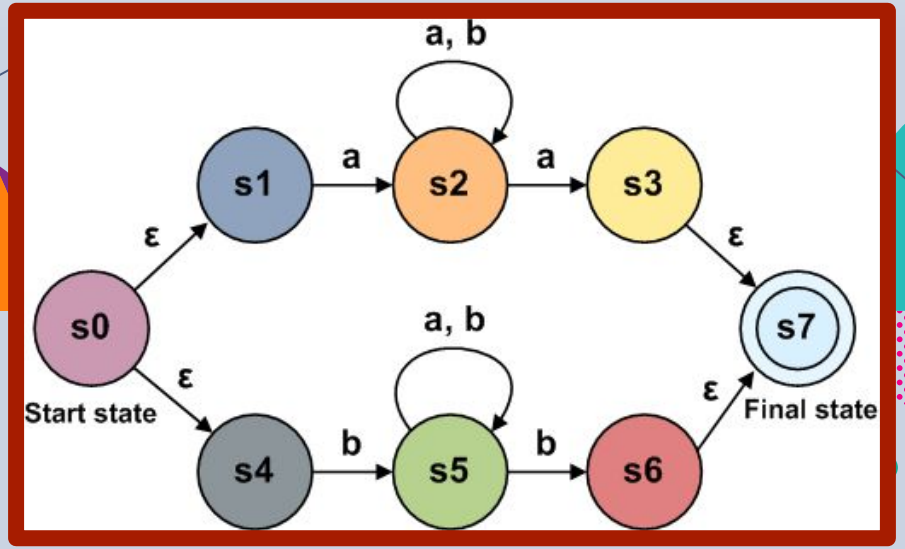
Non-Deterministic Finite Automatons

Another notation

Finite automaton variations



Non-Deterministic Finite Automatons (NFA)

- Can represent the same computation as a DFA
- Any DFA is an NFA
- Any NFA can be converted into a DFA
- The notation is simpler, since it can use less states
- The main difference is the transition function

Non-Deterministic

The transition function δ allows multiple results from a given input.
It is not possible to determine the final state from a given input

NFA

Mathematically described as a 5-tuple:

$$\mathbf{M} = (\mathbf{Q}, \mathbf{\Sigma}, \delta, q_0, \mathbf{F})$$

Where:

Q is the finite set of states

Σ is the finite alphabet

δ is a total function from **Q** x **Σ** to $\mathscr{P}(\mathbf{Q})$, known as the transition function

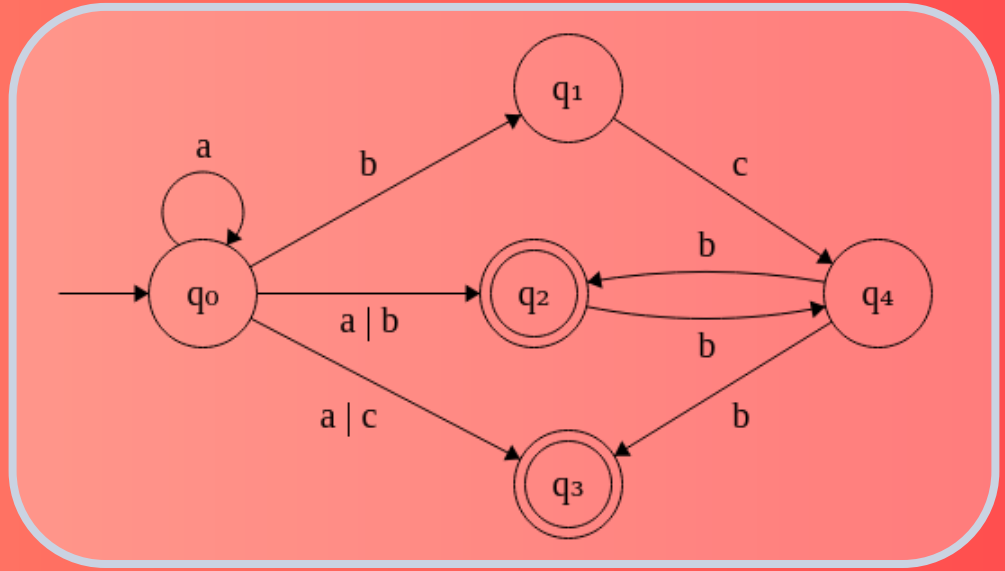
$q_0 \in \mathbf{Q}$ is the initial state

