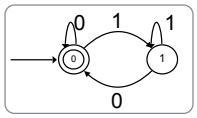
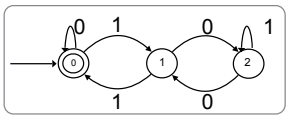
NFA and regex

CI

- the Boolean algebra of languages
- non-deterministic machines
- regular expressions

The intersection of two regular languages is regular



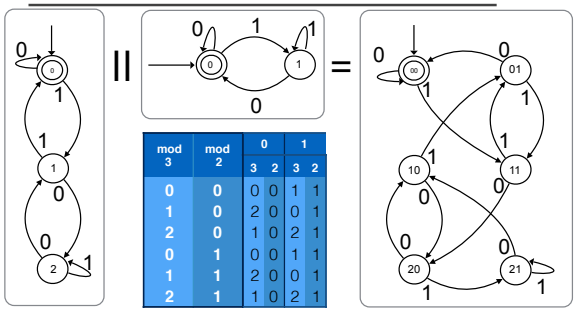


Run both machines in parallel?

Build one machine that simulates two machines running in parallel!

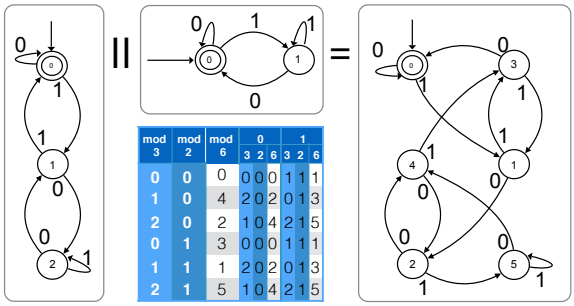
Keep track of the state of each machine.

The intersection of two regular languages is regular



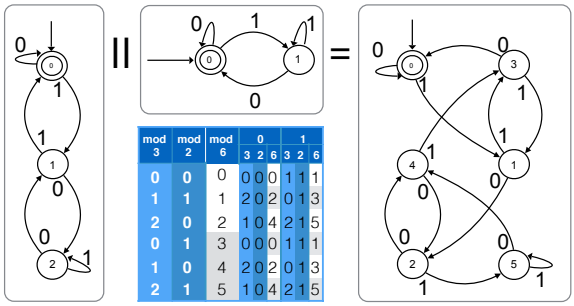
mod 3	mod 2	0		1	
		3	2	3	2
0	0	0	0	1	1
1	0	2	0	0	1
2	0	1	0	2	1
0	1	0	0	1	1
1	1	2	0	0	1
2	1	1	0	2	1

The intersection of two regular languages is regular



mod 3	mod 2	mod 6	0			1		
			3	2	6	3	2	6
0	0	0	0	0	0	1	1	1
1	0	4	2	0	2	0	1	3
2	0	2	1	0	4	2	1	5
0	1	3	0	0	0	1	1	1
1	1	1	2	0	2	0	1	3
2	1	5	1	0	4	2	1	5

The intersection of two regular languages is regular



mod 3	mod 2	mod 6	0			1		
			3	2	6	3	2	6
0	0	0	0	0	0	1	1	1
1	1	1	2	0	2	0	1	3
2	0	2	1	0	4	2	1	5
0	1	3	0	0	0	1	1	1
1	0	4	2	0	2	0	1	3
2	1	5	1	0	4	2	1	5

The regular languages A ⊆ Σ* form a Boolean Algebra

- Since they are closed under intersection and complement.

Determinism

Can always convert to an equivalent DFA for which every state has exactly one transition leaving the state for each input symbol.

For this machine there is exactly one trace for each input string

- Proof

  Add a new “black hole” state, ●

  For every pair (s, a) for which there is no state t with a transition T(s, a, t), add a transition T(s, a, ●).

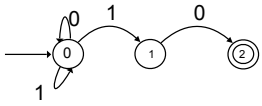
  This includes a transition T(●, a, ●) for each a ∈ Σ . You cannot escape from the black hole.

  The black hole ● is not an accepting state.

