Identificador

Son las palabras que cumplen el iniciar con una letra y pueden estar seguidas de muchas letras o muchos dígitos.

Creando una posible expresión regular

Alfabeto => { { a-z, A-Z}, { 0-9 } } = { a-z A-Z, 0-9 }

Expresión regular: (a-z A-Z) (a-z A-Z | 0-9)*

Convirtiendo a una gramática regular

Para mayor compresión: { a-z, A-Z} => \L
{ 0-9 } = \d

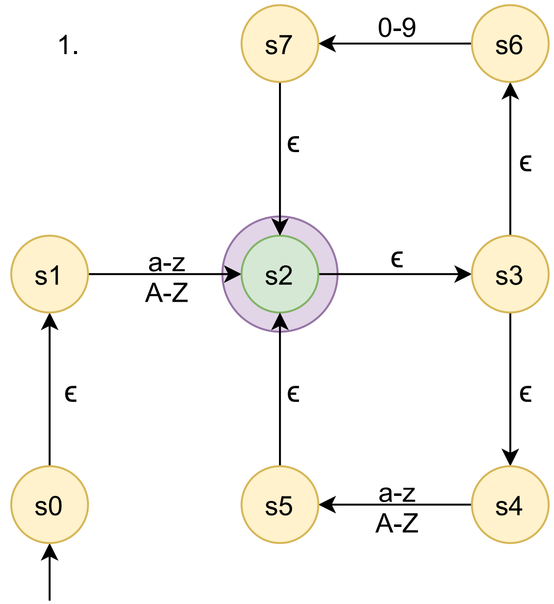
\L (\L | \d)* => S -> \L

\L (\L | \d)* => S -> \L A
A -> BA | ϵ
B -> \L | \d

Resultado: S -> \L A
A -> BA | ϵ
B -> \L | \d

Creando un posible autómata finito determinista

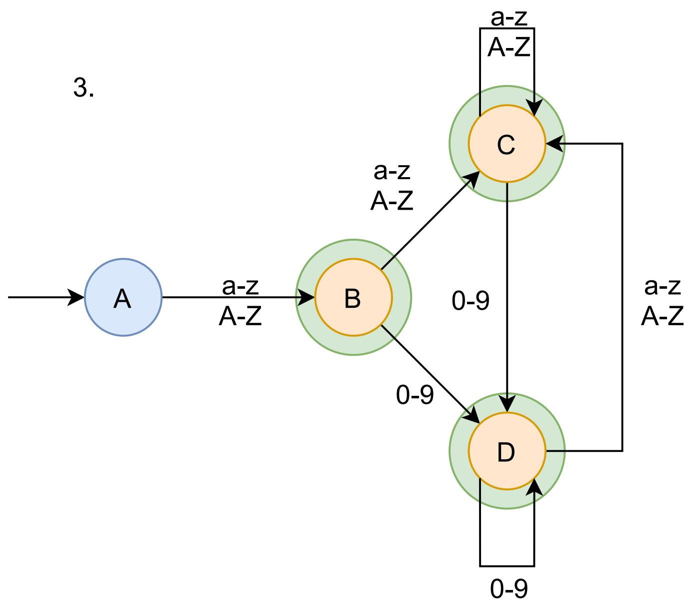
1.



2.

FT	ϵ	a-z A-Z	0-9
s0	s1 = A	(A, a-z A-Z) = s2 = B	(A, 0-9) = {}
s2	s3, s4, s6 = B	(B, a-z A-Z) = s5 = C	(B, 0-9) = s7 = D
s5	s2, s3, s4, s6 = C	(C, a-z A-Z) = s5 = C	(C, 0-9) = s7 = D
s7	s2, s3, s4, s6 = D	(C, a-z A-Z) = s5 = C	(C, 0-9) = s7 = D

3.



Reduciendo AFD

1. No hay estados inaccesibles

2.

Estados	=> Q = { A, B, C, D}
Estado Inicial	=> A
Alfabeto	=> Σ = { { { a-z }, { A-Z } }, {0-9} } = { a-z A-Z, 0-9 }
Aceptación	=> B, C, D
Funciones de Transición	=>

∂(A, a-z A-Z) = B	∂(B, a-z A-Z) = C	∂(C, a-z A-Z) = C	∂(D, a-z A-Z) = C
	∂(B, 0-9) = D	∂(C, 0-9) = D	∂(D, 0-9) = D

3.

	No Aceptación			Aceptacion		
	A			B	C	D
a-z A-Z	B			C	C	C
0-9				D	D	D

4.

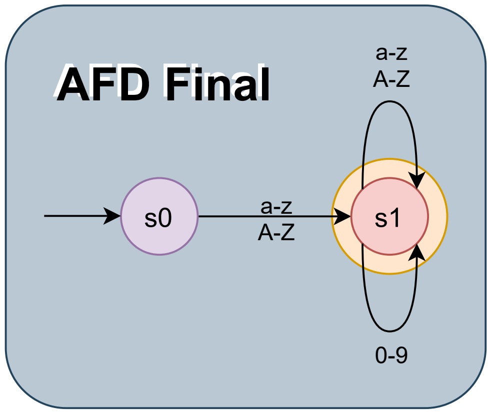
	No Aceptación			Aceptacion		
	A			B	C	D
a-z A-Z	B			C	C	C
0-9				D	D	D

5.

