NFA to DFA

- the Boolean algebra of languages
- regular expressions

A mathematical definition of a Finite State Machine.

$M = (Q, \Sigma, B, A, \delta)$

Q: the set of states,

Σ: the alphabet of the machine

- the tokens the machine can process,

B: the set of **beginning** or start states of the machine

A: the set of the machine's **accepting** states.

δ: the set of **transitions**

is a set of (state, symbol, state) triples

$\delta \subseteq Q \times \Sigma \times Q.$

A *trace* for $s = <x_0, \ldots x_{k-1}> \in \Sigma^*$ (a string of length **k**)

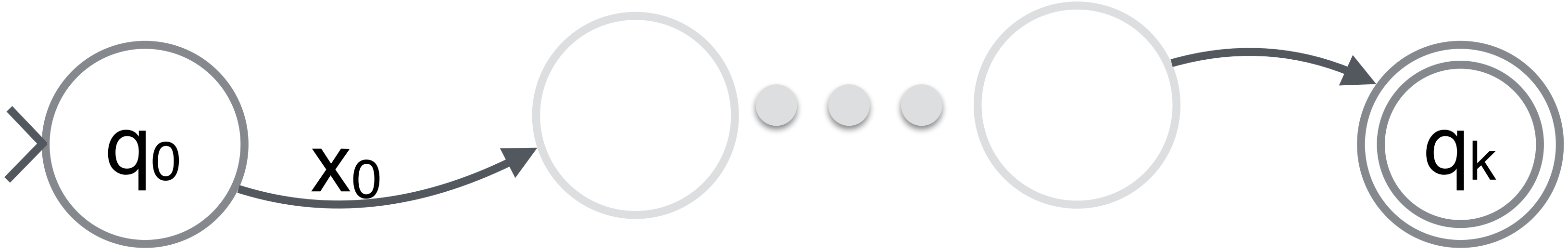
is a sequence of **k+1** states $<q_0, \ldots q_k>$

such that $(q_i, x_i, q_{i+1}) \in \delta$ for each $i < k$

$M = (Q, \Sigma, B, A, \delta)$

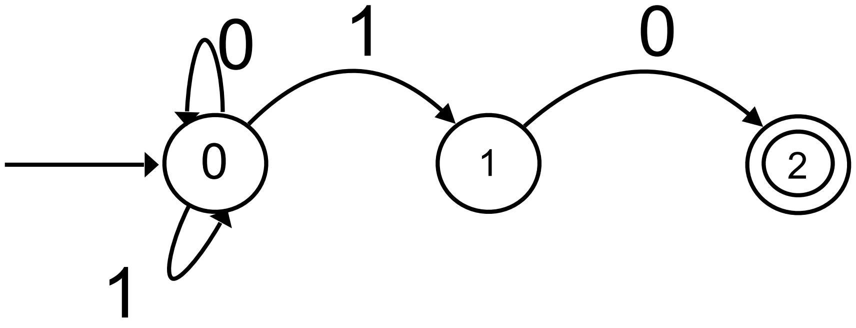
A *trace* for $s = \langle x_0, \ldots, x_{k-1} \rangle \in \Sigma^*$ (a string of length k) is a sequence of $k+1$ states $\langle q_0, \ldots q_k \rangle$ such that $(q_i, x_i, q_{i+1}) \in \delta$ for each $i < k$

We say s is *accepted* by M iff there is a trace $\langle q_0, \ldots q_k \rangle$ for s such that $q_0 \in B$ and $q_k \in A$

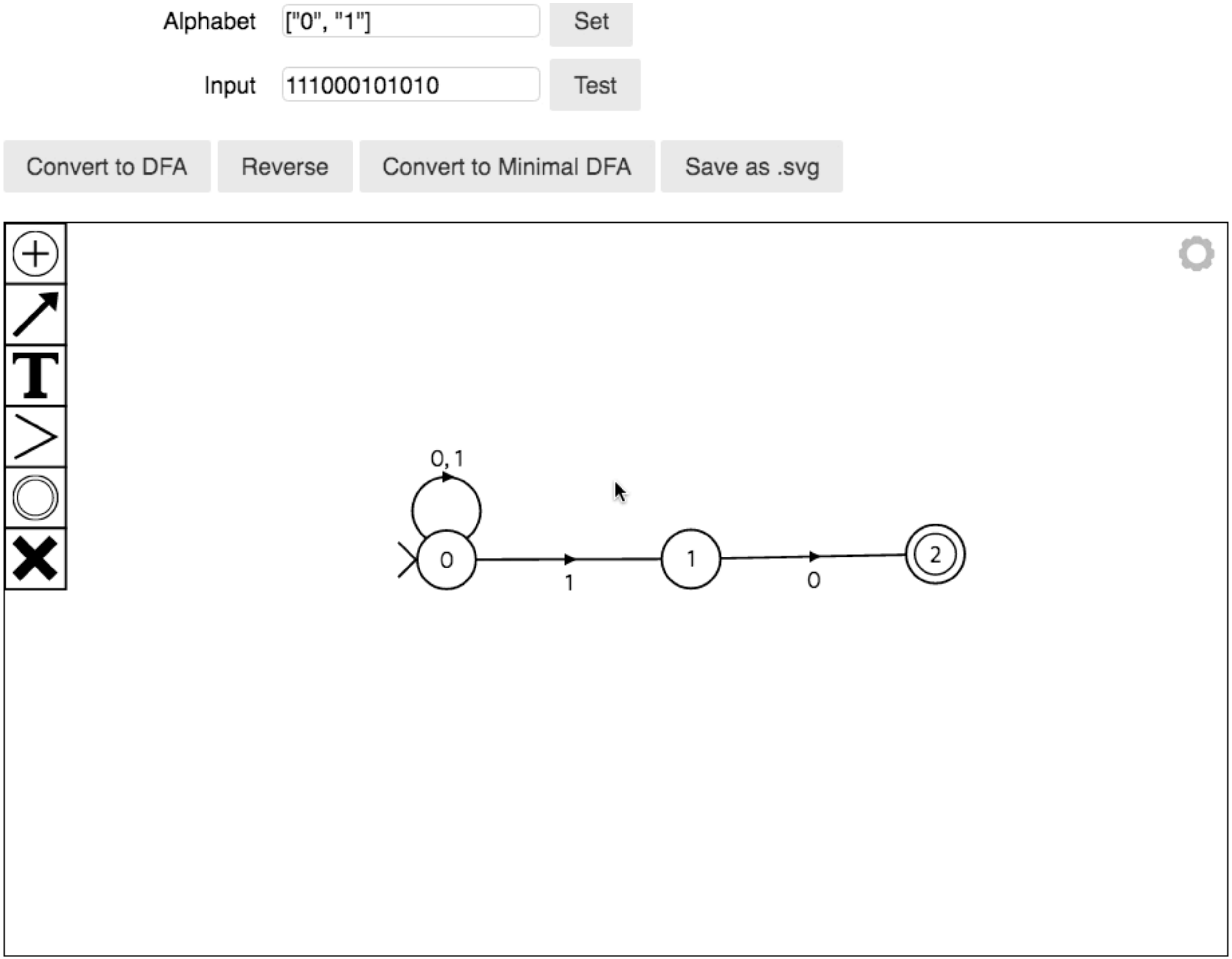


Non Determinism

In a non-deterministic machine (NFA), each state may have any number of transitions with the same input symbol, leaving to different successor states.



