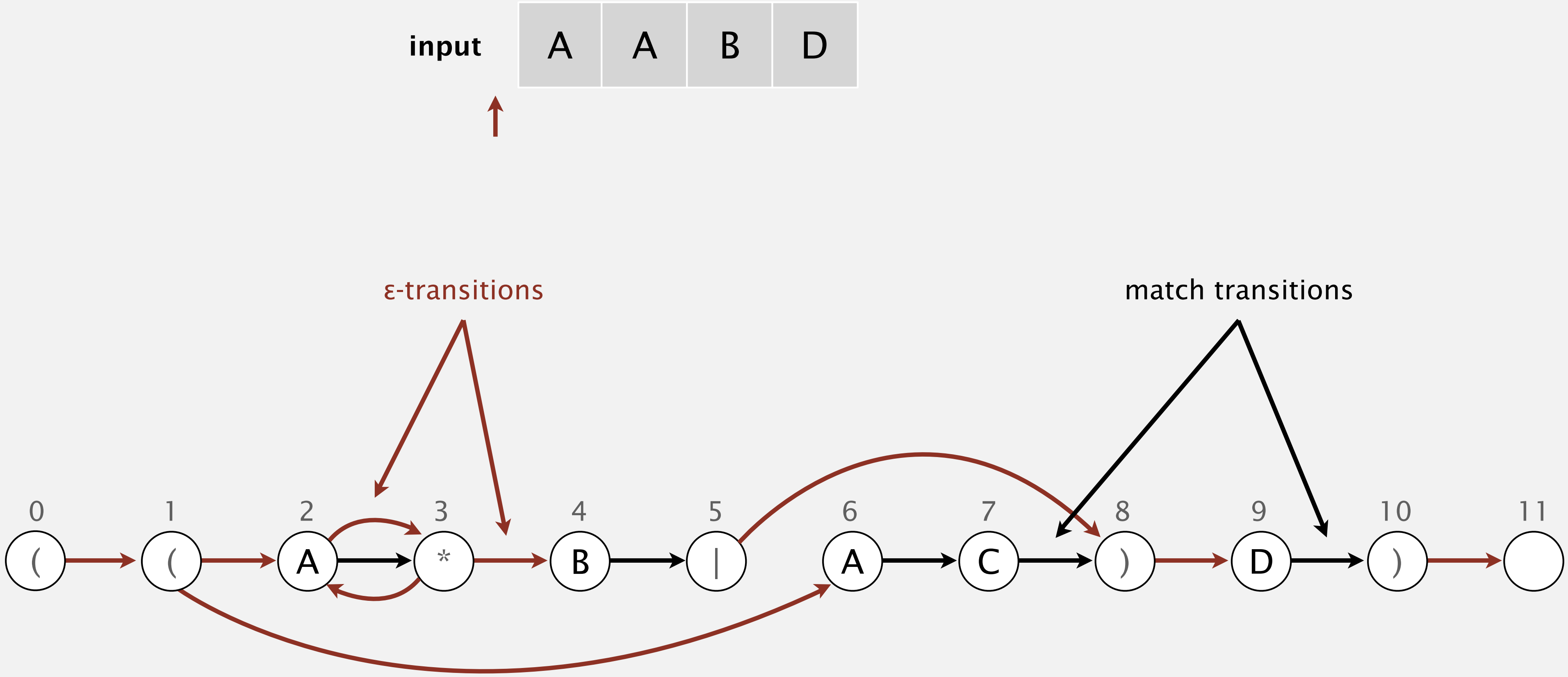


*One step in simulating an NFA*

Q. How to perform reachability?

# NFA simulation

Goal. Check whether input matches pattern.



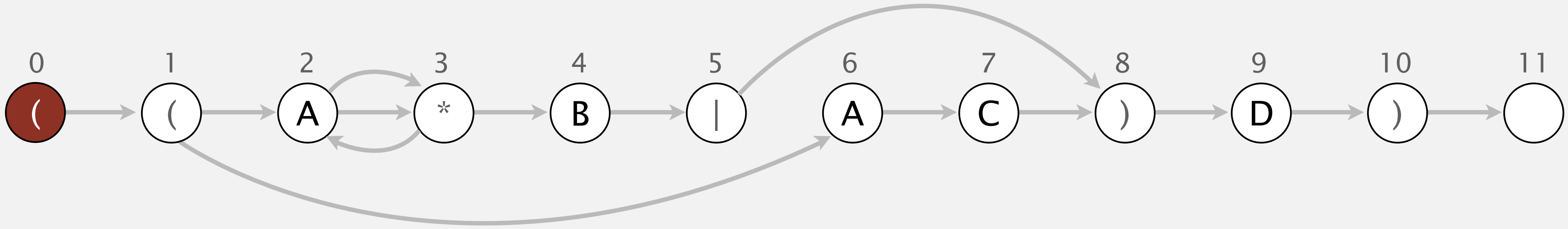
**NFA corresponding to the pattern ( ( A * B | A C ) D )**

# NFA simulation

## Read next input character.

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

| input | A | A | B | D |
|---|---|---|---|---|



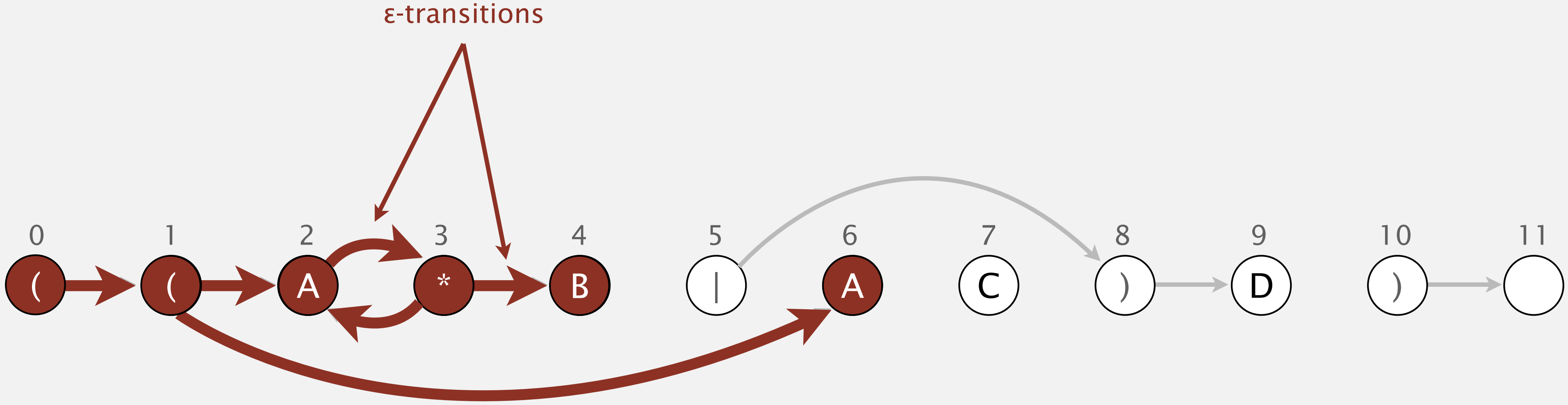
**set of states reachable from start: 0**