	No Aceptación			Aceptacion		
	s0 = A			s1 = { B, C, D }		
a-z A-Z	B			C	C	C
0-9				D	D	D

6.

∂(s0, a-z A-Z) = s1	∂(s1, a-z A-Z) = s1
	∂(s1, 0-9) = s1



7.

Estados => Q = { s0, s1 }
Estado Inicial => s0
Alfabeto => Σ = { { { a-z }, { A-Z } }, {0-9} } = { a-z A-Z, 0-9 }
Aceptación => s1
Funciones de Transición =>

∂(s0, a-z A-Z) = s1

∂(s1, a-z A-Z) = s1
∂(s1, 0-9) = s1

Número

Son palabras que cumplen con tener al menos un dígito o más, y solo puede contener dígitos.

Creando una posible expresión regular

Alfabeto => { { 0-9 } } = { 0-9 }

Expresión regular: (0-9)+

Convirtiendo a una gramática regular

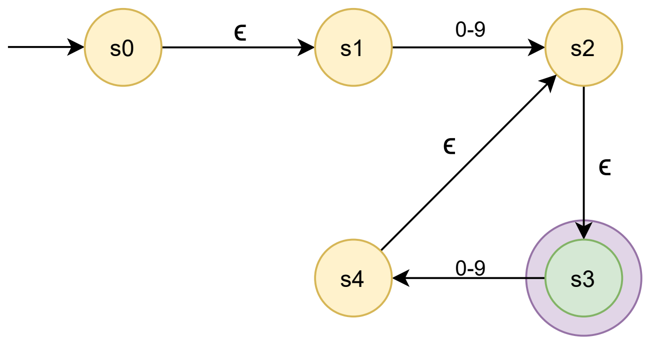
Para mayor compresión: { 0-9 } = \d

\d+ => S -> \d S | \d

Resultado: S -> \d S | \d

Definiendo Posible AFD

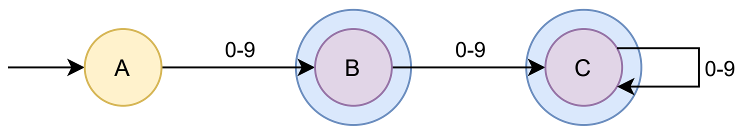
1.



2.

FT	ϵ	0-9
s0	s1 = A	(A, 0-9) = s2 = B
s2	s3 = B	(B, 0-9) = s4 = C
s4	s2, s3 = C	(C, 0-9) = s4 = C

3.



Reduciendo AFD

1. No hay estados inaccesibles

2.

Estados => Q = { A, B, C }
Estado Inicial => A
Alfabeto => Σ = { { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } } = { 0-9 }
Aceptación => B, C
Funciones de Transición =>

∂(A, 0-9) = B ∂(B, 0-9) = C ∂(C, 0-9) = C

3.

	No Aceptación			Aceptacion	
	A			B	C
0-9	B			C	C

4.

	No Aceptación			Aceptacion	
	A			B	C
0-9	B			C	C

5.

	No Aceptación			Aceptacion	
	s0 = A			s1 = { B, C }	
0-9	B			C	C

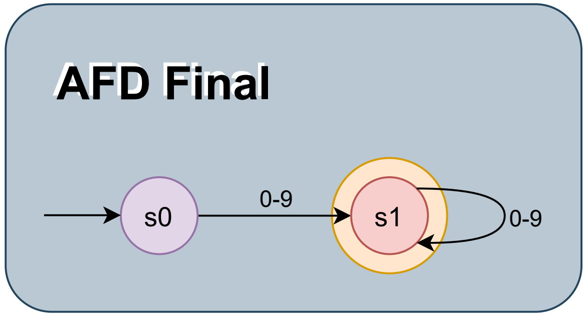
6.

∂(s0, 0-9) = s1	∂(s1, 0-9) = s1

7.

Estados => Q = { s0, s1 }
Estado Inicial => s0
Alfabeto => Σ = { { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } } = { 0-9 }
Aceptación => s1
Funciones de Transición =>

∂(s0, 0-9) = s1 ∂(s1, 0-9) = s1



Decimal

Son palabras que cumplen con tener al menos un dígito o más, seguido de un punto, seguido de uno o más dígitos.

