

Lenguajes de Programación

1. Los humanos un lenguaje para comunicarnos.

La comunicación nos es útil para:

- a. Intercambio de ideas
- b. Expresar emociones
- c. Transmitir un mensaje
- d. Describir aspectos del mundo real

Para lograr lo anterior (y más) los lenguajes deben tener ciertas características:

- a) No ambigüo
- b) Debe permitirnos plantear soluciones a problemas.

Una computadora está prevista de un "lenguaje"

Un lenguaje de programación es el medio por el cual nos comunicamos con la computadora.

Las computadoras se desarrollaron para apoyarnos en la solución de problemas.

- Lenguaje Máquina

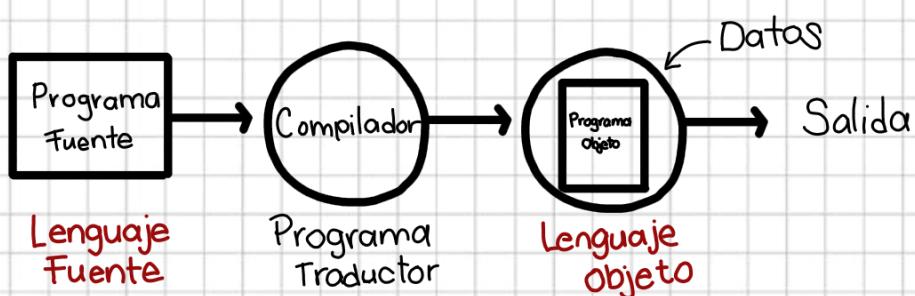
- Lenguaje Ensamblador

{
Traductor
código ensamblador
↓
código máquina

- Lenguaje de alto nivel

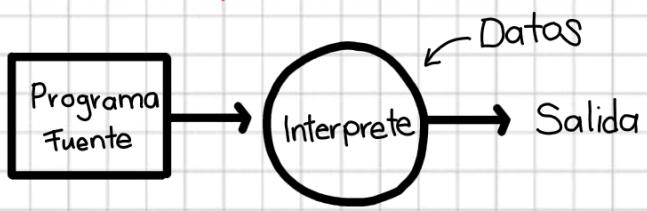
{
Traductor
Compilador
código de alto nivel
↓
código máquina

Compilador Proceso

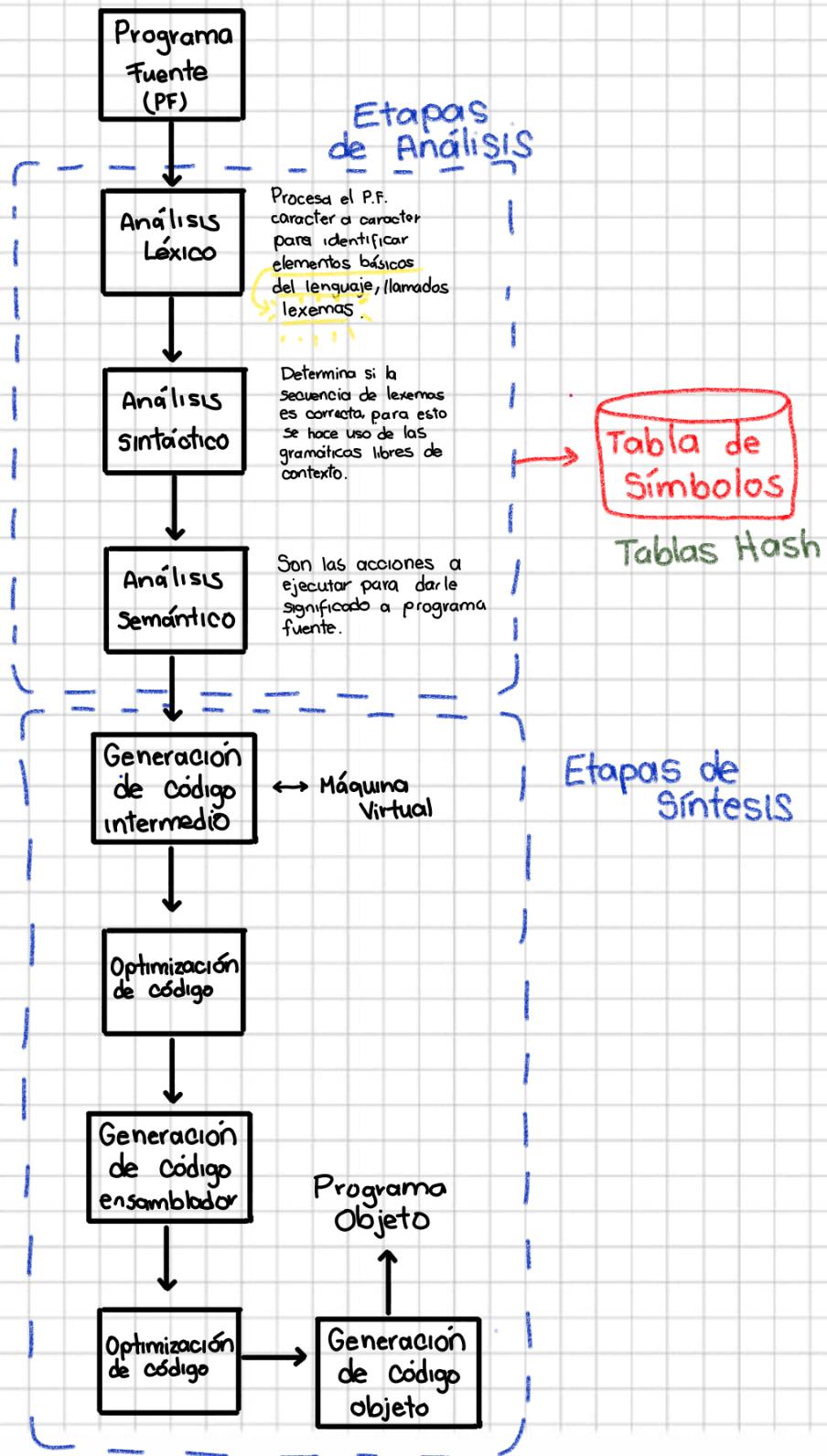


- Tiempo de compilación
- Tiempo de ejecución

Interprete



Proceso o fases de un compilador



Consideraciones:

1. Los lexemas se clasifican en **clases léxicas**.
2. A cada clase léxica se le asigna un ID, valor entero positivo único.

Ejemplos:	Clase léxica:	Identif.:
VAR		10
Núm. Entero		20
Núm. P.F.		30
+		40
;		50

A los **identificadores** se le llama **Tokens**

3. La comunicación entre el analizador léxico y el sintáctico es por medio de tokens.

Lenguaje de alto nivel Primer compilador: Fortran

- 1 Es más fácil que el lenguaje de bajo nivel
- 2 Es independiente de la computadora
- 3 El programador no se tiene que preocupar de como funciona
- 4 Tampoco se debe preocupar de la arquitectura de von Neumann
- 5
- 6 Nos permite obtener más fácil la solución al problema
- 7 Se permite la modulación y reutilización del código
- 8

Análisis Léxico

El análisis léxico se realiza por medio de automátas finitos no determinista y deterministas.

Un AFN (Automata Finito no Determinista)

A es una 5-tupla $(\Sigma, Q, S, \delta, t)$

donde:

Σ : es el alfabeto del afn

Alfabeto := Conjunto Finito no vacío de elementos llamados símbolos del alfabeto

Q : conjunto finito no vacío de elementos llamados estados.