F is the subset of **Q** of accept states

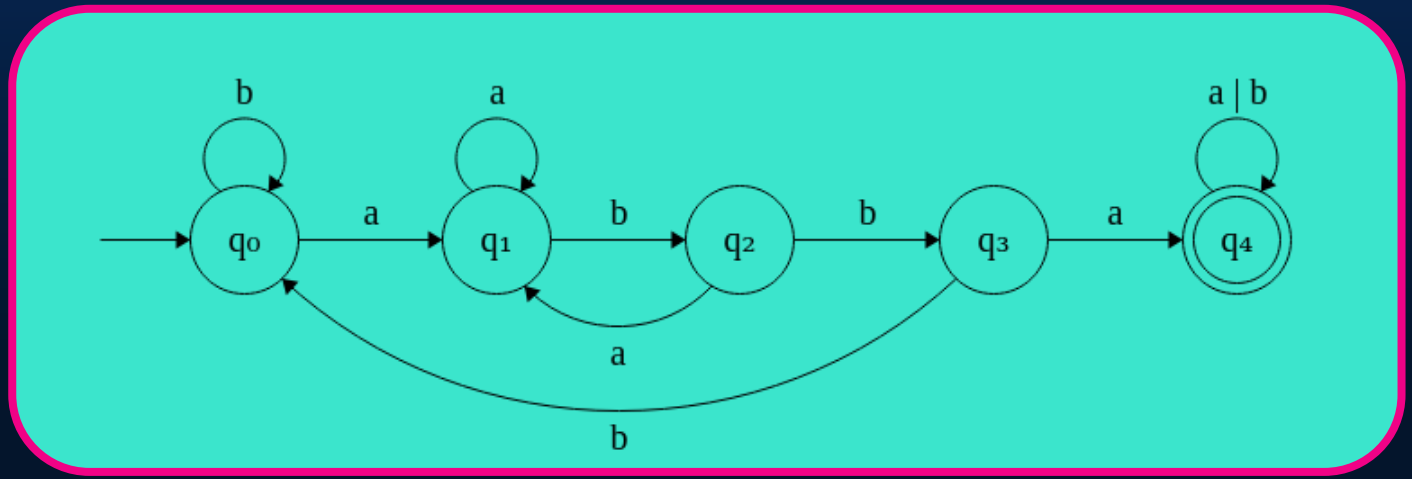
NFA transition function

- The function allows 0, 1 or more transitions from a given state and input
- All possible transitions must be considered
- If at the end of the string, any of the possible paths finishes in an accept state, the string is accepted

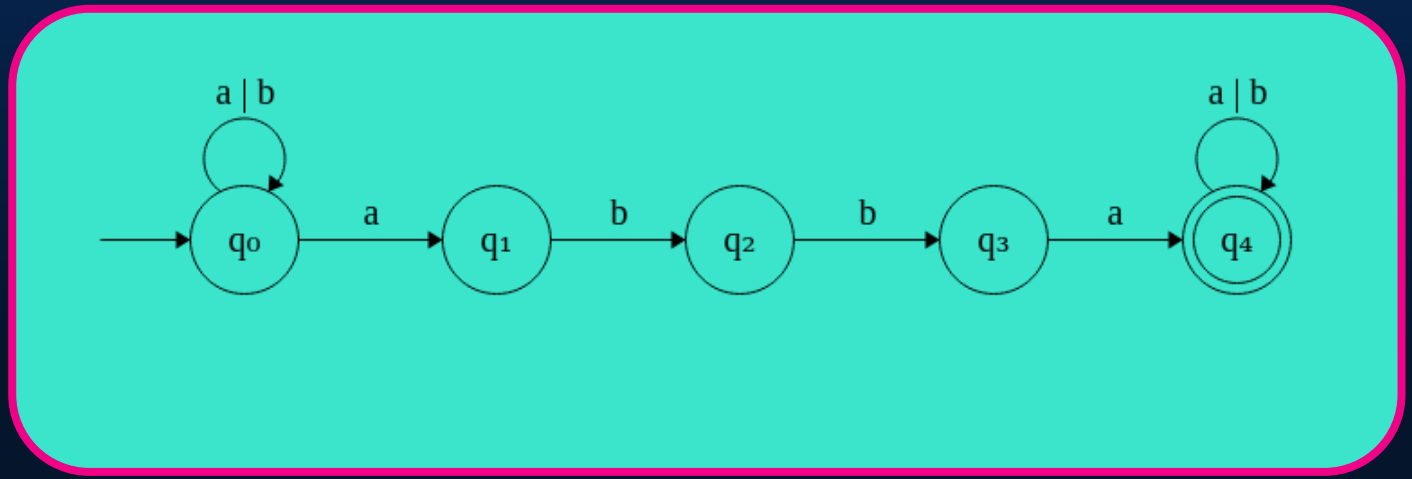
Example of a Non-Deterministic Finite Automaton



