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

http://algs4.cs.princeton.edu

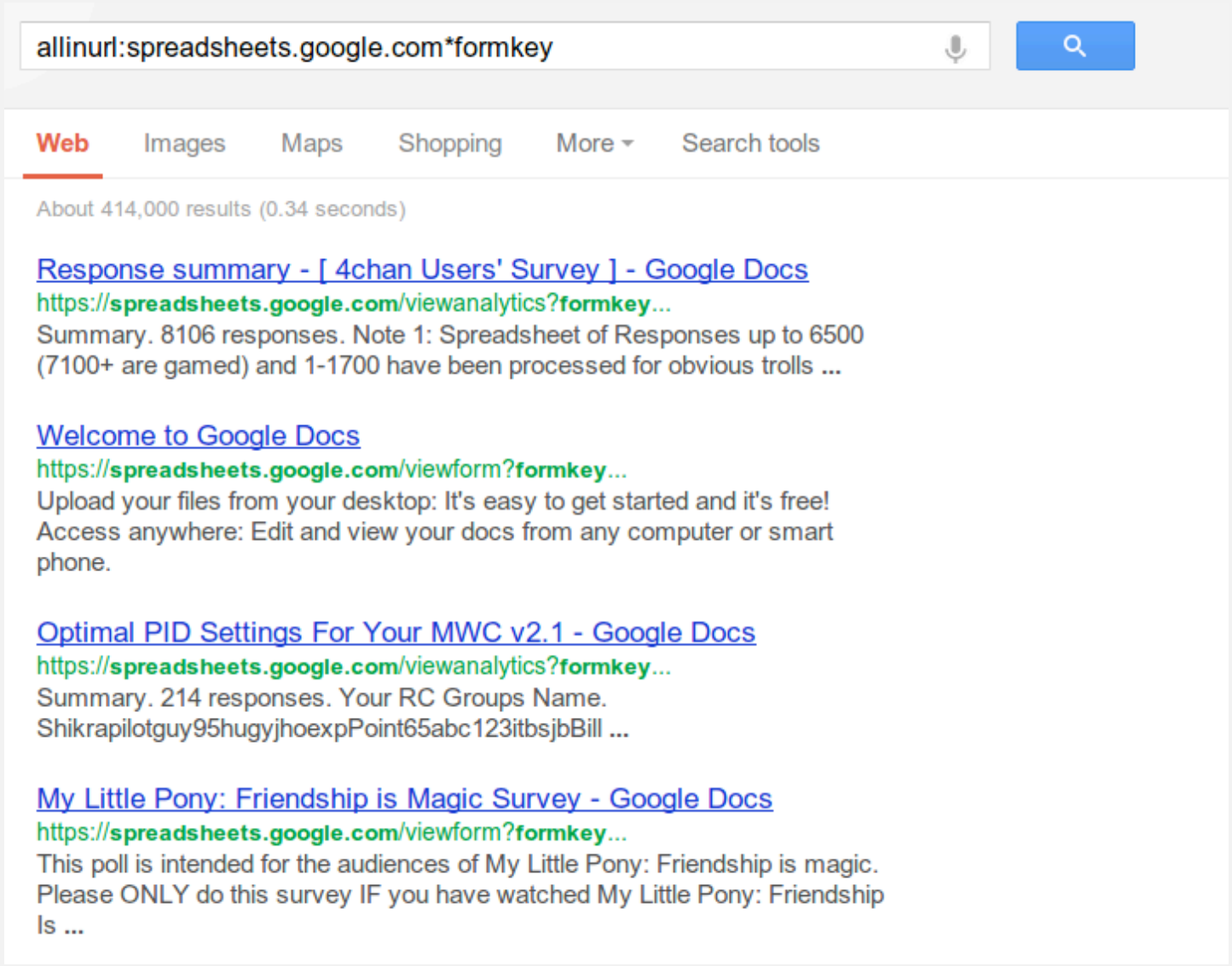
5.4 REGULAR EXPRESSIONS

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

http://algs4.cs.princeton.edu

5.4 Regular Expressions

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


allinurl:spreadsheets.google.com*formkey
Web
Images
Maps
Shopping
More
Search tools
About 414,000 results (0.34 seconds)
Response summary - [4chan Users' Survey] - Google Docs
https://spreadsheets.google.com/viewanalytics?formkey...
Summary. 8106 responses. Note 1: Spreadsheet of Responses up to 6500 (7100+ are gamed) and 1-1700 have been processed for obvious trolls ...
Welcome to Google Docs
https://spreadsheets.google.com/viewform?formkey...
Upload your files from your desktop: It's easy to get started and it's free! Access anywhere: Edit and view your docs from any computer or smart phone.
Optimal PID Settings For Your MWC v2.1 - Google Docs
https://spreadsheets.google.com/viewanalytics?formkey...
Summary. 214 responses. Your RC Groups Name. Shikrapilotguy95hugyjhoexpPoint65abc123itbsjbBill ...
My Little Pony: Friendship is Magic Survey - Google Docs
https://spreadsheets.google.com/viewform?formkey...
This poll is intended for the audiences of My Little Pony: Friendship is magic. Please ONLY do this survey IF you have watched My Little Pony: Friendship Is ...

My Little Pony: Friendship is Magic Survey

This poll is intended for the audiences of My Little Pony: Friendship is magic.

Please ONLY do this survey IF you have watched My Little Pony: Friendship Is Magic And Please do not submit multiple surveys. Thank you

**Survey result can be found here
https://spreadsheets.google.com/viewanalytics?hl=en&formkey=dDB5dy0yTjhlMnlKdEsxRGFzWTNLUlE6MQ
* Required

What is your sex? *

- Male
- Female

What is your age (range)? *

- 5 - 11
- 12 - 14
- 15 - 17
- 18 - 24
- 25 - 30
- 31 - 40
- 41- 50
- 51 - 60
- 60 or above

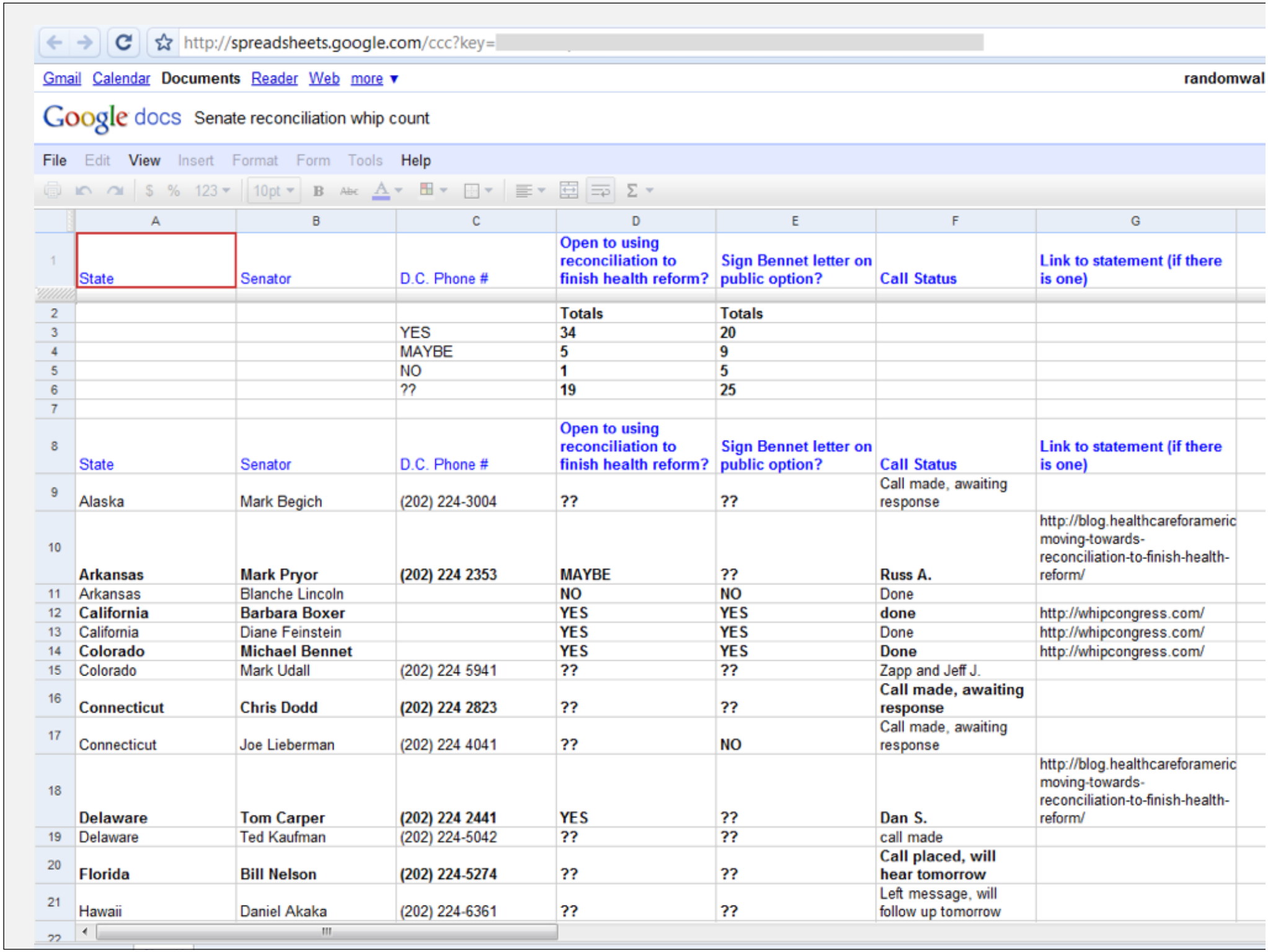
What is your exact age? (optional)

Who is your Favorite Character *

- Twilight Sparkle
- Pinkie Pie
- Fluttershy
- Applejack
- Rainbow Dash
- Rarity
- Princess Celestia
- Derpy Hooves
- Spike
- None of the above/ others
- Luna

Who is your Second Favorite Character *

- Twilight Sparkle
- Pinkie Pie
- Fluttershy



http://spreadsheets.google.com/ccc?key=

Gmail Calendar **Documents** Reader Web more ▼ **randomwal**

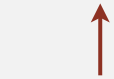
Google docs Senate reconciliation whip count

File Edit View Insert Format Form Tools Help

	A	B	C	D	E	F	G
1	State	Senator	D.C. Phone #	Open to using reconciliation to finish health reform?	Sign Bennet letter on public option?	Call Status	Link to statement (if there is one)
2				**Totals**	**Totals**		
3			YES	**34**	**20**		
4			MAYBE	**5**	**9**		
5			NO	**1**	**5**		
6			??	**19**	**25**		
7							
8	State	Senator	D.C. Phone #	Open to using reconciliation to finish health reform?	Sign Bennet letter on public option?	Call Status	Link to statement (if there is one)
9	Alaska	Mark Begich	(202) 224-3004	??	??	Call made, awaiting response	
10	**Arkansas**	**Mark Pryor**	**(202) 224 2353**	**MAYBE**	**??**	**Russ A.**	http://blog.healthcareforamericmoving-towards-reconciliation-to-finish-health-reform/
11	Arkansas	Blanche Lincoln		NO	NO	Done	
12	**California**	**Barbara Boxer**		**YES**	**YES**	**done**	http://whipcongress.com/
13	California	Diane Feinstein		YES	YES	Done	http://whipcongress.com/
14	**Colorado**	**Michael Bennet**		**YES**	**YES**	**Done**	http://whipcongress.com/
15	Colorado	Mark Udall	(202) 224 5941	??	??	Zapp and Jeff J.	
16	**Connecticut**	**Chris Dodd**	**(202) 224 2823**	??	??	**Call made, awaiting response**	
17	Connecticut	Joe Lieberman	(202) 224 4041	??	NO	Call made, awaiting response	
18	**Delaware**	**Tom Carper**	**(202) 224 2441**	**YES**	**??**	**Dan S.**	http://blog.healthcareforamericmoving-towards-reconciliation-to-finish-health-reform/
19	Delaware	Ted Kaufman	(202) 224-5042	??	??	call made	
20	**Florida**	**Bill Nelson**	**(202) 224-5274**	??	??	**Call placed, will hear tomorrow**	
21	Hawaii	Daniel Akaka	(202) 224-6361	??	??	Left message, will follow up tomorrow	
22							

Regular expressions

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



possibly infinite

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

Regular expression shortcuts

Additional operations are often added for convenience.

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

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

Regular expression examples

RE notation is surprisingly expressive.

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

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

Regular expression examples

regular expression
(ARV|IND*)

ARVND	ARVIND	ARVN	ARVNDDD

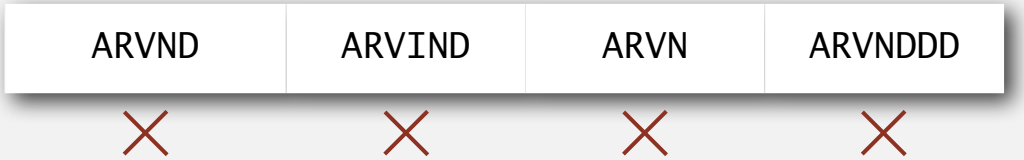
pollEv.com/jhug text to **37607**

Q: How many of the strings match the regular expression?

A. 0 [314389]
B. 1 [314423]
C. 2 [314424]
D. 3 [314425]
E. 4 [314426]

Regular expression examples

regular expression
(ARV|IND*)



Q: How many of the strings match the regular expression?

A. 0 [314389]
B. 1 [314423]
C. 2 [314424]
D. 3 [314425]
E. 4 [314426]

Regular expression examples

regular expression
(A|(B|C*)*)

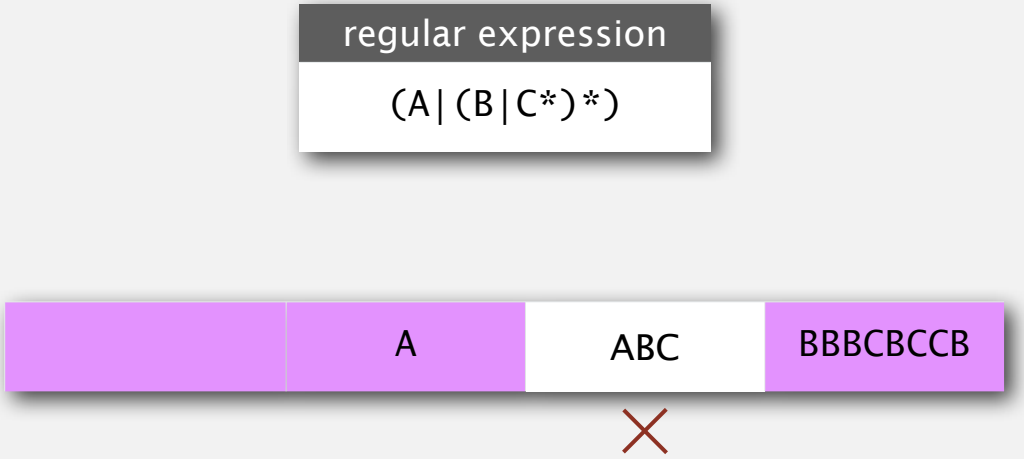
	A	ABC	BBBCBCCB

pollEv.com/jhug text to **37607**

Q: How many of the strings match the regular expression?

A. 0 [314495]
B. 1 [314496]
C. 2 [314551]
D. 3 [314591]
E. 4 [314635]

Regular expression examples



Q: How many of the strings match the regular expression?

A. 0 [314495]
B. 1 [314496]
C. 2 [314551]
D. 3 [314591]
E. 4 [314635]

Regular expression examples

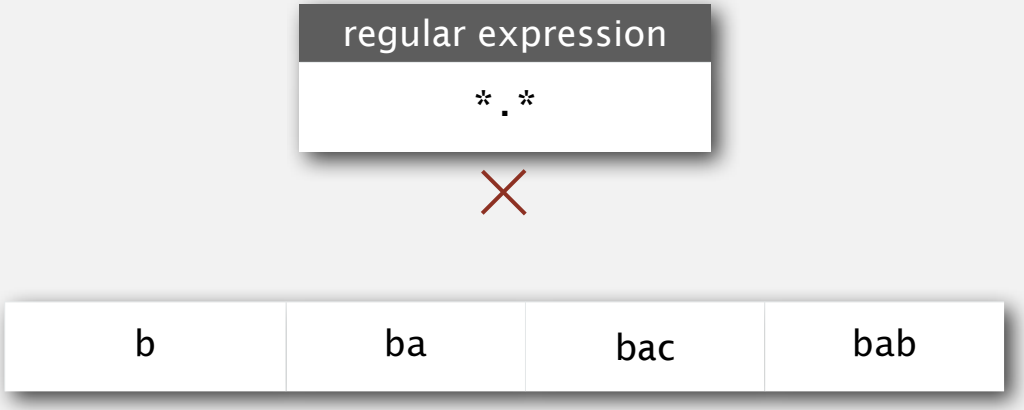
regular expression
.

b	ba	bac	bab

Q: How many of the strings match the regular expression?