S : un elemento de Q llamada estado inicial del autómata

δ : una relación

$$\delta: Q \times \Sigma \rightarrow 2^Q$$

$$\text{Si } Q = \{1, 2, 3\}$$

$$2^Q = \left\{ \{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\} \right\}$$

En un AFD el cambio se da en δ

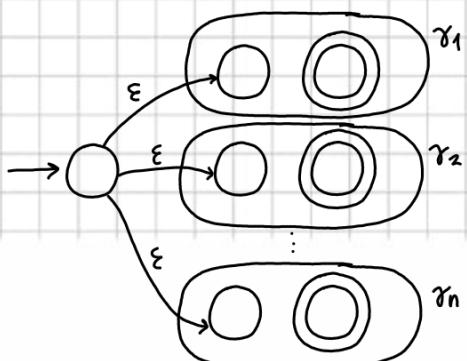
$$\delta: Q \times \Sigma \rightarrow Q$$

Para construir un analizador léxico, cada clase léxica se representa por medio de una expresión regular.

Si tenemos las clases léxicas C_1, C_2, \dots, C_n .

y sus respectivas expresiones regulares $\gamma_1, \gamma_2, \dots, \gamma_n$

Se construye el AFN asociado a cada e.r. utilizando el método de Thompson, y se procede a unir estos n AFN's en un sólo AFN.



A cada estado de aceptación se le asociará su respectivo token.

Expresiones Regulares

Sea Σ un alfabeto, las sig. expresiones son expresiones regulares.

- 1) Si $a \in \Sigma$ entonces a es una e.r. que representa al lenguaje $\{a\}$
- 2) Si $a, b \in \Sigma$ entonces $a|b$ es una e.r. que representa al lenguaje $\{a\} \cup \{b\} = \{a, b\}$
- 3) Si $a, b \in \Sigma$, entonces $a \circ b$ es una e.r. que representa $\{a\} \circ \{b\} = \{ab\}$
- 4) Si $a \in \Sigma$, entonces a^+ es una e.r. que representa al $\{a\}^+ = \{a, aa, \dots\}$
- 5) Si $a \in \Sigma$, entonces a^* es una e.r. que representa al $\{a\}^* = \{\epsilon, a, aa, \dots\}$
- 6) Si $a \in \Sigma$, entonces $a^?$ es una e.r. que representa al lenguaje $\{a, \epsilon\}$

Generalización:

Si E_1 y E_2 son expresiones regulares sobre Σ , y \mathcal{J}_1 , \mathcal{J}_2 son los lenguajes que tienen asociados esas e.r., entonces

- 1) $E_1 | E_2$ es una e.r que representa al lenguaje $\mathcal{J}_1 \cup \mathcal{J}_2$
- 2) $E_1 \circ E_2$ es una e.r que representa al lenguaje $\mathcal{J} = \mathcal{J}_1 \circ \mathcal{J}_2$
- 3) E_1^+ es una e.r que representa al lenguaje $\mathcal{J} = \mathcal{J}_1^+ = \mathcal{J}_1 \cup \mathcal{J}_1^2 \cup \mathcal{J}_1^3 \cup \dots$
 $= \bigcup_{i=1}^{\infty} \mathcal{J}_1^i$

$$\text{donde } \mathcal{J}_1^2 = \mathcal{J}_1 \circ \mathcal{J}_1, \mathcal{J}_1^3 = \mathcal{J}_1^2 \circ \mathcal{J}_1 = \mathcal{J}_1 \circ \mathcal{J}_1^2$$

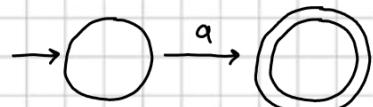
- 4) E_1^* es una e.r que representa al lenguaje $\mathcal{J} = \mathcal{J}_1^* = \{\epsilon\} \cup \mathcal{J}_1^+$
- 5) $E_1^?$ es una e.r que representa al lenguaje $\mathcal{J} = \mathcal{J}_1 \cup \{\epsilon\}$

Construcción de Thompson

La construcción de Thompson nos proporciona un método para construir un AFN asociado a una expresión regular, de tal manera que el AFN reconoce únicamente a las cadenas que pertenecen al lenguaje definido por la expresión regular (e.r.).

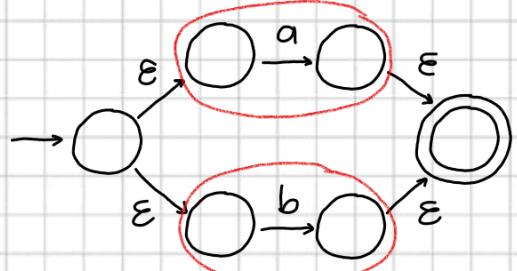
Sea Σ un alfabeto, y $a, b \in \Sigma$ entonces:

- 1) El afn asociado a la e.r. a es: $\mathcal{J} = \{a\}$



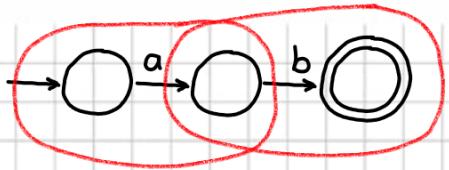
- 2) El afn asociado a la e.r.

$a|b$



3) El afn asociado a la e.r.

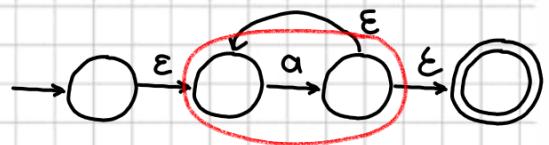
$$ab \equiv a \circ b$$



4) El afn asociado a la e.r.

$$a^+$$

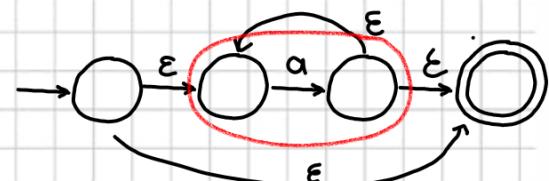
$$\mathcal{L} = \{a, aa, aaa, \dots\}$$



5) El afn asociado a la e.r.

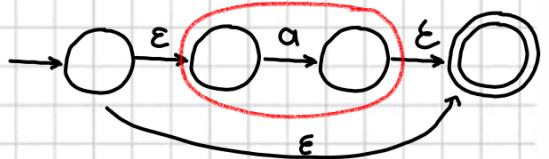
$$a^*$$

$$\mathcal{L} = \{\epsilon, a, aa, aaa, \dots\}$$

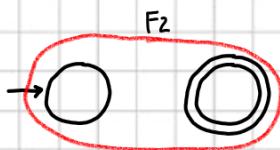
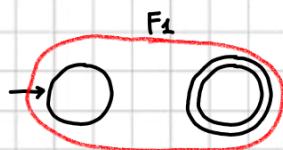


6) El afn asociado a la e.r.