	0	1
0	0	0,1
1	2	
2		

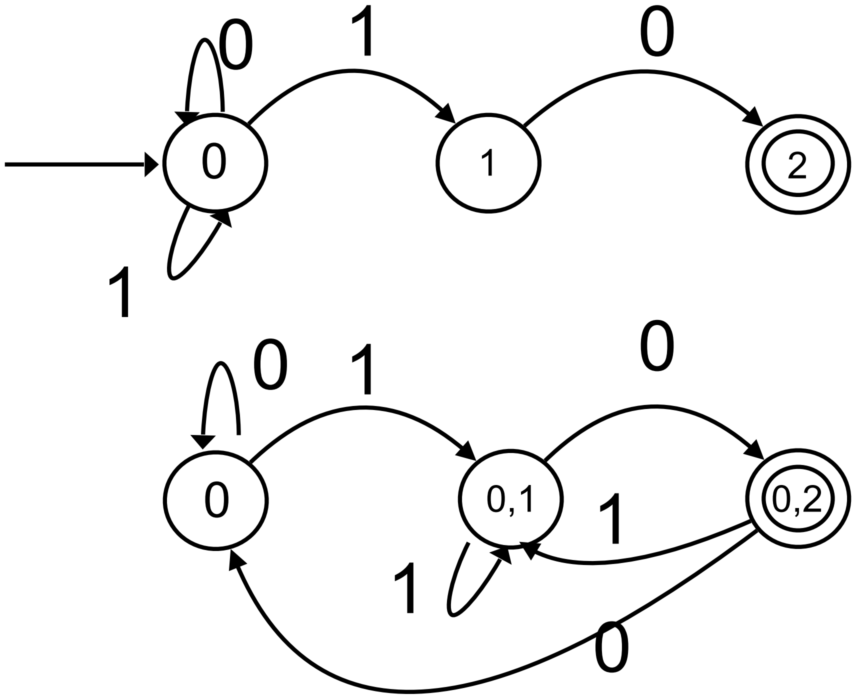


Alphabet ["0", "1"]

Input 111000101010

Non Determinism

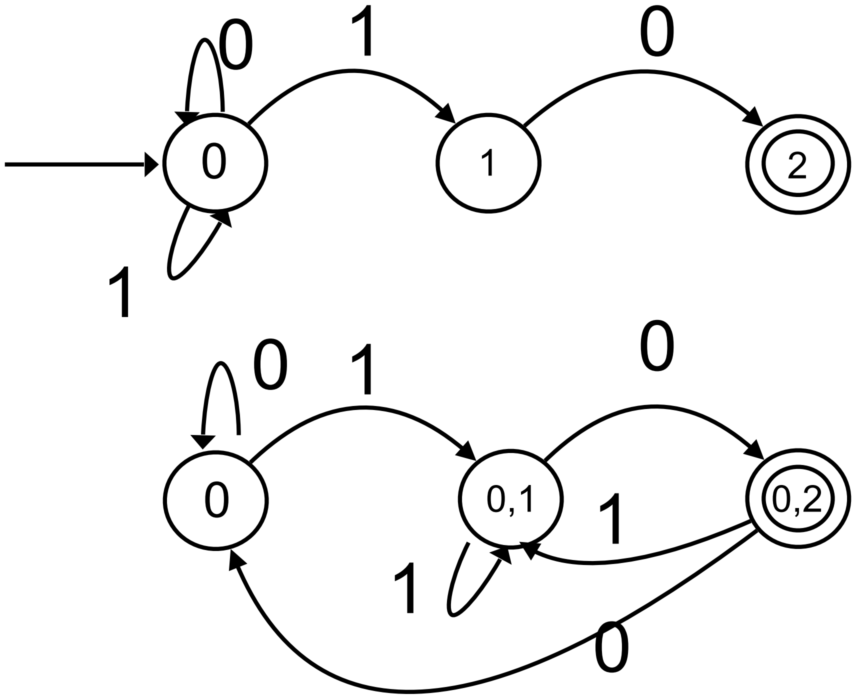
In a non-deterministic machine (NFA), each state may have any number of transitions with the same input symbol, leaving to different successor states.



	0	1
0	0	0,1
1	2	
2		
0,1	0,2	0,1

Non Determinism

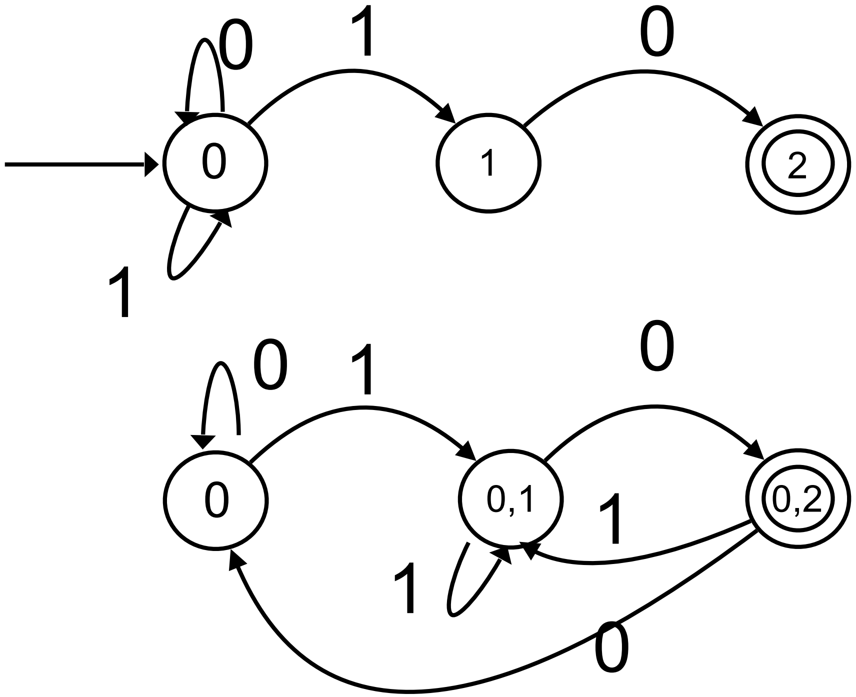
In a non-deterministic machine (NFA), each state may have any number of transitions with the same input symbol, leaving to different successor states.



	0	1
0	0	0,1
1	2	
2		
0,1	0,2	0,1
0,2	0	0,1

Non Determinism

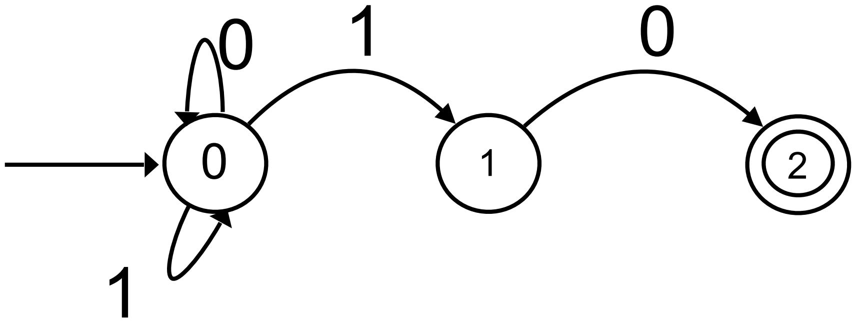
We can simulate a non-deterministic machine using a deterministic machine – by keeping track of the set of states the NFA could possibly be in.

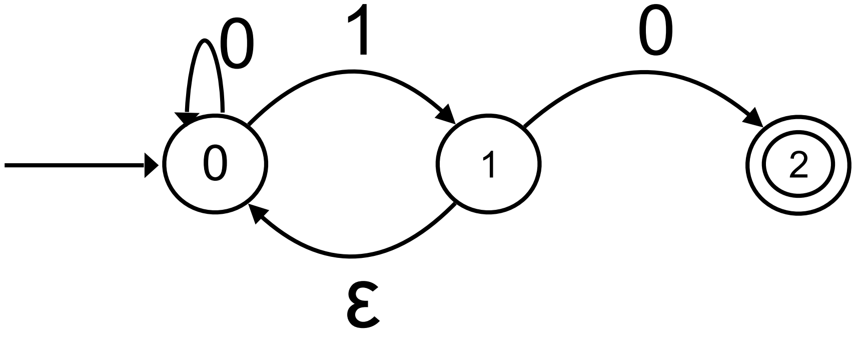


	0	1
0	0	0,1
1	2	
2		
0,1	0,2	0,1
0,2	0	0,1

Internal Transitions

We sometimes add an internal transition ε to a non-deterministic machine (NFA)This is a state change that consumes no input.





	0	1	ε
0	0	1	
1	2		0
2			

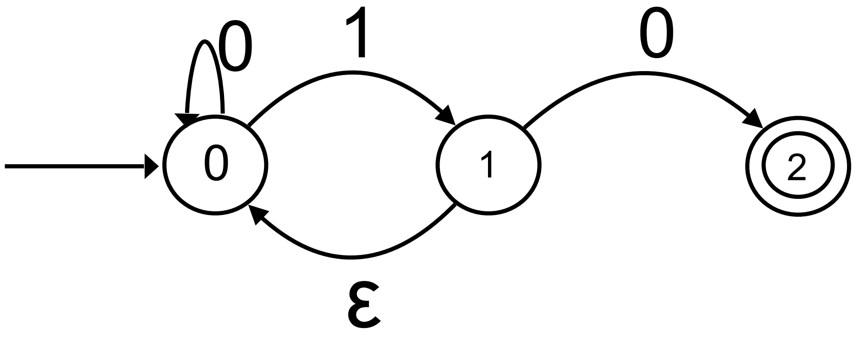
Internal Transitions

We sometimes add **internal transitions** – labelled ε – to a non-deterministic machine (NFA).

