Conceptos generales de los lenguajes de programación

Pedro O. Pérez M., PhD.

Implementación de métodos computacionales
Tecnológico de Monterrey

pperezm@tec.mx

02-2024

Contenido I

1. Introducción a los lenguajes de programación

2. Clasificación de los lenguajes de programación
   - Nivel de abstracción
   - Paradigma
   - Jerarquía de Chomsky-Schützenberger

3. Definición de un lenguaje
   - ¿Cómo se define un lenguaje?
   - ¿Cómo aprende un nuevo lenguaje?
   - Metalenguajes

4 Expresiones regulares
Definición de expresiones regulares
Regex en la práctica

5 Autómatas
Introducción
Implementación práctica de un DFA

6 Analizador léxico
Introducción

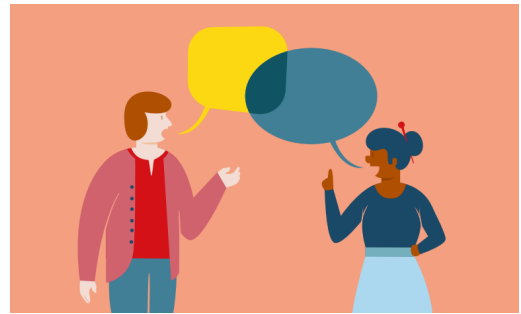
7 Analizador sintáctico
Introducción
Diagramas de sintaxis
Recursión

¿Qué es un lenguaje de programación?

Ejercicio 1

- Coloca tu definición el foro llamado "¿Qué es un lenguaje de programación?" y dale "Me gusta" a los que más te agraden (10 minutos).
- Analizaremos algunas de las propuestas de definición.

- El lenguaje es nuestra herramienta para comunicarnos y necesitamos hablar el mismo “idioma”, no solo en forma verbal o escrita.
- Pero, ¿qué pasa cuando la comunicación no es entre humanos?
- Podemos hablarle a través de una interfaz o directamente en su “idioma”.



- Un lenguaje de programación es una notación para describir los procesos computacionales de manera entendible tanto por la computadora como por el ser humano.
- Un lenguaje de programación es un lenguaje por medio del cual una persona puede expresar el proceso que entenderá y seguirá la computadora para resolver un problema.
- Un lenguaje de programación es un conjunto de reglas, símbolos y convenciones que permiten escribir instrucciones que pueden ser entendidas por una computadora para llevar a cabo una tarea específica.

- Programar es "hablarle" a la computadora por medio de un lenguaje de programación

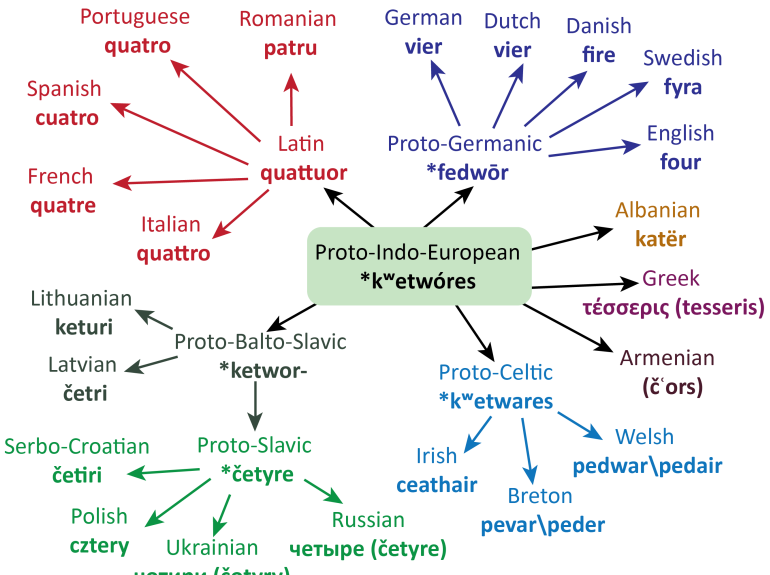
- Estilos de traducción:
  - Interpretación (tiempo real).
  - Compilación (por proyecto).

¿Sabes cuál es la diferencia entre un lenguaje de programación, un programa, un algoritmo y un traductor?

Ejercicio 2

- Coloca tu definición el foro llamado "¿Cuál es la diferencia entre ... ?" y dale "Me gusta" a los que más te agraden (10 minutos)
- Analizaremos algunas de las propuestas de definición.