Creando una posible expresión regular

Alfabeto => { { 0-9 }, punto } = { 0-9, punto }

Expresión regular: (0-9)+ (punto) (0-9)+ <=> (0-9)+ [.] (0-9)+

Convirtiendo a una gramática regular

Para mayor compresión: { 0-9 } = \d ; . = \p

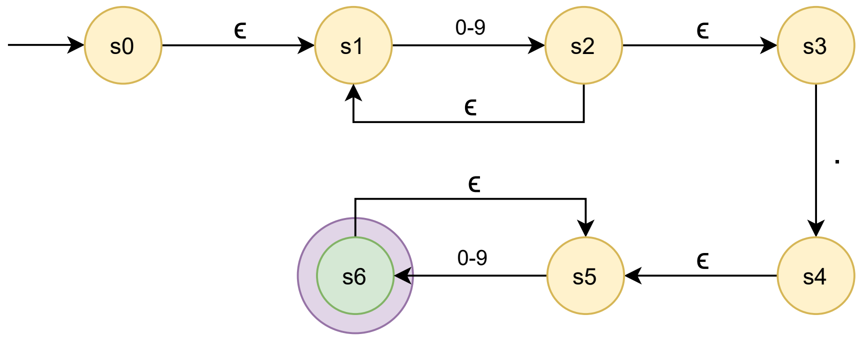
(\d)+ \p (\d)+ => S ->
A -> \d A | \d

(\d)+ \p (\d)+ => S -> A \p A
A -> \d A | \d

Resultado: S -> A \p A <-> A . A
A -> \d A | \d

Definiendo Posible AFD

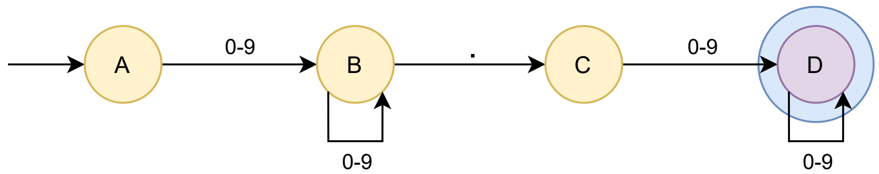
1.



2.

FT	ϵ	0-9	.
s0	s1 = A	(A, 0-9) = s2 = B	(A, .) = {}
s2	s1, s3 = B	(B, 0-9) = s2 = B	(B, .) = s4 = C
s4	s5 = C	(C, 0-9) = s6 = D	(C, .) = {}
s6	s5 = D	(D, 0-9) = s6 = D	(D, 0-9) = {}

3.



Reduciendo AFD

1. No hay estados inaccesibles

2.

Estados => Q = { A, B, C, D }
Estado Inicial => A
Alfabeto => Σ = { { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }, punto } = { 0-9, punto }
Aceptación => D
Funciones de Transición =>

∂(A, 0-9) = B

∂(B, 0-9) = B
∂(B, **.**) = C

∂(C, 0-9) = D

∂(D, 0-9) = D

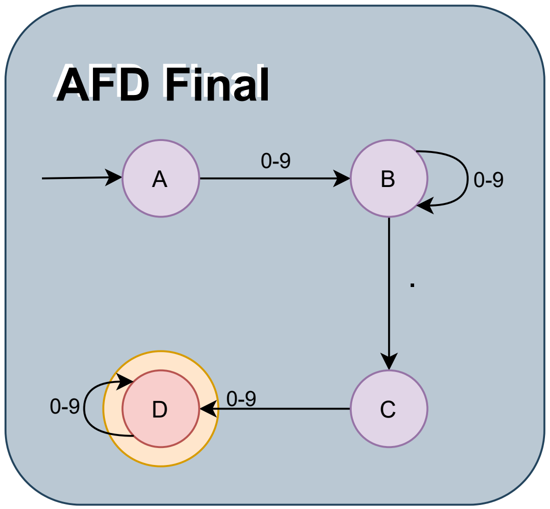
3.

	No Aceptación			Aceptacion	
	A	B	C		D
0-9	B	B	D		D
.		C			

4.

	No Aceptación			Aceptacion	
	A	B	C		D
0-9	B	B	D		D
.		C			

No se puede reducir más el AFD, por lo cual, se tomara el anterior resultado como final.



Puntuación

Ser alguno de los signos de puntuación.

Creando una posible expresión regular

Alfabeto => { [.], [,], [:], [;] }

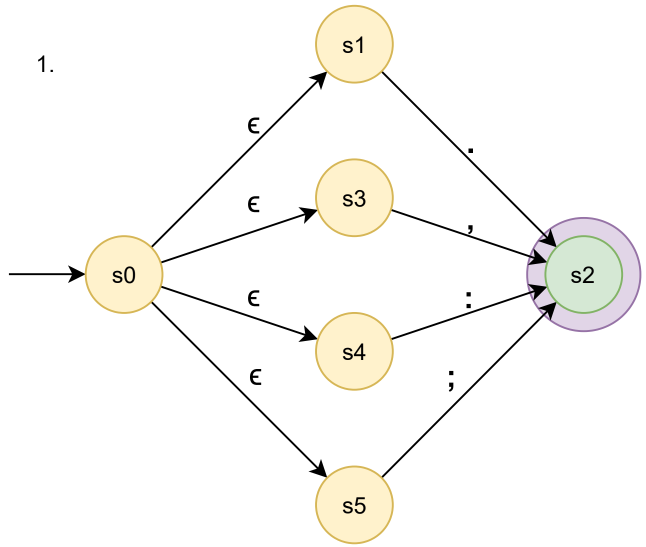
Expresión regular: [.] | [,] | [:] | [;]