DFA to accept strings that contain *abba*

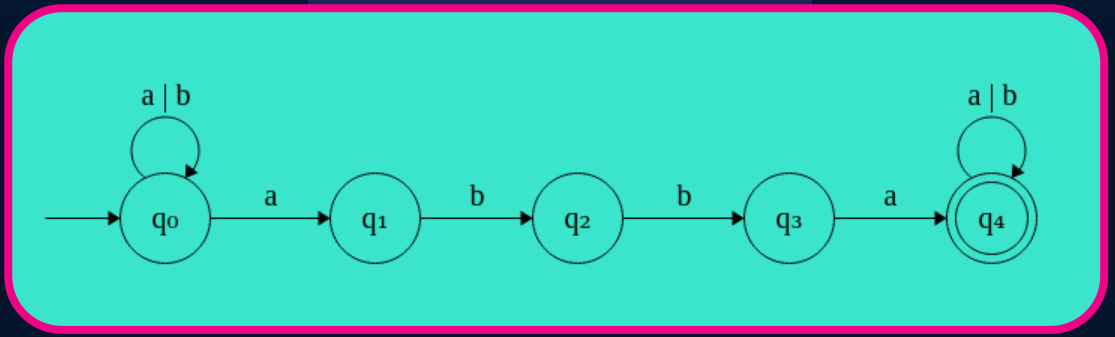


NFA to accept strings that contain *abba*



Building the transition function

Each transition can go to more than one new state



$\delta =$

	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	-	$\{q_2\}$
q_2	-	$\{q_3\}$
q_3	$\{q_4\}$	-
q_4	$\{q_4\}$	$\{q_4\}$

Evaluating the string *aabbaa*

$[q_0, aabbaa]$	$[q_0, aabbaa]$	$[q_0, aabbaa]$	$[q_0, aabbaa]$	$[q_0, aabbaa]$
$[q_0, abbaa]$	$[q_0, abbaa]$	$[q_0, abbaa]$	$[q_0, abbaa]$	$[q_1, abbaa]$
$[q_0, bbaa]$	$[q_0, bbaa]$	$[q_0, bbaa]$	$[q_1, bbaa]$	
$[q_0, baa]$	$[q_0, baa]$	$[q_0, baa]$	$[q_2, baa]$	
$[q_0, aa]$	$[q_0, aa]$	$[q_0, aa]$	$[q_3, aa]$	
$[q_0, a]$	$[q_0, a]$	$[q_1, a]$	$[q_4, a]$	
$[q_0, \lambda]$	$[q_1, \lambda]$		$[q_4, \lambda]$	