# NFA simulation

## 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 from start**

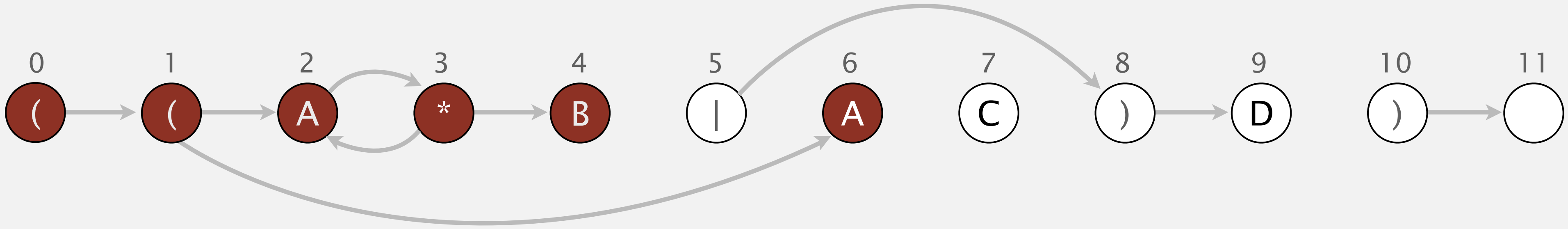
# NFA simulation

## Read next input character.

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

**input**

| A | A | B | D |
|---|---|---|---|

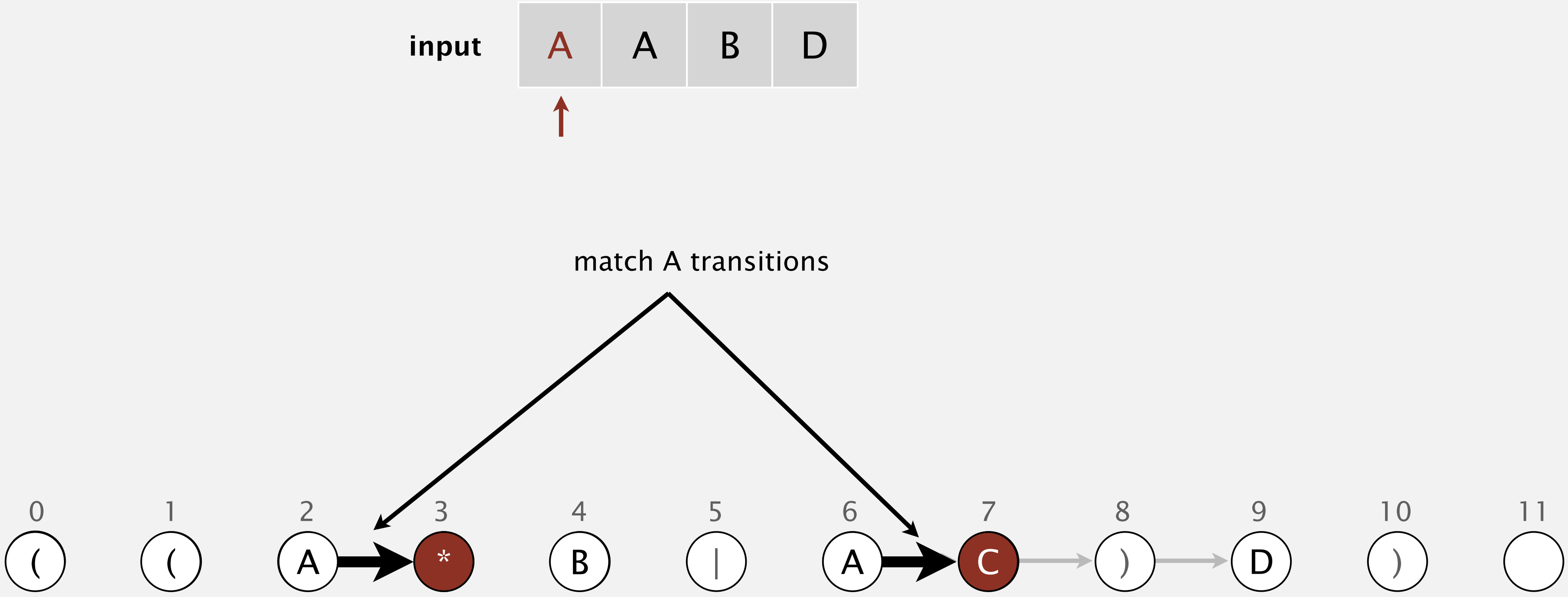


**set of states reachable via ε-transitions from start : { 0, 1, 2, 3, 4, 6 }**

# NFA simulation

## Read next input character.

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

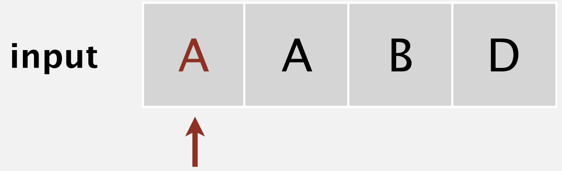


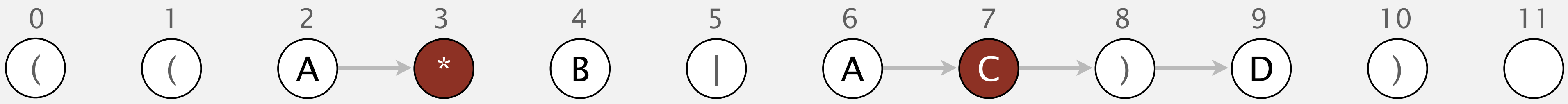
**set of states reachable after matching A**