A. 0
B. 1
C. 2
D. 3
E. 4

Regular expression examples



Q: How many of the strings match the regular expression?

A. 0
B. 1
C. 2
D. 3
E. 4

Regular expression design

Groups of 3

- Give a regular expression that matches a binary string with exactly four 0s.

Hint: Use 1*

√ 0000 × 0010

√ 00100 × 010

- Give a regular expression that matches a binary string with at least two 0s.

Hint: Use (0|1)*

√ 010100 × 011

- Give a regular expression that matches a binary string with an even number of 0s.

√ 00110011000011000011010 1 × 011

√ 00 √ empty string × 00110

- Extra: Derive a regular expression that finds matches ***any*** four letter sequence repeated twice

√ wikiwiki × horsehorse

√ couscous × derpder

- Extra extra: Succinctly state why no regex can tell you if a binary string has an EQUAL number of 0s and 1s.

Regular expression design

Groups of 3

- Give a regular expression that matches a binary string with exactly 4 0s.
  - 1*01*01*01*01*

Regular expression design

Groups of 3

- Give a regular expression that matches a binary string with exactly 4 0s.
  - 1*01*01*01*01*
- Give a regular expression that matches a binary string with at least two 0s.
  - (0|1)*0(0|1)*0(0|1)*

Regular expression design

Groups of 3

- Give a regular expression that matches a binary string with exactly 4 0s.
  - 1*01*01*01*01*
- Give a regular expression that matches a binary string with at least two 0s.
  - (0|1)*0(0|1)*0(0|1)*
- Give a regular expression that matches a binary string with an even number of 0s.
  - 1*(01*01*)*1*

Regular expression design

Groups of 3

- Give a regular expression that matches a binary string with exactly 4 0s.
  - 1*01*01*01*01*
- Give a regular expression that matches a binary string with at least two 0s.
  - (0|1)*0(0|1)*0(0|1)*
- Give a regular expression that matches a binary string with an even number of 0s.
  - 1*(01*01*)*1*
- Extra: Derive a regular expression that finds matches ***any*** four letter sequence repeated twice
  - (aaaaaaaa)|(aaabaaab)|(aaacaaac)| etc... |(zzzzzzzz)

Regular expression design

Groups of 3

- Give a regular expression that matches a binary string with exactly 4 0s.
  - 1*01*01*01*01*
- Give a regular expression that matches a binary string with at least two 0s.
  - (0|1)*0(0|1)*0(0|1)*
- Give a regular expression that matches a binary string with an even number of 0s.
  - 1*(01*01*)*1*
- Extra: Derive a regular expression that finds matches ***any*** four letter sequence repeated twice
  - (aaaaaaaa)|(aaabaaab)|(aaacaaac)| etc... |(zzzzzzzz)
- Extra extra:
  - A regular expression is equivalent to some finite state automata (more shortly). Keeping count requires a potentially unbounded number of states.

Can the average programmer learn to use REs?

Perl RE for valid RFC822 email addresses

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

http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html

Regular expression caveat

Writing a RE is like writing a program.

- Need to understand programming model.
- Can be easier to write than read.
- Can be difficult to debug.

> *" Some people, when confronted with a problem, think 'I know I'll use regular expressions.' Now they have two problems. "*
>
> *— Jamie Zawinski (flame war on alt.religion.emacs)*

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

http://algs4.cs.princeton.edu

5.4 Regular Expressions

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

Duality between REs and DFAs

RE. Concise way to describe a set of strings.

DFA. Machine to recognize whether a given string is in a given set.

Kleene's theorem.

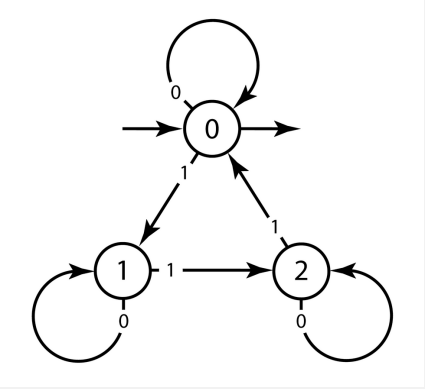
- For any DFA, there exists a RE that describes the same set of strings.
- For any RE, there exists a DFA that recognizes the same set of strings.

RE

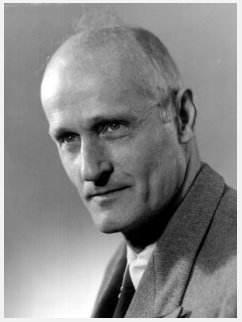
0* | (0*10*10*10*)*

number of 1's is a multiple of 3

DFA



number of 1's is a multiple of 3

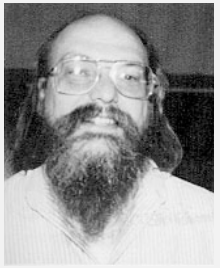


Stephen Kleene
Princeton Ph.D. 1934

Pattern matching implementation: basic plan (first attempt)

Overview is the same as for KMP.

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

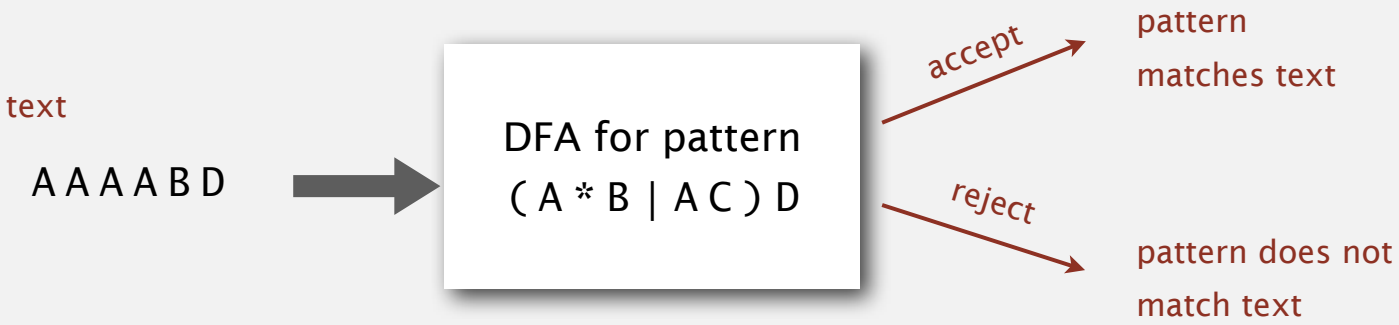


Ken Thompson
Turing Award '83

Underlying abstraction. Deterministic finite state automata (DFA).

Basic plan. [apply Kleene's theorem]

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

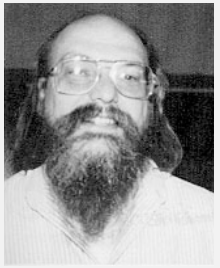


Bad news. Basic plan is infeasible (DFA may have exponential # of states).

Pattern matching implementation: basic plan (revised)

Overview is similar to KMP.

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

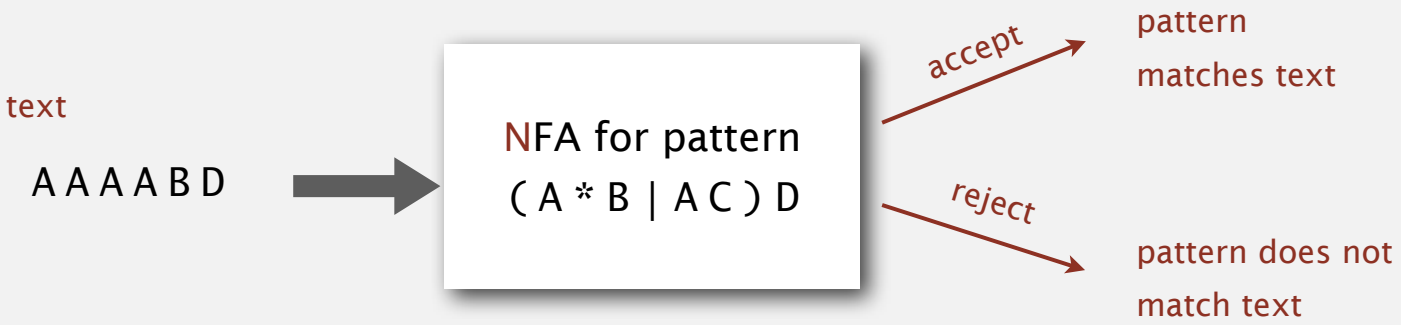


Ken Thompson
Turing Award '83

Underlying abstraction. Nondeterministic finite state automata (NFA).

Basic plan.

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



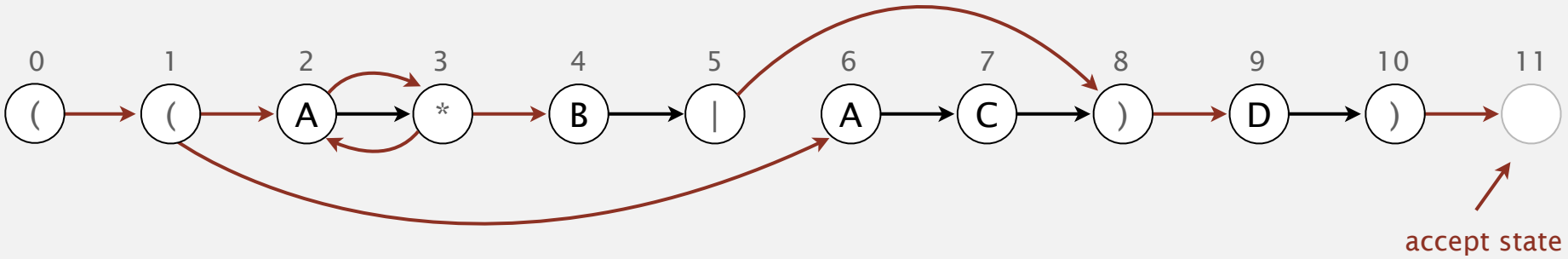
Q. What is an NFA?

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.

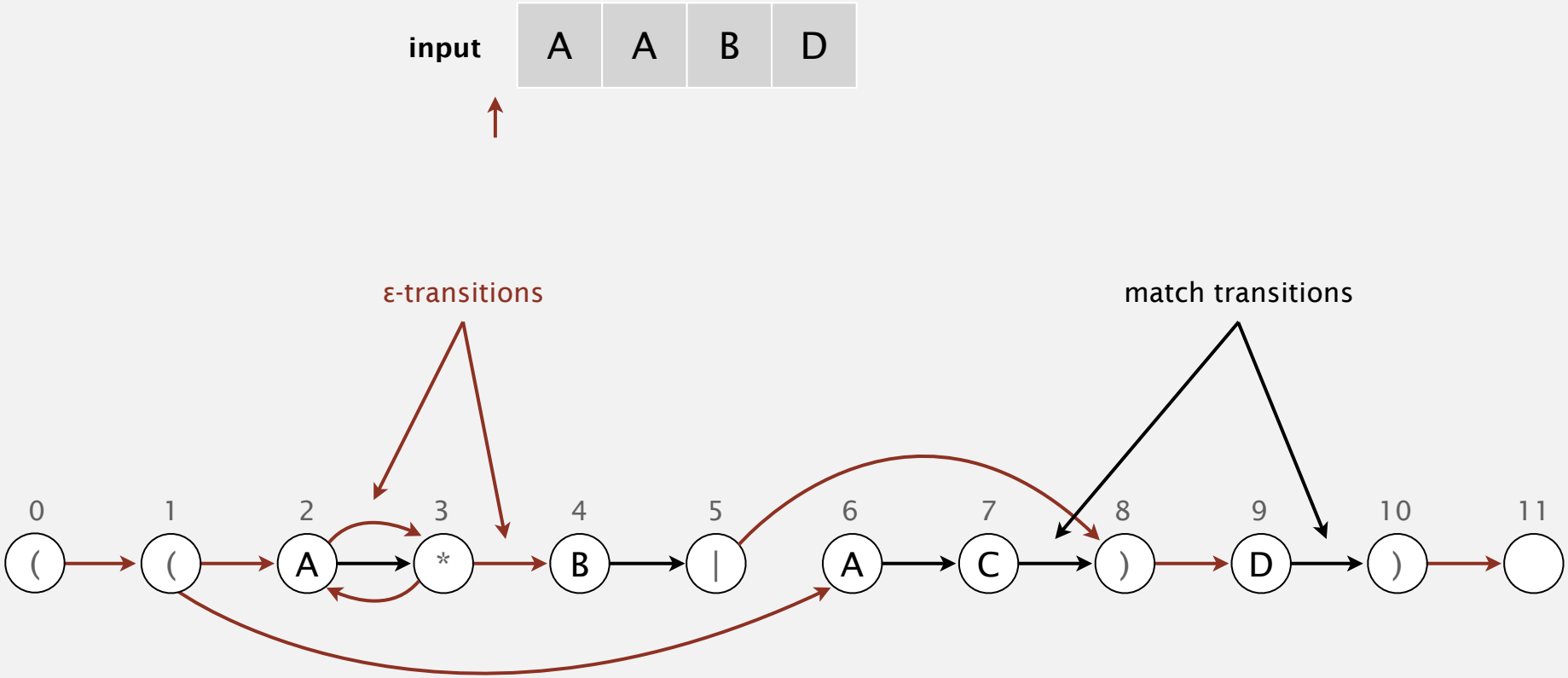
after scanning all text characters



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

NFA simulation demo

Goal. Check whether input matches pattern.

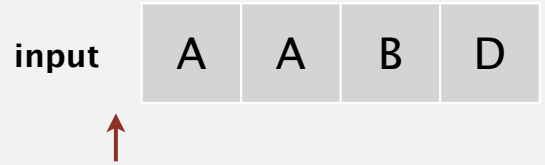


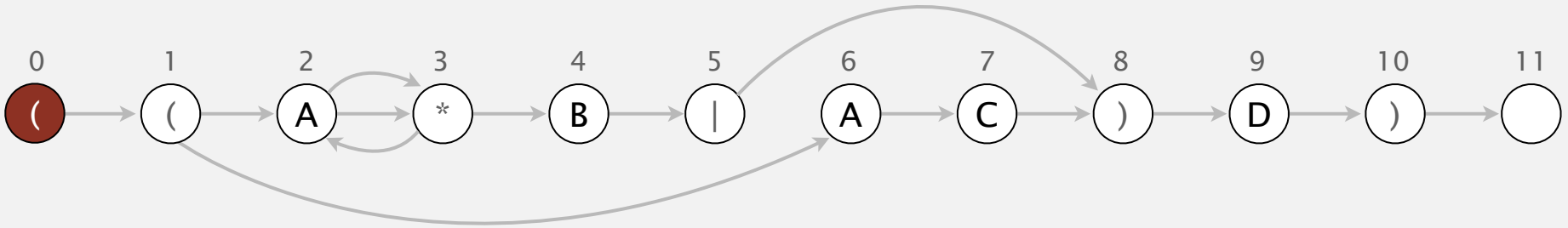
NFA corresponding to the pattern ((A * B | A C) D)

NFA simulation demo

Read next input character.