$$a^?$$

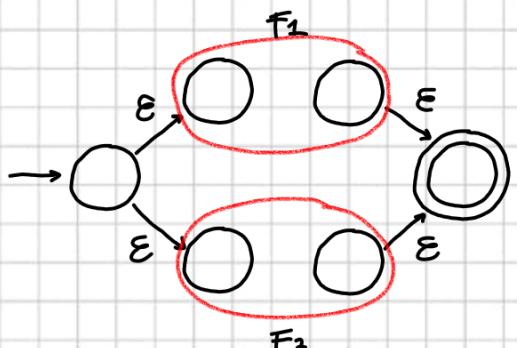


Generalizando: Si e_1 y e_2 son e.r. con sus respectivos afn's, F_1 y F_2



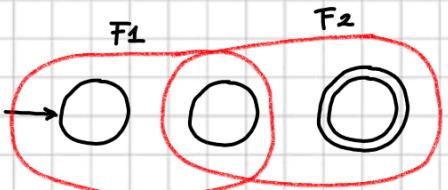
a) El afn para $e_1 | e_2$ es:

$$\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$$



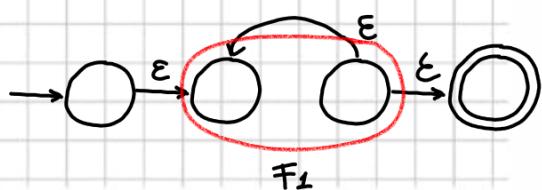
b) El afn para $e_1 \circ e_2$ es:

$$\mathcal{L} = \mathcal{L}_1 \circ \mathcal{L}_2$$

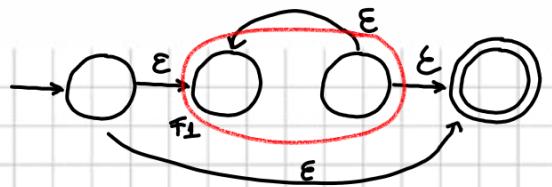


c) El afn para e_1^+ es:

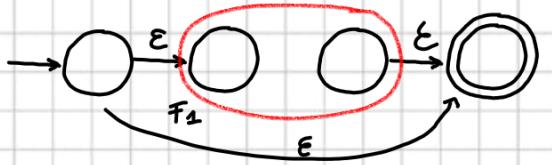
$$\mathcal{L}_1 = \mathcal{L}_1^+ = \mathcal{L}_1 \cup \mathcal{L}_1^2 \cup \dots \cup$$



d) El afn para e_1^* es:



e) El afn para $e_1^?$ es:



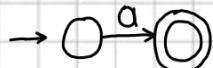
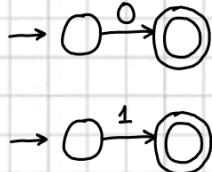
Ejemplos:

$$\text{Si } \Sigma = \{a, b, 0, 1\}$$

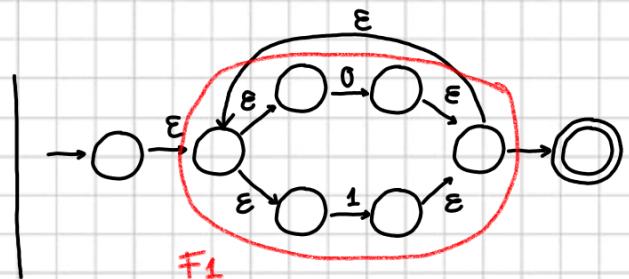
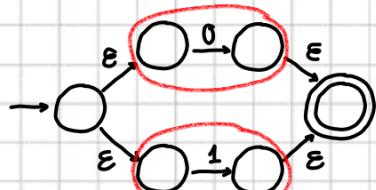
obtener los afn's asociados a las sig. e.r. (utilizando Thompson)

a) $(0|1)^+ \circ a^*$

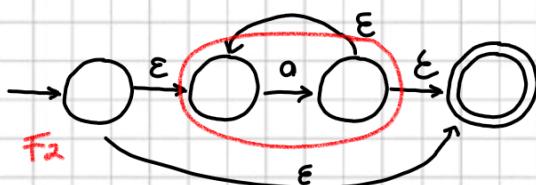
Solución:



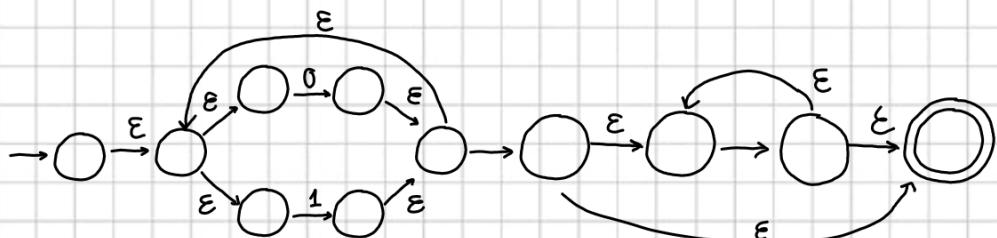
Para $0|1$



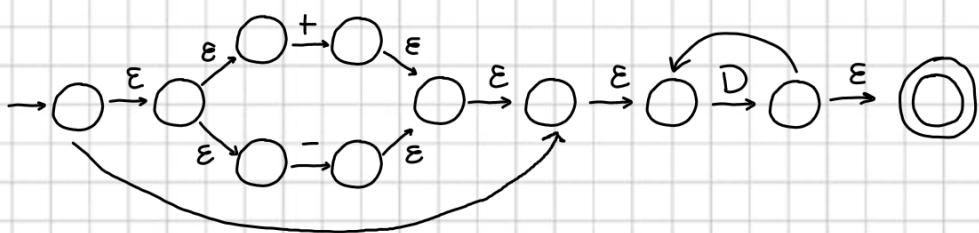
Ahora el afn para a^* es:



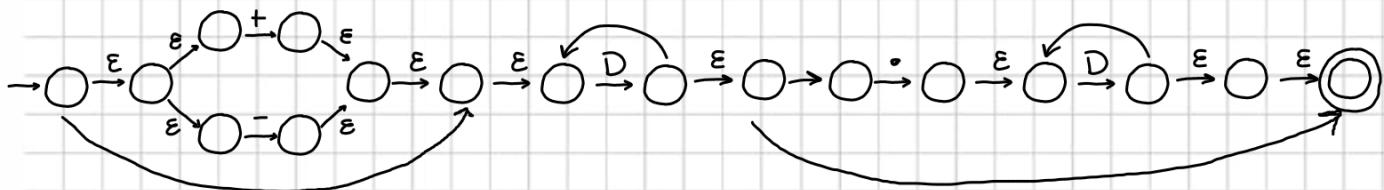
Finalmente para $(0|1)^+ \circ a^*$ tiene:



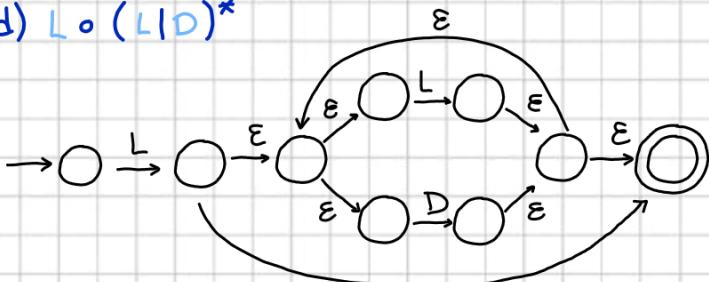
b) $(+|-)? \circ D^+$



c) $(+|-)? \circ D^+ \circ (.\circ D^+)?$



d) $L \circ (L \cap D)^*$



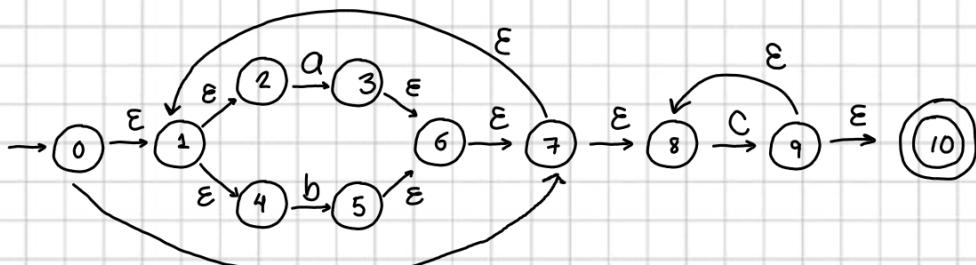
Operaciones sobre un AFN con transiciones epsilon