# NFA simulation

## Read next input character.

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



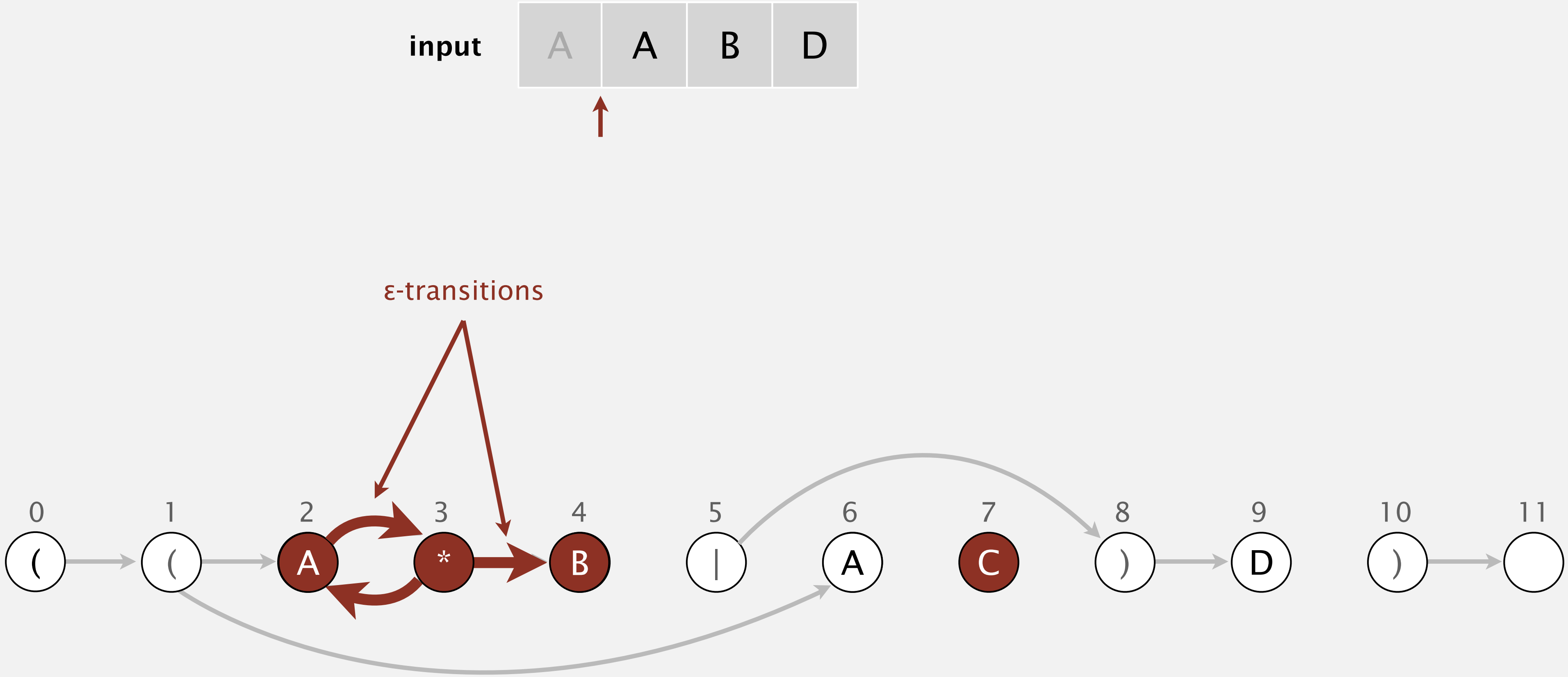


**set of states reachable after matching A : { 3, 7 }**

# NFA simulation

## Read next input character.

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

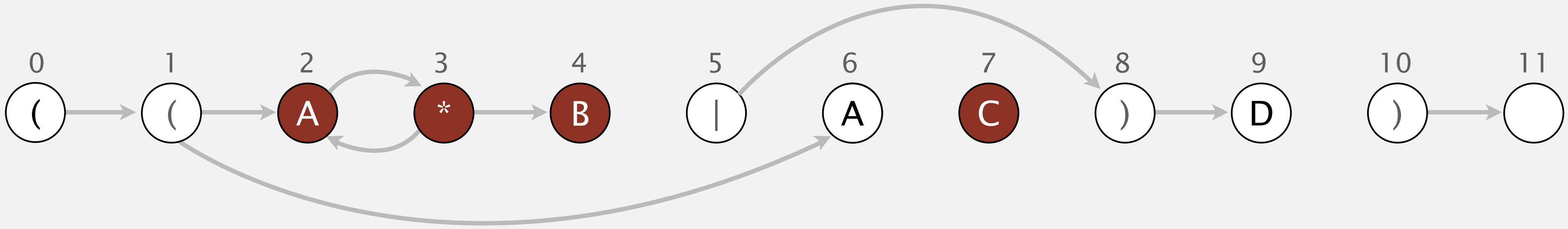


**set of states reachable via ε-transitions after matching A**

# NFA simulation

## Read next input character.

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

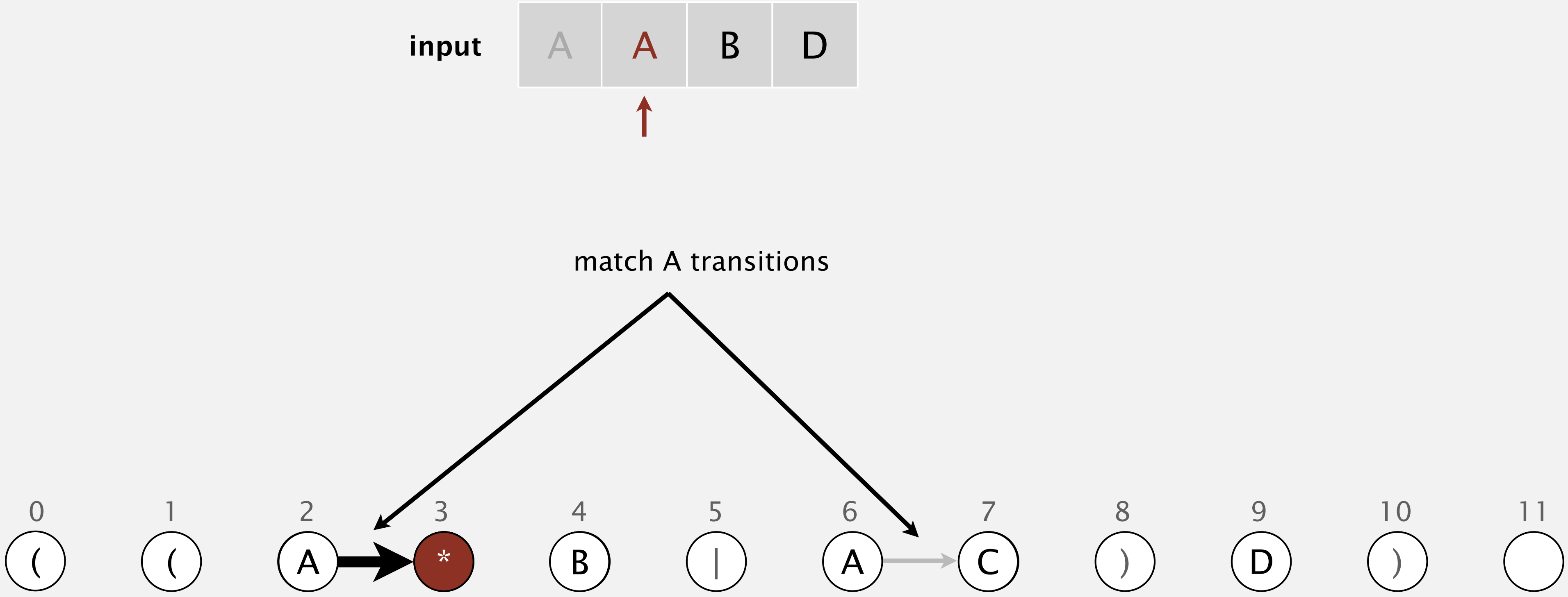


**set of states reachable via ε-transitions after matching A : { 2, 3, 4, 7 }**

# NFA simulation

## Read next input character.

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

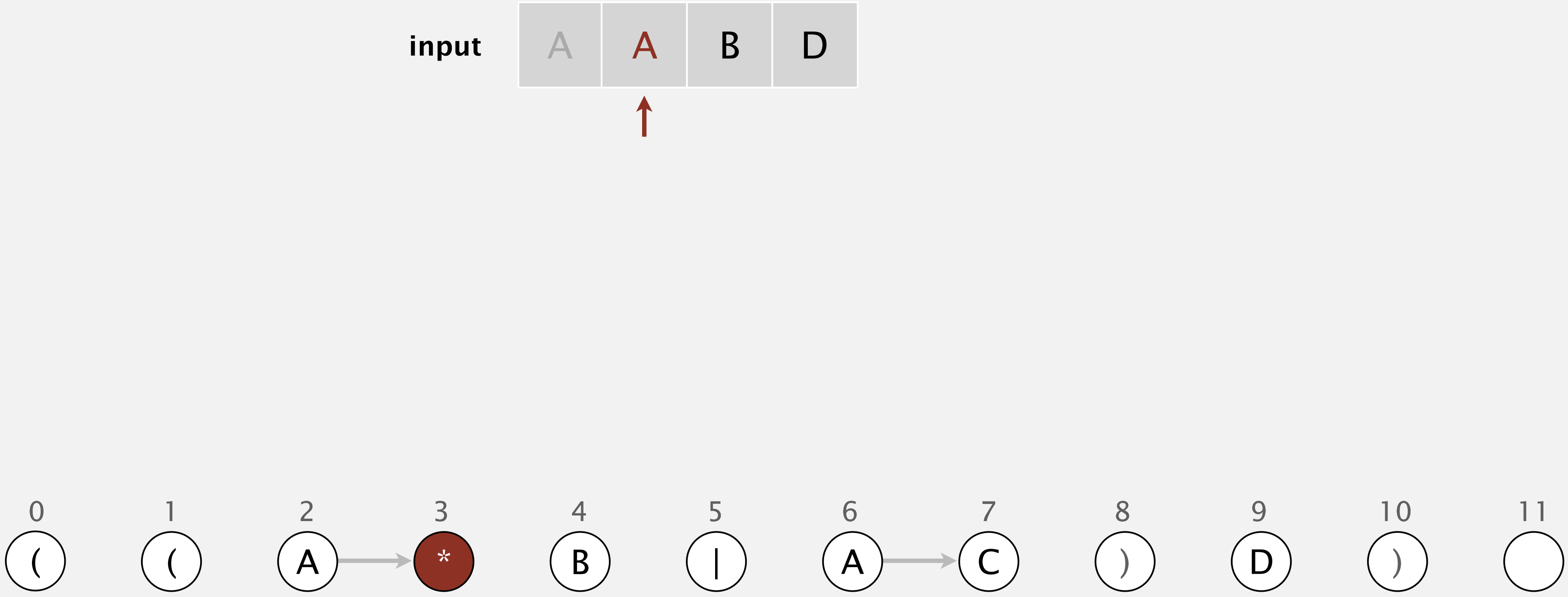


**set of states reachable after matching A A**

# NFA simulation

## Read next input character.

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

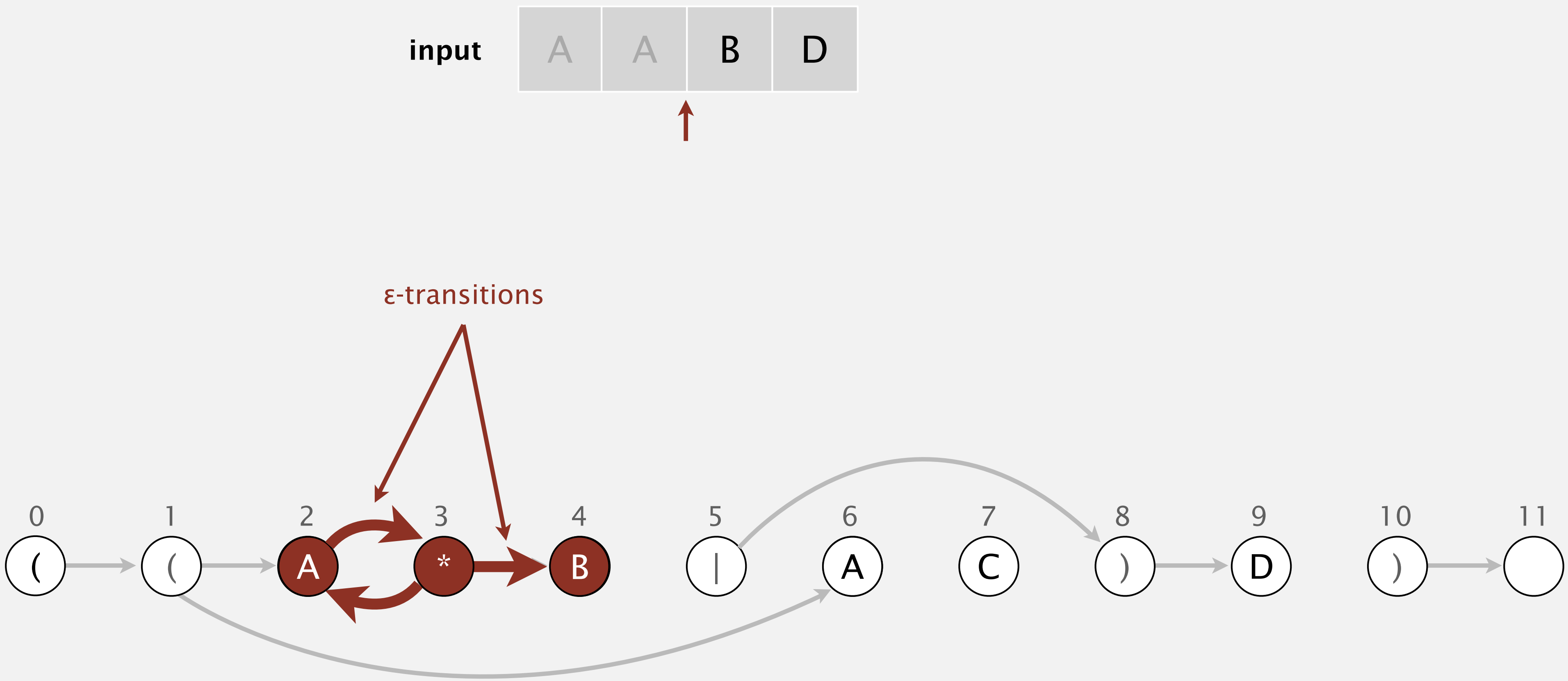


**set of states reachable after matching A A : { 3 }**

# NFA simulation

## Read next input character.

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

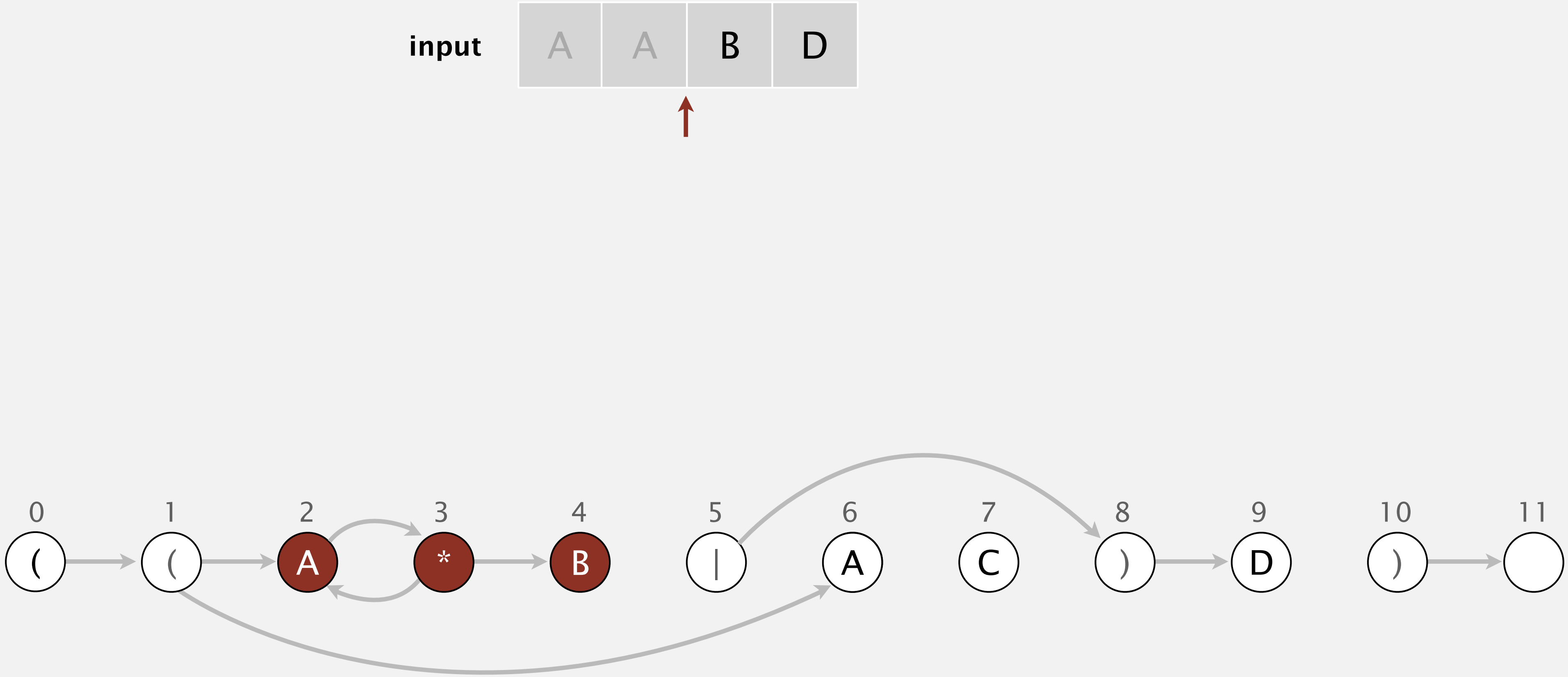