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



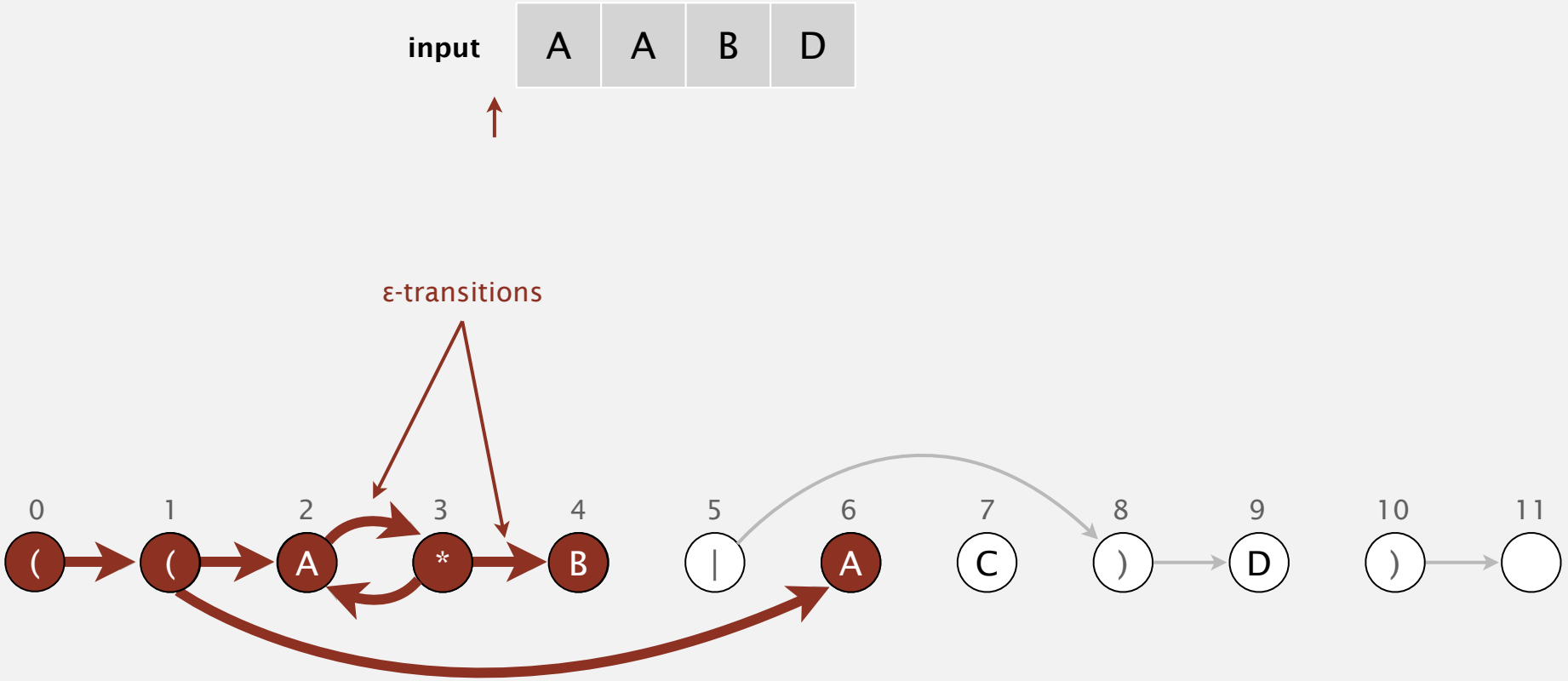


set of states reachable from start: 0

NFA simulation demo

Read next input character.

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

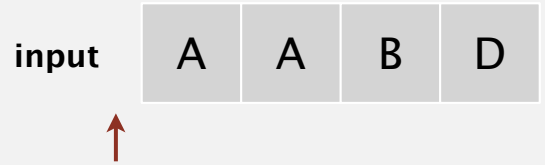


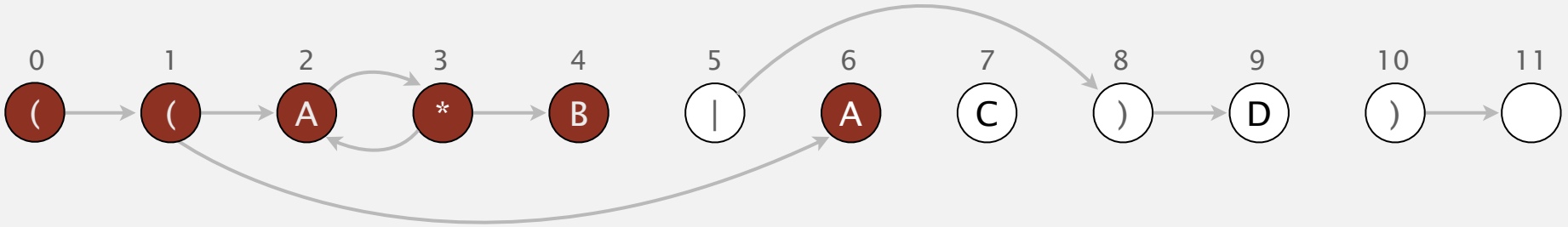
set of states reachable via ε-transitions from start

NFA simulation demo

Read next input character.

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



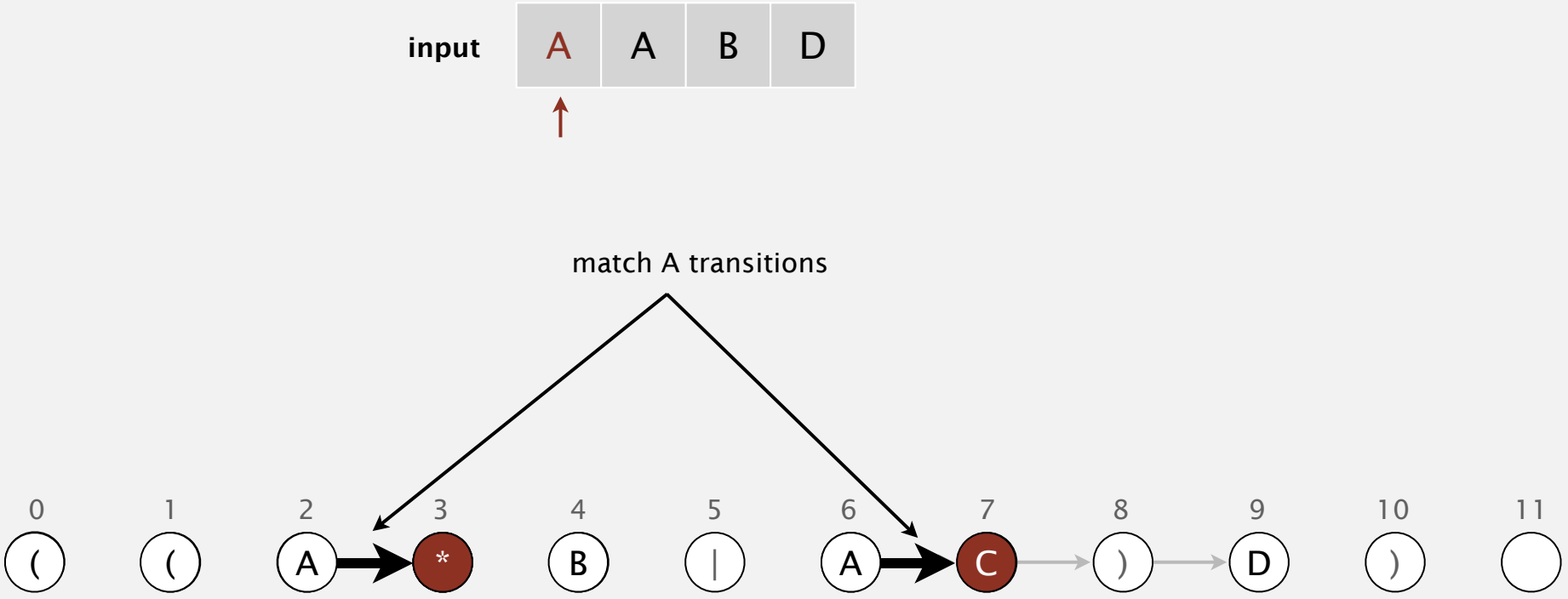


set of states reachable via ε-transitions from start : { 0, 1, 2, 3, 4, 6 }

NFA simulation demo

Read next input character.

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

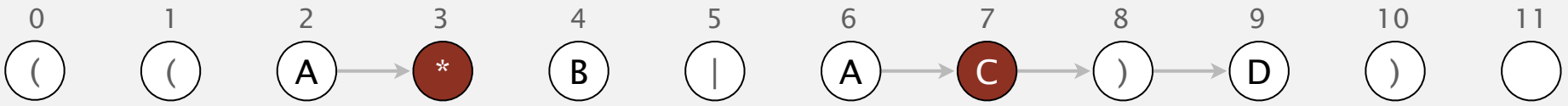


set of states reachable after matching A

NFA simulation demo

Read next input character.

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

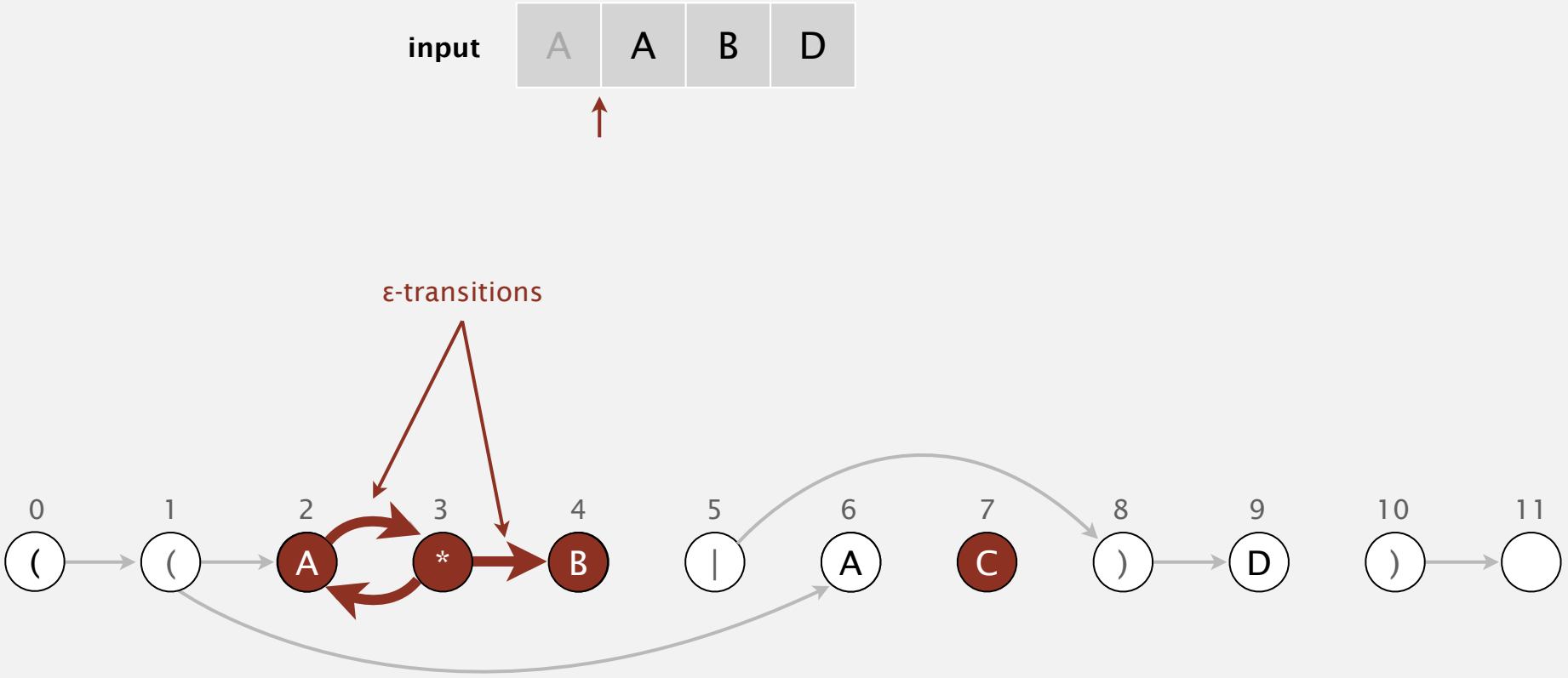


set of states reachable after matching A : { 3, 7 }

NFA simulation demo

Read next input character.

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

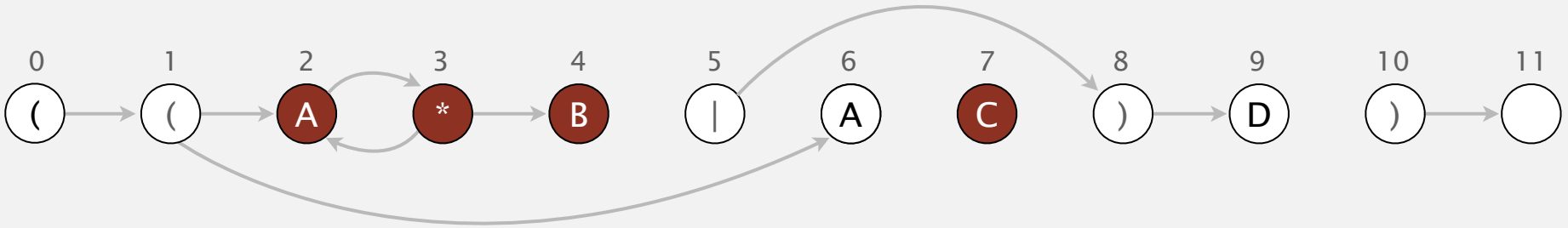


set of states reachable via ε-transitions after matching A

NFA simulation demo

Read next input character.

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

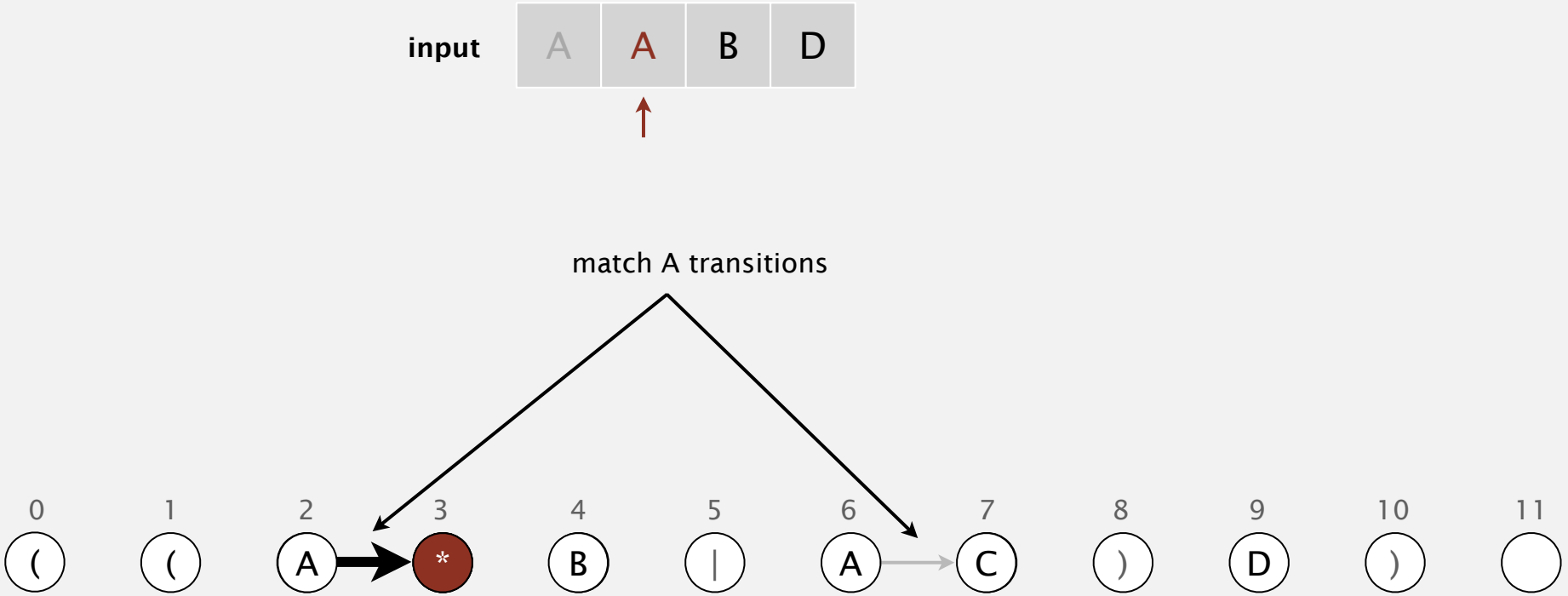


set of states reachable via ε-transitions after matching A : { 2, 3, 4, 7 }

NFA simulation demo

Read next input character.