This machine accepts the same language as the original.

Non Determinism

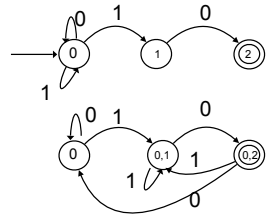
In a non-deterministic machine (NFA), each state may have any number of transitions with the same input symbol, leaving to different successor states.



	0	1
0	0	0,1
1	2	
2		

Non Determinism

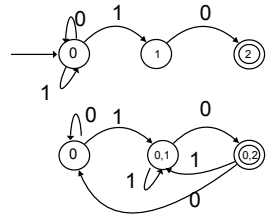
In a non-deterministic machine (NFA), each state may have any number of transitions with the same input symbol, leaving to different successor states.



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

Non Determinism

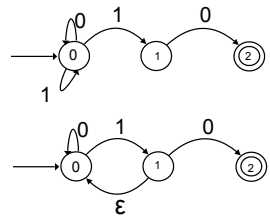
We can simulate a non-deterministic machine using a deterministic machine – by keeping track of the set of states the NFA could possibly be in.



	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.



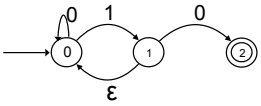
	0	1	ε
0	0	1	
1	2		0
2			

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.