Evaluating the string *aabbaa*

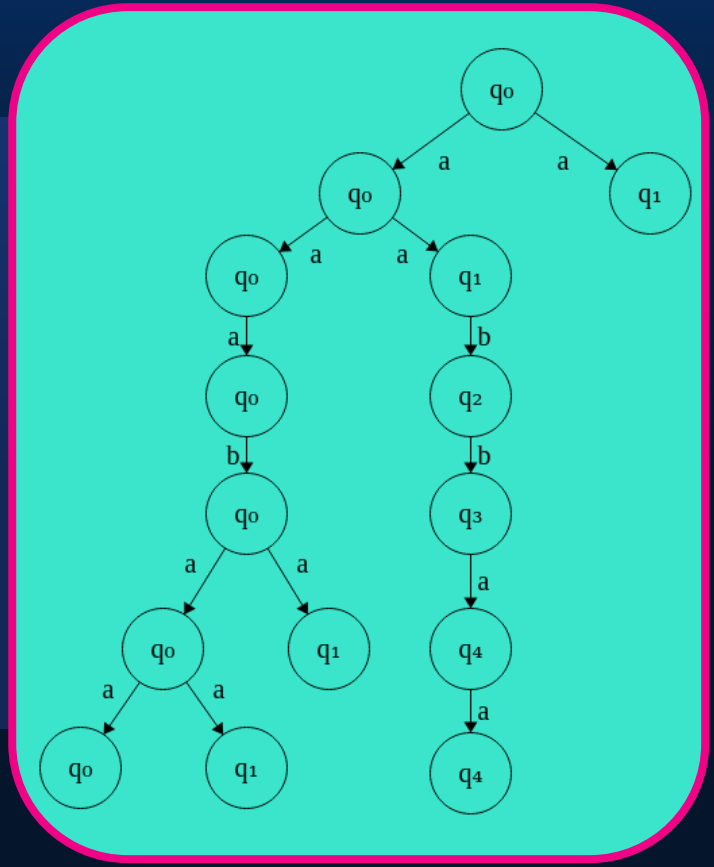
$[q_0, aabbaa]$	$[q_0, aabbaa]$	$[q_0, aabbaa]$	$[q_0, aabbaa]$	$[q_0, aabbaa]$
$[q_0, abbaa]$	$[q_0, abbaa]$	$[q_0, abbaa]$	$[q_0, abbaa]$	$[q_1, abbaa]$
$[q_0, bbaa]$	$[q_0, bbaa]$	$[q_0, bbaa]$	$[q_1, bbaa]$	
$[q_0, baa]$	$[q_0, baa]$	$[q_0, baa]$	$[q_2, baa]$	
$[q_0, aa]$	$[q_0, aa]$	$[q_0, aa]$	$[q_3, aa]$	
$[q_0, a]$	$[q_0, a]$	$[q_1, a]$	$[q_4, a]$	
$[q_0, \lambda]$	$[q_1, \lambda]$		$[q_4, \lambda]$	

How can this be computed?

- Multiple paths to follow at the same time
- Using concurrency

How can this be computed?

- Multiple paths to follow at the same time
- Using concurrency



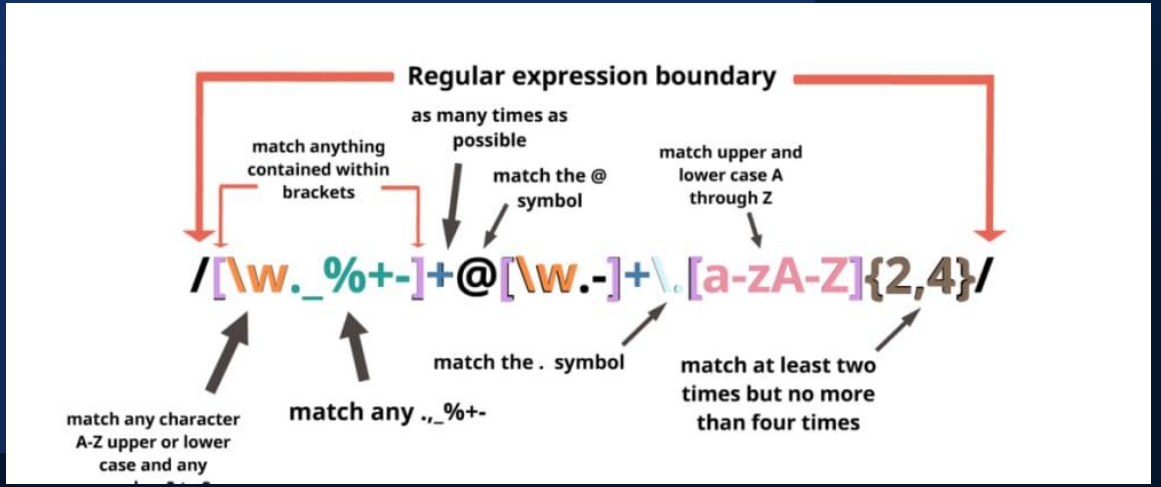
Example of a Non-Deterministic Finite Automaton