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



set of states reachable after matching A A

NFA simulation demo

Read next input character.

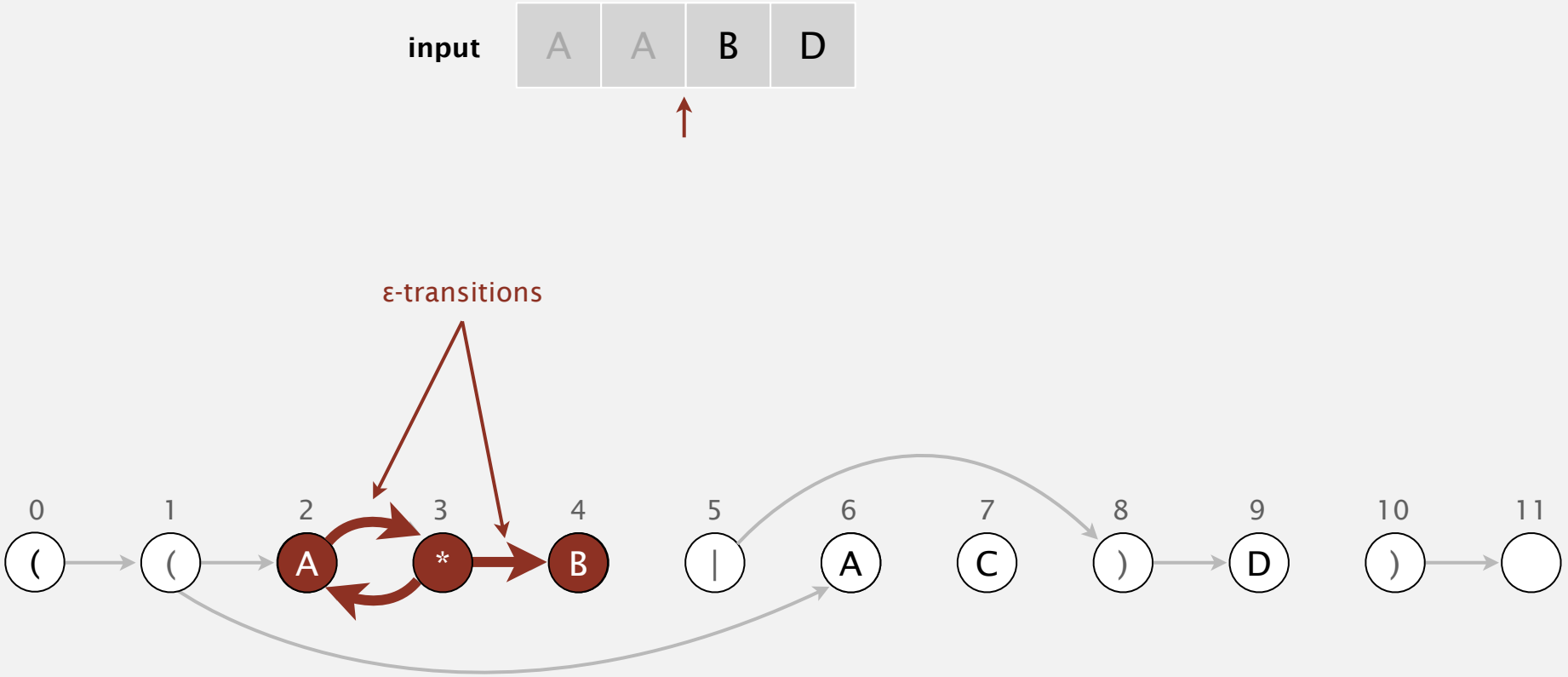
- Find states reachable by match transitions.
- Find states reachable by ε-transitions

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

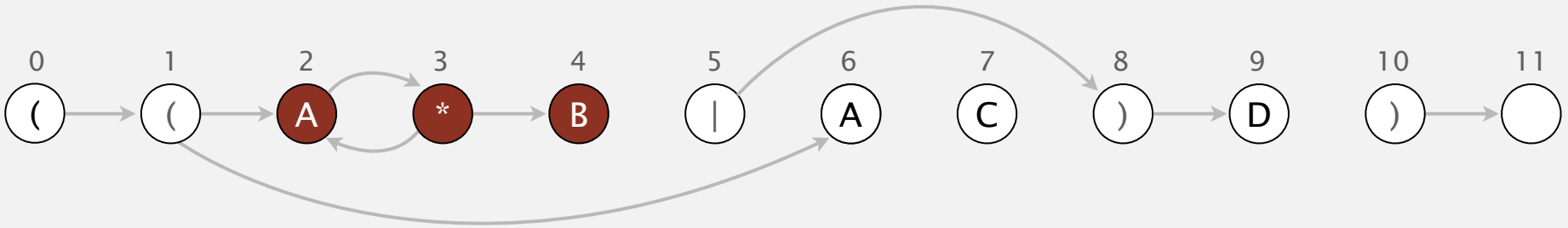


set of states reachable via ε-transitions after matching A A

NFA simulation demo

Read next input character.

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

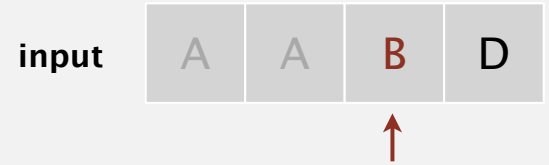


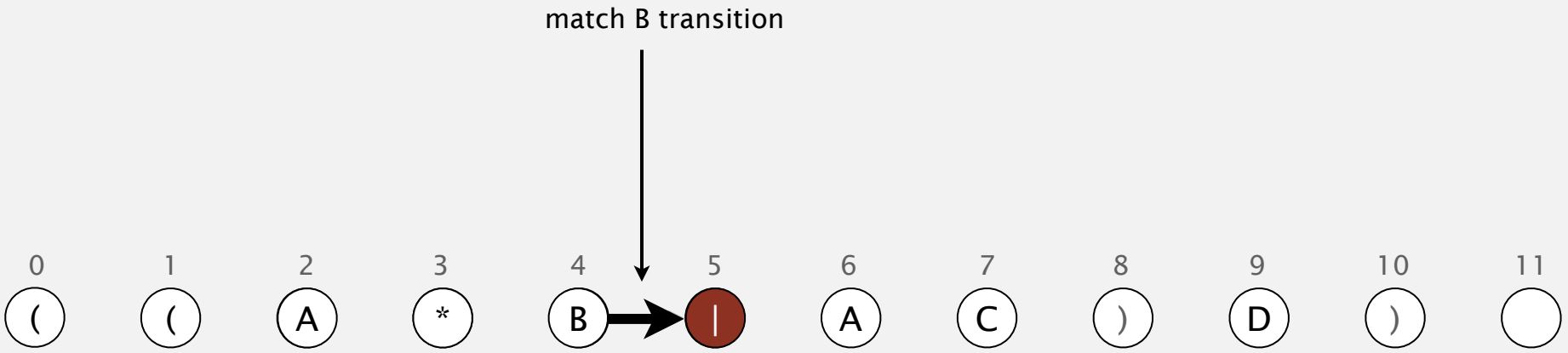
set of states reachable via ε-transitions after matching A A : { 2, 3, 4 }

NFA simulation demo

Read next input character.

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



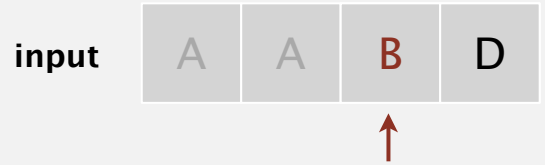


set of states reachable after matching A A B

NFA simulation demo

Read next input character.

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

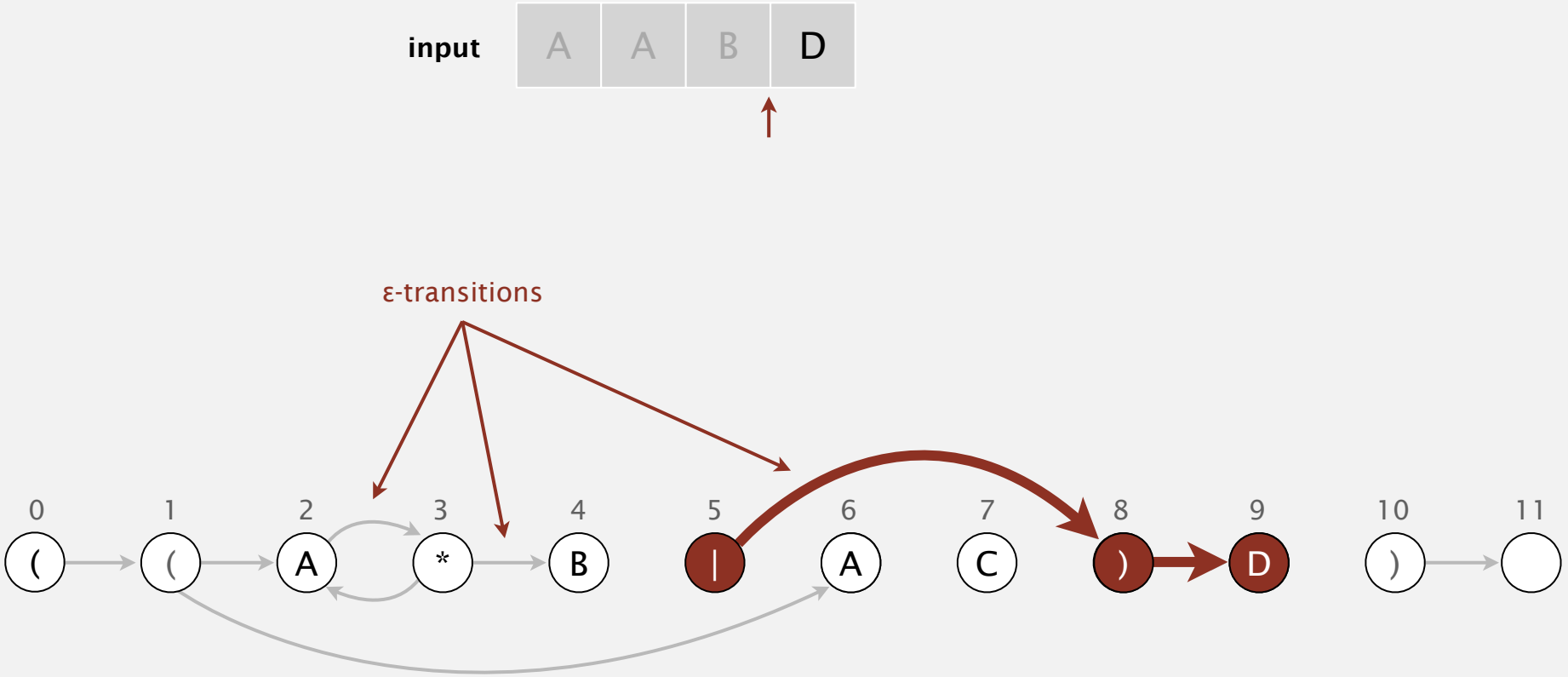


set of states reachable after matching A A B : { 5 }

NFA simulation demo

Read next input character.

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

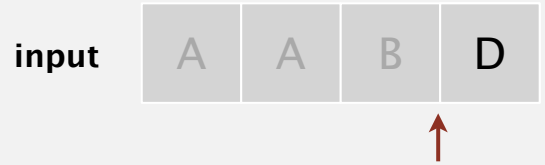


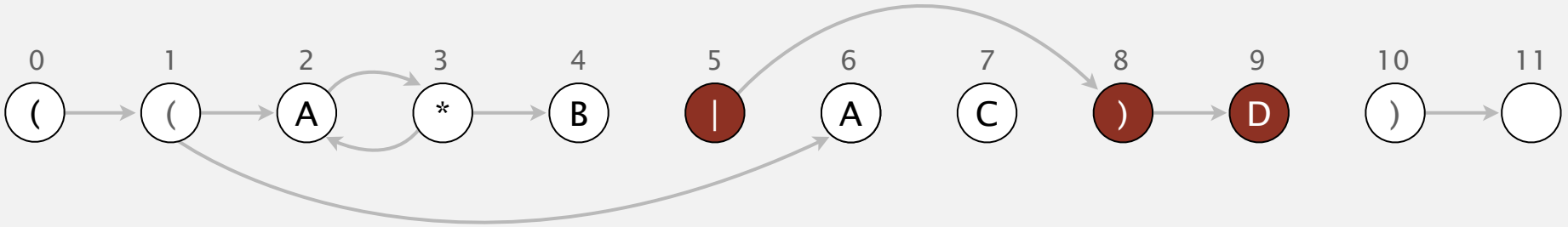
set of states reachable via ε-transitions after matching A A B

NFA simulation demo

Read next input character.

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



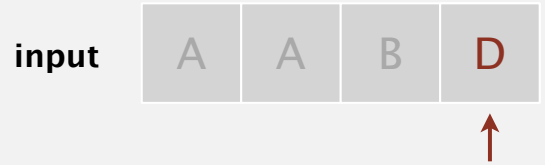


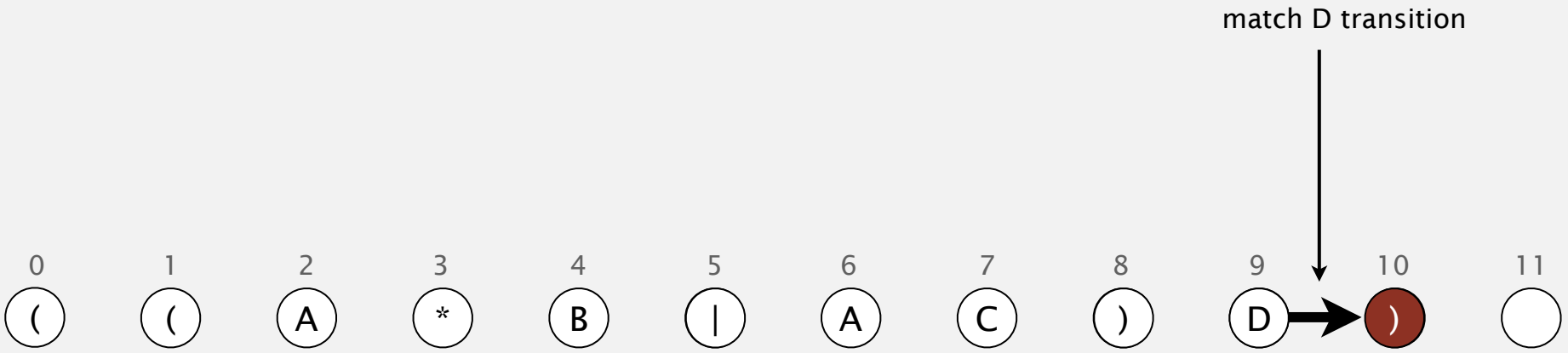
set of states reachable via ε-transitions after matching A A B : { 5, 8, 9 }

NFA simulation demo

Read next input character.

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



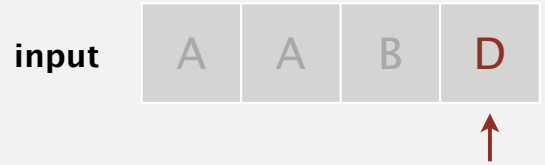


set of states reachable after matching A A B D

NFA simulation demo

Read next input character.