Clasificación de los lenguajes de programación

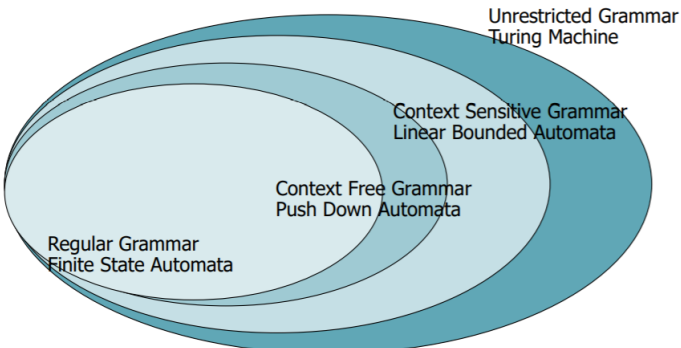


- Los idiomas se clasifican (taxonomías) y están interrelacionados por su origen e historia (originalidad y pureza).

- Los lenguajes de programación son una abstracción que nos permiten hacer abstracciones.
- Entre mayor el nivel de abstracción, más es el trabajo que tiene hacer el traductor:
  - Muy alto nivel (lenguajes declarativos): ¿Qué quiero hacer?
  - Alto nivel (lenguajes imperativos): ¿Cómo lo puedo hacer?
  - Bajo nivel (lenguaje ensamblador).

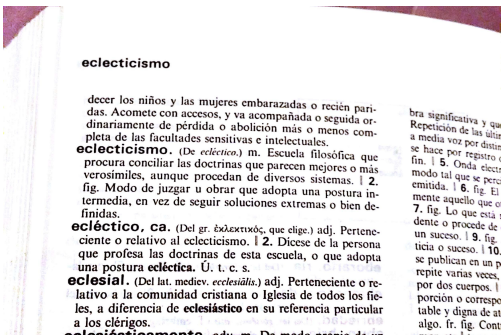
- Paradigma se define como un modelo, patrón o ejemplo que debe seguir en determinada situación.
- Existen diferentes paradigmas de programación:
  - Programación Imperativa.
  - Programación Orientada a Objetos.
  - Programación Funcional.
  - Programación Lógica.
  - Programación Paralela.

- La jerarquía de Chomsky-Schützenberger es la base formal para describir un lenguaje (natural o artificial). "How Complex is Natural Language? The Chomsky Hierarchy".



- Todo lenguaje se define con un conjunto de palabras (**léxico**) y un conjunto de reglas para utilizarlas ordenadamente (**sintaxis**) y congruentemente (**semántica**).

- Vocabulario, palabras válidas y clasificadas.



eclecticismo

decer los niños y las mujeres embarazadas o recién paridas. Acomete con accesos, y va acompañada o seguida ordinariamente de pérdida o abolición más o menos completa de las facultades sensitivas e intelectuales.

eclecticismo. (De *ecléctico.*) m. Escuela filosófica que procura conciliar las doctrinas que parecen mejores o más verosímiles, aunque procedan de diversos sistemas. ǁ **2.** fig. Modo de juzgar u obrar que adopta una postura intermedia, en vez de seguir soluciones extremas o bien definidas.

ecléctico, ca. (Del gr. ἐκλεκτικός, que elige.) adj. Perteneciente o relativo al eclecticismo. ǁ **2.** Dícese de la persona que profesa las doctrinas de esta escuela, o que adopta una postura **ecléctica.** Ú. t. c. s.

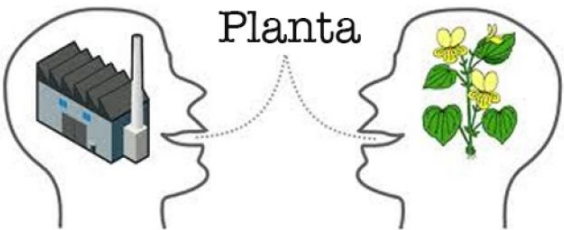
eclesial. (Del lat. mediev. *ecclesiālis.*) adj. Perteneciente o relativo a la comunidad cristiana o Iglesia de todos los fieles, a diferencia de **eclesiástico** en su referencia particular a los clérigos.

eclesiásticamente. adv. m. De modo propio de un

bra significativa y qu
Repetición de las últi
a media voz por distin
se hace por registro
fin. ǁ **5.** Onda elect
modo tal que se perc
emitida. ǁ **6.** fig. El
mente aquello que
7. fig. Lo que está
dente o procede de
un suceso. ǁ **9.** fig.
ticia o suceso. ǁ **10.**
se publican en un p
repite varias veces,
por dos cuerpos. ǁ
porción o correspo
table y digna de at
algo. fr. fig. Cont
mor. etc. ǁ **tener ec**

- Orden en que se usan las palabras, reglas para conformar frases.