This is a state change that consumes no input.

It introduces non-determinism in the observed behaviour of the machine.

	0	1	ε
0	0	1	
1	2		0
2			



	0ε*	1ε*
0	0	1,0
1	2	
2		

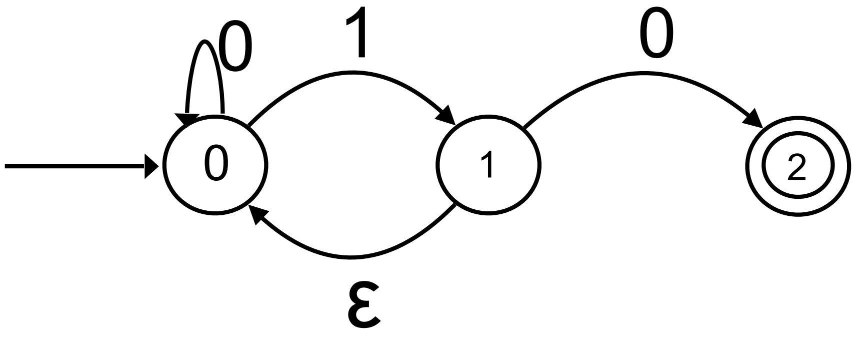
Internal Transitions

We sometimes add **internal transitions** – labelled ε – to a non-deterministic machine (NFA).

This is a state change that consumes no input.

It introduces non-determinism in the observed behaviour of the machine.

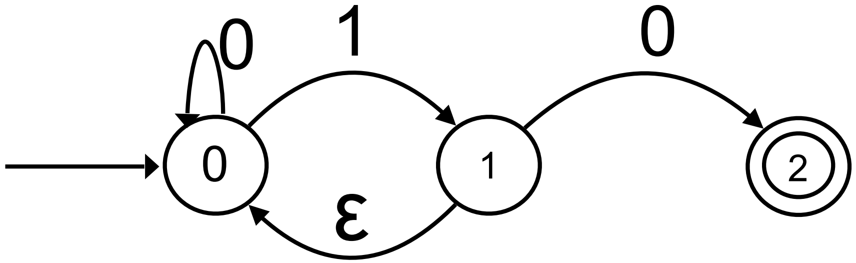
	0	1	ε
0	0	1	
1	2		0
2			



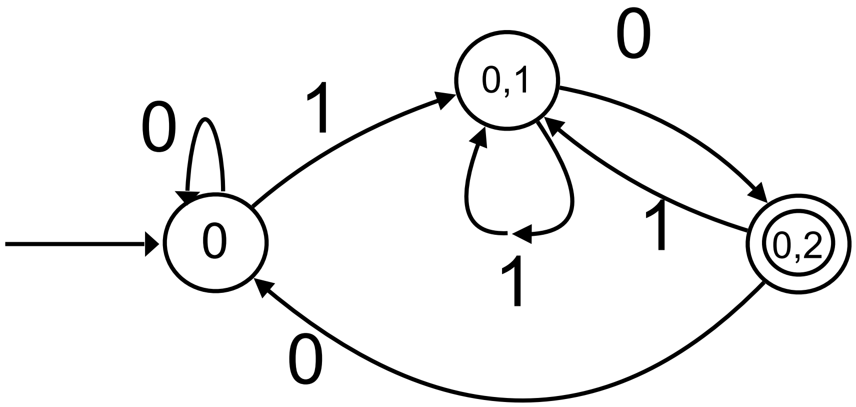
	0ε*	1ε*
0	0	0,1
1	2	
2		
0,1	0,2	1

Internal Transitions

We sometimes add **internal transitions** – labelled ε – to a non-deterministic machine (NFA).