**set of states reachable via ε−transitions after matching A A**

# NFA simulation

## Read next input character.

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

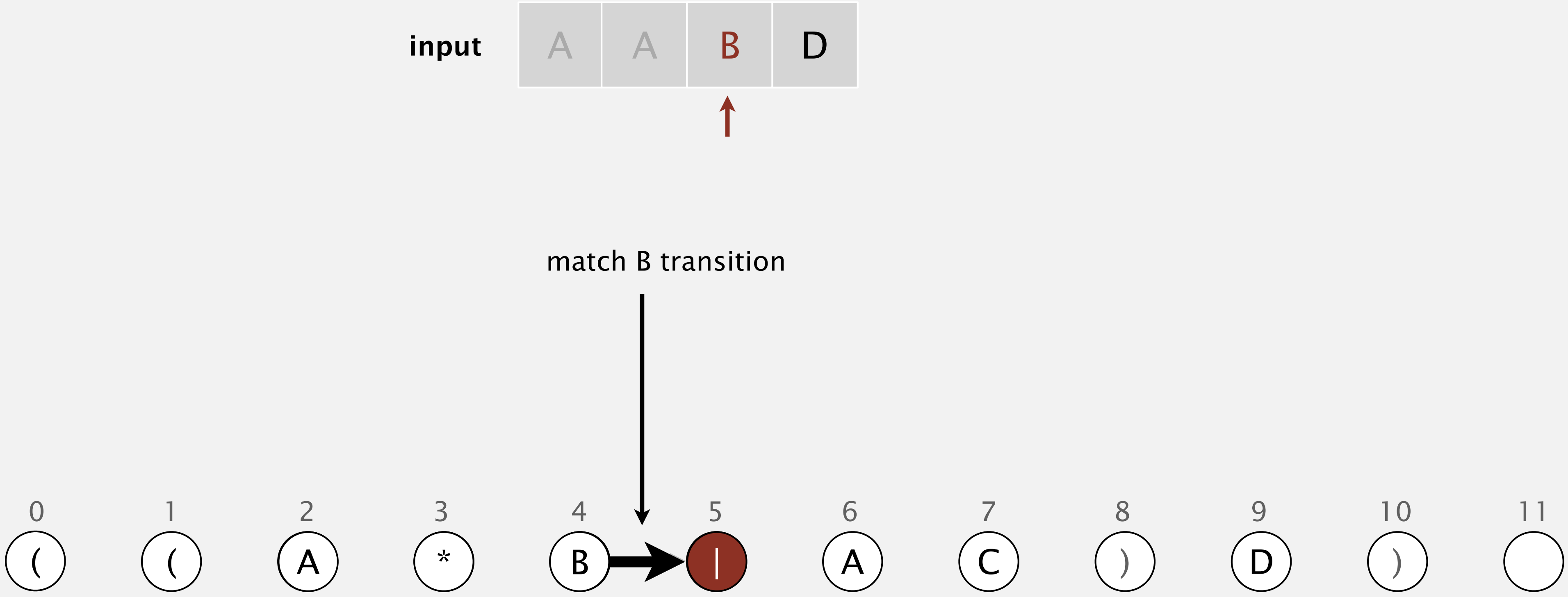


**set of states reachable via ε−transitions after matching A A : { 2, 3, 4 }**

# NFA simulation

## Read next input character.

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

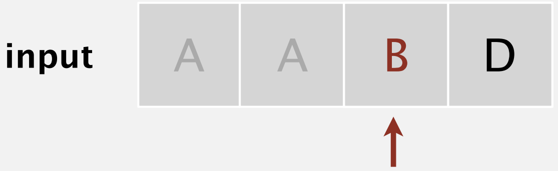


**set of states reachable after matching A A B**

# NFA simulation

## Read next input character.

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

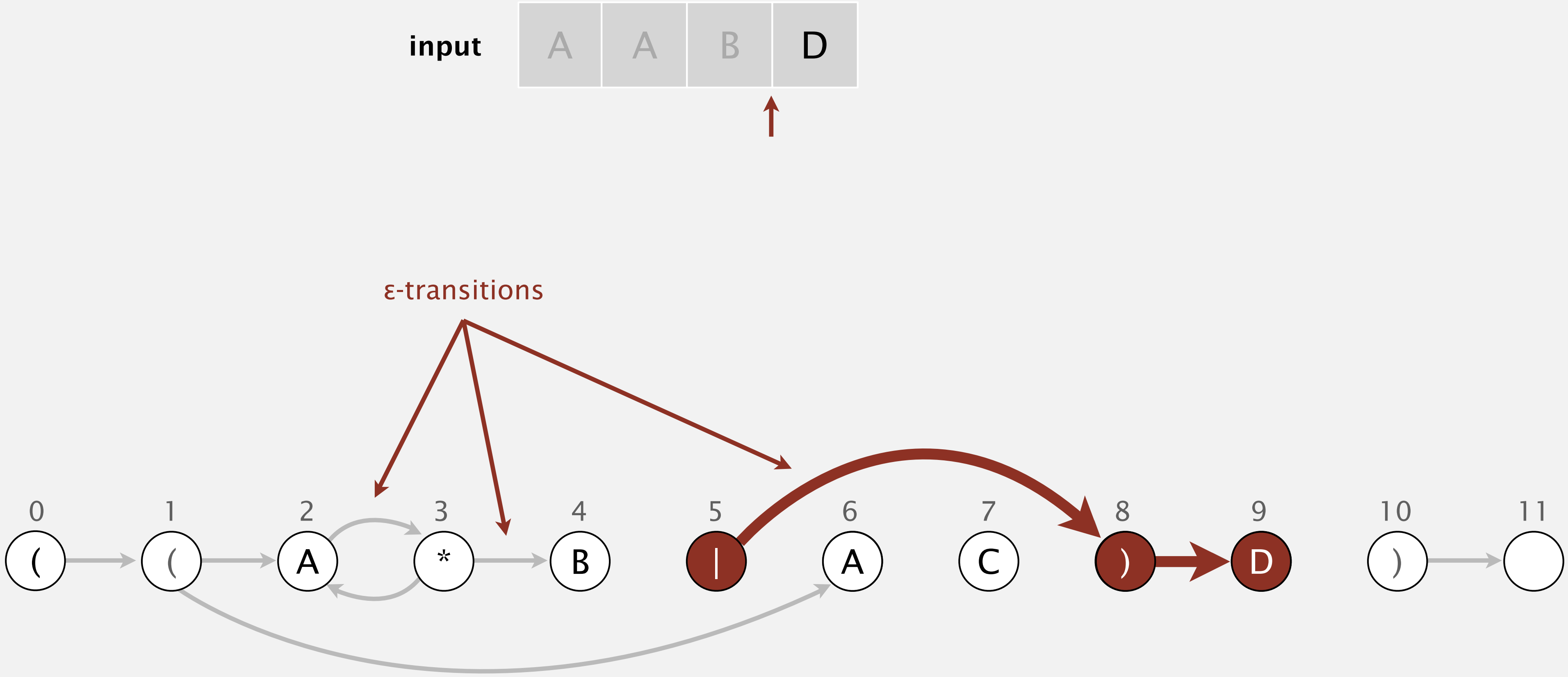


**set of states reachable after matching A A B : { 5 }**

# NFA simulation

## Read next input character.

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

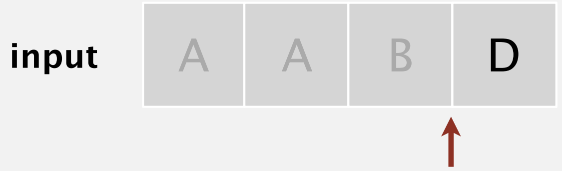


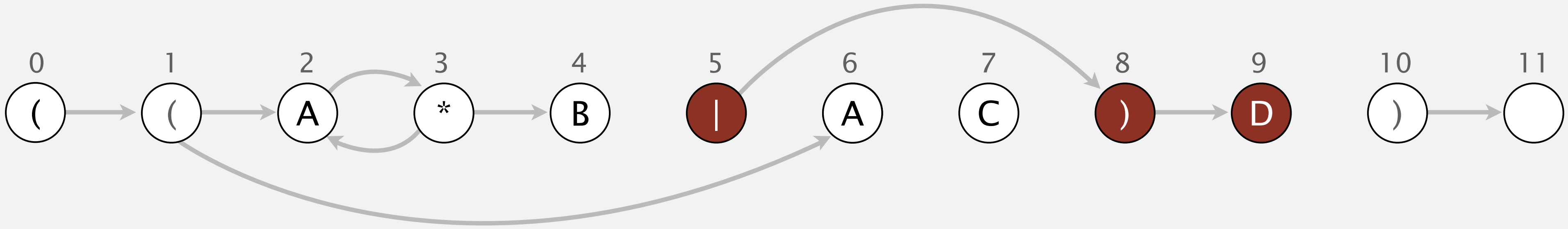
**set of states reachable via ε-transitions after matching A A B**

# NFA simulation

## Read next input character.

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



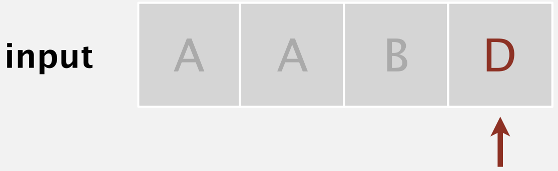


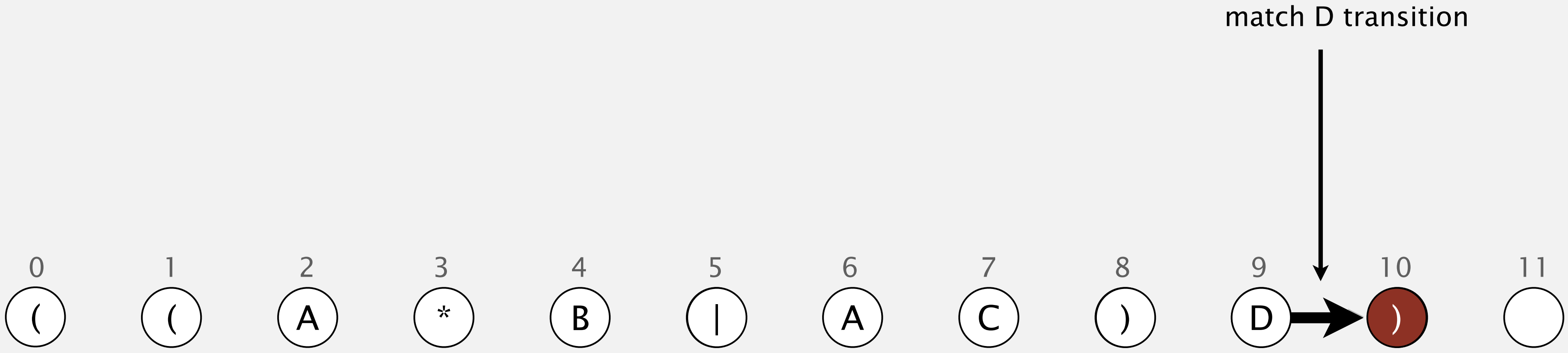
**set of states reachable via ε-transitions after matching A A B : { 5, 8, 9 }**