- Significado de una frase.



Definición de un lenguaje de programación

- Léxico
  - Palabras reservadas.
  - Identificadores.
  - Valores constantes.
  - Símbolos especiales: operadores, delimitadores.
- Sintaxis
  - Reglas de construcción de estatus e instrucciones.
- Semántica
  - Reglas de congruencia entre tipos de datos, unicidad de nombres, etc.

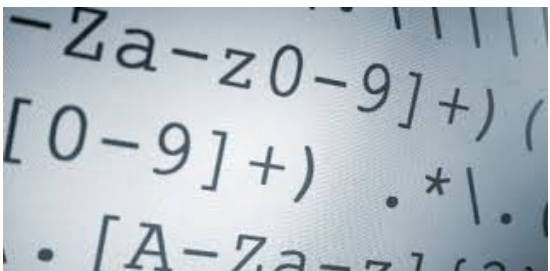
¿Cómo aprende un nuevo lenguaje?

1. Conoce un poco de su historia y características (taxonomía, paradigma).
2. Conoce un poco de su vocabulario (léxico).
3. Conoce las principales reglas que te permitan formar frases válidas con ese vocabularios (sintaxis).
4. Conoce las principales reglas de significado congruente de las frases (semántica).
5. Ponlo en práctica (úsalo) resolviendo problemas que ya conoces y comienza a generar confianza en ti mismo.
6. Repite los puntos anteriores tantas veces lo desees o necesites y cada vez incrementando tu conocimiento y dominio del lenguaje.

- Lenguajes regulares.
  - Expresiones Regulares (regex).
  - Autómatas de Estados Finitos (AFN, AFD).
- Lenguajes libres de contexto.
  - Gramáticas.
  - Diagramas de sintaxis.

Definición de expresiones regulares

- Las expresiones regulares son una forma de representar a los lenguajes regulares.
- Se utilizan para buscar patrones o combinaciones de caracteres de cadenas de texto.



¿Qué es una expresión?

- Una expresión es una secuencia de operandos unidos por operadores para dar un resultado.
- Los operadores pueden ser unarios o binarios.
- Los operadores tienen prioridades para su ejecución.
- Los paréntesis permiten cambiar las prioridades de ejecución.
- Los operandos pueden ser simbólicos (variables) o valores constantes.

- Notación de expresiones que permite describir patrones válidos en secuencias simbólicas.
- Operandos básicos de una expresión regular:
  - Símbolo del alfabeto.
  - ϵ (representan una cadena vacía).
- Operadores básicos de una expresión regular:
  - Concatenación. No tiene símbolo asociado y se representa por la secuencia directa de operandos. Por ejemplo, *abc* describe el patrón de una *a* seguida de una *b* seguida de una *c*; 2759 describe el patrón de 2 seguido de un 7 seguido de un 5 y terminando con un 9.

- Operadores básicos de una expresión regular:
  - Alternativa. El operador binario | se utiliza para representar la posibilidad de elegir uno de los dos operandos en el patrón. Por ejemplo, $a|b$ describe el patrón de una a o una b; $+|-$ describe el patrón con el signo positivo o el signo negativo.
  - Repetición:
    - Cerradura de Kleen (*) se utiliza para representar la posibilidad de repetir **cero o más veces** el operando. Por ejemplo, a^* describe el patrón de cero o más a's concatenadas.
    - Cerradura positiva (+) se utiliza para representar la posibilidad de repetir **uno o más veces** el operando. Por ejemplo, a^+ describe el patrón de uno o más a's concatenadas.

Expresiones regulares compuestas

- Los paréntesis son utilizado en las expresiones regulares para priorizar la ejecución de operadores cuando sea necesario.
- Prioridad:
  - Los operadores de repetición son unitarios, tienen la más alta prioridad y son asociativos por la izquierda.
  - La concatenación es segundo en precedencia y asociativo por la izquierda.
  - $|$ tiene la precedencia más baja y es asociativo por la izquierda.
- Por ejemplo,
  - $(a|b)^*c$ describe el patrón de cero o más a's o b's concatenadas que terminan con una c.
  - $a|b^*c$ describe el patrón de una a o cero o más b's concatenadas que terminan con una c.