1) Operación Cerradura Epsilon

Sed F un AFN y e un estado de F_1 . La cerradura epsilon de e se define como:

$$\zeta_{-\epsilon}(e) = \{e\} \cup \{d \mid d \text{ es "alcanzable" desde } e, \text{ únicamente con transiciones epsilon}\}$$

Considerando el sig. AFN



que corresponde a la e.r. $(a|b)^* \circ C^*$
calcular:

$$\zeta_{-\epsilon}\{0\} = \{0\} \cup \{1, 2, 4, 7, 8\} = \{0, 1, 2, 4, 7, 8\}$$

Ejemplos:

Calcular

$$\mathcal{L}_\varepsilon(3) = \{3, 6, 1, 2, 4, 7, 8\}$$

$$\mathcal{L}_\varepsilon(5) = \{5, 6, 1, 2, 4, 7, 8\}$$

Generalización: Si A es un conjunto de estados y $a \in \Sigma$, entonces:

$$\mathcal{L}_\varepsilon(A) = \bigcup_{e \in A} \mathcal{L}_\varepsilon(e)$$

2) Operación Mover

$a \in \Sigma$ y e un estado de F .

$$\text{Mover}(e, a) = \{\ell \mid \text{Se tiene una transición de } e \text{ con } a \text{ a } \ell\}$$

$$S(e, a) = \ell$$

Ejemplos:

$$\text{Mover}(2, a) = \{3\}$$

$$\text{Mover}(2, b) = \{\}$$

$$\text{Mover}(4, b) = \{5\}$$

$$\text{Mover}(8, C) = \{9\}$$

$$\text{Mover}(8, a) = \{\}$$

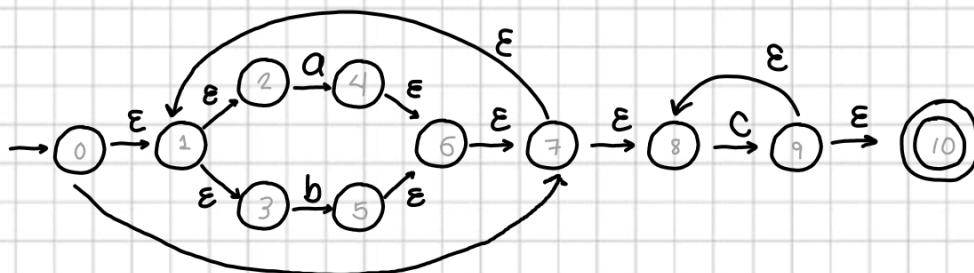
Generalización: Si A es un conjunto de estados y $a \in \Sigma$, entonces:

$$\text{Mover}(A, a) = \bigcup_{e \in A} \text{Mover}(e, a)$$

3) Operación Ir A

$$\text{Ir A}(A, a) = \mathcal{L}_\varepsilon(\text{Mover}(A, a))$$

Ejemplos: Calcular si $A = \{0, 1, 2, 3, 7, 8\}$



$$\begin{aligned}\text{Ir A}(A, a) &= \mathcal{L}_\varepsilon(\text{Mover}(A, a)) \\ &= \mathcal{L}_\varepsilon(\{4\}) = \{1, 2, 3, 4, 6, 7, 8\}\end{aligned}$$

$$\begin{aligned}\text{Ir A}(A, b) &= \mathcal{L}_\varepsilon(\text{Mover}(A, b)) \\ &= \mathcal{L}_\varepsilon(\{5\}) = \{1, 2, 3, 5, 6, 7, 8\}\end{aligned}$$

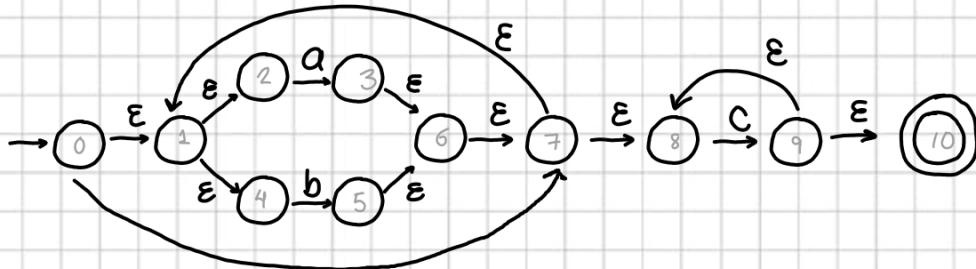
$$\begin{aligned}\text{Ir A}(A, C) &= \mathcal{L}_\varepsilon(\text{Mover}(A, C)) \\ &= \mathcal{L}_\varepsilon(\{9\}) = \{8, 9, 10\}\end{aligned}$$

Conversión de AFN a AFD

Método de la construcción de conjunto de estados.

Ilustraremos el proceso por medio de un ejemplo:

Convertir a AFD el sig. AFN



que corresponde a la e.r. $(a|b)^* \circ c^+$

1er Paso: Se calcula la \mathcal{L}_ϵ del estado inicial

$$S_0 = \mathcal{L}_\epsilon(\{0\}) = \{0, 1, 2, 4, 7, 8\}$$

2do Paso: Se calcula IrA del conjunto S_0 con cada símbolo del alfabeto
 $\Sigma = \{a, b, c\}$

$$S = IrA(S_0, a) = \mathcal{L}_\epsilon(S_0, a) = \mathcal{L}_\epsilon(\{3\}) = \{1, 2, 3, 4, 6, 7, 8\}$$

Se verifica si S es igual a algún conjunto ya creado.

En este caso es un conjunto nuevo, que será S_1 .

$$S_1 = IrA(S_0, a) = \{1, 2, 3, 4, 6, 7, 8\}$$

$$S_2 = IrA(S_0, b) = \mathcal{L}_\epsilon(\{5\}) = \{1, 2, 4, 5, 7, 8\}$$

$$S_3 = IrA(S_0, c) = \mathcal{L}_\epsilon(\{9\}) = \{8, 9, 10\}$$

Finaliza el análisis de S_0 .

Se repite el procedimiento para cada conjunto S_j no analizado.