Convirtiendo a una gramática regular

Resultado [.] | [,] | [:] | [;] => S -> A | B | C | D
A -> .
B -> ,
C -> :
D -> ;

Definiendo Posible AFD

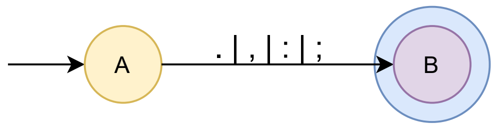
1.



2.

FT	ϵ	.	,	:	;
s0	s1, s3, s4, s5 = A	(A, .) = s2 = B	(A, ,) = s2 = B	(A, :) = s2 = B	(A, ;) = s2 = B
s2	-				

3.



Reduciendo AFD

1. No hay estados inaccesibles

2.

Estados => Q = { A, B }
Estado Inicial => A
Alfabeto => Σ = { [.], [,], [:], [;] }
Aceptación => B
Funciones de Transición =>

$\partial(A, .) = B$ $\partial(A, ,) = B$ $\partial(A, :) = B$ $\partial(A, ;) = B$

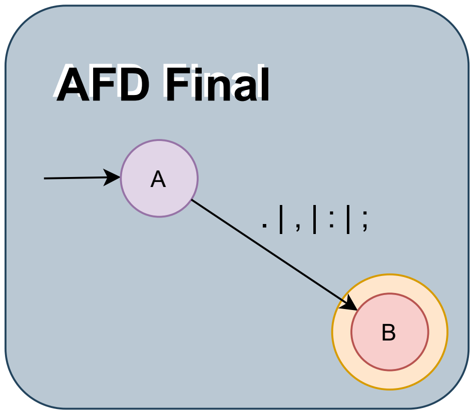
3.

	No Aceptación		Aceptacion	
	A			B
.	B			
,	B			
:	B			
;	B			

4.

	No Aceptación		Aceptacion	
	A			B
.	B			
,	B			
:	B			
;	B			

No se puede reducir más el AFD, por lo cual, se tomara el anterior resultado como final.



Operador

Ser alguno de los operadores aritméticos.

Creando una posible expresión regular

Alfabeto => { +, -, *, /, % }

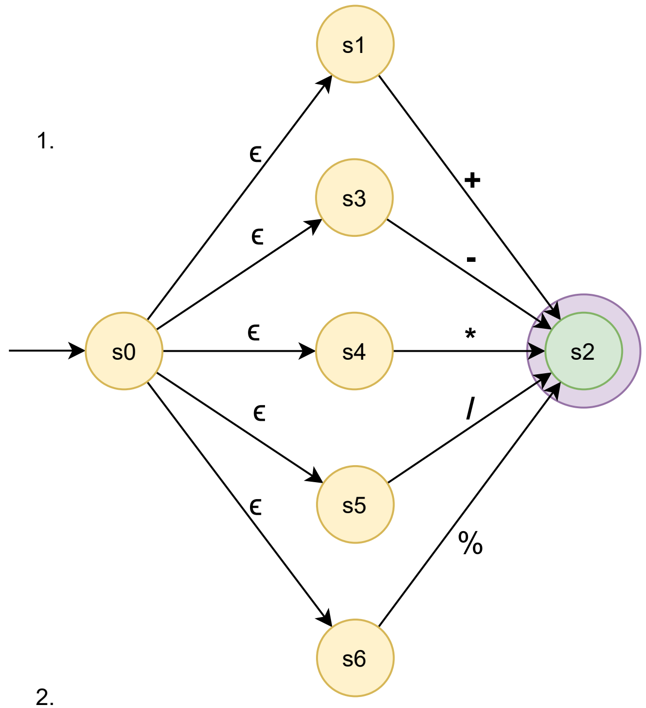
Expresión regular: [+] | [-] | [*] | [/] | [%]

Convirtiendo a una gramática regular

Resultado [+] | [-] | [*] | [/] | [%] => S -> A | B | C | D | E
A -> +
B -> -
C -> *
D -> /
E -> %

Definiendo Posible AFD

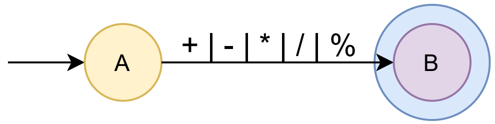
1.



2.

FT	ϵ	+	-	*	/	%
s0	s1, s3, s4, s5, s6 = A	(A, +) = s2 = B	(A, -) = s2 = B	(A, *) = s2 = B	(A, /) = s2 = B	(A, %) = s2 = B
s2	-					

3.



Reduciendo AFD

1. No hay estados inaccesibles

2.