Regular Expressions (RE)

Compact syntax to describe sequences of symbols

Basic operations:

- Concatenation: *ab*
- Selection: *a* | *b* or [*a-z*]
- Repetition:
  - *a**
  - *a*+ => *aa**
- Range: [*a-z*]
- Amount: {min, max}



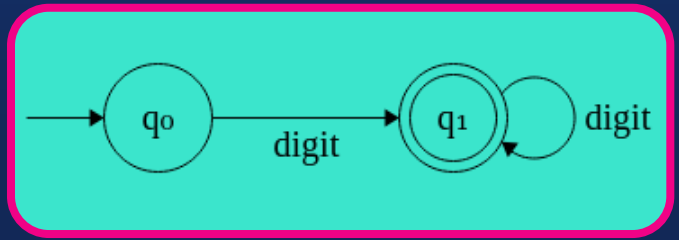
Token recognition

Compilers use FAs or REs to identify valid tokens

Example: identifying integer numbers

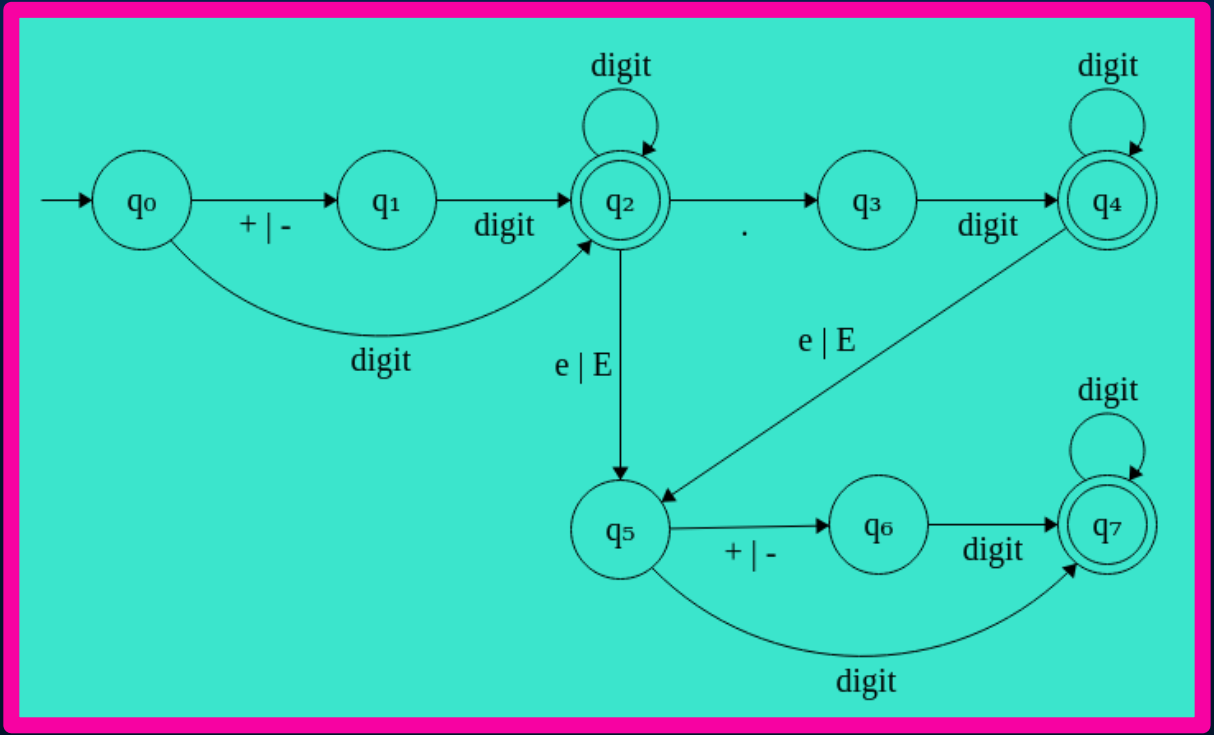
Regular expression: $\{0,1,2,3,4,5,6,7,8,9\}^+$, also represented as $[0\text{-}9]^+$.