Analizar S_1 :

$$S_1 = IrA(S_1, a) = \mathcal{L}_\epsilon(\{3\})$$

$$S_2 = IrA(S_1, b) = \mathcal{L}_\epsilon(\{5\})$$

$$S_3 = IrA(S_1, c) = \mathcal{L}_\epsilon(\{9\})$$

Analizar S_2 :

$$S_1 = IrA(S_2, a) = \mathcal{L}_\epsilon(\{3\})$$

$$S_2 = IrA(S_2, b) = \mathcal{L}_\epsilon(\{5\})$$

$$S_3 = IrA(S_2, c) = \mathcal{L}_\epsilon(\{9\})$$

Analizar S_3 :

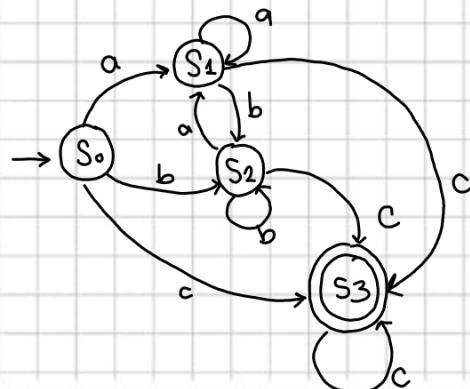
$$S_1 = IrA(S_3, a) = \mathcal{L}_\epsilon(\{\}) = \{\}$$

$$S_2 = IrA(S_3, b) = \mathcal{L}_\epsilon(\{\}) = \{\}$$

$$S_3 = IrA(S_3, c) = \mathcal{L}_\epsilon(\{9\})$$

Ya no hay conjuntos S_j por finalizar.

Procedemos a construir el AFD. Cada S_j es un estado del AFD y S_0 es estado inicial.



Problema: Obtengal, utilizando Thompson, los afn asociados a:

$$1) L \circ (L \mid D)^*$$

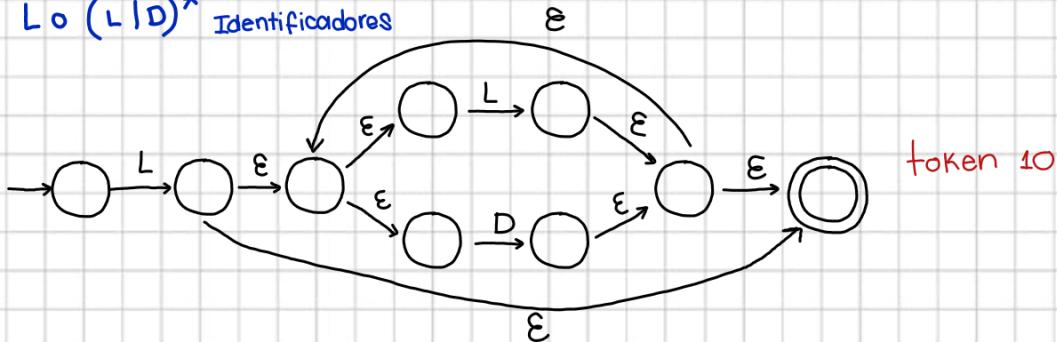
$$2) D \circ (.\circ D^+)?$$

$$3) S^+$$

$$4) + \mid -$$

$$1) L \circ (L \mid D)^*$$

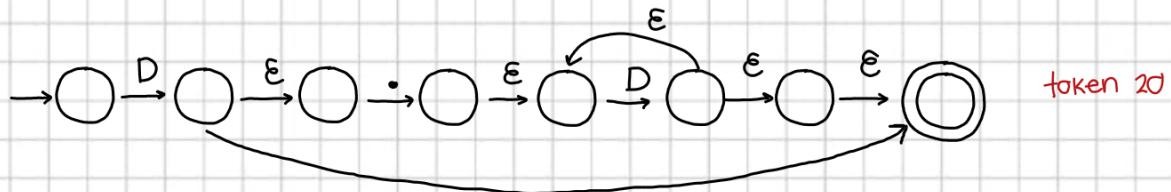
Identificadores



token 10

$$2) D \circ (.\circ D^+)?$$

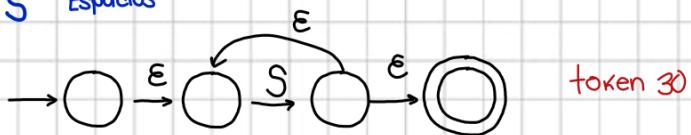
Números



token 20

$$3) S^+$$

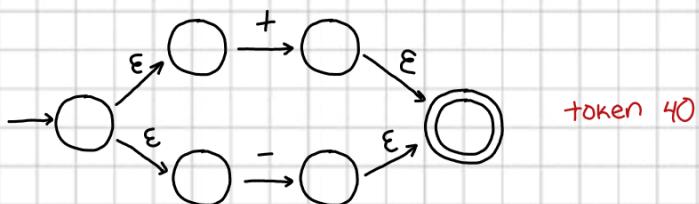
Espacios



token 30

$$4) + \mid -$$

Suma o resta



token 40

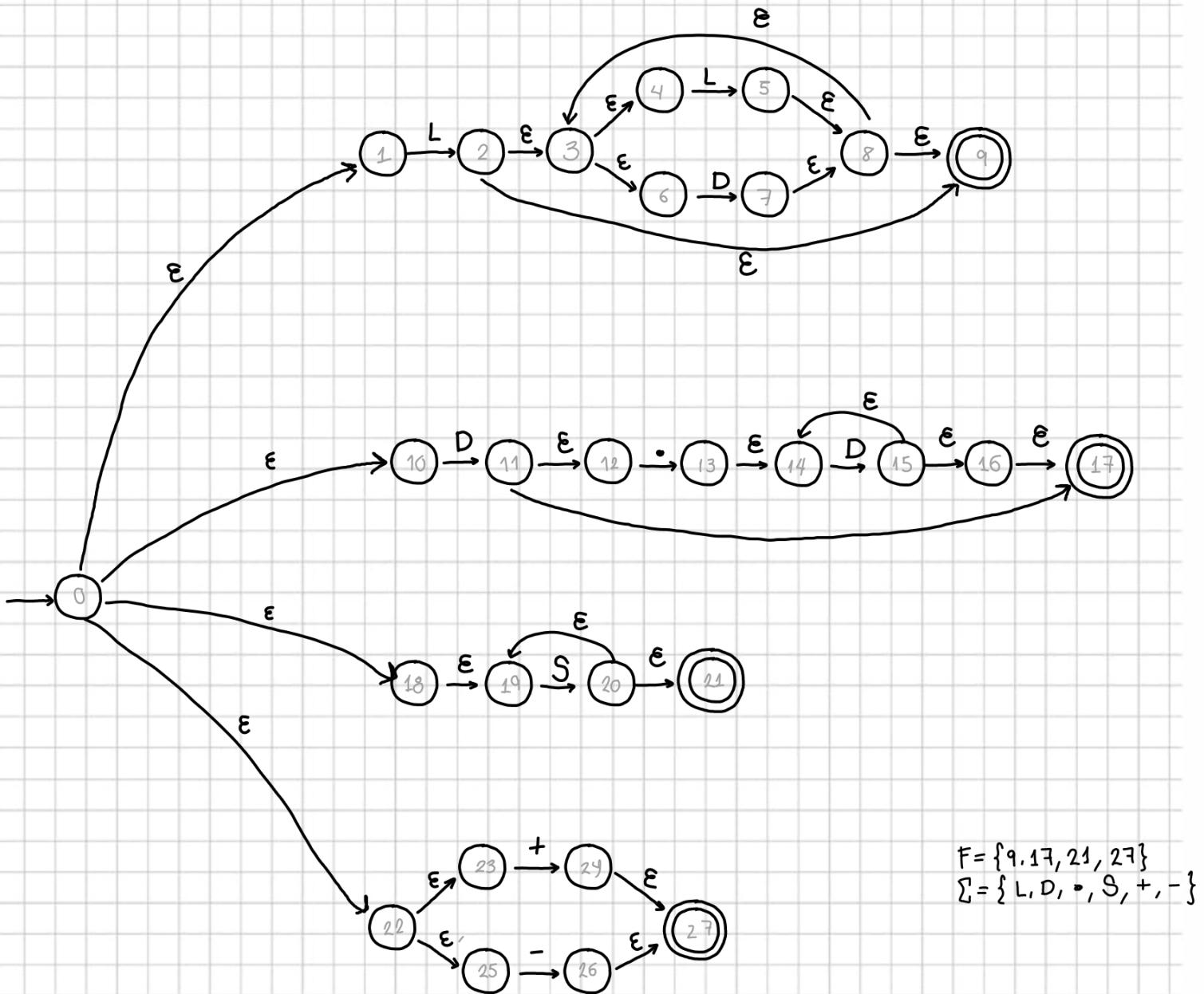
Cada una de las e.r. anteriores representa una clase léxica.