# NFA simulation

## Read next input character.

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



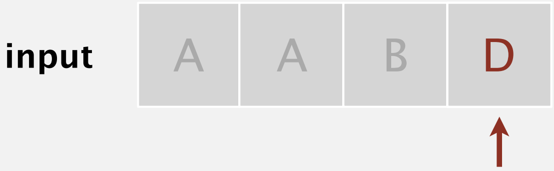


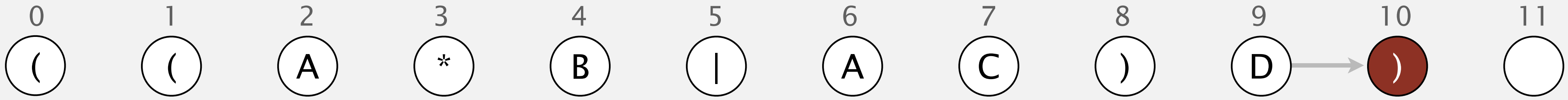
**set of states reachable after matching A A B D**

# NFA simulation

## Read next input character.

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



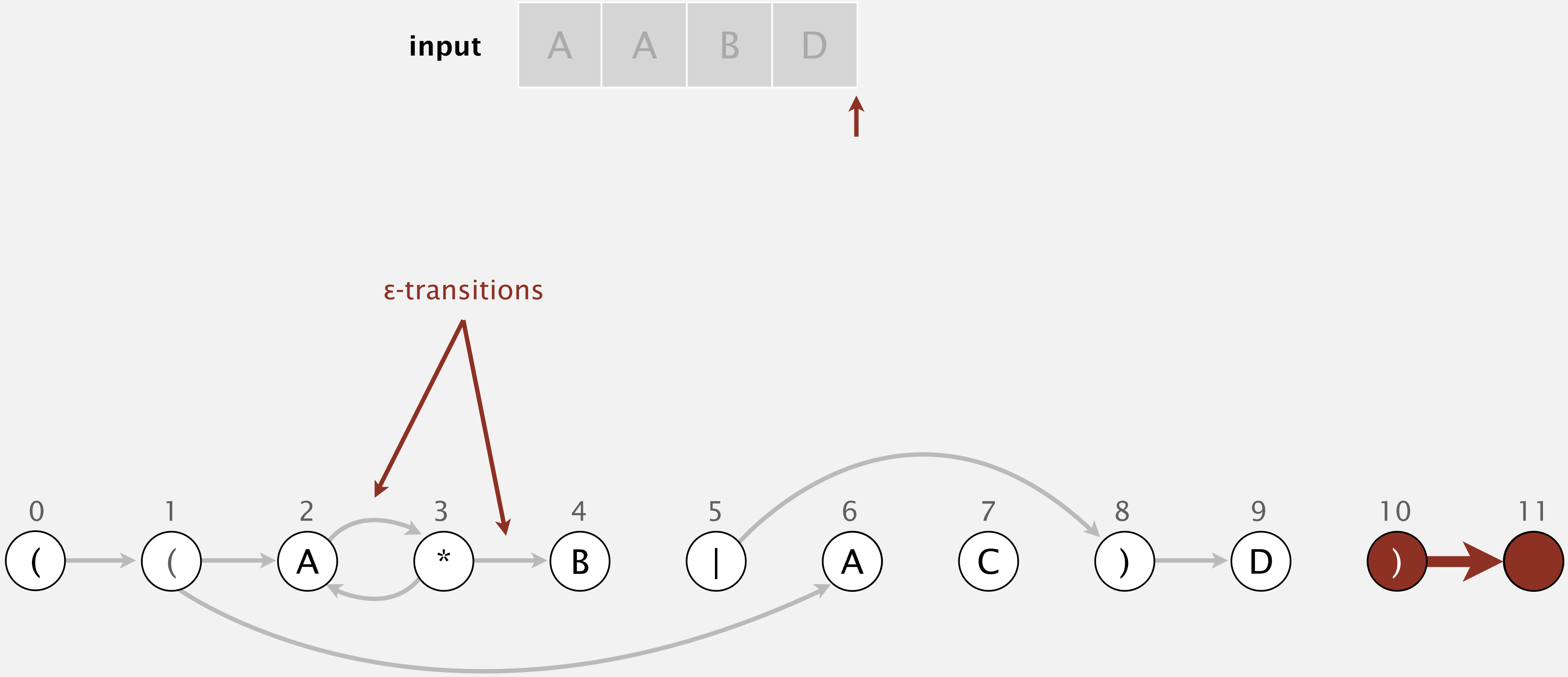


**set of states reachable after matching A A B D : { 10 }**

# NFA simulation

## Read next input character.

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

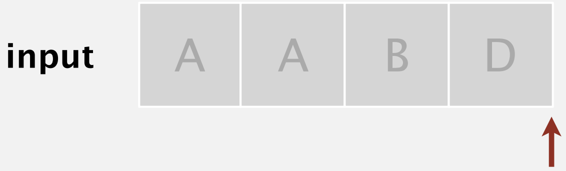


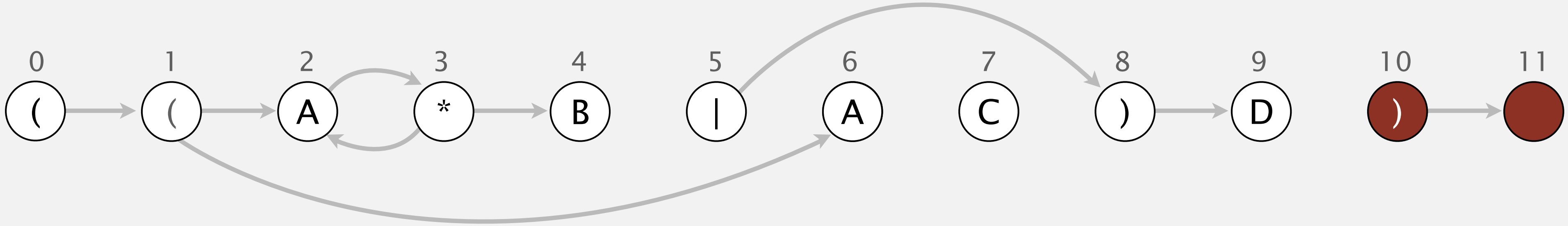
**set of states reachable via ε−transitions after matching A A B D**

# NFA simulation

## Read next input character.

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





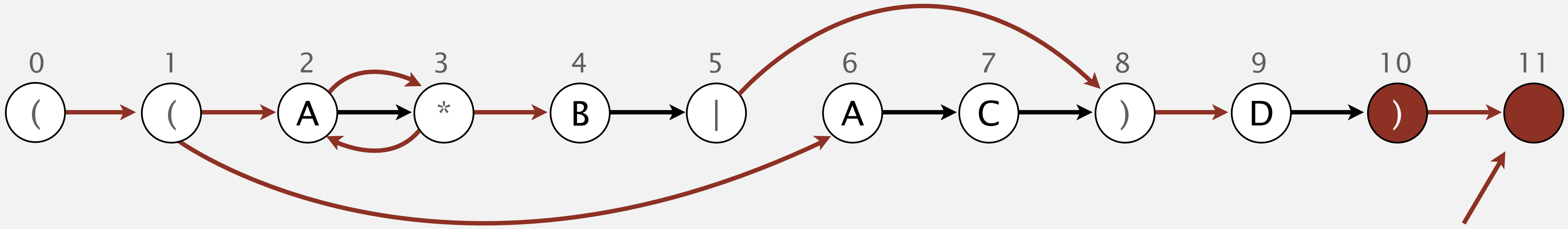
**set of states reachable via ε−transitions after matching A A B D : { 10, 11 }**

# NFA simulation

## When no more input characters:

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

**input** A A B D



accept !

**set of states reachable : { 10, 11 }**

# Digraph reachability

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

`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 = buildEpsilonTransitionsDigraph();
   }

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

   public Digraph buildEpsilonTransitionDigraph()
   {  /* stay tuned */  }

}
```

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

      dfs = new DirectedDFS(G, match);
      pc = new Bag<Integer>();
      for (int v = 0; v < G.V(); v++)
         if (dfs.marked(v)) pc.add(v);
   }

   for (int v : pc)
      if (v == M) return true;
   return false;
}
```

states reachable from start by ε-transitions

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

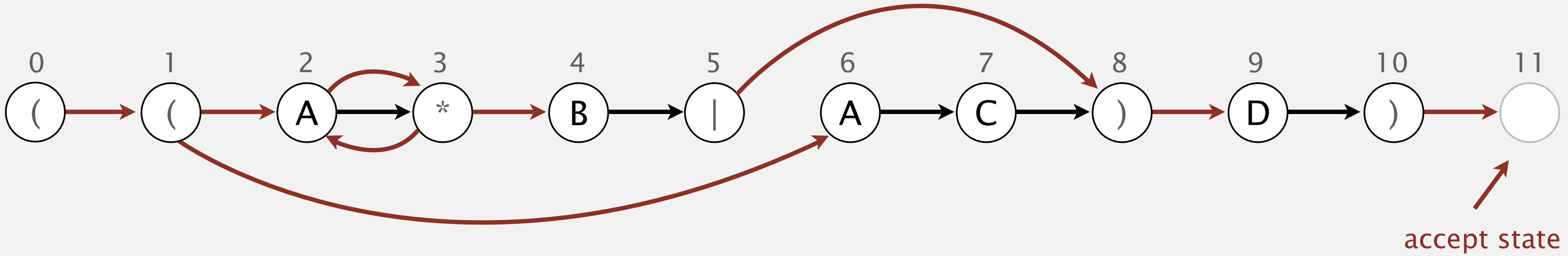
follow ε-transitions

accept if can end in state M

# NFA simulation: analysis

**Proposition.** Determining whether an $N$-character text is recognized by the NFA corresponding to an $M$-character pattern takes time proportional to $M N$ 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 $\leq 3M$.]