Estados => Q = { A, B }
Estado Inicial => A
Alfabeto => Σ = { +, -, *, /, % }
Aceptación => B
Funciones de Transición =>

$\partial(A, +) = B$ $\partial(A, -) = B$ $\partial(A, *) = B$ $\partial(A, /) = B$ $\partial(A, \%) = B$

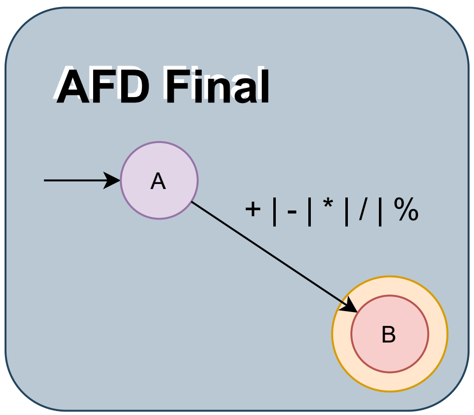
3.

	No Aceptación		Aceptacion	
	A			B
+	B			
-	B			
*	B			
/	B			
%	B			

No se puede reducir más el AFD, por lo cual, se tomara el anterior resultado como final.

4.

	No Aceptación		Aceptacion	
	A			B
+	B			
-	B			
*	B			
/	B			
%	B			



Agrupación

Ser alguno de los signos de agrupación.

Creando una posible expresión regular

Alfabeto => { (,), [,], {, } }

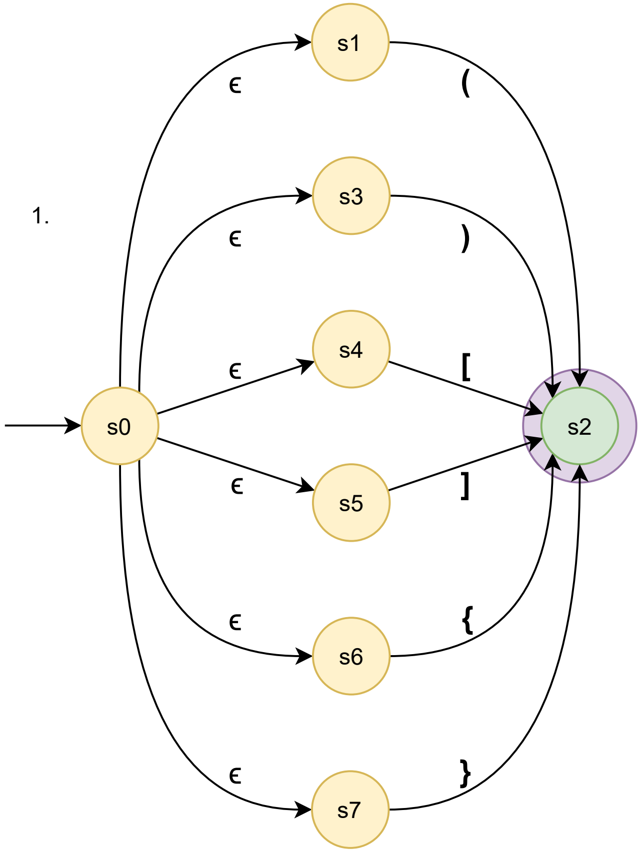
Expresión regular: [(] | [)] | [[] | []] | [{] | [}]

Convirtiendo a una gramática regular

Resultado [(] | [)] | [[] | []] | [{] | [}] => S -> A | B | C | D | E | F
A -> (
B ->)
C -> [
D ->]
E -> {
F -> }

Definiendo Posible AFD

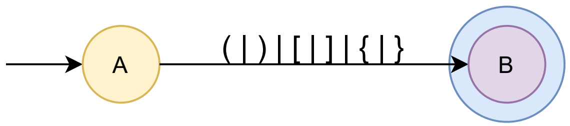
1.



2.

FT	ϵ	(	)	[	]	{	}
s0	s1, s3, s4, s5, s6, s7 = A	(A, () = s2 = B	(A,)) = s2 = B	(A, [) = s2 = B	(A,]) = s2 = B	(A, {) = s2 = B	(A, }) = s2 = B
s2	-						

3.



Reduciendo AFD

1. No hay estados inaccesibles

2.

Estados => Q = { A, B }
Estado Inicial => A
Alfabeto => Σ = { (,), [,], {, } }
Aceptación => B
Funciones de Transición =>

∂(A, () = B ∂(A,)) = B ∂(A, [) = B ∂(A,]) = B ∂(A, {) = B ∂(A, }) = B

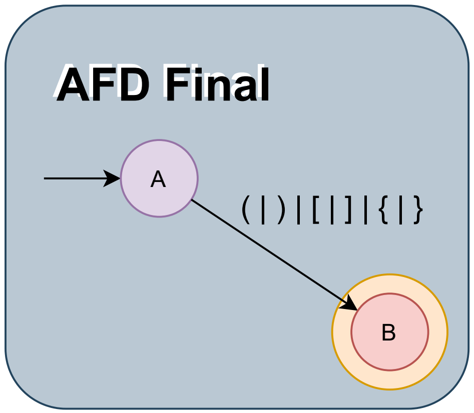
3.