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

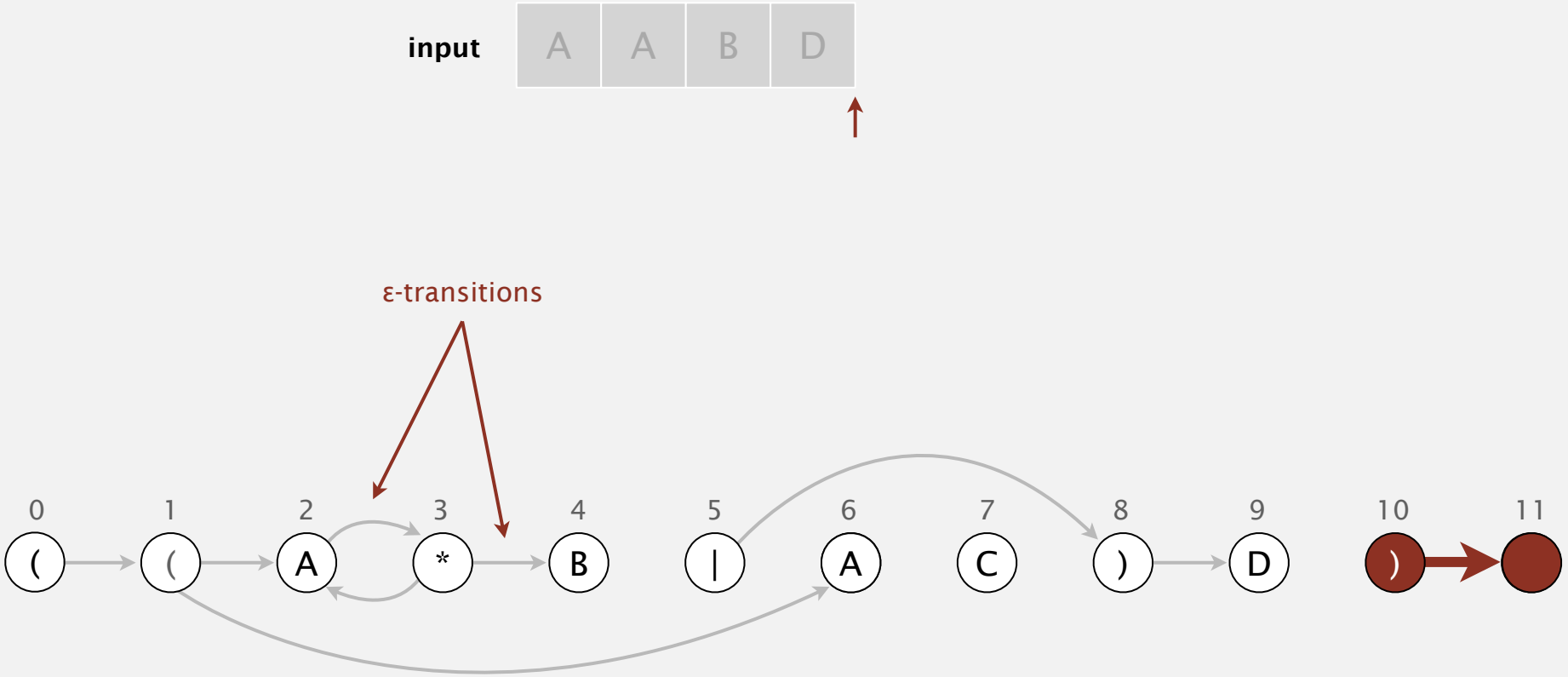


set of states reachable after matching A A B D : { 10 }

NFA simulation demo

Read next input character.

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

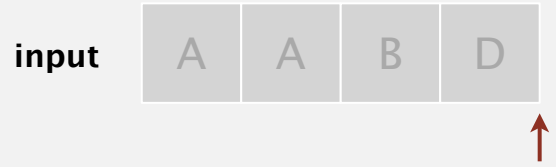


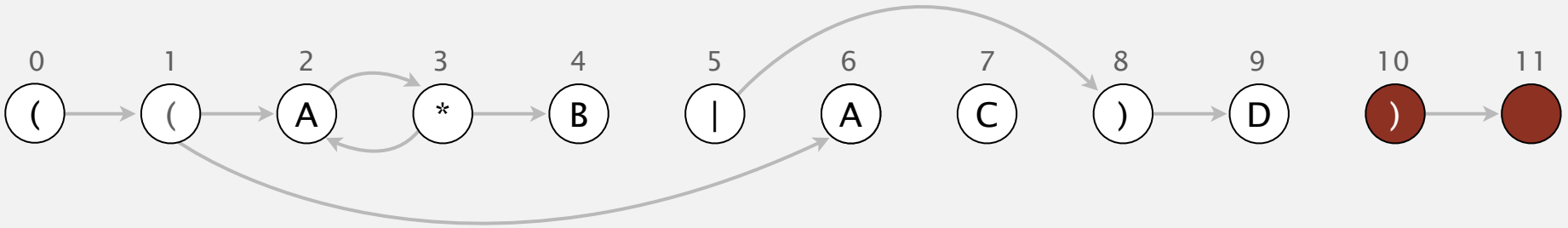
set of states reachable via ε-transitions after matching A A B D

NFA simulation demo

Read next input character.

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



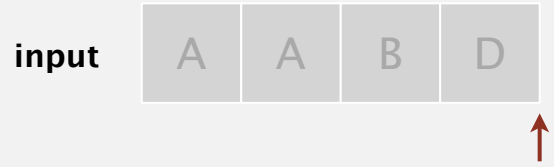


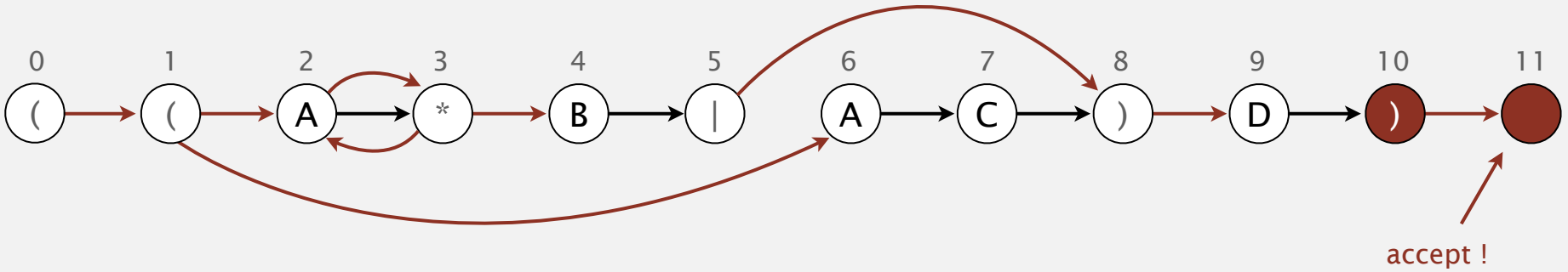
set of states reachable via ε-transitions after matching A A B D : { 10, 11 }

NFA simulation demo

When no more input characters:

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



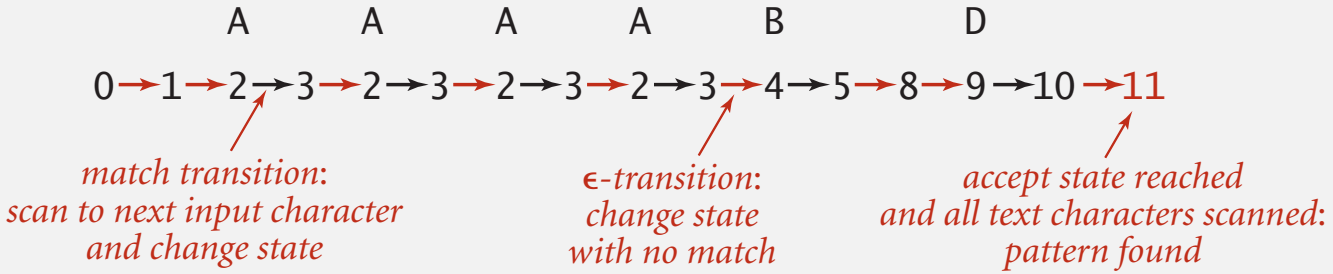


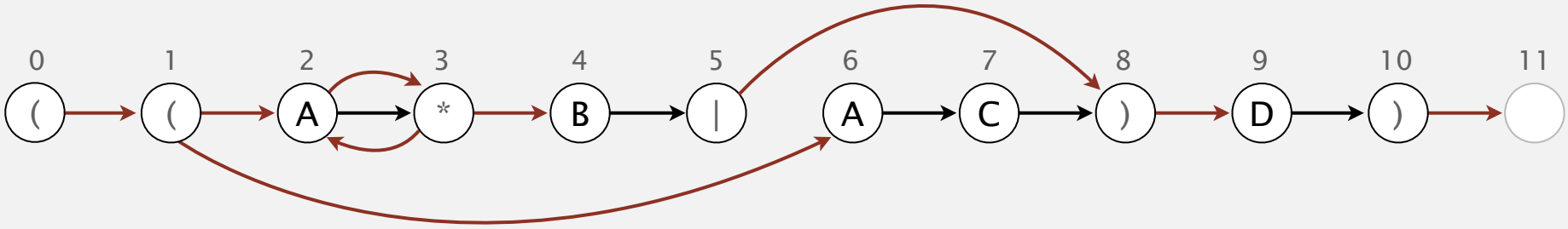
set of states reachable : { 10, 11 }

Nondeterministic finite-state automata

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

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





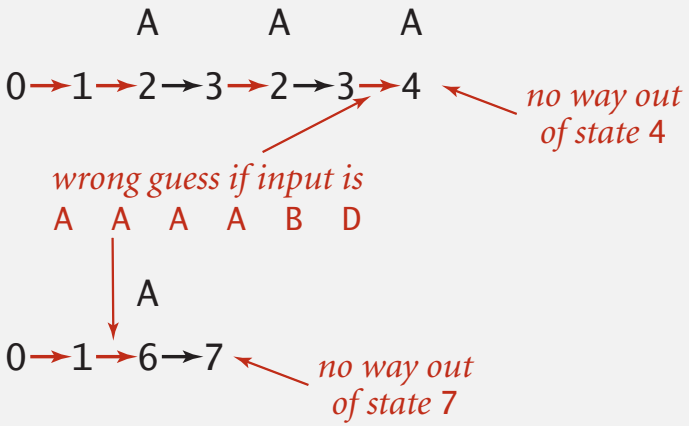
NFA corresponding to the pattern ((A * B | A C) D)

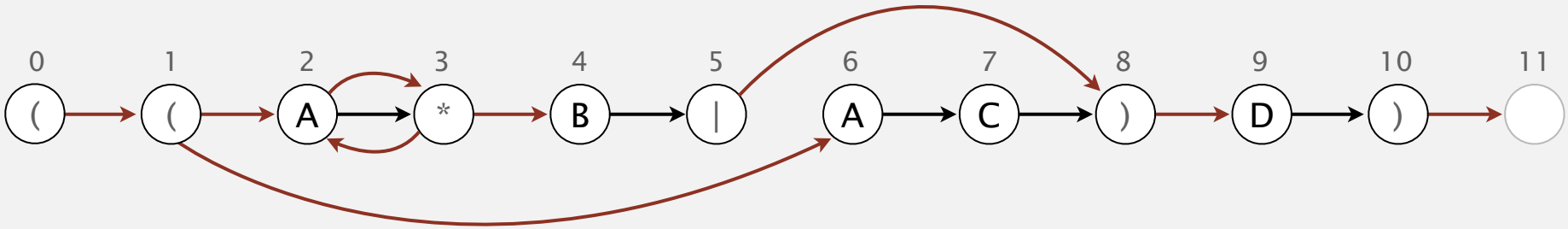
Nondeterministic finite-state automata

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

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

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





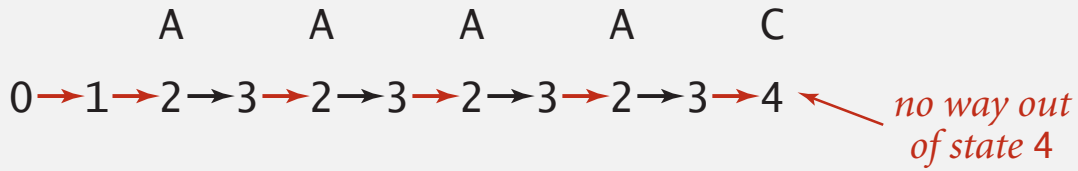
NFA corresponding to the pattern ((A * B | A C) D)

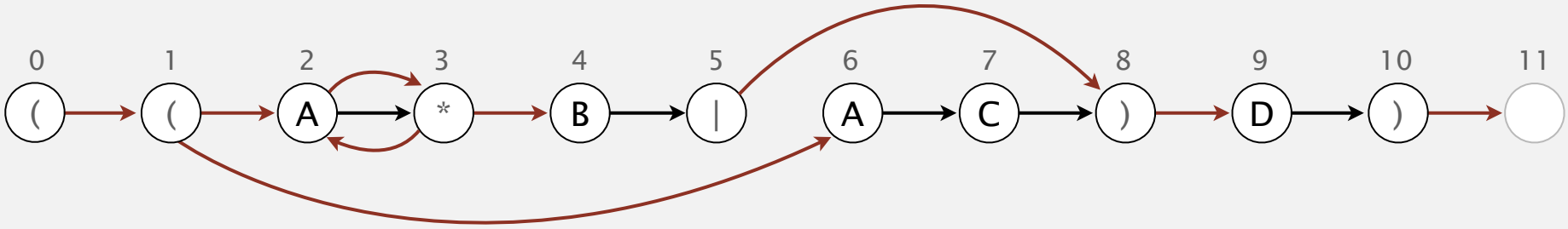
Nondeterministic finite-state automata

Q. Is A A A C matched by NFA?

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

[but need to argue about all possible sequences]





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

Nondeterminism

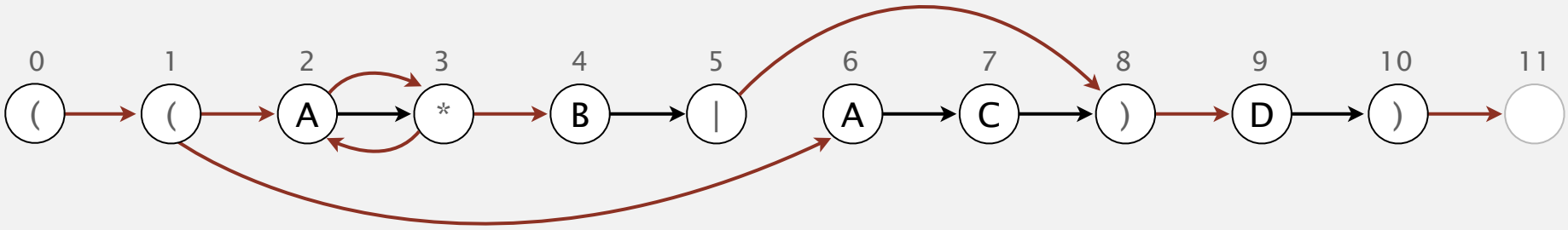
Q. How to determine whether a string is matched by an automaton?

DFA. Deterministic ⇒ easy because exactly one applicable transition.

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

Q. How to simulate NFA?

A. Systematically consider all possible transition sequences.



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

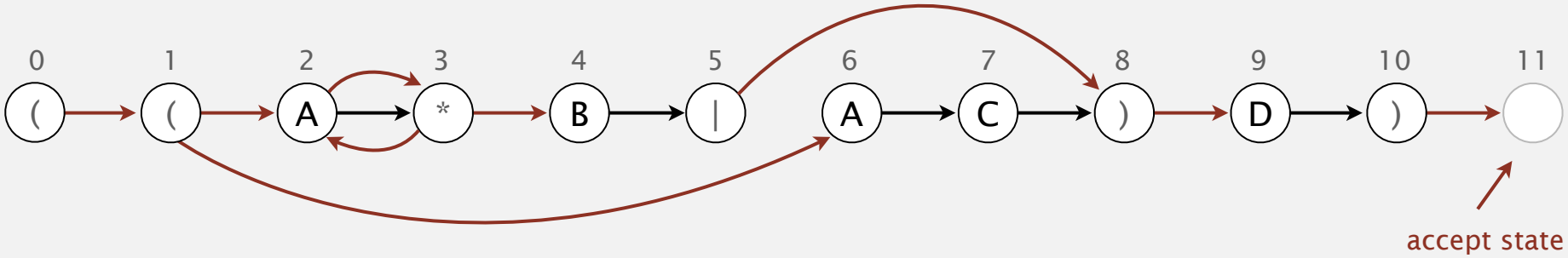
Runtime for NFA simulation

NFA simulation steps for each text character:

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

Let:

- N = length of text
- M = length of pattern (also number of DFA states)
- R = alphabet size



pollEv.com/jhug text to **37607**

Q: What is the runtime to complete NFA simulation in the worst case?