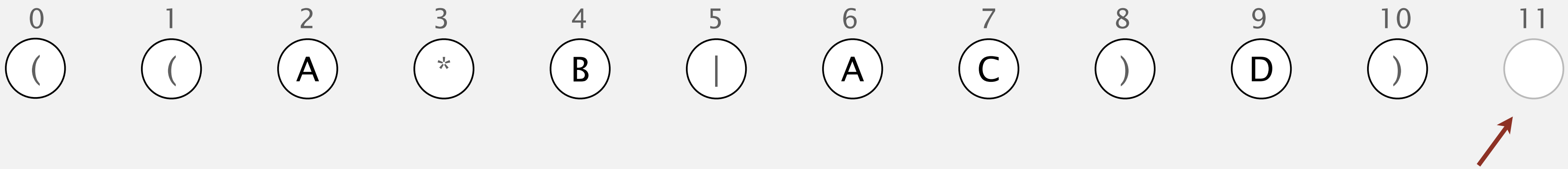
NFA corresponding to the pattern ( ( A * B | A C ) D )

# 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.



accept state

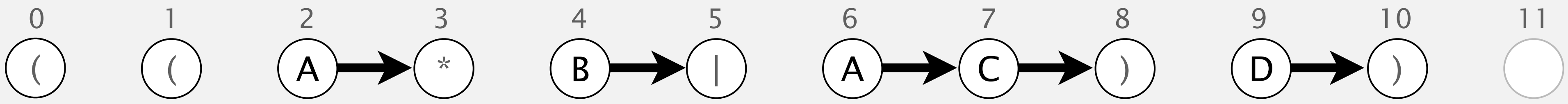
**NFA corresponding to the pattern ( ( A * B | A C ) D )**

# 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`

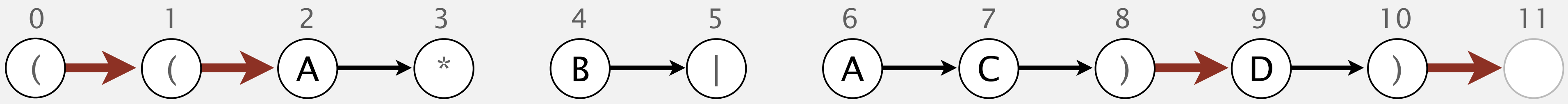
**Metacharacters.** `( ) . * |`



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

# Building an NFA corresponding to an RE

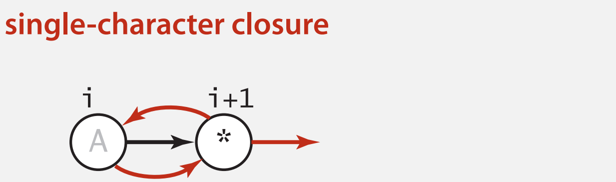
**Parentheses.** Add ε-transition edge from parentheses to next state.



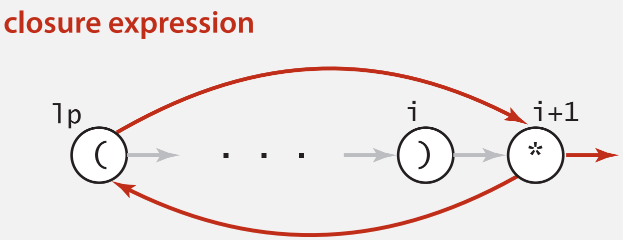
**NFA corresponding to the pattern ( ( A * B | A C ) D )**

# Building an NFA corresponding to an RE

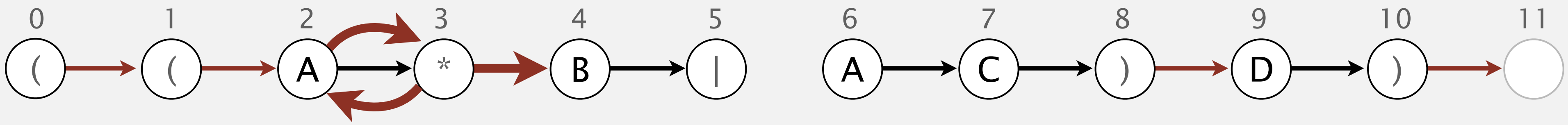
**Closure.** Add three ε-transition edges for each * operator.



```
G.addEdge(i, i+1);
G.addEdge(i+1, i);
```



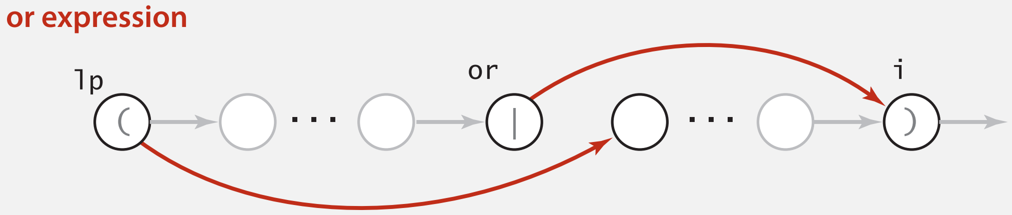
```
G.addEdge(lp, i+1);
G.addEdge(i+1, lp);
```



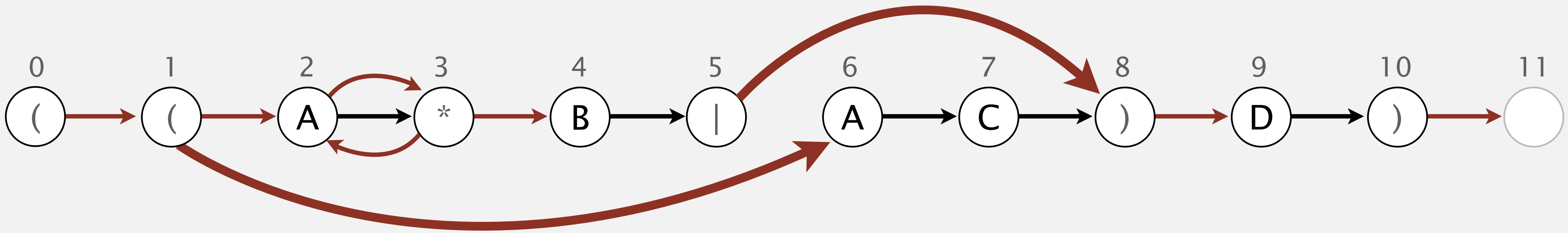
NFA corresponding to the pattern ( ( A * B | A C ) D )

# Building an NFA corresponding to an RE

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



```
G.addEdge(lp, or+1);
G.addEdge(or, i);
```



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

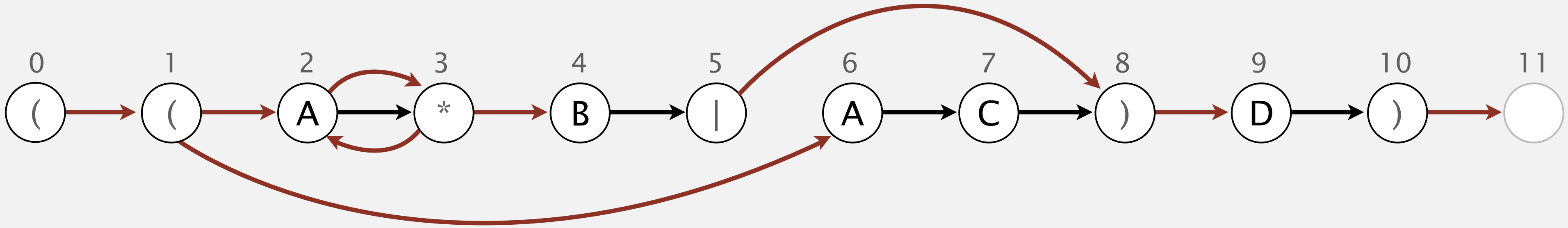
# NFA construction:  implementation

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

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

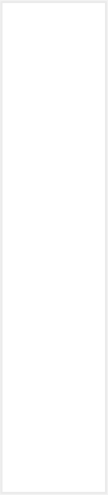
**Solution.** Maintain a stack.

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



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

# NFA construction



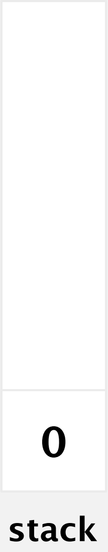
stack

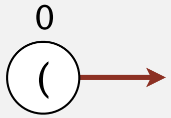
( ( A * B | A C ) D )

# NFA construction

## Left parenthesis.

- Add ε-transition to next state.
- Push index of state corresponding to `(` onto stack.



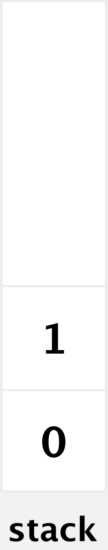


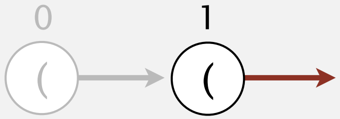
**(** ( A * B | A C ) D )

# NFA construction

## Left parenthesis.

- Add ε-transition to next state.
- Push index of state corresponding to `(` onto stack.





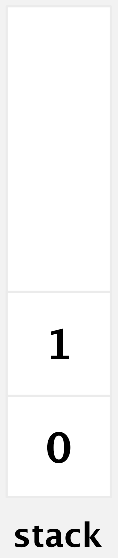
( ( A * B | A C ) D )

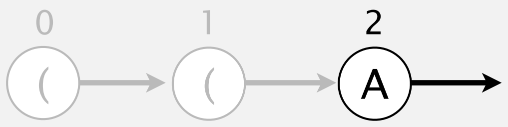
# NFA construction

## Alphabet symbol.

- Add match transition to next state.
- Do one-character lookahead:

  add ε-transitions if next character is *.





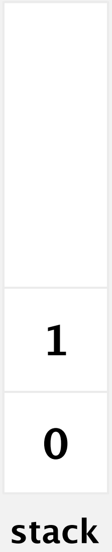
( ( A * B | A C ) D )

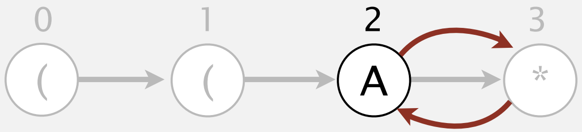
# NFA construction

## Alphabet symbol.

- Add match transition to next state.
- Do one-character lookahead:

  add ε-transitions if next character is *.



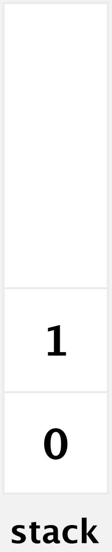


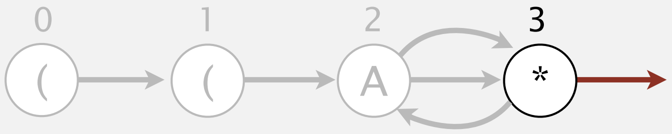
( ( A * B | A C ) D )

# NFA construction

## Closure symbol.

- Add ε-transition to next state.