Finite Automaton:



The implementation uses a transition table with all the states and inputs in the FA.

FA (incomplete) example: number token



Formal definition

Recognizing a number:

Q = $\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$
Σ = {digit, e, E, +, -, .}
F = $\{q_2, q_4, q_7\}$

δ =

	digit	e | E	- | +	.
q_0	q_2	-	q_1	-
q_1	q_2	-	-	-
q_2	q_2	q_5	-	q_3
q_3	q_4	-	-	-
q_4	q_4	q_5	-	-
q_5	q_7	-	q_6	-
q_6	q_7	-	-	-
q_7	q_7	-	-	-

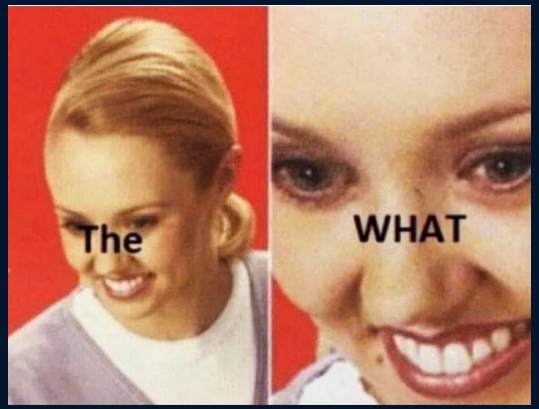
RE example: number token

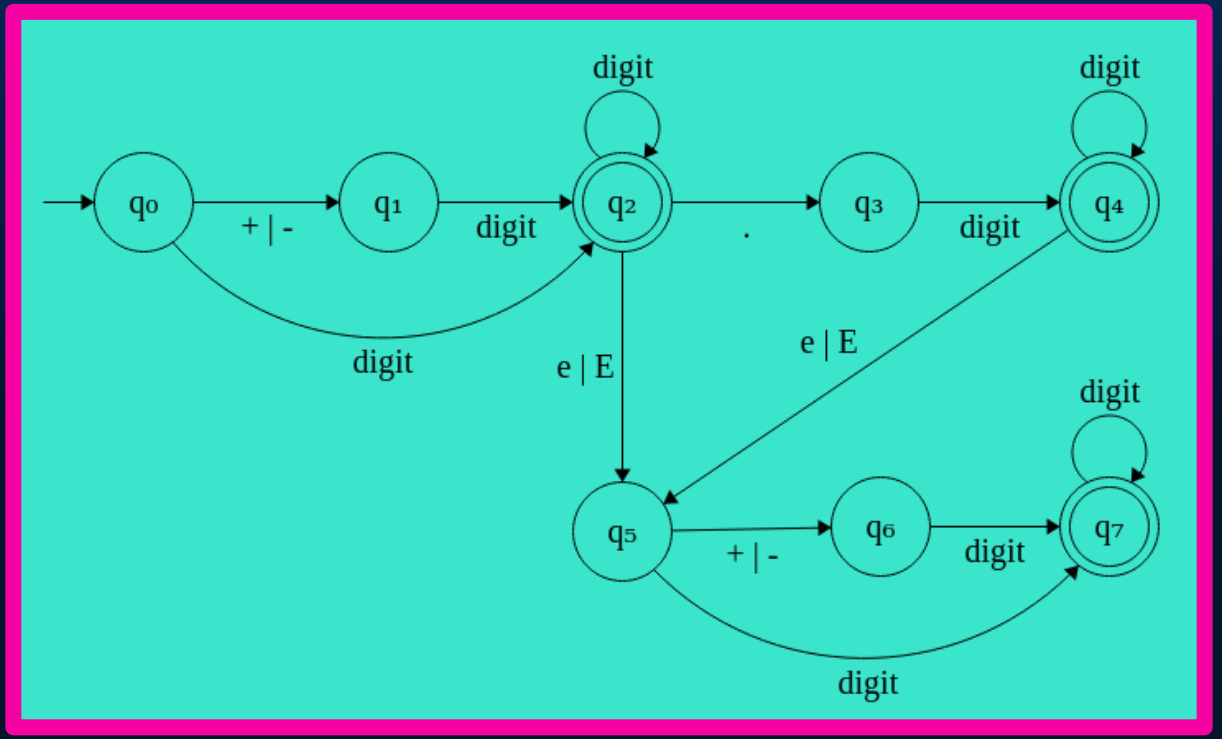
RE:

^[+-]\d+([.]\d+)?([eE][+-]?\d+)?$

Non capturing groups:

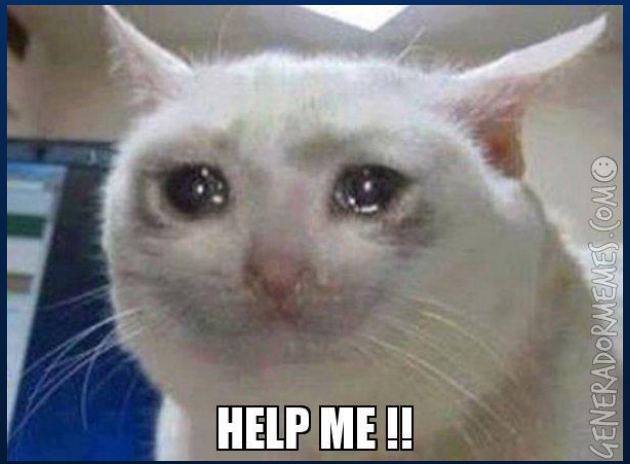
^[+-]\d+(?:[.]\d+)?(?:[eE][+-]?\d+)?$

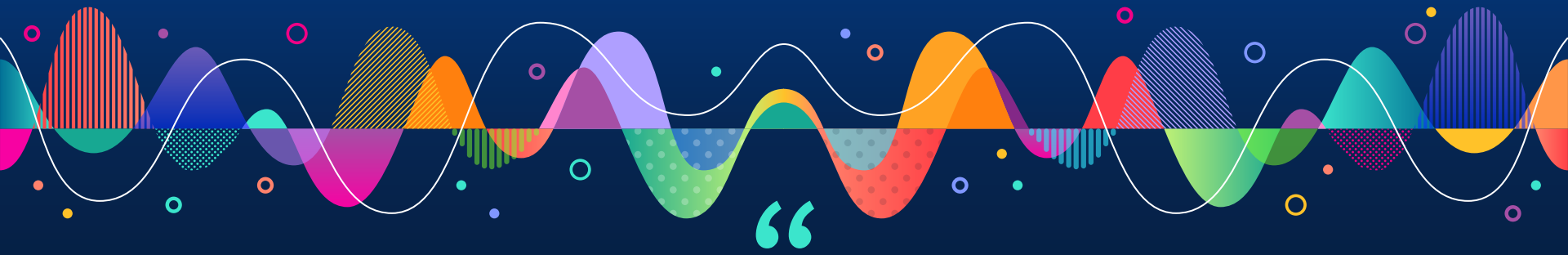




Example

- Create a DFA to identify additions or subtractions of integer numbers
- Create the transition table for the DFA
- Examples of valid expressions:
  - 43 + 22
  - 29.47 – 89.524
  - 67 + 74.213





“

A NFA is easier to describe than a DFA,

but is more complex to validate

Conclusion

THANKS!

Do you have any questions?
Contact me at:
g.echeverria@tec.mx

References:

Nombre: Languages and machines: an introduction to the theory of computer science

Autor: Thomas Sudkamp

Edición: 3rd

Año: 2016

Editorial: Addison-Wesley

ISBN: 9780321322210

References:

https://devopedia.org/chomsky-hierarchy

https://regex101.com/

http://madebyevan.com/fsm/

https://hackernoon.com/lexical-analysis-861b8bfe4cb0

https://dev.to/mconner89/regular-expressions-grouping-and-string-methods-3ijn

DIAGRAMS AND INFOGRAPHICS