A. N	[320067]	D. N + M	[320257]
B. M	[320191]	E. N + M + R	[320258]
C. NM	[320231]	F. NMR	[320259]

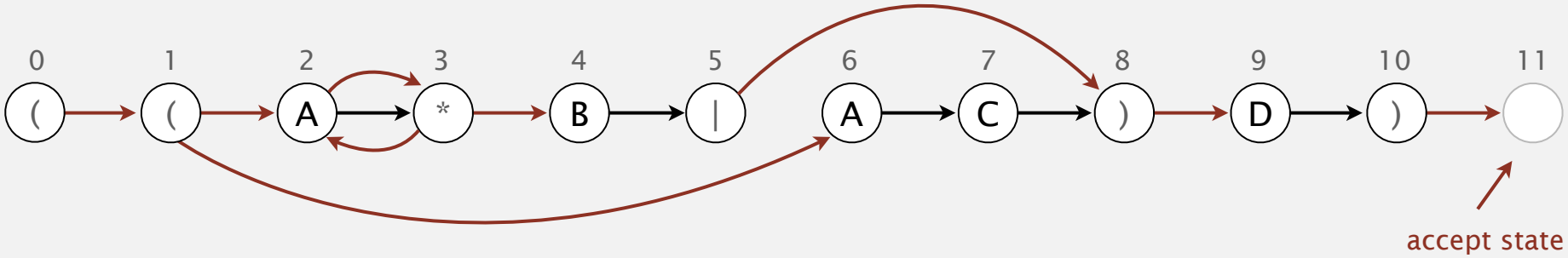
Runtime for NFA simulation

NFA simulation steps for each text character:

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

Let:

- N = length of text
- M = length of pattern (also number of DFA states)
- R = alphabet size



Q: What is the runtime to complete NFA simulation in the worst case?

C. NM

Finding states reachable by match transitions: Constant time

Finding states reachable by match transitions: V + E

Number of vertices: M. Number of edges: O(M) (as Arvind will explain)

NFA representation

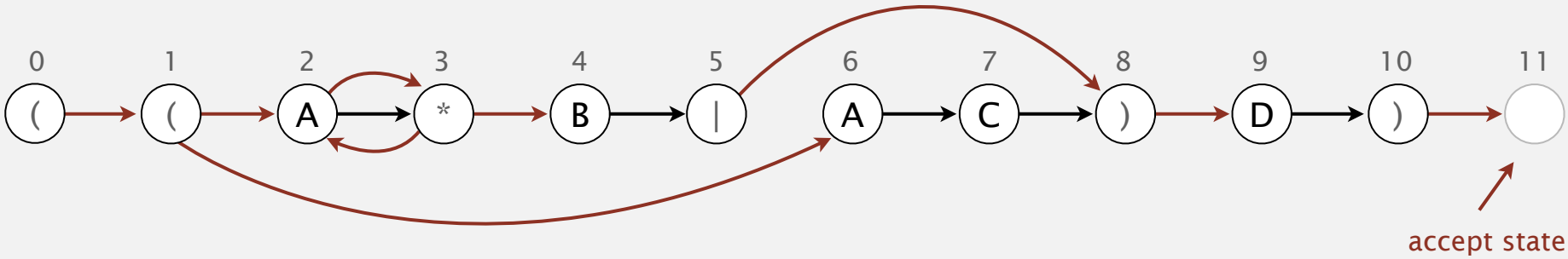
State names. Integers from 0 to M.

number of symbols in RE

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

ε-transitions. Store in a digraph G.

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

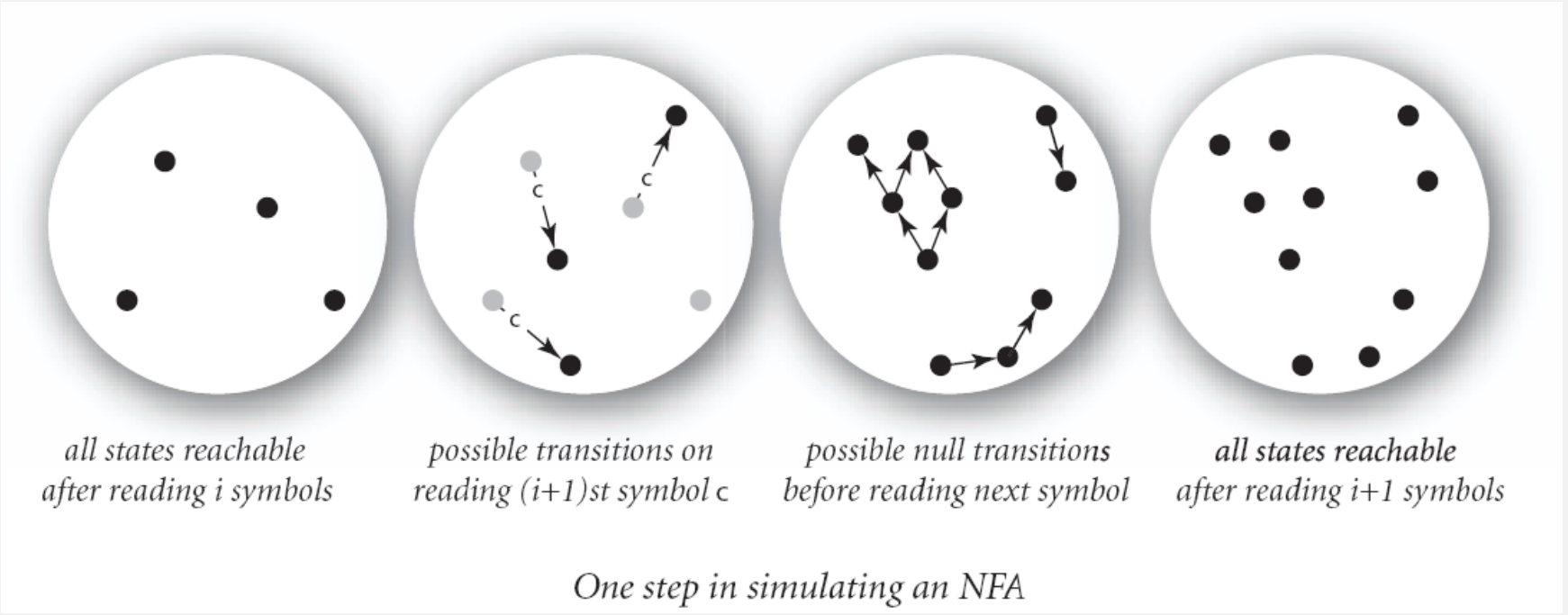


NFA corresponding to the pattern ((A * B | A C) D)

NFA simulation

Q. How to efficiently simulate an NFA?

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



One step in simulating an NFA

Q. How to perform reachability?

Digraph reachability

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

recall Section 4.2

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

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

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

NFA simulation: Java implementation

```
public class NFA
{
   private char[] re;       // match transitions
   private Digraph G;       // epsilon transition digraph
   private int M;           // number of states

   public NFA(String regexp)
   {
      M  = regexp.length();
      re = regexp.toCharArray();
      G = buildEpsilonTransitionsDigraph();
   }

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

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

}
```

stay tuned (next segment)