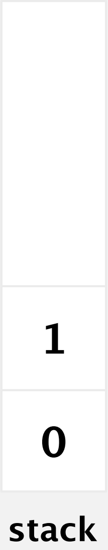
( ( A * B | A C ) D )

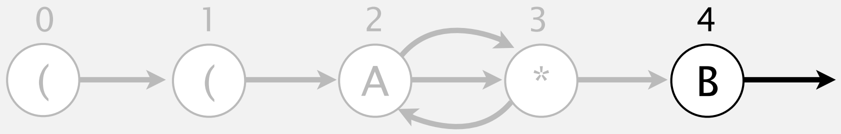
# NFA construction

## Alphabet symbol.

- Add match transition to next state.
- Do one-character lookahead:

  add ε-transitions if next character is *.



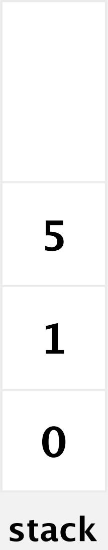


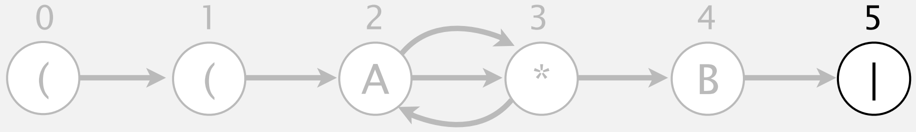
( ( A * B | A C ) D )

# NFA construction

## Or symbol.

- Push index of state corresponding to `|` onto stack.



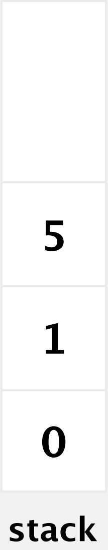


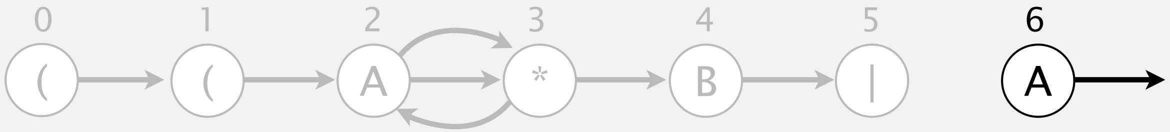
( ( A * B | A C ) D )

# NFA construction

## Alphabet symbol.

- Add match transition to next state.
- Do one-character lookahead:
  add ε-transitions if next character is *.





( ( A * B | A C ) D )

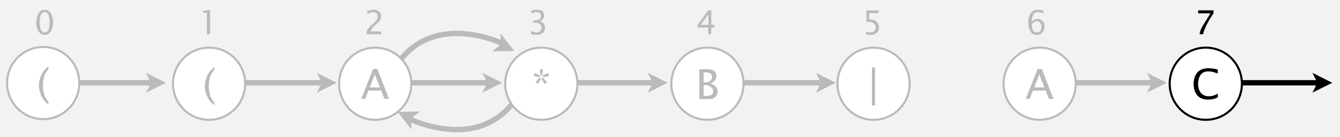
# NFA construction

Alphabet symbol.

- Add match transition to next state.
- Do one-character lookahead:
  add ε-transitions if next character is *.

| |
|---|
| |
| 5 |
| 1 |
| 0 |

stack

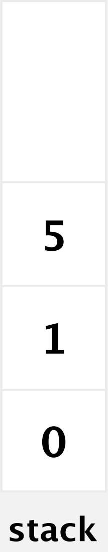


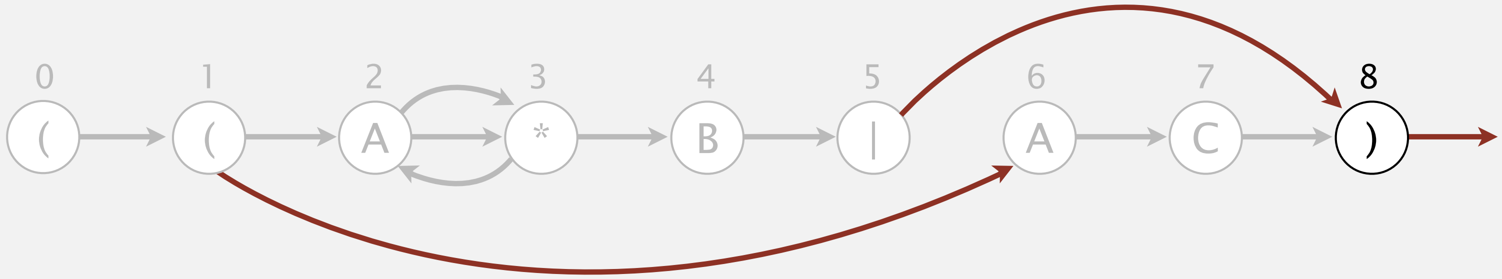
( ( A * B | A C ) D )

# NFA construction

## Right parenthesis.

- Add ε-transition to next state.
- Pop corresponding `(` and possibly intervening `|`; add ε-transition edges for or.
- Do one-character lookahead: add ε-transitions if next character is `*`.



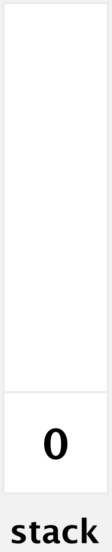


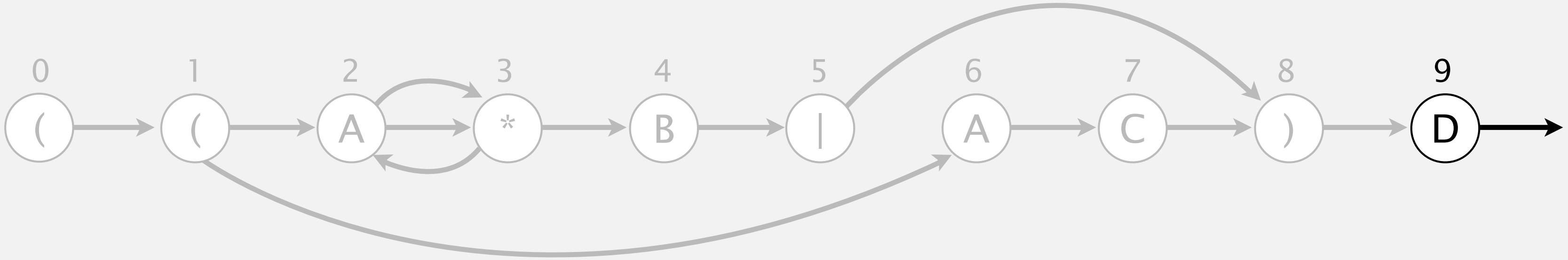
( ( A * B | A C ) D )

# NFA construction

## Alphabet symbol.

- Add match transition to next state.
- Do one-character lookahead:
  add ε-transitions if next character is *.



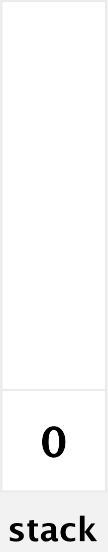


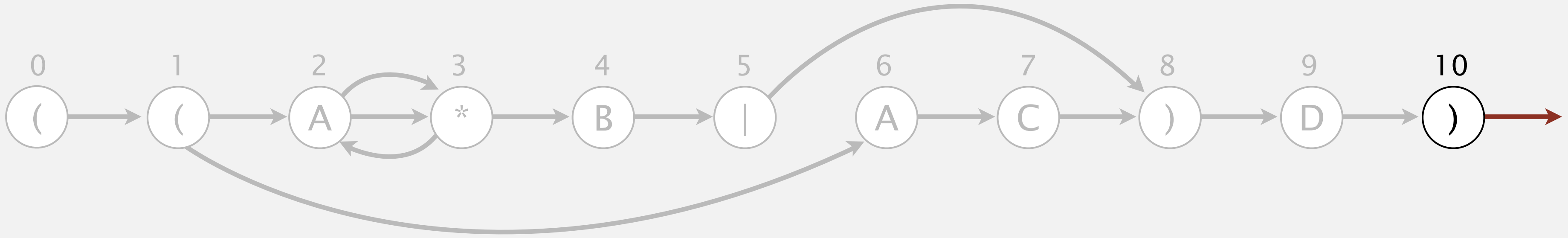
( ( A * B | A C ) D )

# NFA construction

## Right parenthesis.

- Add ε-transition to next state.
- Pop corresponding `(` and possibly intervening `|`; add ε-transition edges for or.
- Do one-character lookahead: add ε-transitions if next character is `*`.



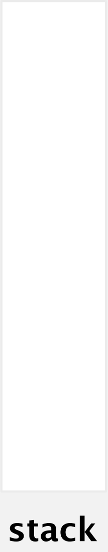


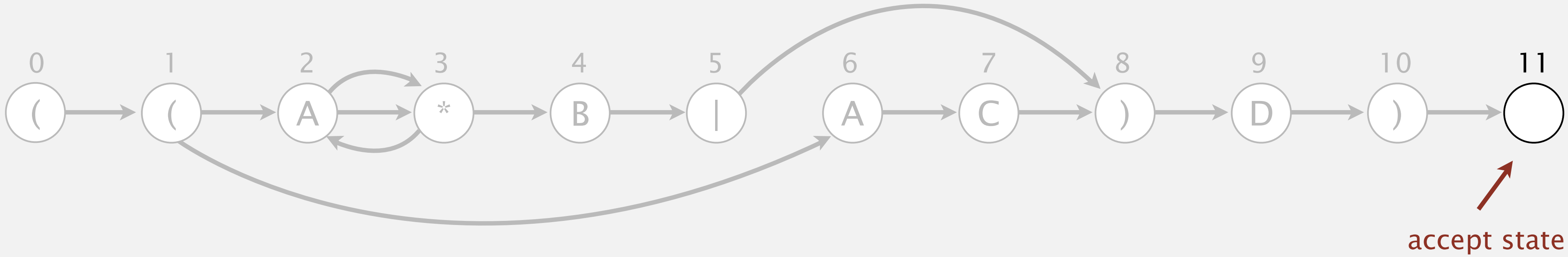
( ( A * B | A C ) D )

# NFA construction

## End of regular expression.

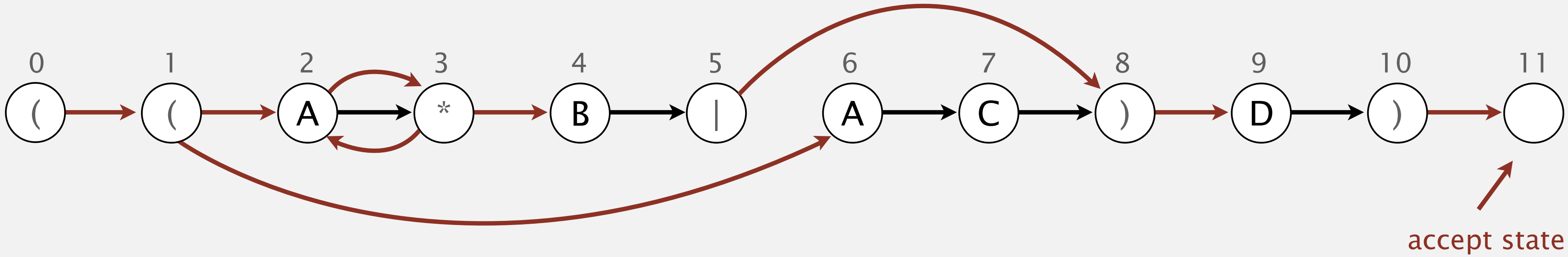
- Add accept state.