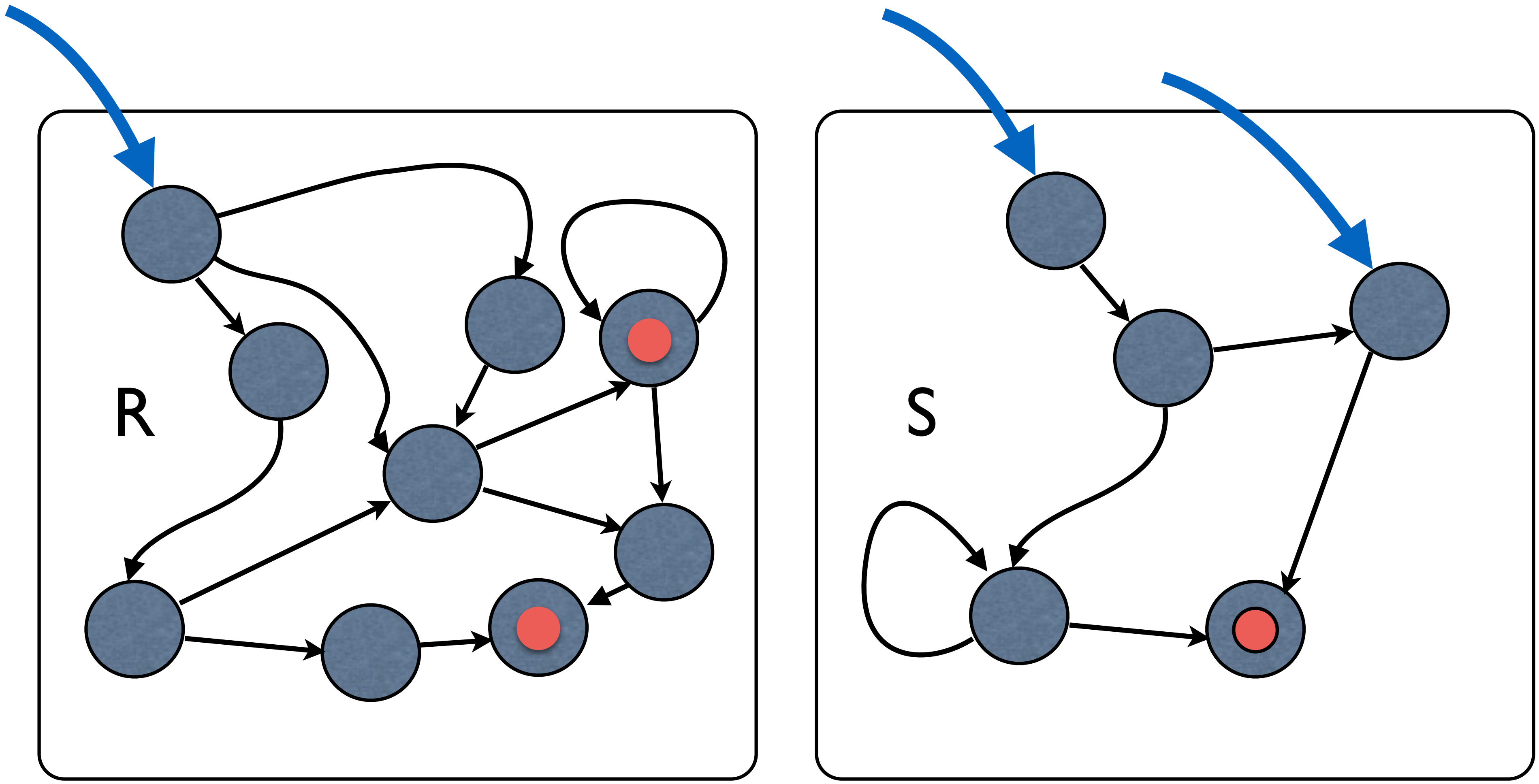


	0	**1**	**ε**
0	0	1	
1	2		0
2			

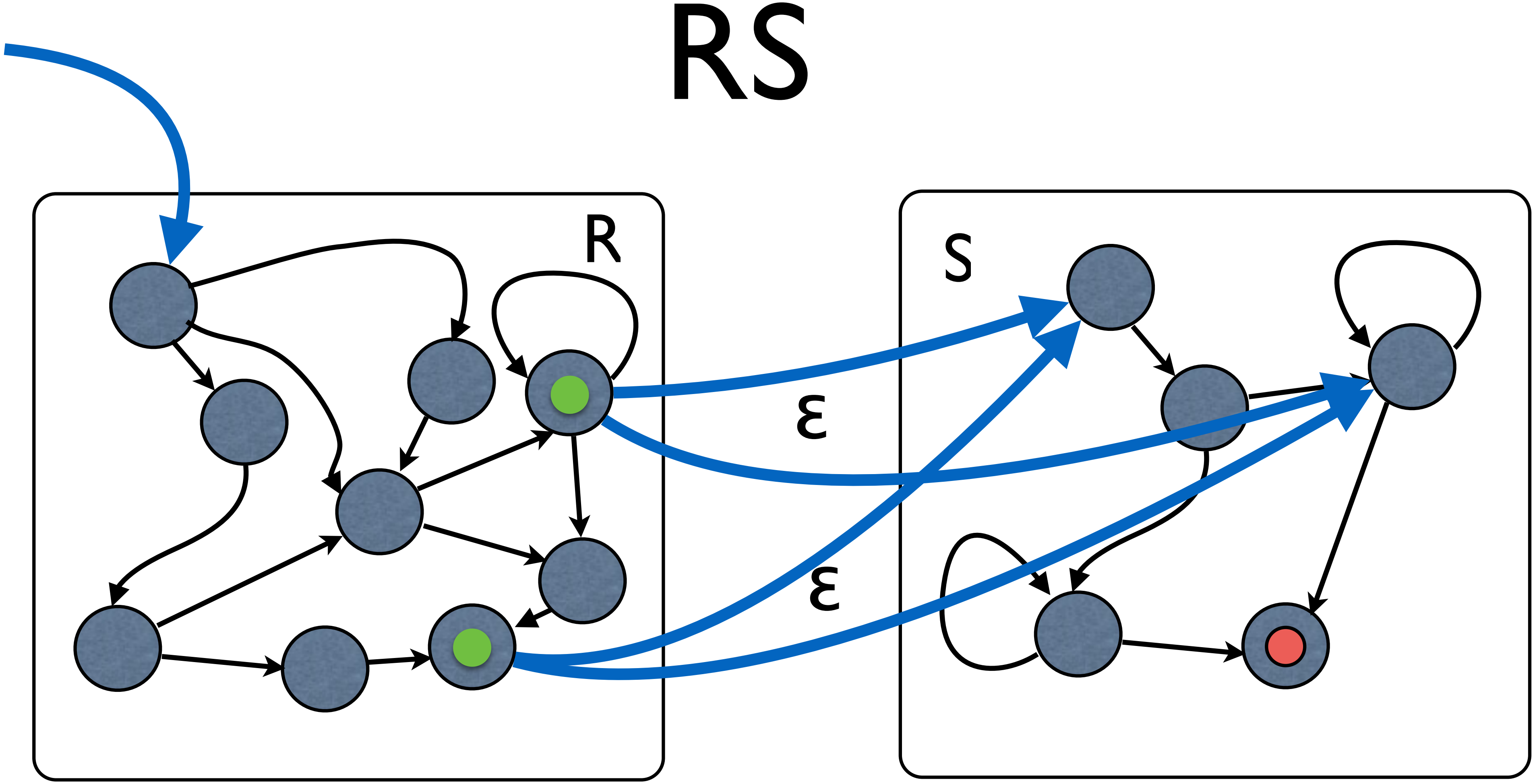


	0ε*	**1ε***
0	0	0,1
1	2	
2		
0,1	0,2	0,1
0,2	0	0,1

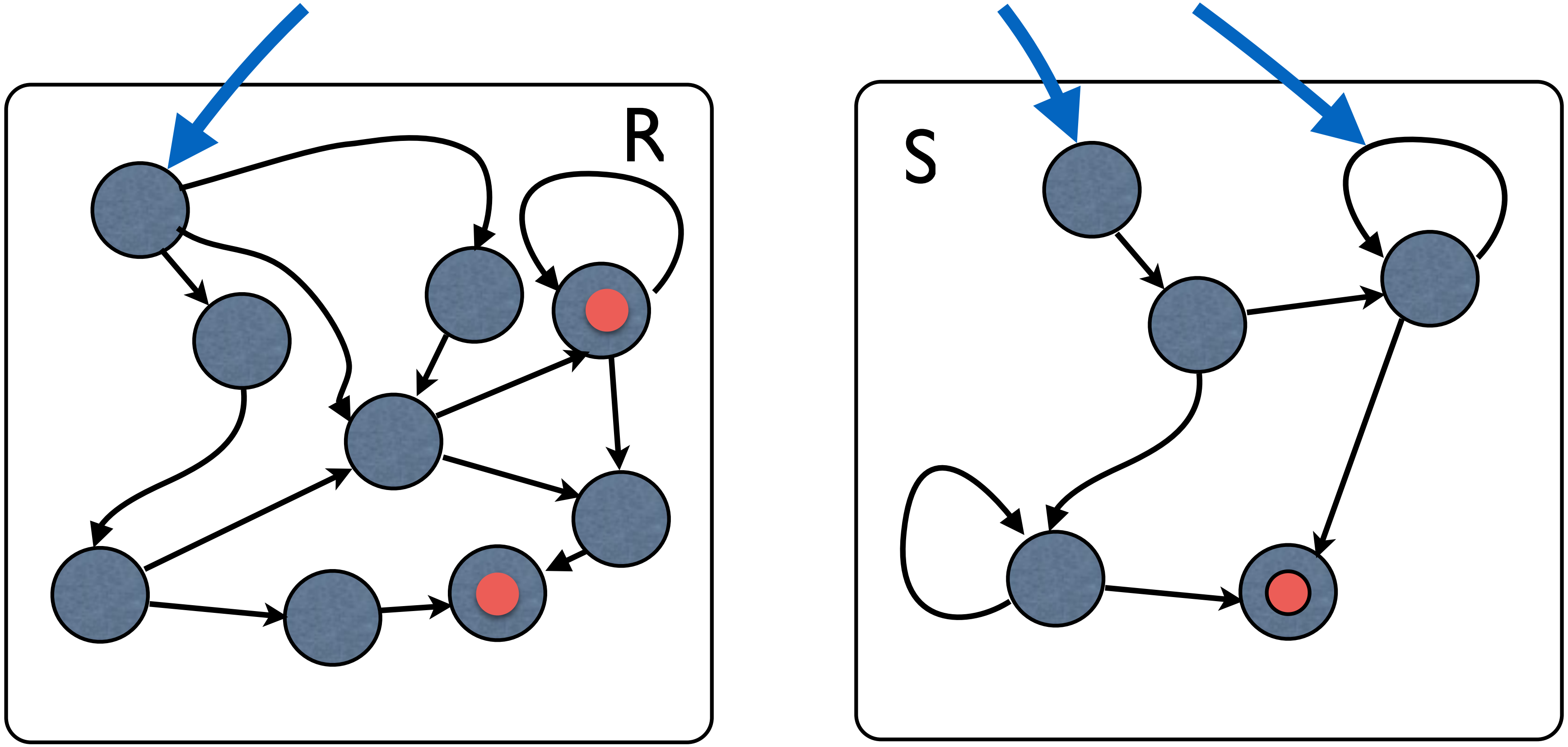
NFA any number of start states and accepting states