NFA simulation: Java implementation

```
public boolean recognizes(String txt)
{
   Bag<Integer> pc = new Bag<Integer>();
   DirectedDFS dfs = new DirectedDFS(G, 0);
   for (int v = 0; v < G.V(); v++)
      if (dfs.marked(v)) pc.add(v);

   for (int i = 0; i < txt.length(); i++)
   {
      Bag<Integer> 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

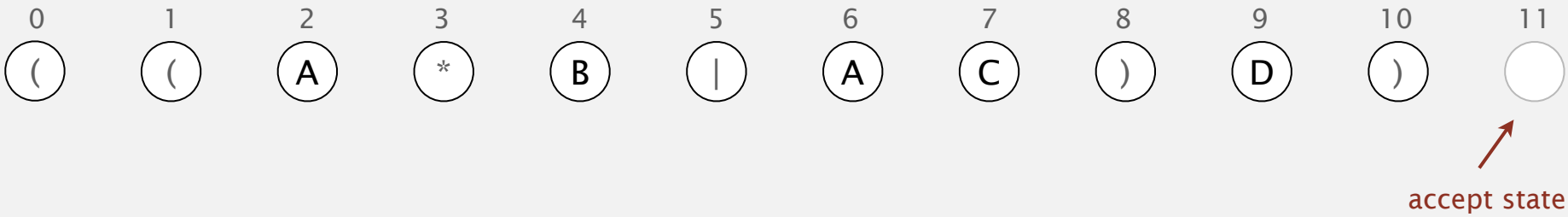
http://algs4.cs.princeton.edu

5.4 Regular Expressions

- *regular expressions*
- *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.



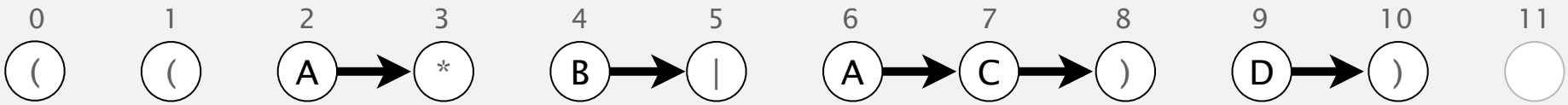
NFA corresponding to the pattern ((A * B | A C) D)

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`

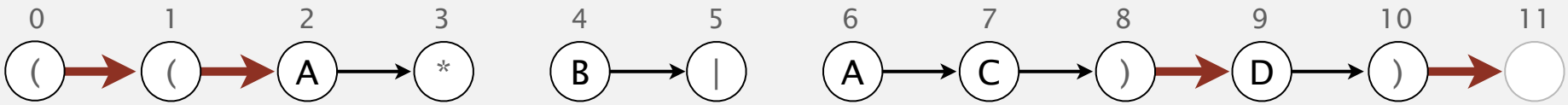
Metacharacters. `( ) . * |`



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.

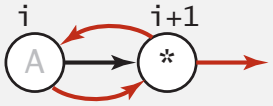


NFA corresponding to the pattern ((A * B | A C) D)

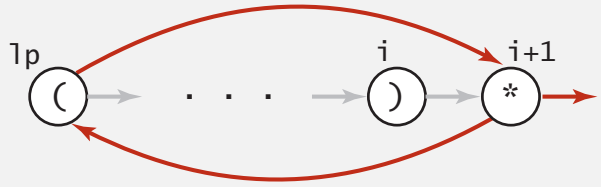
Building an NFA corresponding to an RE

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

single-character closure



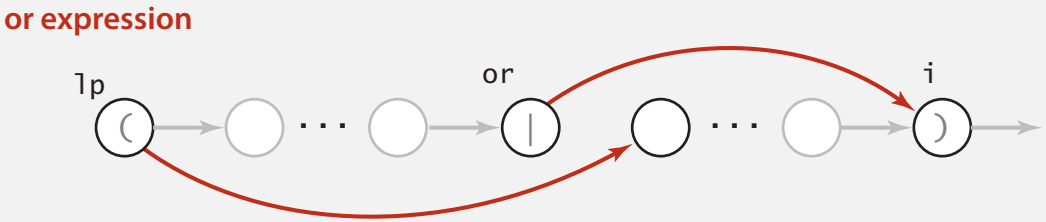
closure expression

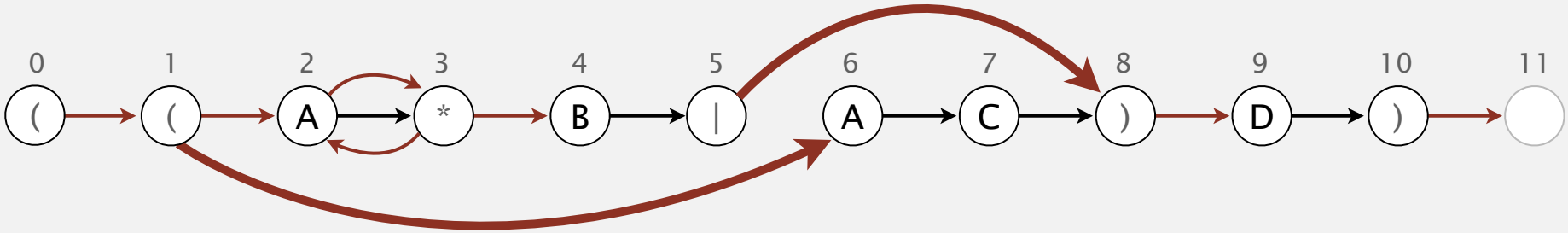


NFA corresponding to the pattern ((A * B | A C) D)

Building an NFA corresponding to an RE

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





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

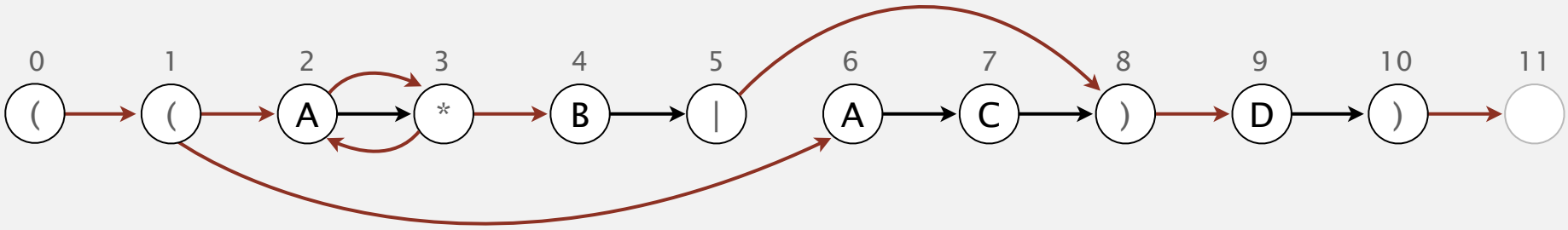
NFA construction: implementation

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

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

Solution. Maintain a stack.

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



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

NFA construction demo

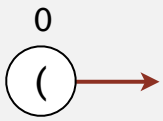
stack

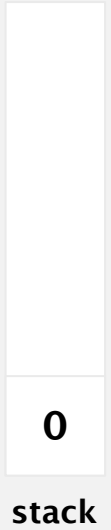
((A * B | A C) D)

NFA construction demo

Left parenthesis.

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



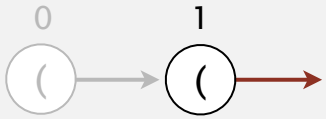


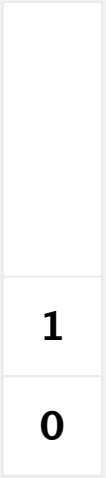
((A * B | A C) D)

NFA construction demo

Left parenthesis.

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



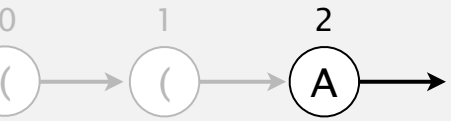


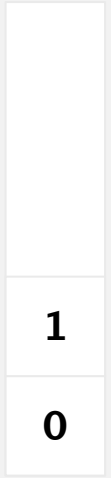
((A * B | A C) D)

NFA construction demo

Alphabet symbol.

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





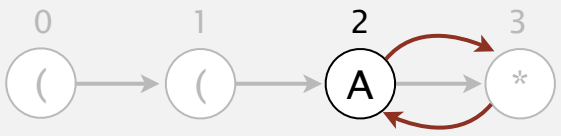
((A * B | A C) D)

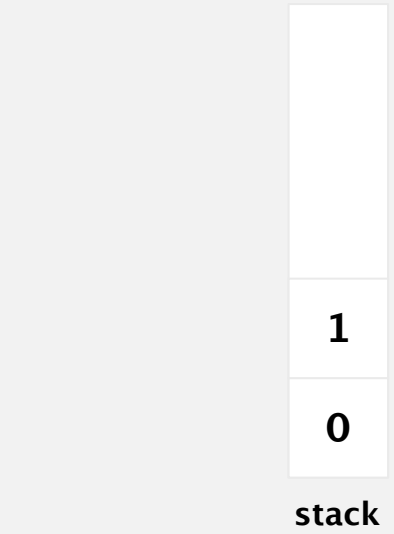
NFA construction demo

Alphabet symbol.

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

  add ε-transitions if next character is *.



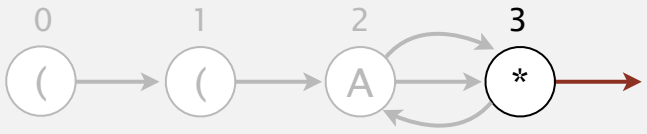


((A * B | A C) D)

NFA construction demo

Closure symbol.

- Add ε-transition to next state.

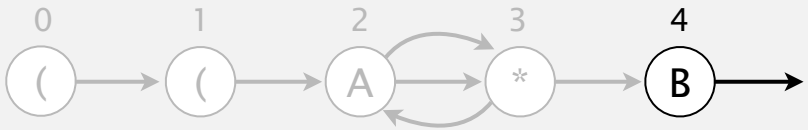


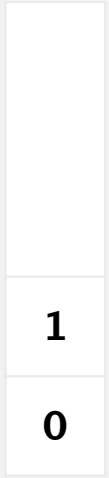
((A * B | A C) D)

NFA construction demo

Alphabet symbol.

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



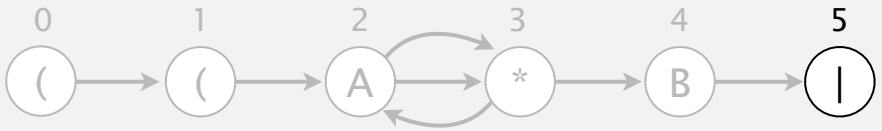


((A * B | A C) D)

NFA construction demo

Or symbol.

- Push index of state corresponding to | onto stack.

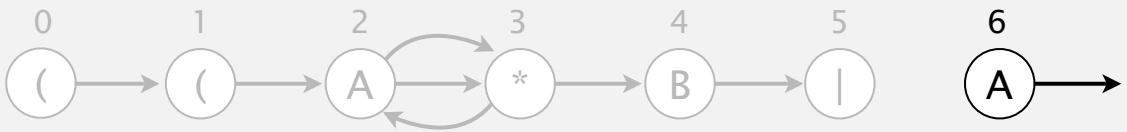


((A * B | A C) D)

NFA construction demo

Alphabet symbol.

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

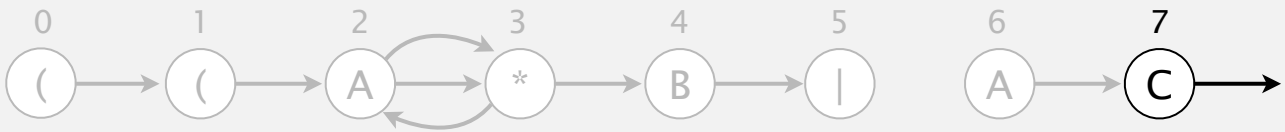


((A * B | A C) D)

NFA construction demo

Alphabet symbol.

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

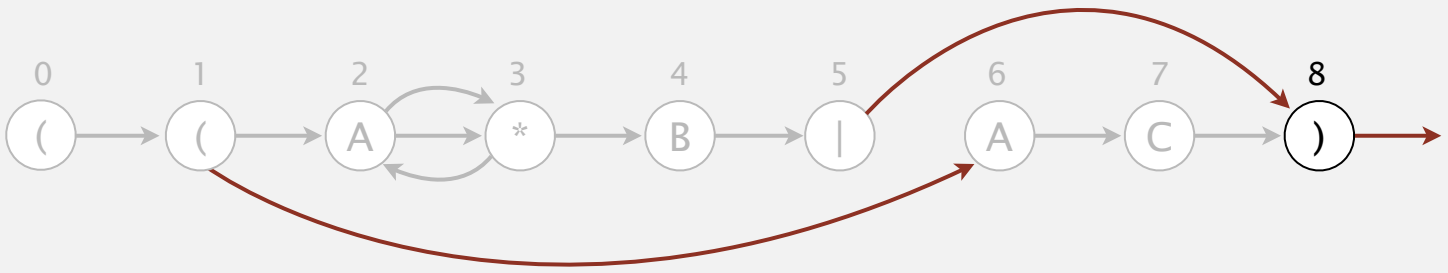


((A * B | A C) D)

NFA construction demo

Right parenthesis.

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

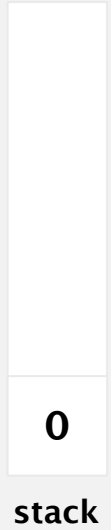


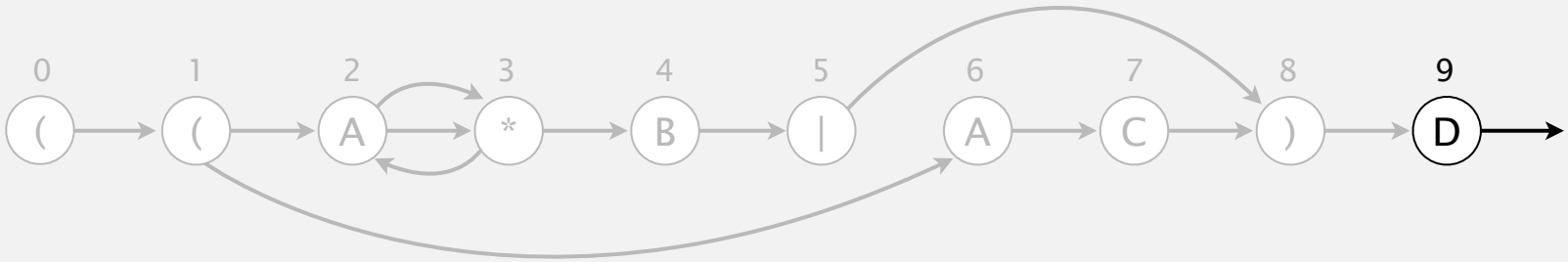
((A * B | A C) D)

NFA construction demo

Alphabet symbol.

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



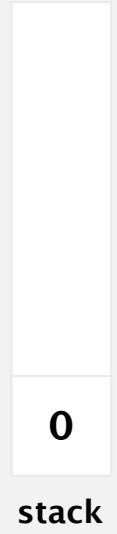


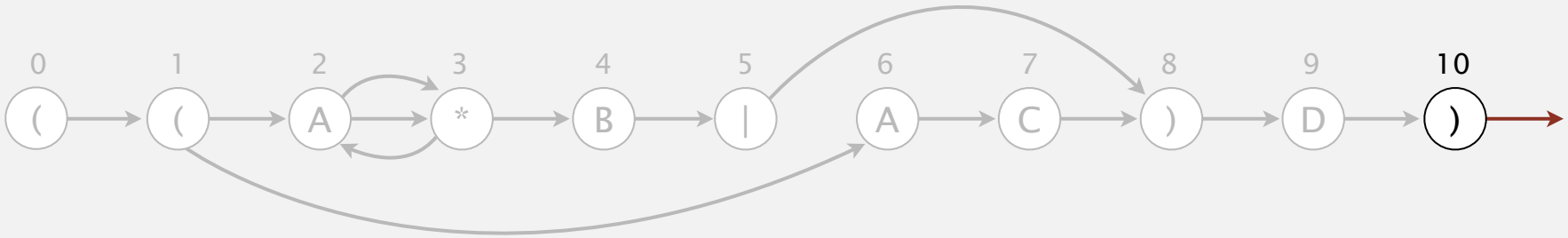
((A * B | A C) D)

NFA construction demo

Right parenthesis.

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



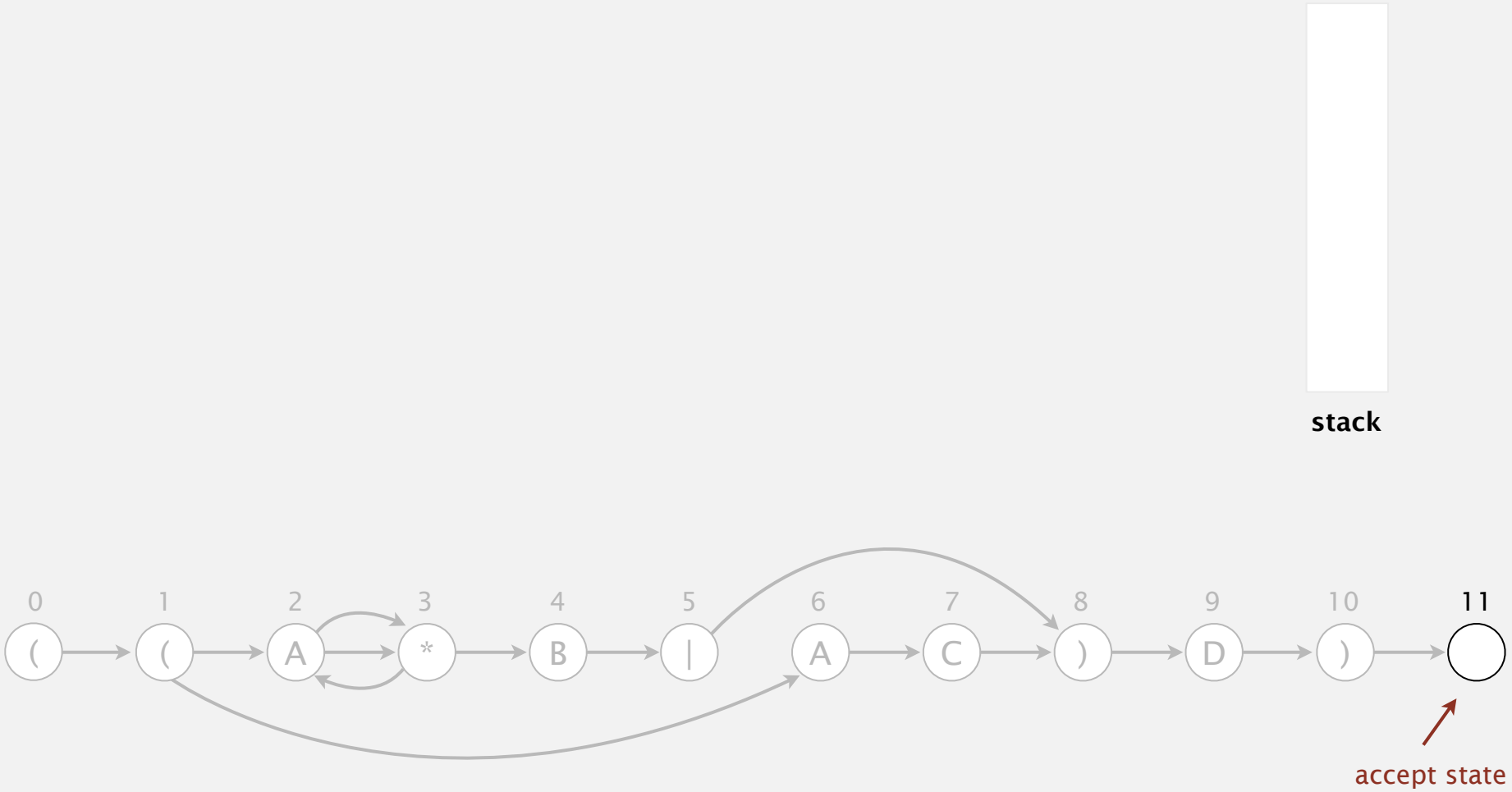


((A * B | A C) D)

NFA construction demo

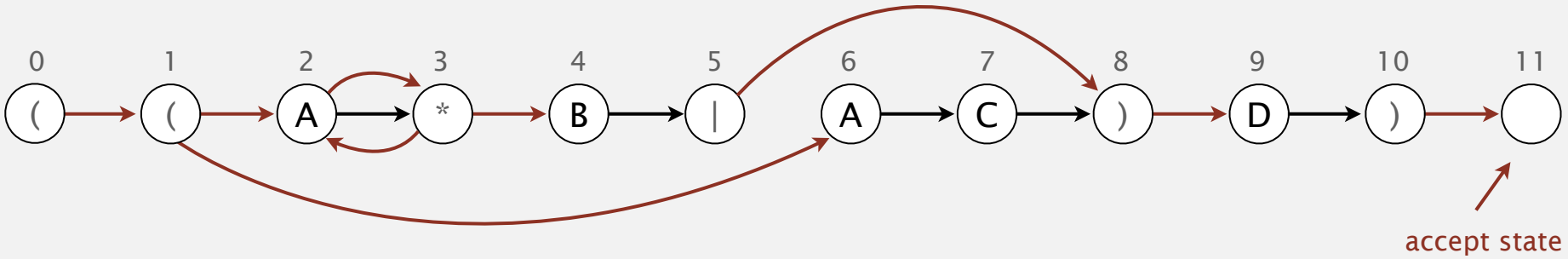
End of regular expression.

- Add accept state.



NFA construction demo

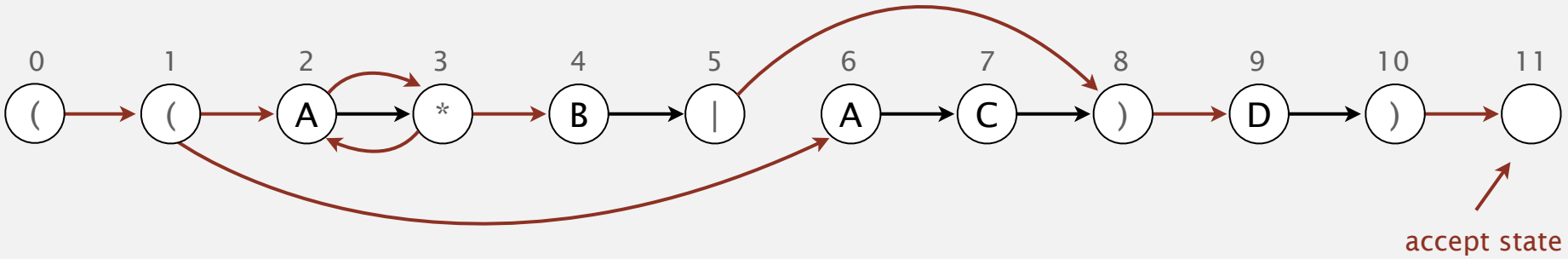
stack



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

NFA construction demo

stack



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

NFA construction: Java implementation

```
private Digraph buildEpsilonTransitionDigraph() {
   Digraph G = new Digraph(M+1);
   Stack<Integer> ops = new Stack<Integer>();
   for (int i = 0; i < M; i++) {
      int lp = i;