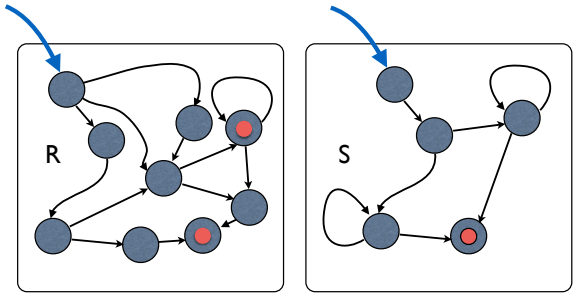


	0	1	ε
0	0	1	
1	2		0
2			

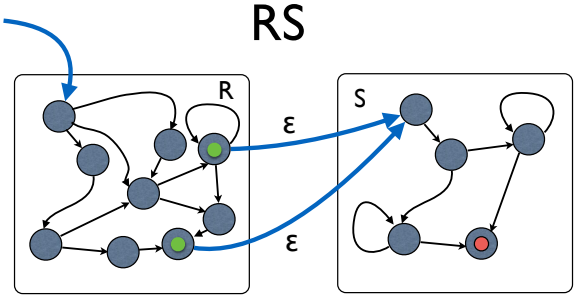
	0ε*	1ε*
0	0	1,0
1	2	
2		

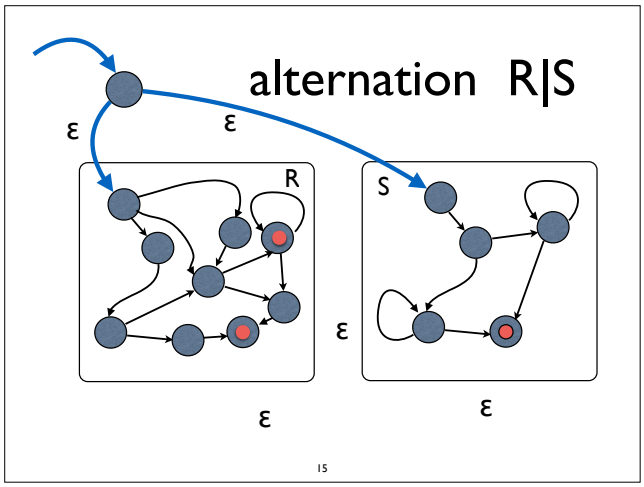
NFA single start state

any number of accepting states

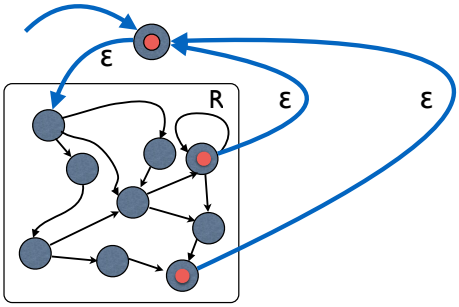


sequence
RS



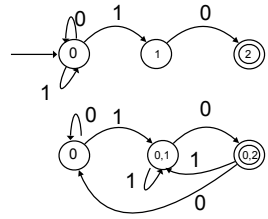

alternation R|S
ε
ε
R
S
ε
ε
ε

iteration R*



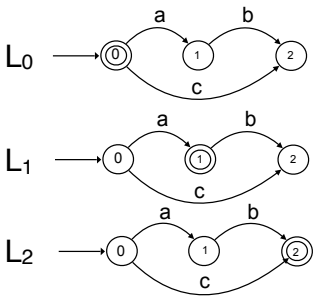
Demo

http://ivanzuzak.info/noam/webapps/fsm_simulator/



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

Is there a regular expression for every FSM?



Let L_i be the language accepted if **i** is the accepting state

$L_0 = \varepsilon$
$L_1 = L_0\, a$
$L_2 = L_1\, b \mid L_0\, c$

$L_2 = L_0\, a\, b \mid \varepsilon\, c$
$L_2 = \varepsilon\, a\, b \mid \varepsilon\, c$
$L_2 = a\, b \mid c$

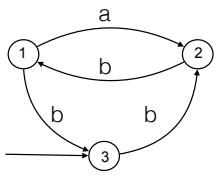
Is there a regular expression for every FSM?

$L_1 = L_2\,b$

$L_2 = L_3\,b \mid L_1\,a$

$L_3 = \varepsilon \mid L_1\,b$

$\quad = \varepsilon \mid L_2\,b\,b$



$L_2 = (\varepsilon \mid L_2\,b\,b)\,b \mid L_2\,b\,a$

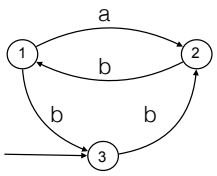
$\quad = b \mid L_2\,b\,b\,b \mid L_2\,b\,a$

$\quad = b \mid L_2\,(b\,b\,b \mid b\,a)$

Is there a regular expression for every FSM?

$L_2 = b \mid L_2\,(b\,b\,b \mid b\,a)$

$L_2 = b\,(b\,b\,b \mid b\,a)^*$



$L_1 = L_2\,b = b\,(b\,b\,b \mid b\,a)^*\,b$

$L_3 = \varepsilon \mid L_2\,b\,b = \varepsilon \mid b\,(b\,b\,b \mid b\,a)^*\,bb$

Arden's Lemma

If R and S are regular expressions then the equation

$$X = R \mid X\,S$$

has a solution $X = R\,S^*$

If $\varepsilon \notin L(S)$ then this solution is unique.

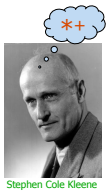
$L_2 = b \mid L_2\,(b\,b\,b \mid b\,a)$

$L_2 = b\,(b\,b\,b \mid b\,a)^*$

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 R|S
  - 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 0 and 1
  - 0 matches nothing; 1 matches the empty string

Kleene *, +



Stephen Cole Kleene

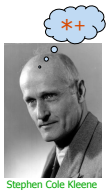
1909-1994

regular expressions denote
regular sets

- any character a is a regexp
  - {<a>}
- if R and S are regexs, so is RS
  - { r s | r ∈ R and s ∈ S }
- if R and S are regexps, so is R|S
  - R ∪ S
- if R is a regexp, so is R*
  - { r^n | n ∈ N and r ∈ R
- 0 0 | S = S = S | 0
  - ∅ empty set
- 1 1 S = S = S 1
  - {<>} singleton empty sequence:

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

Kleene *, +



Stephen Cole Kleene

1909-1994

Regular Expressions

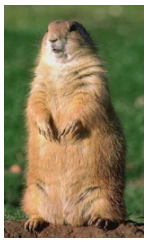
- using REs to find patterns
- implementing REs using finite state automata

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

http://xkcd.com/

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