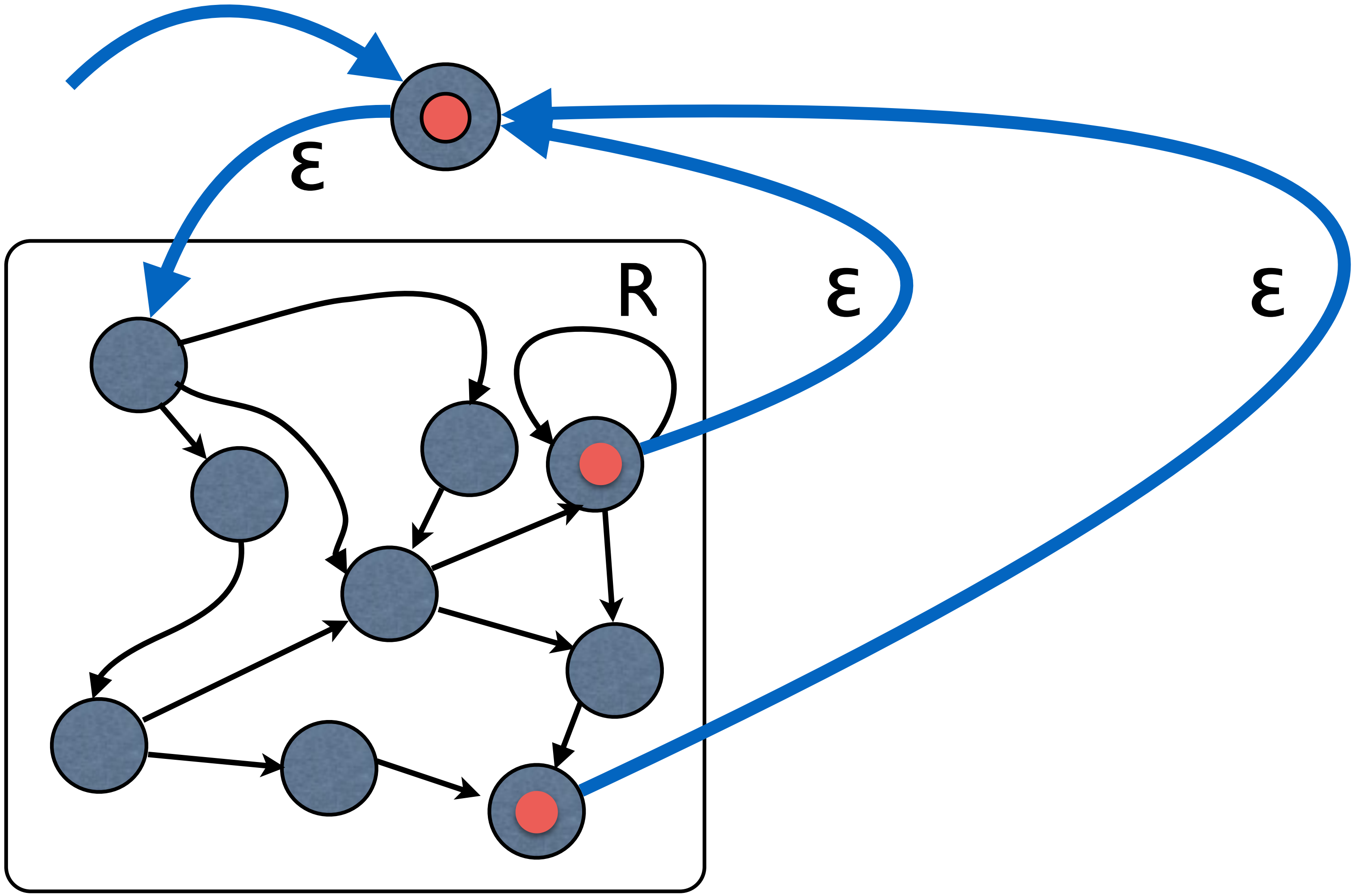
sequence
RS



alternation R|S



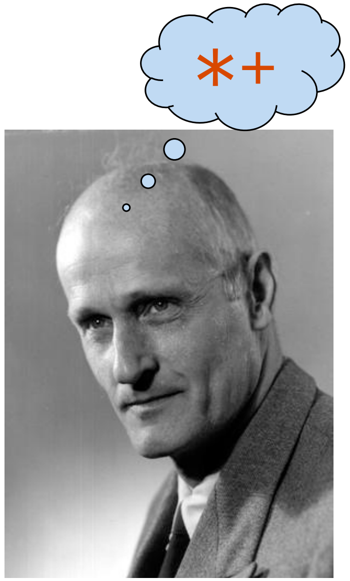
iteration R*



regular expressions

- any character is a regexp
  - matches itself
- if R and S are regexps, so is RS
  - matches
    a match for R followed by a match for S
- if R and S are regexps, so is RIS
  - matches
    any match for R or S (or both)
- if R is a regexp, so is R*
  - matches
    any sequence of 0 or more matches for R
- The algebra of regular expressions also includes elements ∅ and ε
  - ∅ matches nothing; ε matches the empty string

Kleene *, +



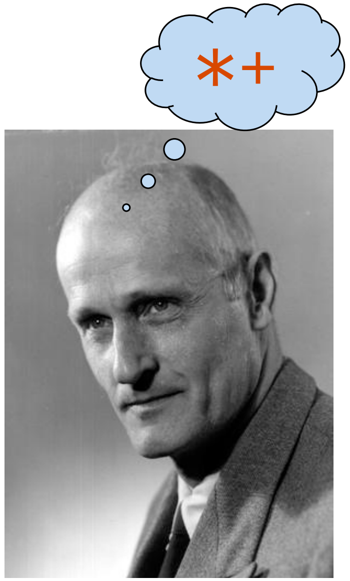
Stephen Cole Kleene

1909-1994

regular expressions denote
regular sets

- any character a is a regexp
  - {<a>}
- if R and S are regexs, so is RS
  - $\{ r\,s \mid r \in R$ and $s \in S \}$
- if R and S are regexps, so is R|S
  - $R \cup S$
- if R is a regexp, so is R*
  - $\{ r^n \mid n \in N$ and $r \in R$
- $\varnothing$ $\varnothing \mid S = S = S \mid \varnothing$
  - $\varnothing$ empty set
- ε $\varepsilon\, S = S = S\, \varepsilon$
  - {<>} singleton empty sequence:

Kleene *, +



Stephen Cole Kleene

1909-1994

https://en.wikipedia.org/wiki/Kleene_algebra

Regular Expressions

- using REs to find patterns
- implementing REs using finite state automata

REs and FSAs

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata
- Finite-state automata are a way of implementing regular expressions
- Regular expressions denote regular sets of strings - each regular set is recognised by some FSA

Regular expressions

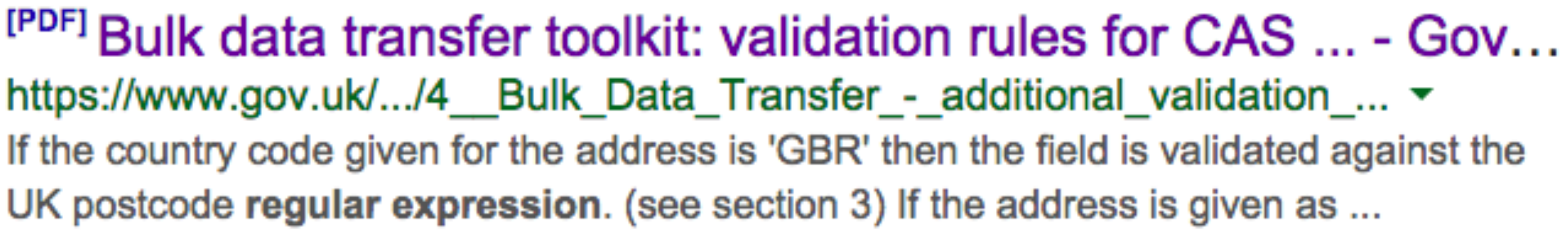
- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions for Textual Searches

Who does it?

Everybody:

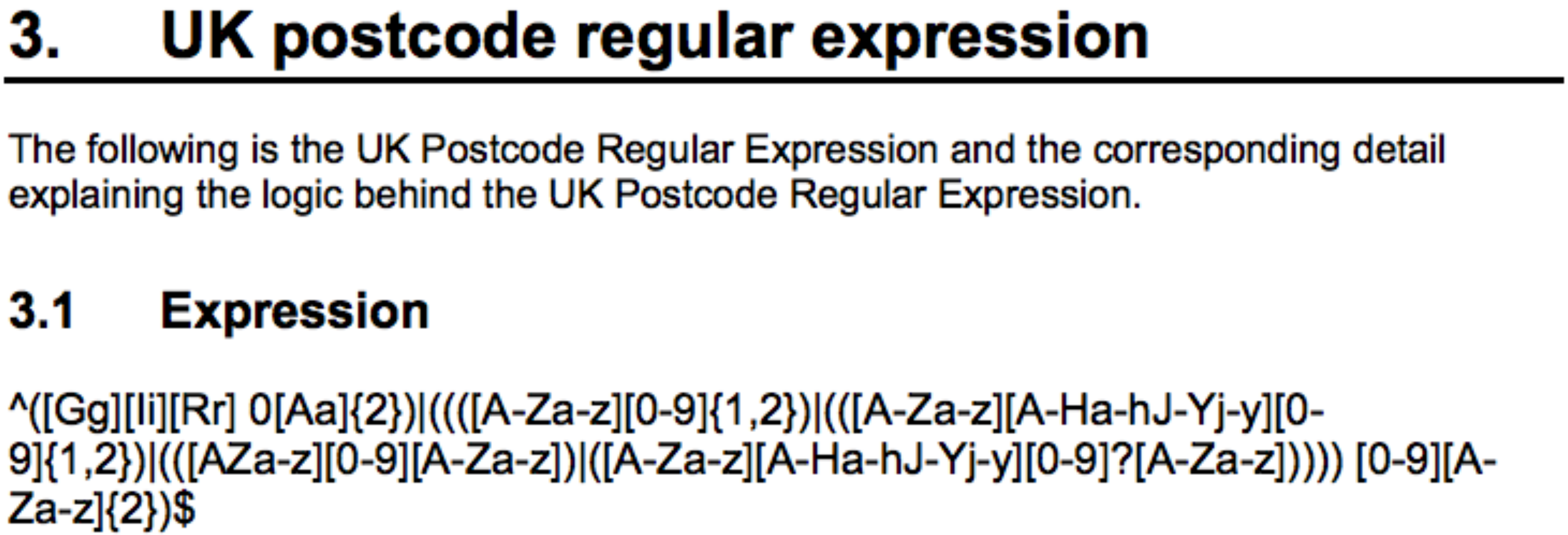
- Web search engines, CGI scripts
- Information retrieval
- Word processing (Emacs, vi, MSWord)
- Linux tools (sed, awk, grep)
- Computation of frequencies from corpora
- Perl



[PDF] Bulk data transfer toolkit: validation rules for CAS ... - Gov…

https://www.gov.uk/.../4__Bulk_Data_Transfer_-_additional_validation_... ▾

If the country code given for the address is 'GBR' then the field is validated against the UK postcode **regular expression**. (see section 3) If the address is given as ...