Le designaremos su respectivo token.

Identificadores
Números
Espacios
Suma o resta

token 10
token 20
token 30
token 40

Ahora procederemos a unir todos estos AFN's. Conservando los estados de aceptación.



Procedemos a convertir el AFN a AFD

$$S_0 = \emptyset - \epsilon (\{f_0\}) = \{0, 1, 10, 18, 19, 23, 25\}$$

Analizar S_0 :

$$S_1 = \text{IrA}(S_0, L) = \emptyset - \epsilon (\{f_2\}) = \{2, 3, 4, 6, 9\}$$

$$S_2 = \text{IrA}(S_0, D) = \emptyset - \epsilon (\{f_{11}\}) = \{11, 12, 17\}$$

$$\text{IrA}(S_0, \bullet) = \emptyset$$

$$S_3 = \text{IrA}(S_0, S) = \emptyset - \epsilon (\{f_{20}\}) = \{19, 20, 21\}$$

$$S_4 = \text{IrA}(S_0, +) = \emptyset - \epsilon (\{f_4\}) = \{24, 27\}$$

$$S_5 = \text{IrA}(S_0, -) = \emptyset - \epsilon (\{f_{26}\}) = \{26, 27\}$$

Analizar S_1 :

$$S_6 = \text{IrA}(S_1, L) = \emptyset - \epsilon (\{f_5\}) = \{3, 4, 5, 6, 8, 9\}$$

$$S_7 = \text{IrA}(S_1, D) = \emptyset - \epsilon (\{f_7\}) = \{3, 4, 6, 7, 8, 9\}$$

Analizar S_2 :

$$S_8 = \text{IrA}(S_2, \bullet) = \emptyset - \epsilon (\{f_{13}\}) = \{13, 14\}$$

Analizar S_3 :

$$S_9 = \text{IrA}(S_3, S) = \emptyset - \epsilon (\{f_{20}\})$$

Analizar S_4 :

Vacio

Analizar S_5 :

Vacio

Analizar S_6 :

$$S_{10} = \text{IrA}(S_6, L) = \emptyset - \epsilon (\{f_5\})$$

$$S_{11} = \text{IrA}(S_6, D) = \emptyset - \epsilon (\{f_7\})$$

Analizar S_7 :

$$S_{12} = \text{IrA}(S_7, L) = \emptyset - \epsilon (\{f_5\})$$

$$S_{13} = \text{IrA}(S_7, D) = \emptyset - \epsilon (\{f_7\})$$

Analizar S_8

$$S_9 = \text{IrA}(S_8, D) = \mathcal{L}_E(\{15\}) = \{14, 15, 16, 17\}$$

Analizar S_9

$$S_9 = \text{IrA}(S_9, D) = \mathcal{L}_E(\{15\})$$

FIN

$$S_0 \cap F = \emptyset$$

$$S_1 \cap F = \{9\} \text{ Ident.}$$

$$S_2 \cap F = \{17\} \text{ Núm.}$$

$$S_3 \cap F = \{21\} \text{ Espacios}$$

$$S_4 \cap F = \{27\} \text{ Suma o resta}$$

$$S_5 \cap F = \{27\} \text{ Suma o resta}$$

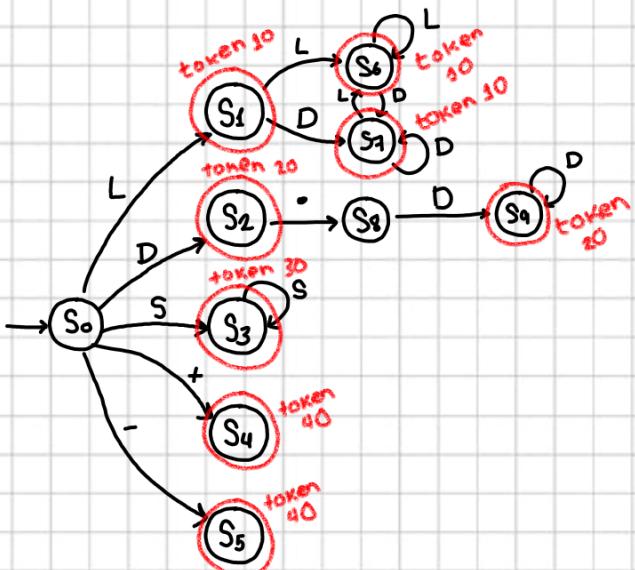
$$S_6 \cap F = \{9\} \text{ Ident.}$$

$$S_7 \cap F = \{9\} \text{ Ident.}$$

$$S_8 \cap F = \emptyset$$

$$S_9 \cap F = \{17\} \text{ Núm.}$$

Con esto obtenemos el AFD



Representación tablas del AFD

	L	D	.	S	+	-	Edo Acept.
0	1	2	-1	3	4	5	-1
1	6	7	-1	-1	-1	-1	10
2	-1	-1	8	-1	-1	-1	20
3	-1	-1	-1	3	-1	-1	30
4	-1	-1	-1	-1	-1	-1	40
5	-1	-1	-1	-1	-1	-1	40
6	6	7	-1	-1	-1	-1	10
7	6	7	-1	-1	-1	-1	10
8	-1	9	-1	-1	-1	-1	-1
9	-1	9	-1	-1	-1	-1	20

Analizar la cadena $\sigma = D \cdot DDD$

Solución: El estado inicial es S_0 , (i.e) es el renglón \emptyset del arreglo

$$S(0, D) = 2$$

$$S(2, \cdot) = 8$$

$$S(8, D) = 9$$

$$S(9, D) = 9$$

$$S(9, \cdot) = 9$$

El estado 9 es de aceptación $token = 20 \equiv \text{Número}$

$\therefore \sigma \in \text{Clase de los números}$

Table 2.4. Algorithm Used by the LEX State-Machine Driver

```

current_state = 0;
previously_seen_accepting_state = none_seen;

if( lookahead character is end-of-input )
    return 0;

while( lookahead character is not end-of-input )
{
    if( there is a transition from the current state on the current lookahead character)
    {
        current_state = that state;
        advance the input;

        if( the current state is an accepting state )
        {
            remember the current position in the input
            and the action associated with the current state;
        }
    }
    else
    {
        if( no accepting state has been seen )
        {
            There's an error:
            Discard the current lexeme and input character.
            Current_state = 0;
        }
        else
        {
            back up the input to the position it was in when it saw the last accepting state
            perform the action associated with that accepting state;
        }
    }
}

```

Problema:

Si $\sigma = D \cdot DD + SSS$

Analizar léxicamente σ

EdoActual = \emptyset ; IndCaracActual = \emptyset
 IndCaracActual = \emptyset
 IndCaracFinal = -1
 PasoXEdoAcept = false

Tabla [EdoActual, CaracActual] = 2

\emptyset D
 ¿ 2 es un edo acept? Sí \therefore IndCaracFinal = \emptyset
 PasoXEdoAcept = true
 IndCaracActual++ ; EdoActual = 2

Tabla [2, .] = 8
 IndCaracActual++ ; EdoActual = 8

Tabla [8, D] = 9
 EdoActual = 9
 PasoXEdoAcept = true
 IndCaracFinal = 2
 Token = 20
 IndCaracActual++

Tabla [9 , D] = 9
EdoActual = 9
Parse X EdoAcept = true
IndCaracFinal = 3
TOKEN = 20
IndCaracActual++

Tabla $[q, +] = -1$
 Pero anteriormente pasó \times edo. acept \therefore

IndCaractActual = IndCaractFinal + 1 ;

Sig. llamada
IndCaractActual = 4
IndCaractInicial = 4
IndCaractFinal = -1
Paso x EdoAcept = false
Token = -1

Tabla $[\emptyset, +] = 4$
Edo. Aceptación : 4 ∴
IndCaract Final = IndCaractActual = 4
Paso X Edo Acept = true
IndCaracActual ++
Token = 40

Implementación del Analizador Léxico

Vamos a requerir:

La clase AFN

Elementos:

- Estados : Conjunto de estados
- Alfabeto : Conjunto de caracteres
- Estado Inicial : Estado
- Estados de Aceptación : Conjunto de estados

Clase Estado:

Identificador : Valor Entero > 0

EdoAcept : Boolean

Token : Valor Entero > 0

Transiciones : Conjunto de transición

Clase Transición:

Símbolo : char

EdoDestino : Estado

Métodos de la Clase AFN

- AFN CrearAFNBasico (char c);
- AFN CrearAFNBasico (char c1, char c2);
- AFN UnirAFN (AFN F2)
- AFN Concatenar (AFN F2)
- AFN CerraduraPos();
- AFN CerraduraKleen();
- AFN Opcional();

Clase AFN

Class AFN

```
{  
    Conjunto <Estado> Estados;  
    Estado EdоГnicial;  
    Conjunto <char> Alfabeto;  
    Conjunto <Estado> EdosAcept;
```

AFN ()

```
Estados = new Conjunto <Estado>();  
Estado.clear();  
EdоГnicial = NULL;  
Alfabeto = new Conjunto <char>;  
Alfabeto.clear();
```

```
EdosAcept = new Conjunto <Estado>();  
EdosAcept.clear();
```

```
}
```

```
AFN CrearAFNBasico (char c)  
{  
    Estado e1, e2;  
    e1 = new Estado();  
    e2 = new Estado();
```

```
This.Estudios.Add (e1);  
This.Estudios.Add (e2);  
This.EdоГnicial = e1;
```

```
e1.Transiciones.Add (new Transicion (c,e2));  
e2.EdosAcept = true;  
This.Alfabeto.Add (c);  
This.EdosAcept.Add (e2);  
return This;
```

```
}
```

AFN CreaAFNBasico (char c1, char c2)

```
{  
    Estado e1, e2;  
    e1 = new Estado();  
    e2 = new Estado();  
  
    This.Estudios.Add(e1);  
    This.Estudios.Add(e2);  
    This.EdoInicial = e1;  
  
    e1.Transiciones.Add(new Transicion(c1, c2, e2));  
    e2.EdosAcept = true;  
    This.EdosAcept.Add(e2);  
    for(int i=c1; i <= c2; i++)  
        This.Alfabeto.Add(c);  
  
    return This;  
}
```

AFN UnirAFN (AFN F2)

```
{  
    Estado e1, e2;  
    e1 = new Estado();  
    e2 = new Estado();  
  
    e1.Transiciones.Add(new Transicion(SimbEsp.EPSILON, This.EdoInicial));  
    e1.Transiciones.Add(new Transicion(SimbEsp.EPSILON, F2.EdoInicial));  
    foreach(Estado e in This.EdosAcept)  
    {  
        e.Transiciones.Add(new Transicion(SimbEsp.EPSILON, e2));  
        e.EdosAcept = false;  
    }  
  
    foreach(Estado e in F2.EdosAcept)  
    {  
        e.Transiciones.Add(new Transicion(SimbEsp.EPSILON, e2));  
        e.EdosAcept = false;  
    }  
    e.EdosAcept = true;  
  
    This.EdoInicial = e1;  
    This.Estudios.Union(F2.Estudios);  
    This.Estudios.Add(e1);  
    This.Estudios.Add(e2);  
    This.EdosAcept.clear();  
    This.EdosAcept.Add(e2);  
    This.Alfabeto.Union(F2.Alfabeto);  
    F2 = null;  
    return This;  
}
```

AFN Concatenar (AFN F2)

```
{  
    foreach (Estado e in This.EdosAcept)  
    {  
        foreach (Transicion t in F2.EdoInicial.Transiciones)  
            e.Transiciones.Add (t);  
        e.EdoAcept = false;  
    }  
}
```

```
This.EdosAcept.clear();  
This.EdosAcept.Union (F2.EdosAcept);  
This.Alfabeto.Union (F2.Alfabeto);  
F2.Estados.Quitar (F2.EdoInicial);  
This.Estados.Union (F2.Estados);  
return This;
```

}

AFN CerraduraPos()

```
{  
    Estado e1, e2;  
    e1 = new Estado();  
    e2 = new Estado();  
  
    foreach (Estado e in This.EdosAcept)  
    {  
        e.Transiciones.Add (new Transicion (SimbEsp.EPSILON, This.EdoInicial));  
        e.Transiciones.Add (new Transicion (SimbEsp.EPSILON, e2));  
        e.EdosAcept = false;  
    }  
}
```

```
e1.Transiciones.Add (new Transicion (SimbEsp.EPSILON, This.EdoInicial));  
This.EdoInicial = e1;  
This.EdosAcept.Clear();  
This.EdosAcept.Add (e2);  
This.Estados.Add (e1);  
This.Estados.Add (e2);
```

}

AFN CerraduraKleen()

```
{  
    Estado e1, e2;  
    e1 = new Estado();  
    e2 = new Estado();  
  
    foreach (Estado e in This.EdosAcept)  
    {  
        e.Transiciones.Add (new Transicion (SimbEsp.EPSILON, This.EdoInicial));  
        e.Transiciones.Add (new Transicion (SimbEsp.EPSILON, e2));  
        e.EdosAcept = false;  
    }  
}
```

```

e1.Transiciones.Add(new Transicion(SimbEsp.EPSILON, This.EdoInicial));
This.EdoInicial = e1;
This.EdosAcept.Clear();
This.EdosAcept.Add(e2);
This.Estados.Add(e1);
This.Estados.Add(e2);
e1.Transiciones.Add(new Transicion(SimbEsp.EPSILON, e2));
return this;
}

```

AFN Opcional()

```

{
Estado e1, e2;
e1 = new Estado();
e2 = new Estado();

foreach(Estado e in This.EdosAcept)
{
    e.Transiciones.Add(new Transicion(SimbEsp.EPSILON, e2));
    e.EdosAcept = false;
}