Regex	Descripción	Cadenas válida
abc^*	**a** seguida de **b** seguida de cero o más c's.	ab abc abccccccccc
$(a\|b)^+$	Secuencias de, al menos, una **a**'s o **b**'s en cualquier orden.	a b aaaa bb ababaaabbababb
x^*yz^+	Cero o más **x**'s, seguidas de una **y**, seguida de un o más **z**´s.	yz xyz xxxyzzzzzz
$a\|b^*$	Una **a** o cero o más **b**'s.	a b bbbbb ϵ
$z(zz)^*$	Secuencia con un número impar de **z**'s.	z zzz zzzzzzz

Ejemplos de patrones que que identifican elementos del léxico de un lenguaje de programación.

- Números enteros decimales sin signo: $(0|1|2|3|4|5|6|7|8|9)^+$.
- Números enteros decimales con signo: $(+|-|\epsilon)(0|1|2|3|4|5|6|7|8|9)^+$.
- Palabra reservada: **if**.
- Identificador de variable: $(A..Z|a..z)(A..Z|a..z|0..,9|\_)^*$.
- Operador de incremento: $++$.

- Muchas interfaces permiten hacer búsquedas con regex.
- Sirven también para hacer validaciones o testring.
- Muchos lenguajes de alto nivel, proveen librerías para facilitar la programación de búsqueda de patrones con regex.

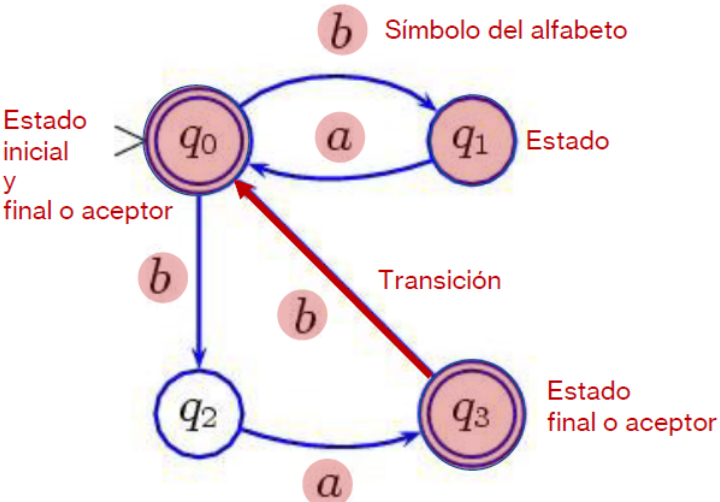
Ejercicio 3

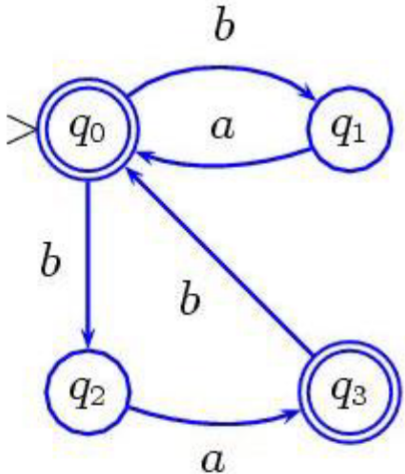
- Busca en Internet información sobre el uso de regex en Python.
- Usando regex en Python, escribe la expresión regular ...
  1. Para verificar que la cadena contiene la matrícula de un alumno del Tec.
  2. Para verificar que una cadena contenga solo un determinado conjunto de caracteres (en este caso, a-z, A-Z y 0-9).
  3. Para verificar que coincida con una cadena que tenga una **a** seguida de tres **b**.
  4. Que coincida con una cadena que tenga una **a** seguida de cualquier cosa que termine en **b**.
  5. Que coincida con una palabra al final de una cadena, con puntuación opcional.
  6. Para buscar números (0-9) de longitud entre 1 y 3 en una cadena determinada.
  7. Que coincida con una palabra que contenga z, no el principio o el final de la palabra.
  8. Para buscar un número al final de una cadena.