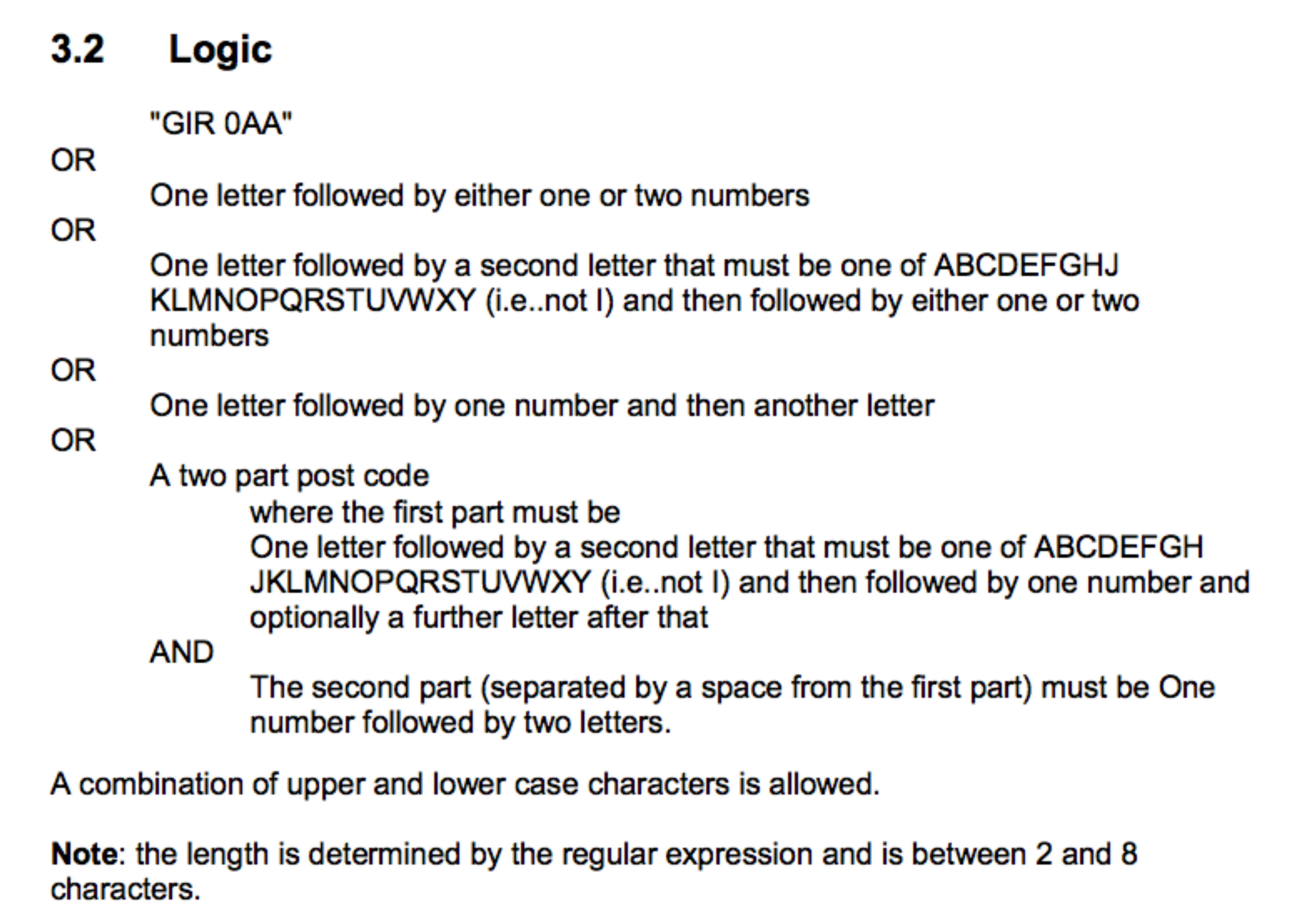


3. UK postcode regular expression

The following is the UK Postcode Regular Expression and the corresponding detail explaining the logic behind the UK Postcode Regular Expression.

3.1 Expression

```
^([Gg][Ii][Rr] 0[Aa]{2})|((([A-Za-z][0-9]{1,2})|(([A-Za-z][A-Ha-hJ-Yj-y][0-9]{1,2})|(([AZa-z][0-9][A-Za-z])|([A-Za-z][A-Ha-hJ-Yj-y][0-9]?[A-Za-z]))))) [0-9][A-Za-z]{2})$
```



3.2 Logic

"GIR 0AA"

OR

One letter followed by either one or two numbers

OR

One letter followed by a second letter that must be one of ABCDEFGHJ KLMNOPQRSTUVWXY (i.e..not I) and then followed by either one or two numbers

OR

One letter followed by one number and then another letter

OR

A two part post code

where the first part must be
One letter followed by a second letter that must be one of ABCDEFGH JKLMNOPQRSTUVWXY (i.e..not I) and then followed by one number and optionally a further letter after that

AND

The second part (separated by a space from the first part) must be One number followed by two letters.

A combination of upper and lower case characters is allowed.

Note: the length is determined by the regular expression and is between 2 and 8 characters.

http://xkcd.com/

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Regular Expression

- **Regular expression:** formula in algebraic notation for specifying a set of strings
- **String:** any sequence of alphanumeric characters
  - letters, numbers, spaces, tabs, punctuation marks
- **Regular expression search**
  - **pattern:** specifying the set of strings we want to search for
  - **corpus:** the texts we want to search through

Basic Regular Expression Patterns

- Case sensitive: d is not the same as D
- Disjunctions: `[dD]` `[0123456789]`
- Ranges: `[0-9]` `[A-Z]`
- Negations: `[^Ss]` *(only when ^ occurs immediately after [ )*
- Optional characters: `?` and `*`
- Wild : `.`
- Anchors: `^` and `$`, also **\b** and **\B**
- Disjunction, grouping, and precedence: **|** **(pipe)**

Caret for negation, ^ , or anchor

RE	Match (single characters)	Example Patterns Matched
`[^A-Z]`	not an uppercase letter	“Oyfn pripetchik”
`[^Ss]`	neither ‘S’ nor ‘s’	“I have no exquisite reason for’t”
`[^\.]`	not a period	“our resident Djinn”
`[e/]`	either ‘e’ or ‘^’	“look up ^ now”
`a^b`	the pattern ‘`a^b`’	“look up a^b now”
`^T`	T at the beginning of a line	“The Dow Jones closed up one”

Optionality and Counters

RE	Match	Example Patterns Matched
`woodchucks?`	woodchuck or woodchucks	“The woodchuck hid”
`colou?r`	color or colour	“comes in three colours”
`(he){3}`	exactly 3 “he”s	“and he said hehehe.”