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

      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

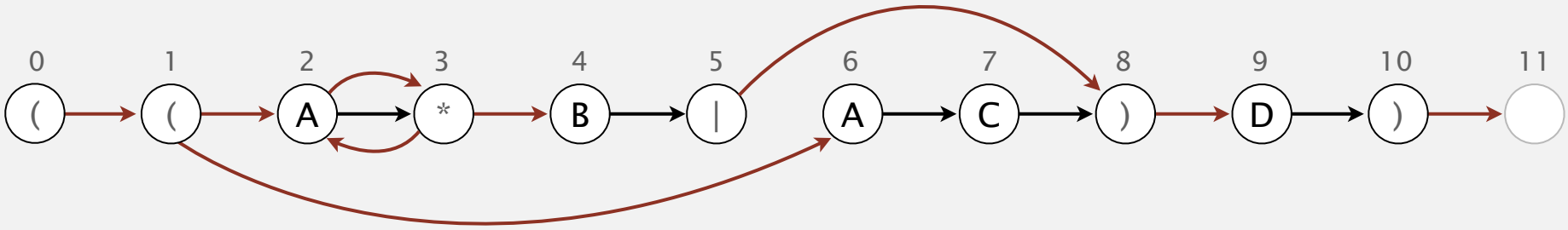
(needs 1-character lookahead)

metasymbols

NFA construction: analysis

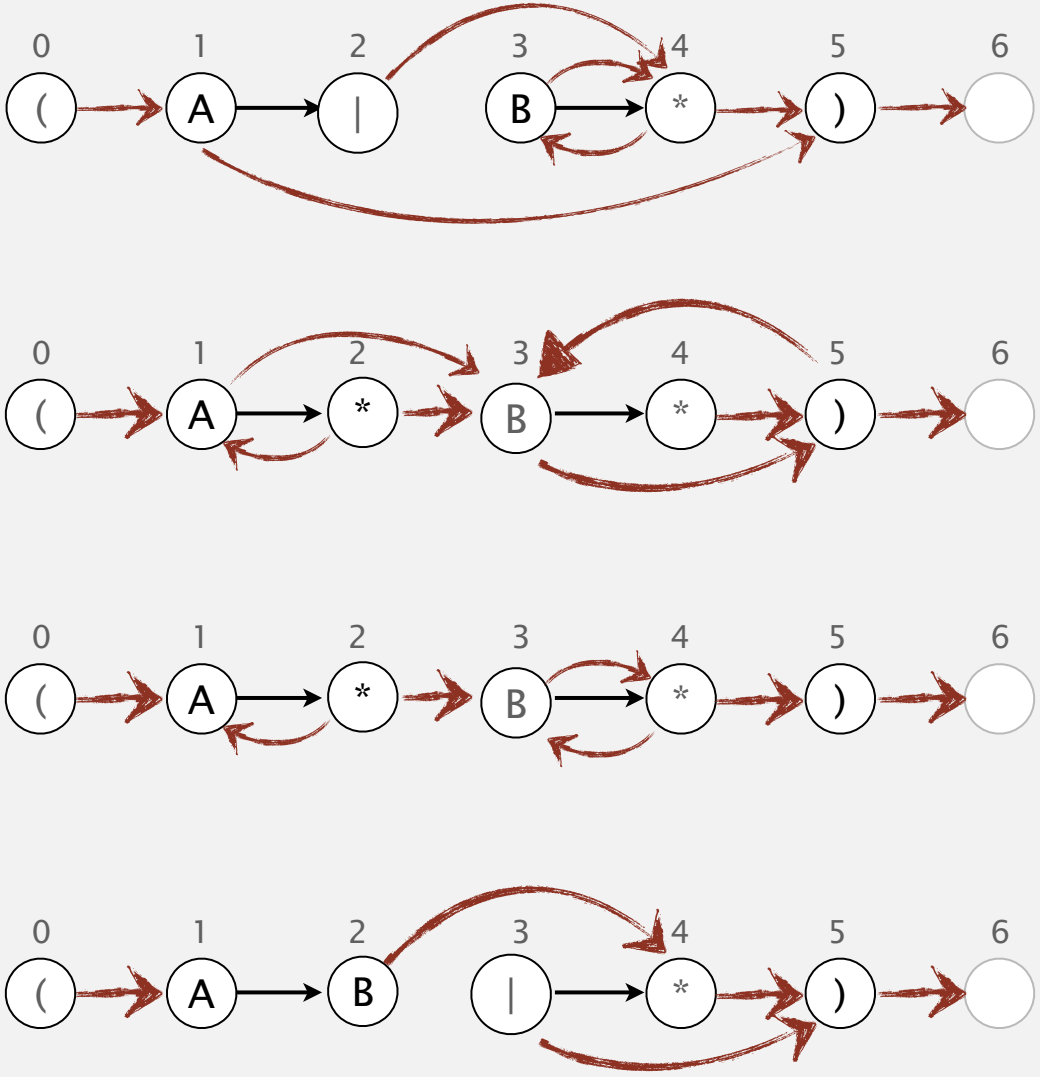
Proposition. Building the NFA corresponding to an M-character RE takes time and space proportional to M.

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)

What regEx (if any) are equivalent to the following NFAs?

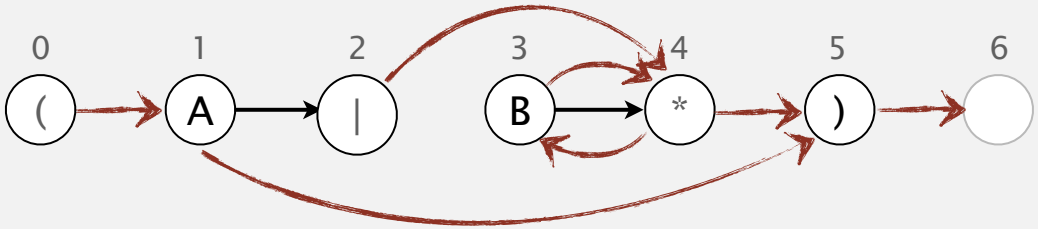


Warning: These NFAs were NOT generated by our mechanical procedure on the previous slide!

What regEx (if any) are equivalent to the following NFAs?

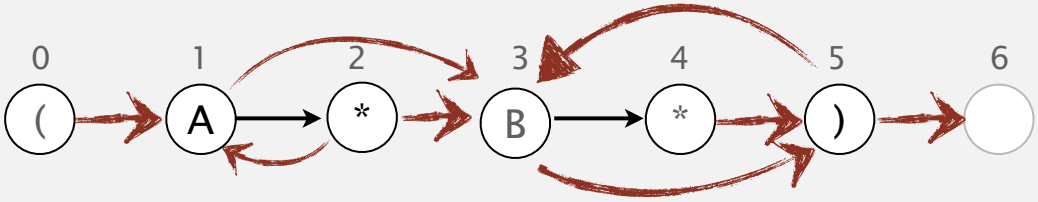
(Ø | AB*)

Ø is the empty set!



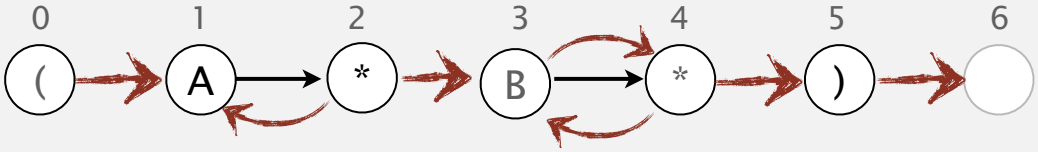
Epsilon from 2 to 4 should be from 2 to 5.

(A*B*)



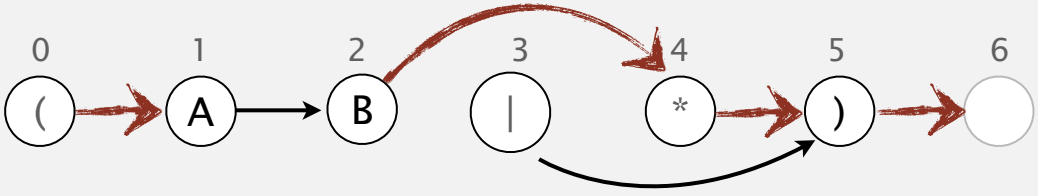
Epsilon from 1 should go to 2. Epsilon from 3 should go to 4. Epsilon from 5 to 3 should go to 4.
Despite these three 'errors' the regex is exactly as in the nodes.

(A+B*) (AA*B*)



There is a missing epsilon from 2 to 1, this effectively transforms the * into a +.

Invalid NFA, black arrow from |



http://algs4.cs.princeton.edu

5.4 Regular Expressions

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

Generalized regular expression print

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

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

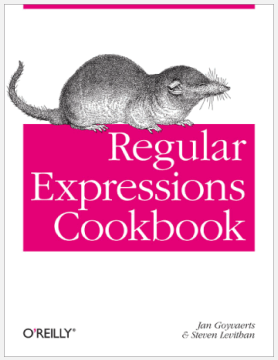
contains RE as a substring

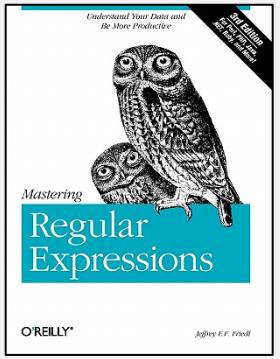
Bottom line. Worst-case for grep (proportional to MN) is the same as for brute-force substring search.

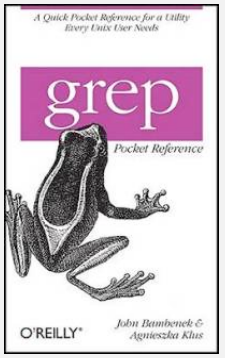
Industrial-strength grep implementation

To complete the implementation:

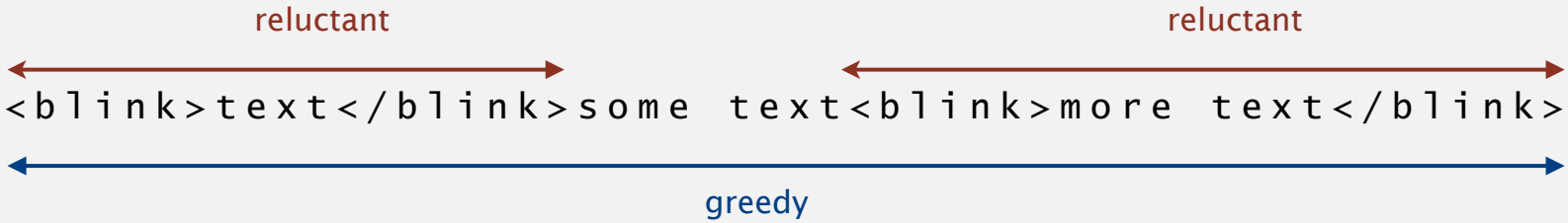
- Add character classes. [0-9][0-9]
- Handle metacharacters. a\.b
- Add capturing capabilities.
- Extend the closure operator. a+b
- Error checking and recovery. *|*|*
- Greedy vs. reluctant matching. to.*?be







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 and text editors 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
```

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));
   }
}
```

Does not build NFA!

```
% 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

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

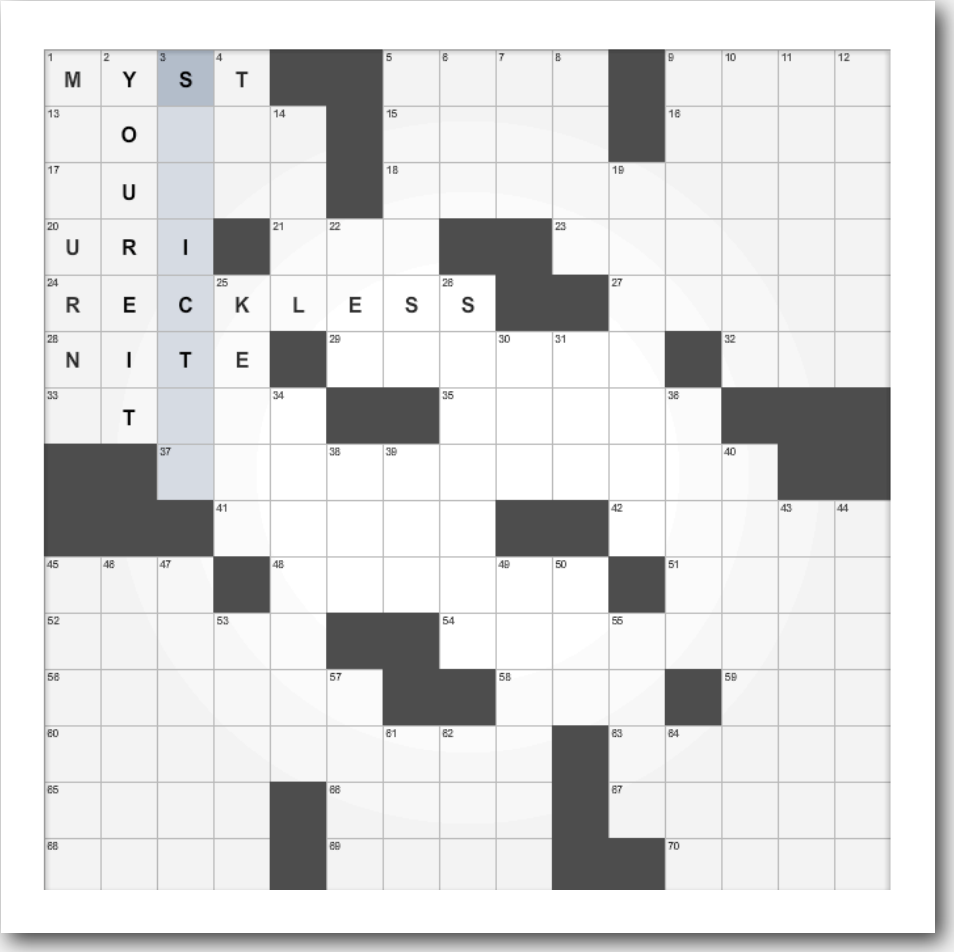
```
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaac              1.6 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaac             3.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac            9.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac          23.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac         62.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac       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.

Typical grep application: crossword puzzles



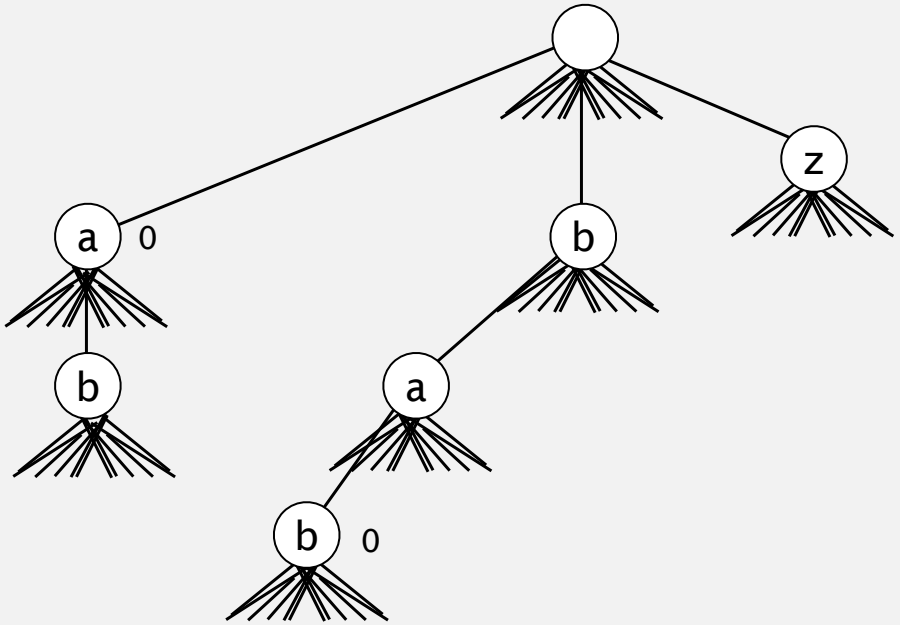
```
% more words.txt
a
aback
abacus
abalone
abandon
...

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

dictionary
(standard in Unix)
also on booksite

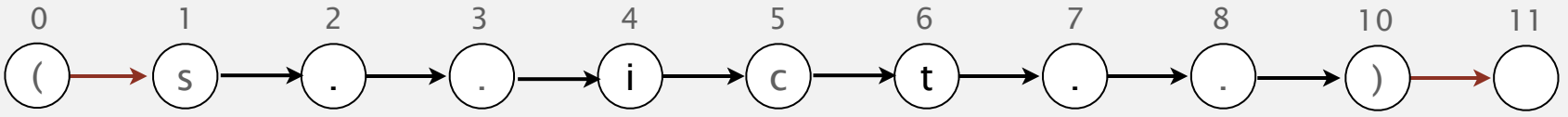
Tries vs. Regex

Group Discussion

- Is it better to implement this particular search using an r-way trie or an NFA? (Assume we have the memory somehow to store the r-way trie)




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

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



NFA corresponding to the pattern (s . . i c t . .)

accept state

Not-so-regular expressions

Back-references.

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

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

Some non-regular languages.

- Strings of the form ww for some string w: `beriberi`.
- Unary strings with a composite number of 1s: `111111`.
- Bitstrings with an equal number of 0s and 1s: `01110100`.
- Watson-Crick complemented palindromes: `atttcggaaat`.

Remark. Pattern matching with back-references is intractable.

Context

Abstract machines, languages, and nondeterminism.

- Basis of the theory of computation.
- Intensively studied since the 1930s.
- Basis of programming languages.

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

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

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

Some programs produce themselves as output

*Quine.java

```
public class Quine
{
  public static void main( String[] args )
  {
    char q = 34;       // Quotation mark characte
    String[] l = {     // Array of source code
    "public class Quine",
    "{",
    "  public static void main( String[] args )",
```

Console

<terminated> Quine [Java Application] C:\Program Files\Java\jre6\bin\javaw.ex

```
public class Quine
{
  public static void main( String[] args )
  {
    char q = 34;       // Quotation mark character
    String[] l = {     // Array of source code
    "public class Quine",
    "{",
    "  public static void main( String[] args )",
```

What about programs that take themselves as input?