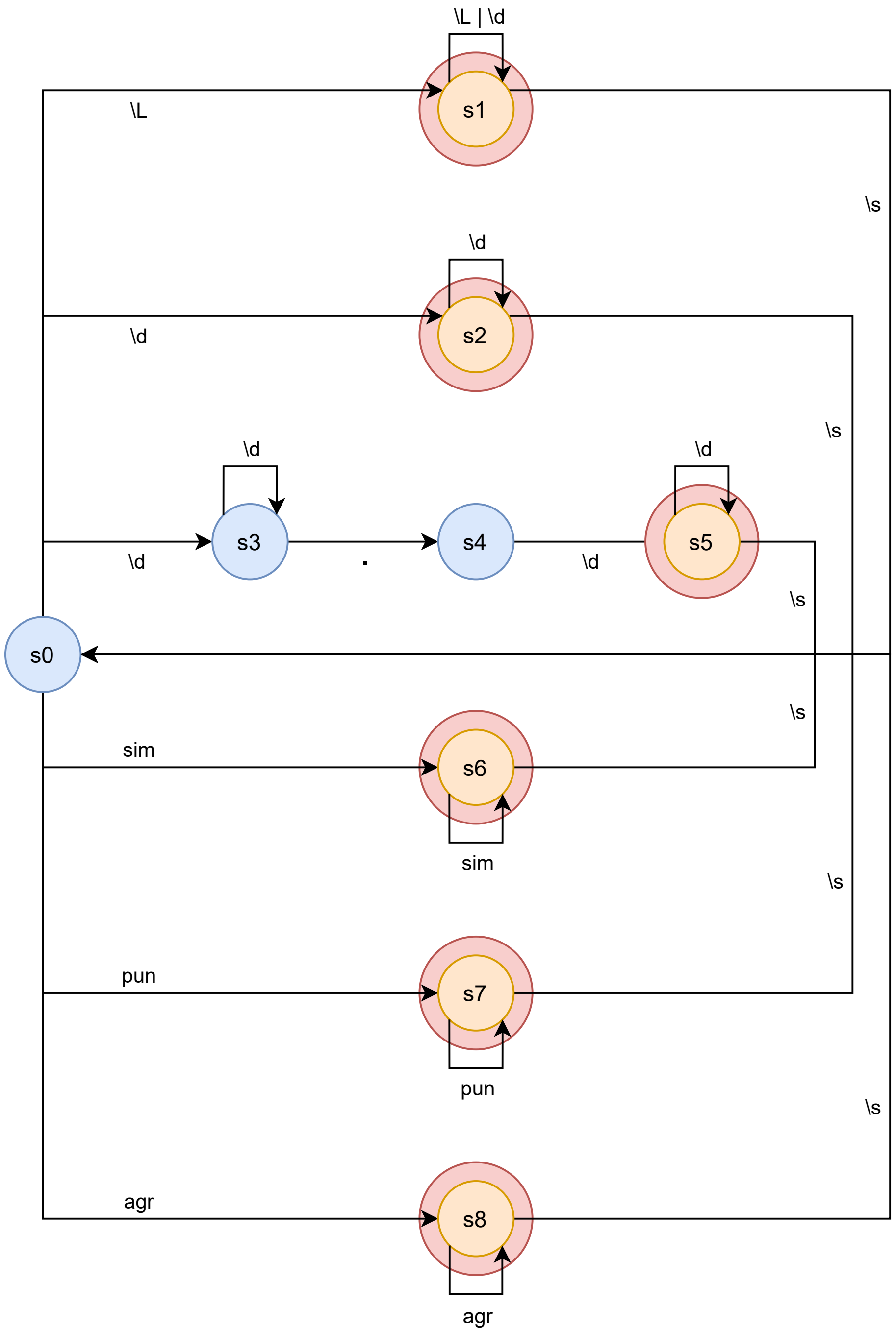
	No Aceptación		Aceptacion	
	A			B
(	B			
)	B			
[	B			
]	B			
{	B			
}	B			

4.