? zero or one occurrences of previous char or expression

* zero or more occurrences of previous char or expression

+ one or more occurrences of previous char or expression

{n} exactly n occurrences of previous char or expression

{n, m} between n to m occurrences

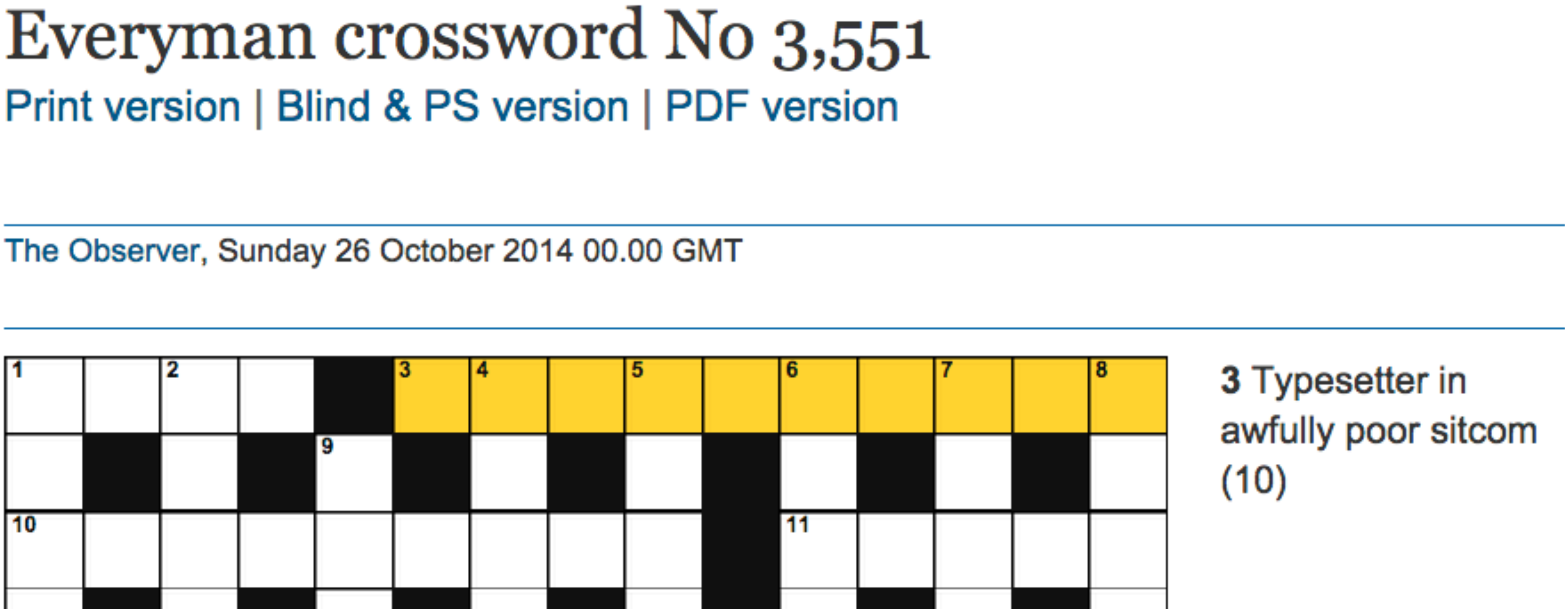
{n, } at least n occurrences

Wild card ‘ .’

RE	Match	Example Patterns Matched
`beg.n`	any char between *beg* and *n*	begin, beg’n, begun
`big.*dog`	find lines where big and dog occur	the big dog bit the little the big black dog bit the

.	any character (but newline)
*	previous character or group, repeated 0 or more time
+	previous character or group, repeated 1 or more time
?	previous character or group, repeated 0 or 1 time
^	start of line
$	end of line
[...]	any character between brackets
[^..]	any character not in the brackets
[a-z]	any character between a and z
\	prevents interpretation of following special char
\\|	or
\w	word constituent
\b	word boundary

\{3\}	previous character or group, repeated 3 times
\{3,\}	previous character or group, repeated 3 or more times
\{3,6\}	previous character or group, repeated 3 to 6 times



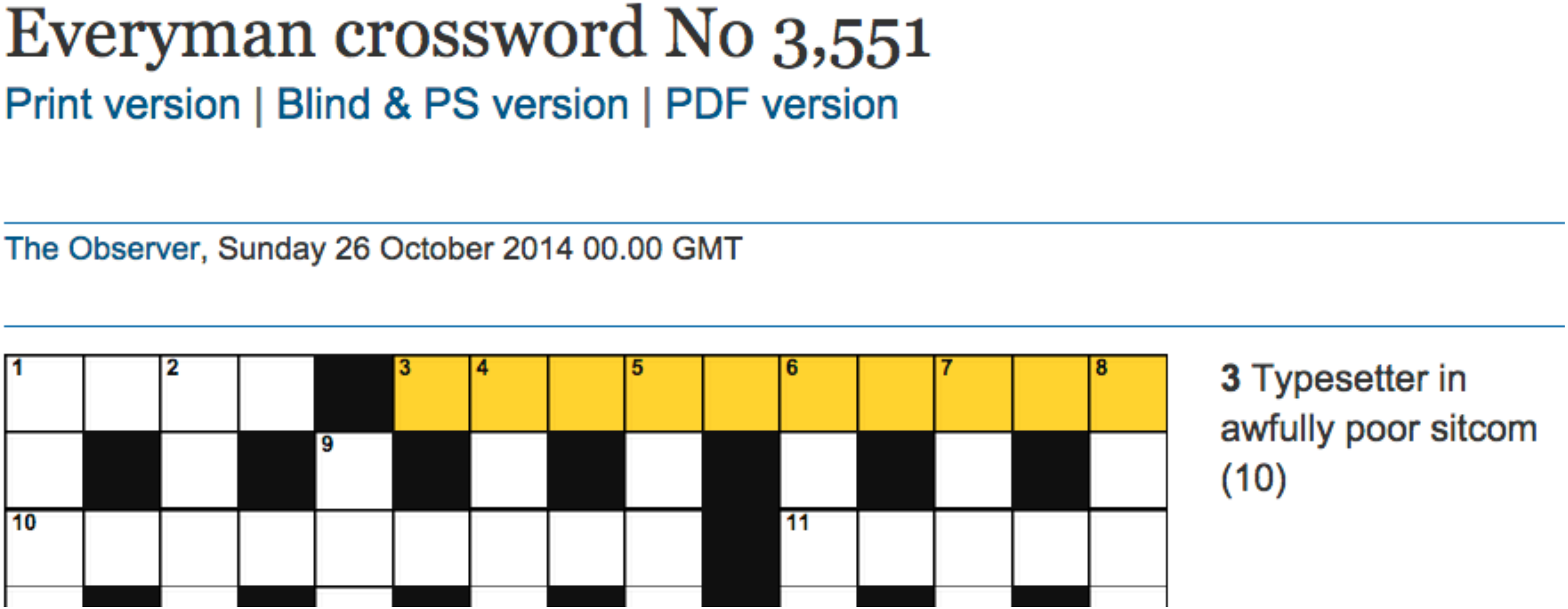
Everyman crossword No 3,551

Print version | Blind & PS version | PDF version

The Observer, Sunday 26 October 2014 00.00 GMT

1		2		■	3	4		5		6		7		8
	■		■	9	■		■		■		■		■	
10									■	11				
	■		■		■		■		■		■		■	

3 Typesetter in awfully poor sitcom (10)



Everyman crossword No 3,551

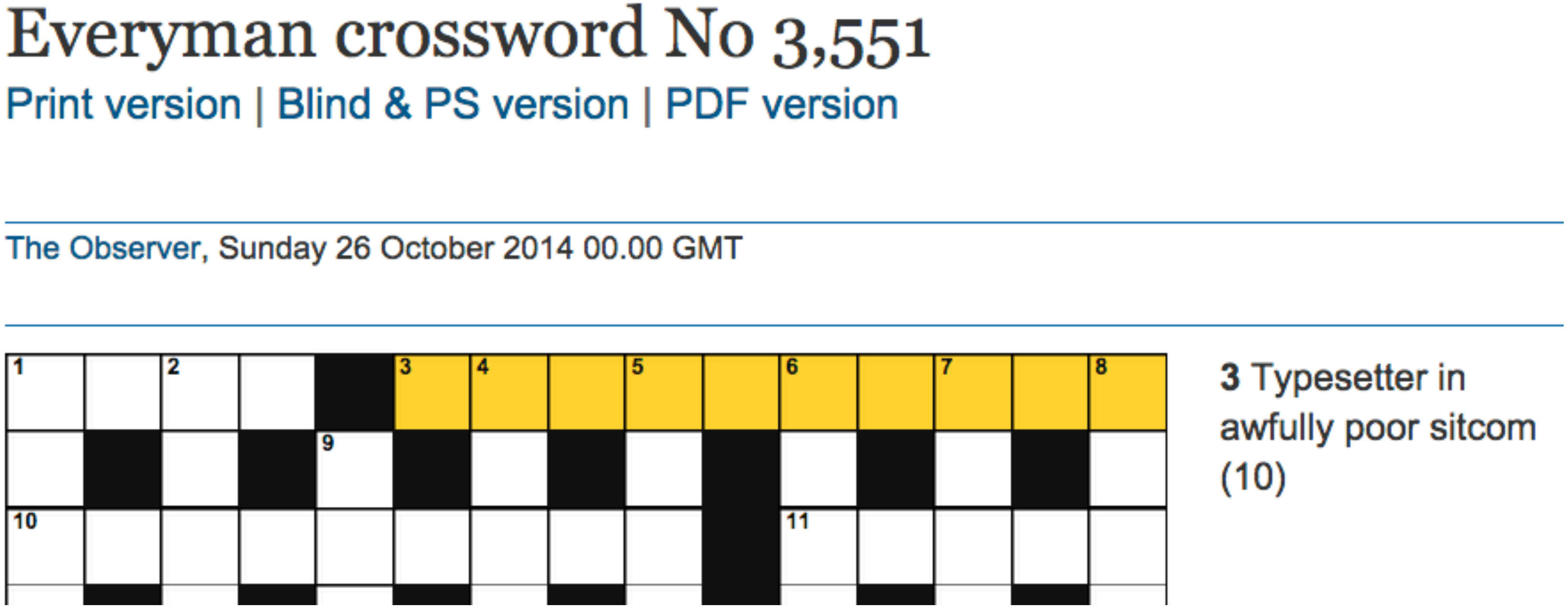
Print version | Blind & PS version | PDF version