```

```

e1.Transiciones.Add(new Transicion(SimbEsp.EPSILON, This.EdoInicial));
This.EdoInicial = e1;
This.EdosAcept.Clear();
This.EdosAcept.Add(e2);
This.Estados.Add(e1);
This.Estados.Add(e2);
e1.Transiciones.Add(new Transicion(SimbEsp.EPSILON, e2));
return this;
}

```

Conjunto <Estado> CerraduraEpsilon (Estado e)

```

{
Conjunto <Estado> C = new Conjunto <Estado> ();
Stack <Estado> P = new Stack <Estado> ();
Estado e2;

```

C.clear();

P.clear();

P.Push(e);

while (!P.Empty())

```

{
    e2 = P.pop();
    if (!C.Contiene(e2))
    {
        C.Add(e2);
        foreach(Transicion t in e2.Transiciones)
            if (t.Simbolo == SimbEspecial.EPSILON)

```

```
    p.Push(t.EdoDestino);  
}  
}  
return C;  
}
```

Conjunto <Estado> CerraduraEpsilon (Conjunto <Estado> C)

```
{  
    Conjunto <Estado> R = new Conjunto <Estado> C;  
    R.Clear();  
    foreach (Estado e in C)  
        R.Union(CerraduraEpsilon(e));  
    return R;  
}
```

Conjunto <Estado> Mover (Estado e, char c)

```
{  
    Conjunto <Estado> R = new Conjunto <Estado> ();  
    R.Clear();  
    foreach (Transicion t in e.Transiciones)  
        if (t.Simbolo == c)  
            R.Add(t.EdoDestino);  
    return R;  
}
```

Conjunto <Estado> Mover (Conjunto <Estado> E, char c)

```
{  
    Conjunto <Estado> R = new Conjunto <Estado> ();  
    R.clear();  
    foreach (Estado e in E)  
        foreach (Transicion t in e.Transiciones)  
            if (t.Simbolo == c)  
                R.Add(t.EdoDestino);  
    return R;  
}
```

Conjunto <Estado> IrA (Estado e, char c)

```
{  
    return CerraduraEpsilon(Mover(e, c))  
}
```

Conjunto <Estado> IrA (Conjunto <Estado> E, char c)

```
{  
    return CerraduraEpsilon(Mover(e, c))  
}
```

```
Class Estado
{
    int IdEdo;
    bool EdоАcept;
    int Token;
    Conjunto <Transicion> Transiciones;
    static int NumEstados = 0;
```

```
Estado ()
{
    IdEdo = NumEstados++;
    EdоАcept = false;
    Token = -1;
    Transiciones = new Conjunto <Transicion> ();
    Transiciones. clear();
}
:
```

```
}
```

```
Class Transicion
```

```
{
    char SimboloInf;
    char SimboloSup;
    Estado EdоДestino;
```

```
Transicion ()
{
    EdоДestino = NULL;
}
```

```
Transicion (char c, Estado e)
{
    SimboloInf = SimboloSup = c;
    EdоДestino = e;
}
```

```
Transicion (char c.inf, char c.sup, Estado e)
{
    SimboloInf = c.inf;
    SimboloSup = c.sup;
    EdоДestino = e;
}
```

```
Conjunto <Estado> TieneTransicionCon (char c)
```

```
{
    Conjunto <Estado> R = new Conjunto <EstadoC>;
    R. clear();
    foreach (Transicion t in this.Transiciones)
        if (t.SimboloInf <= c && c <= t.SimboloSup)
            R.Add(t.EdоДestino);
```

Nota: Agregar un elemento a la clase AFN para identificar cada AFN creado.
Se requiere un lugar para guardar los AFN's static Conjunto<AFN> Automatos;

return R;

}

Class ElemSj

{

int Id;

Conjunto<Estado> S;

ELEM Sj()

{

Id = -1;

S = new Conjunto<Estado>();

S.clear();

}

:

}

Class EdoAFD

{

int [257] TransAFD;

int Id;

EdoAFD()

{

TransAFD = new int[257];

Id = -1;

for (int i=0; i<=256; i++)

TransAFD[i] = -1;

}

EdoAFD(int idEdo)

{

:

}

:

}

Class AFD

{

EdoAFD[] EdosAFD;

Conjunto<char> Alfabeto;

int NumEdos;

AFD()

{

NumEdos = 0;

Alfabeto = new Conjunto<char>();

}

AFD(int n)

{

EdosAFD = new EdoAFD[n];

EdoAFD

0 1

255 256

 | |

 | |

0 1

255 256

 | |

 | |

 | |

 | |

 | |

 | |

Afd-J.txt

```

NumEdos = n;
Alfabeto = new Conjunto <char>();
}

AFD (int n, Conjunto <char> Alf)
{
    EdosAFD = new EdoAFD [n];
    NumEdos = n;
    Alfabeto = new Conjunto <char>();
    Alfabeto.clear();
    Alfabeto.Union (Alf);
}

```

```

bool GuardarAFD (String NomArch)
{
    :
}

```

```

bool CargarAFD (string NombArch)
{
    :
}

```

Arch

NumEdos
-1,-1,... -1,10
⋮
-1,-1,-1...,6,...,-1

Método de la clase AFN

```

AFD ConvertirAAFD ()
{
    AFD Afld_Conv = new AFD ();
    Conjunto < ElemenSj > C = new Conjunto < ElemenSj > ();
    Conjunto < ElemenSj > Q = new Queue < ElemenSj > ();
    int NumElemSj = 0;
    int NumEdo;
    ElemenSj SjAux = new ElemenSj ();
    C.clear ();
    Q.clear ();
    ElemenSj SjAct = new ElemenSj ();
}

```

```

SjAux.S = CerraduraEpsilon (this. EdoInicial);
SjAux.Id = NumElemSj++; // So
C.Add (SjAux);
Q.Add (SjAux);

while (!Q.Empty ())
{
    SjAct = Q.DeQueue ();
    foreach (char c in This.Alfabeto)
    {
        SjAux.S = IrA (SjAct.S, c);
        if ((NumEdo = Buscar (C, SjAux)) == -1)

```

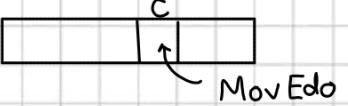
```

    {
        SjAux.Id = NumElemSj++  

        // Transición SjAct
        :
        Q.Add(SjAux);  

        C.Add(SjAux);
    }  

else  

{
    // Poner la transición
    //
    
}
.
}

return Afd.Conv;
}

```

SEGUNDO PARCIAL

Análisis Sintáctico

Para llevar a cabo el análisis sintáctico de una cadena σ se hace uso de gramáticas. Únicamente consideraremos gramáticas libres de contexto.

Recordatorio:

Una gramática G es una cuatupla

$$(V_T, V_N, S, \Phi)$$

donde:

V_T : es un conjunto finito no vacío de símbolos llamados símbolos terminales

V_N : Conjunto finito no vacío de símbolos llamados símbolos no terminales

$S \in V_N$: llamado símbolo inicial de la gramática.

Φ : Conjunto de elementos llamados reglas gramaticales.

$$\Phi : V_N \rightarrow (V_N \cup V_T)^*$$

para las gramáticas libres de contexto.

Una gramática G define un lenguaje \mathcal{L}

donde:

$$\mathcal{L} = \{\sigma \mid S \xrightarrow{*} \sigma \text{ y } \sigma \in V_T^*\}$$

Con esto, una cadena σ pertenece al lenguaje definido por una gramática G si:

$$\sigma \in \mathcal{L}(G)$$