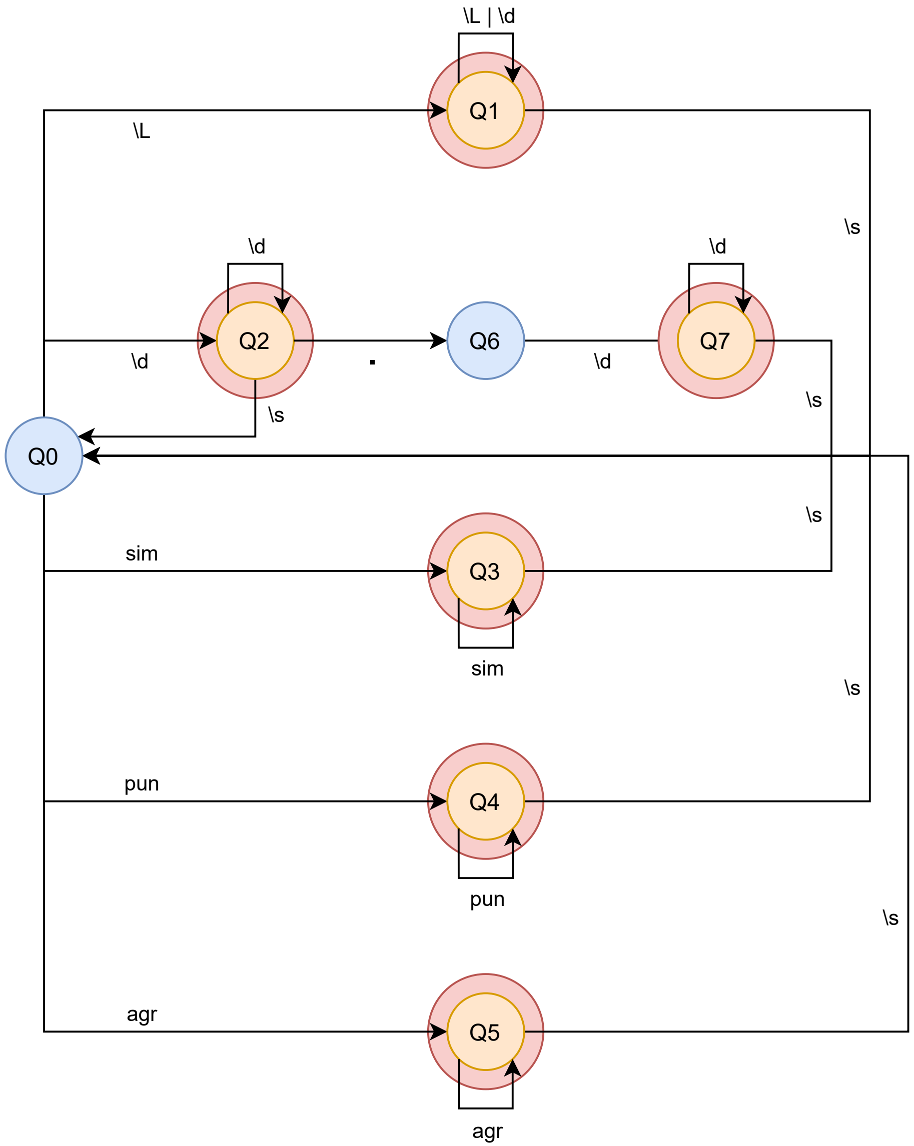
	No Aceptación		Aceptacion	
	A			B
(	B			
)	B			
[	B			
]	B			
{	B			
}	B			

No se puede reducir más el AFD, por lo cual, se tomara el anterior resultado como final.