The Observer, Sunday 26 October 2014 00.00 GMT

3 Typesetter in awfully poor sitcom (10)

```
% cat /usr/share/dict/words| egrep ^[poorsitcom]{10}$
```

```
$ cat /usr/share/dict/words| egrep ^[poorsitcom]{10}$
compositor
copromisor
crisscross
isoosmosis
isotropism
microtomic
optimistic
poroscopic
postcosmic
postscript
prioristic
promitosis
proproctor
protoprism
tricrotism
troostitic
```



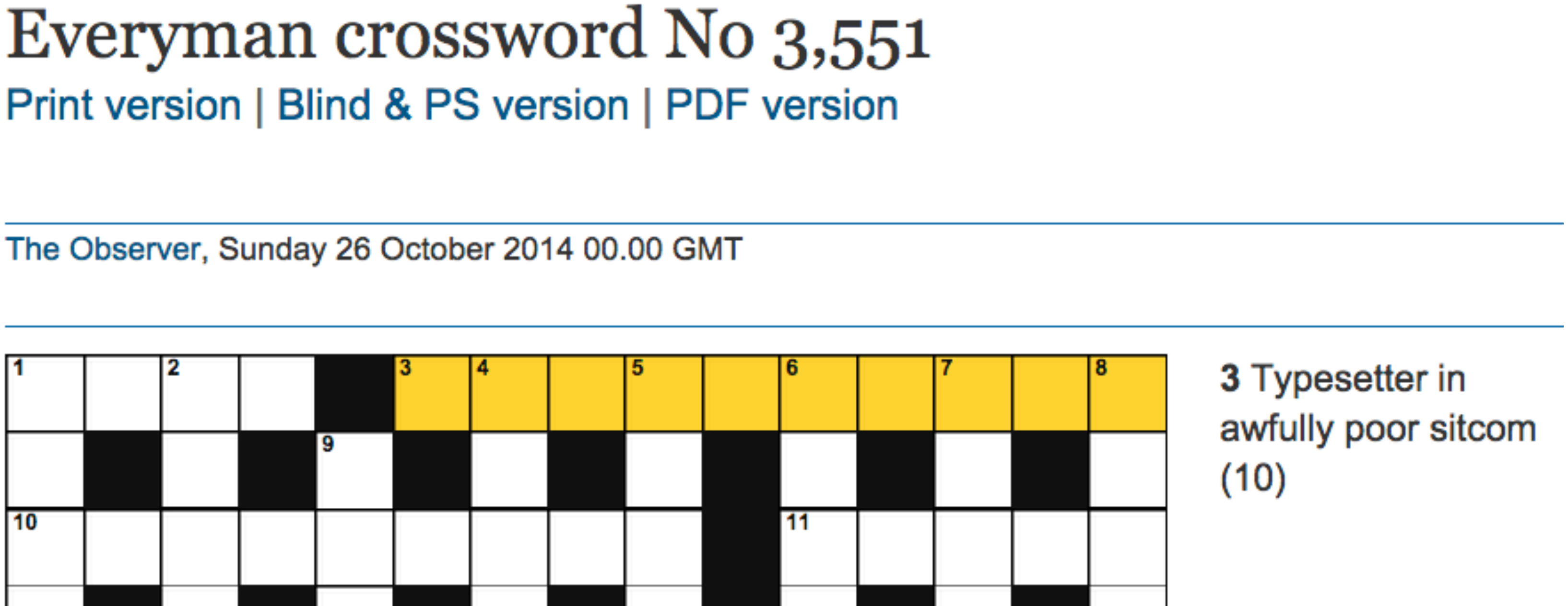
Everyman crossword No 3,551

Print version | Blind & PS version | PDF version

The Observer, Sunday 26 October 2014 00.00 GMT

3 Typesetter in awfully poor sitcom (10)

```
% cat /usr/share/dict/words| egrep ^[poorsitcom]{10}$ | grep o.*o.*o
compositor
copromisor
isoosmosis
poroscopic
proproctor
```



Everyman crossword No 3,551

Print version | Blind & PS version | PDF version

The Observer, Sunday 26 October 2014 00.00 GMT

3 Typesetter in awfully poor sitcom (10)

Regular Expressions

- Basic regular expression patterns
- Java-based syntax

- **Disjunctions** `[mM]`

Reg Exp	Match	Example Patterns
`[mM]other`	mother or Mother	"Mother"
`[abc]`	a or b or c	"you are"
`[1234567890]`	any digit	"3 times a day"

Regular Expressions

- **Ranges** `[A-Z]`

RE	Match	Examples Patterns Matched
[A-Z]	an uppercase letter	“call me Eliza”
[a-z]	a lowercase letter	“call me Eliza”
[0-9]	a single digit	“I’m off at 7”

- **Negations** `[^Ss]`

RE	Match	Examples Patterns Matched
[^A-Z]	not an uppercase letter	“You can call me Eliza”
[^Ss]	neither s nor S	“Say hello Eliza”
[^\.]	not a period	“Hello.”

Regular Expressions

- **Optional characters: ? ,* and +**

- ? (0 or 1)

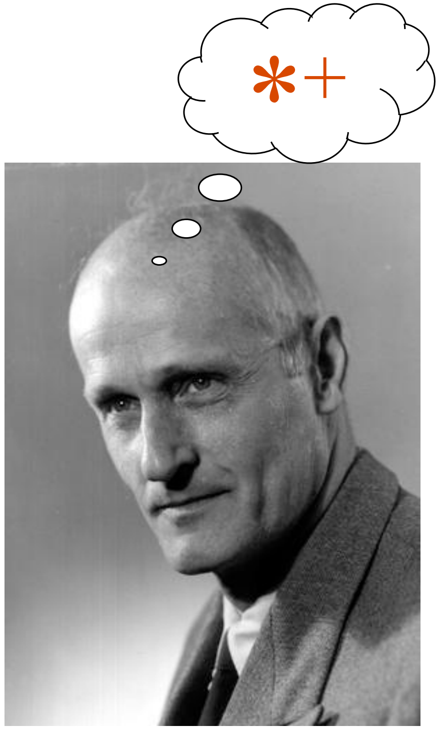
  colou?r ➔ color *or* colour

- * (0 or more)

  oo*h! ➔ oh! *or* ooh! *or* ooooh!

- + (1 or more)

  o+h! ➔ oh! *or* ooh! *or* ooooh!



Stephen Cole Kleene

- . any char except newline

  beg.n ➔ begin *or* began *or* begun

Regular Expressions

- **Anchors ^ and $**
  - `^[A-Z]` → "France", "Paris"
  - `^[^A-Z]` → "¿verdad?", "really?"
  - `\.$` → "It's over."
  - `moo$` → "moo", but not "mood"
- **Boundaries \b and \B**
  - `\bon\b` → "on my way" "Monday"
  - `\Bon\b` → "automaton"
- **Disjunction |**
  - `yours|mine` → "it's either yours or mine"

Regular Expressions

http://www.inf.ed.ac.uk/teaching/courses/il1/2010/labs/2010-10-28/regexrepl.xml

- **Replacement**
  - in emacs
  - in javascript
  - in python and perl
  - …

```
s/\bI(’m| am)\b /ARE YOU/g
```

- Syntax varies - the ideas are universal

Experiment

http://www.inf.ed.ac.uk/teaching/courses/il1/2010/labs/2010-10-28/regexrepl.xml

- **Replacement**
  - in emacs
  - in javascript
  - in python and perl
  - …

```
s/\bI('m| am)\b /ARE YOU/g
```

- Syntax varies - the ideas are universal