**( ( A * B | A C ) D )**

# NFA construction



**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);

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

      if (i < M-1 && re[i+1] == '*') {
         G.addEdge(lp, i+1);
         G.addEdge(i+1, lp);
      }

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

left parentheses and |

or

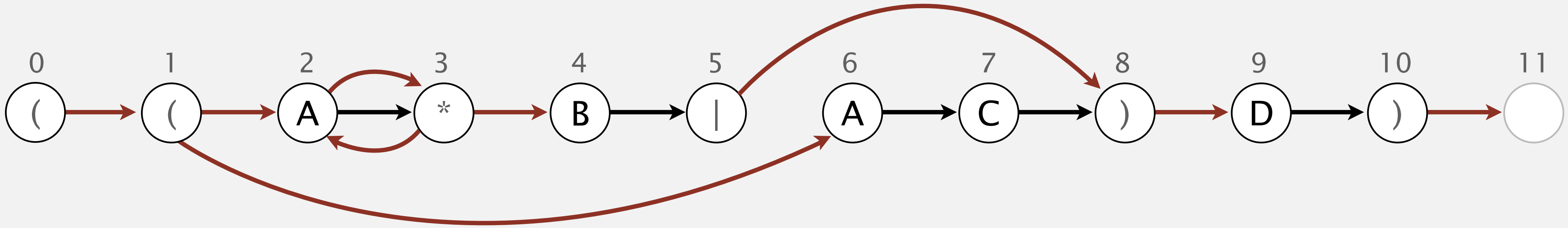
closure
(needs 1-character lookahead)

metasymbols

# NFA construction: analysis

Proposition. Building the NFA corresponding to an $M$-character RE takes time and space proportional to $M$.

Pf. For each of the $M$ characters in the RE, we add at most three ε-transitions and execute at most two stack operations.



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

# REGULAR EXPRESSIONS

- REs and NFAs
- NFA simulation
- NFA construction
- **Applications**

# Generalized regular expression print

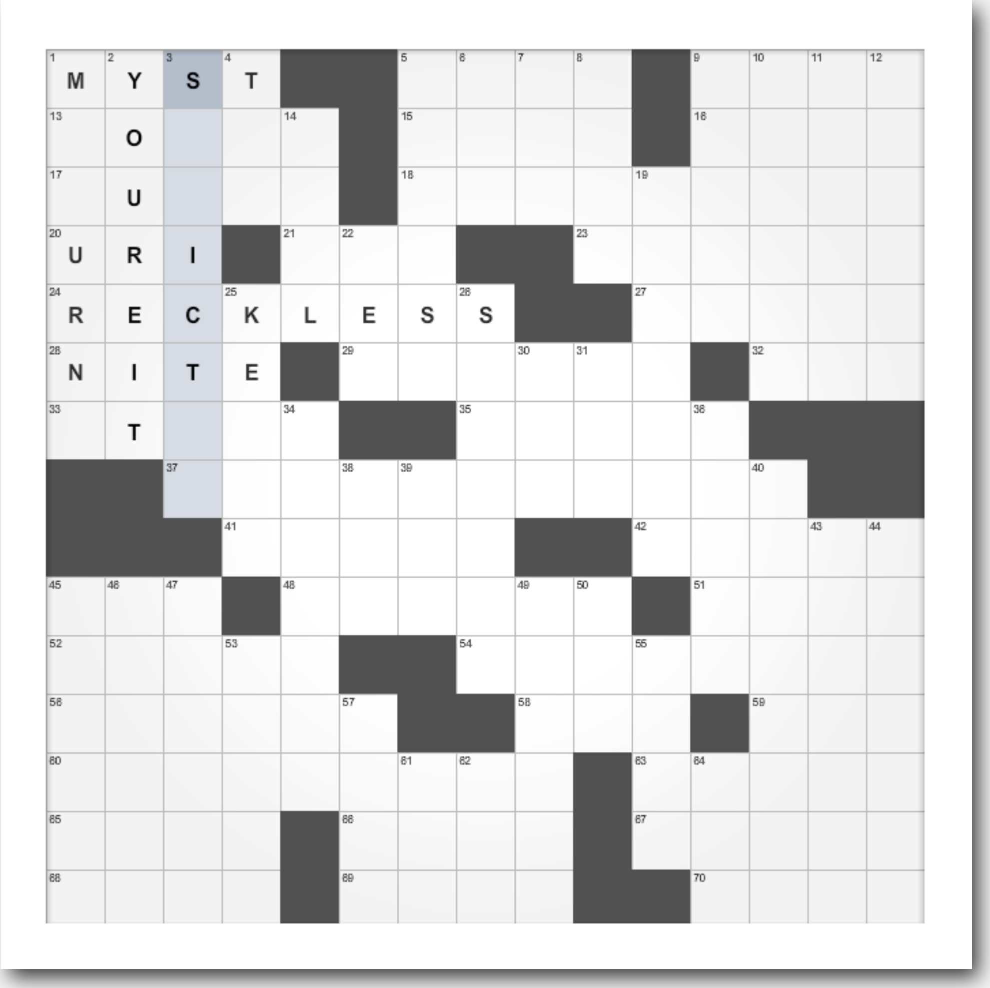
Grep. Take a RE as a command-line argument and print the lines from standard input having some substring that is matched by the RE.

```
public class GREP
{
   public static void main(String[] args)
   {
      String regexp = "(.*" + args[0] + ".*)";
      NFA nfa = new NFA(regexp);
      while (StdIn.hasNextLine())
      {
         String line = StdIn.readLine();
         if (nfa.recognizes(line))
            StdOut.println(line);
      }
   }
}
```

contains RE as a substring

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

# Typical grep application: crossword puzzles



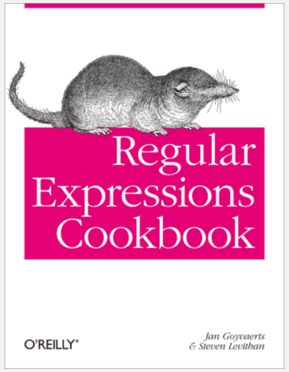
```
% more words.txt
a
aback
abacus
abalone
abandon
...

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

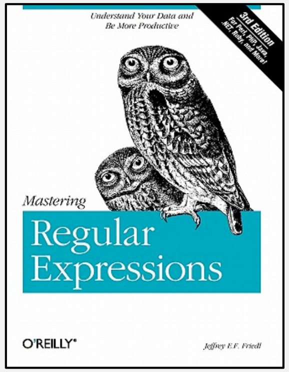
dictionary
(standard in Unix)
also on booksite

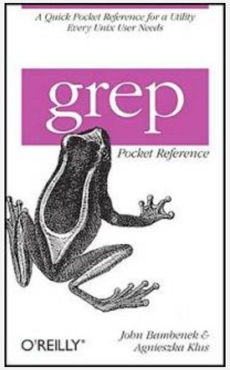
# Industrial-strength grep implementation

## To complete the implementation:

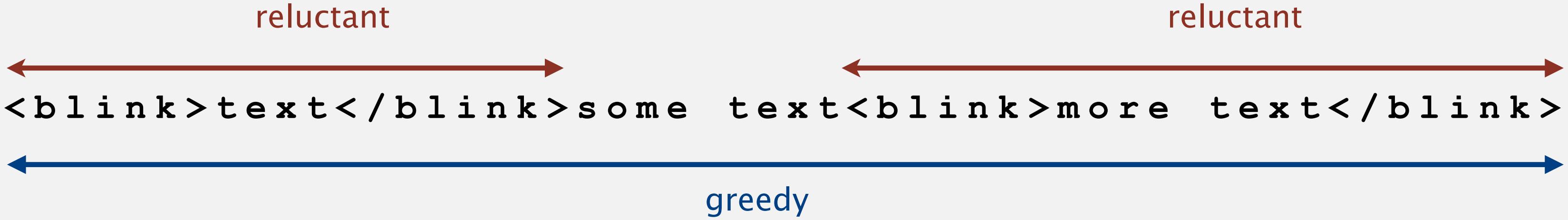
- Add character classes.
- Handle metacharacters.
- Add capturing capabilities.
- Extend the closure operator.
- Error checking and recovery.
- Greedy vs. reluctant matching.







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



# Regular expressions in other languages

## Broadly applicable programmer's tool.

- Originated in Unix in the 1970s.
- Many languages support extended regular expressions.
- Built into grep, awk, emacs, Perl, PHP, Python, JavaScript, ...

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

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

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

## PERL. Practical Extraction and Report Language.

```
% perl -p -i -e 's|from|to|g' input.txt
```

replace all occurrences of **`from`** with **`to`** in the file **`input.txt`**

```
% perl -n -e 'print if /^[A-Z][A-Za-z]*$/' words.txt
```

print all words that start with uppercase letter

do for each line

# Regular expressions in Java

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

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

```
public class Validate
{
   public static void main(String[] args)
   {
      String regexp = args[0];
      String input  = args[1];
      StdOut.println(input.matches(regexp));
   }
}
```

```
% java Validate "[$_A-Za-z][$_A-Za-z0-9]*" ident123
true

% java Validate "[a-z]+@([a-z]+\.)+(edu|com)" rs@cs.princeton.edu
true

% java Validate "[0-9]{3}-[0-9]{2}-[0-9]{4}" 166-11-4433
true
```

legal Java identifier

valid email address (simplified)

Social Security number

# 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

```
% 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 $w\,w$ 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 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.