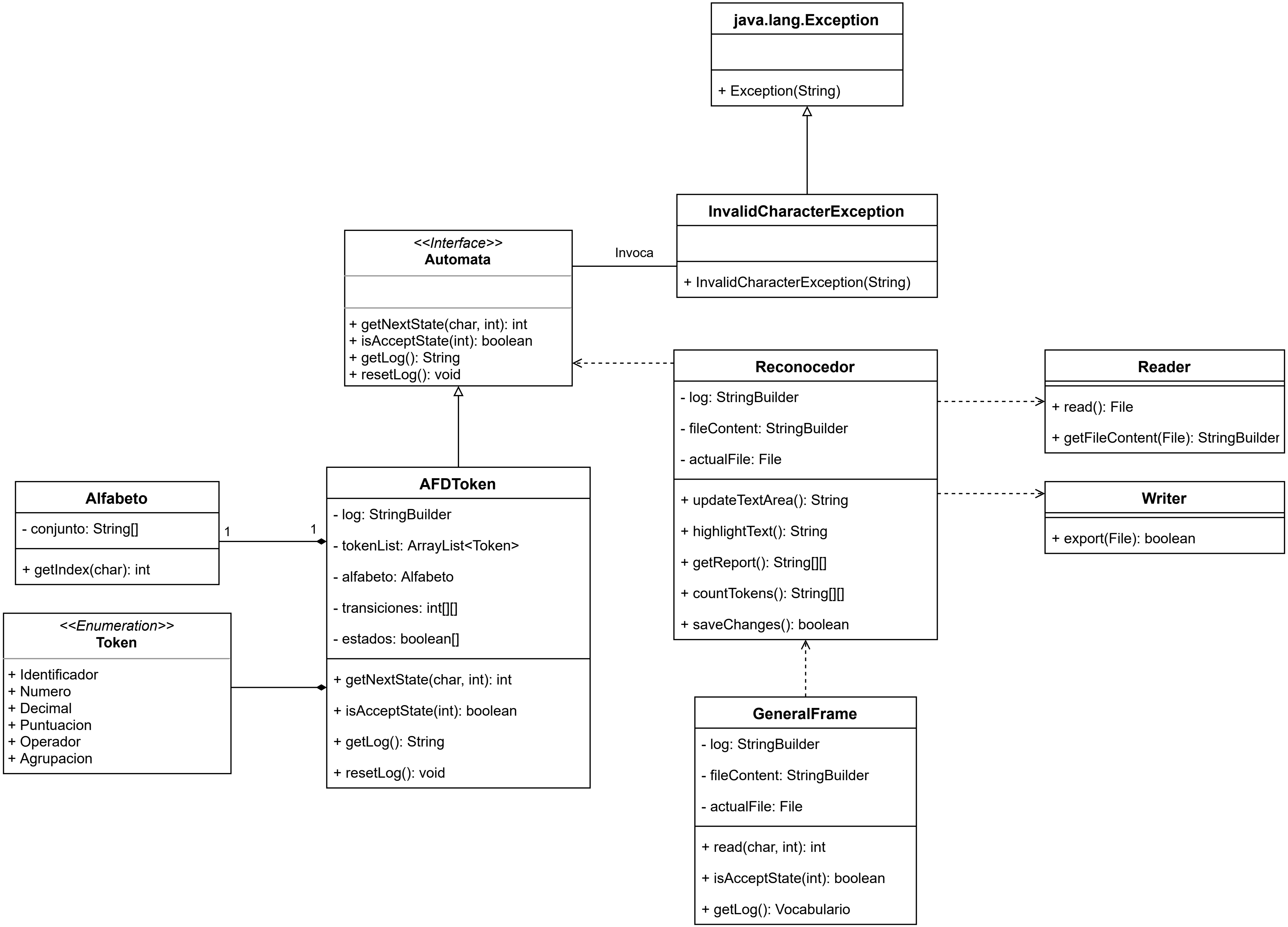
Estado	\L	\d	.	pun	ope	agr	\s
s0 = Q0	s1 = Q1	s2, s3 = Q2	s6 = Q3	s6 = Q3	s7 = Q4	s8 = Q5	
Q1	s1 = Q1	s1 = Q1					s0 = Q0
Q2		s2, s3 = Q2	s4 = Q6				s0 = Q0
Q3			s6 = Q3	s6 = Q3			s0 = Q0
Q4					s7 = Q4		s0 = Q0
Q5						s8 = Q5	s0 = Q0
Q6		s5 = Q7					
Q7		s5 = Q7					s0 = Q0



Definición Formal

Estados => Q = { Q0, Q1, Q2, Q4, Q5, Q6, Q7 }
Estado Inicial => Q0
Alfabeto => Σ = { a-z A-z, 0-9, ., { [.], [,], [:], [;] }, { +, -, *, /, % }, { (,), [,], {, } } }
Aceptación => B, C, D
Funciones de Transición =>

∂(Q0, a-z A-Z) = Q1	∂(Q1, a-z A-Z) = Q1	
∂(Q0, 0-9) = Q2	∂(Q1, 0-9) = Q1	∂(Q2, 0-9) = Q2
∂(Q6, 0-9) = Q7	∂(Q7, 0-9) = Q7	
∂(Q0, .) = Q3	∂(Q2, .) = Q6	∂(Q3, .) = Q3
∂(Q0, pun) = Q3	∂(Q3, pun) = Q1	
∂(Q0, ope) = Q4	∂(Q4, ope) = Q4	
∂(Q0, agr) = Q5	∂(Q5, agr) = Q5	
∂(Q1, \s) = Q0	∂(Q2, \s) = Q0	∂(Q3, \s) = Q0
∂(Q4, \s) = Q0	∂(Q5, \s) = Q0	∂(Q7, \s) = Q0


java.lang.Exception
+ Exception(String)
InvalidCharacterException
+ InvalidCharacterException(String)
Invoca
<<Interface>>
Automata
+ getNextState(char, int): int
+ isAcceptState(int): boolean
+ getLog(): String
+ resetLog(): void
Reconocedor
- log: StringBuilder
- fileContent: StringBuilder
- actualFile: File
+ updateTextArea(): String
+ highlightText(): String
+ getReport(): String[][]
+ countTokens(): String[][]
+ saveChanges(): boolean
Reader
+ read(): File
+ getFileContent(File): StringBuilder
Writer
+ export(File): boolean
Alfabeto
- conjunto: String[]
+ getIndex(char): int
1
1
AFDToken
- log: StringBuilder
- tokenList: ArrayList<Token>
- alfabeto: Alfabeto
- transiciones: int[][]
- estados: boolean[]
+ getNextState(char, int): int
+ isAcceptState(int): boolean
+ getLog(): String
+ resetLog(): void
<<Enumeration>>
Token
+ Identificador
+ Numero
+ Decimal
+ Puntuacion
+ Operador
+ Agrupacion
GeneralFrame
- log: StringBuilder
- fileContent: StringBuilder
- actualFile: File
+ read(char, int): int
+ isAcceptState(int): boolean
+ getLog(): Vocabulario