- Un autómata es un **modelo matemático formal** que sirve para describir un **proceso** de transformación (computacional) de una **entrada** en una **salida**.
- Se categorizan dependiendo del tipo de problemas que pueden resolver (capacidad o potencia computacional):
  - **Autómatas finitos**.
  - **Autómatas de pila**.
  - **Máquinas de Turing**.
- A su vez, pueden ser **deterministas** (AFD) o **no deterministas** (AFN).

- Se describe con base en los **estados del proceso** y las **transiciones** entre los estados según los **datos de entrada**, hasta llegar a un estado de **aceptación** o **rechazo**.
- Matemáticamente se define con la quíntupla $(Q, \Sigma, \delta, q_0, F)$, tal que Q es un conjunto finito de estados, Σ es un alfabeto de entrada, $q_0 \ \epsilon \ Q$ es el estado inicial, $F \subseteq Q$ es el conjunto de estados finales y δ es la función de transición que proyecta $Q \times \Sigma$. Es decir, $\delta(q, a)$ es un estado para cada estado q y cada símbolo de entrada a.

Diagrama de estados o transiciones





EQUIVALENCIA

entre la definición formal y la definición gráfica

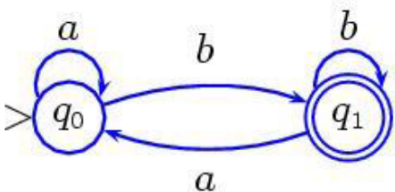
(Q, Σ, δ, q_0, F)

Q = $\{q_0, q_1, q_2, q_3\}$

Σ = $\{a, b\}$

F = $\{q_0, q_3\}$

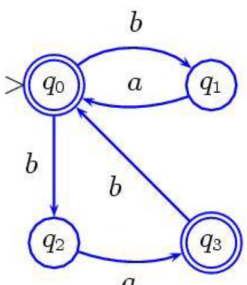
$\delta(q_0, b, q_1)$
$\delta(q_0, b, q_2)$
$\delta(q_1, a, q_0)$
$\delta(q_2, a, q_3)$
$\delta(q_3, b, q_0)$



- ¿Cuáles son las entradas aceptadas por el autómata?
- ¿Cuál es el lenguaje aceptado por el autómata?

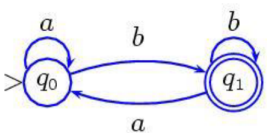
No determinísticos (NFA)

- Pueden existir **varias transiciones** para un mismo estado y símbolo de entrada.
- Pueden existir transiciones con ϵ.
- Fáciles de diseñar.

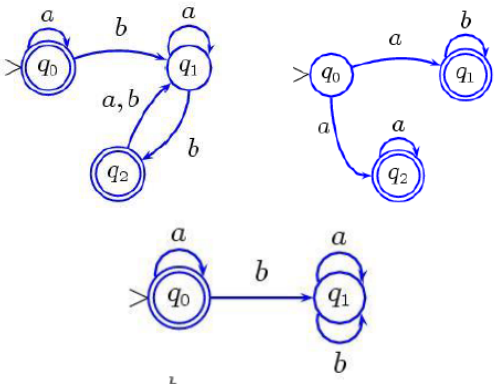


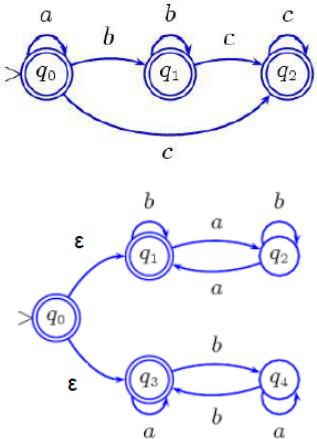
Determinísticos (DFA)

- Para cada estado y símbolos de entrada, existe una **transición única** a otro estado.
- Sin ambigüedades para implementar.



Clasifica los autómatas (DFA, NFA)

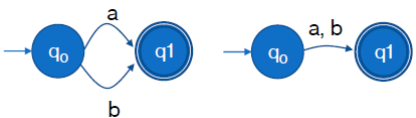




Equivalencia de expresiones regulares con autómatas finitos

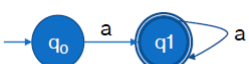
- Concatenación (ab):

- Alternativa ($a|b$):



- Cerradura de Kleen (a^*):

- Cerradura positiva (a^+):



¿Cuál es el autómata finito de cada expresión regular

Regex	Descripción	Cadenas válida
abc^*	**a** seguida de **b** seguida de cero o más c's.	ab abc abccccccccc
$(a\|b)^+$	Secuencias de, al menos, una **a**'s o **b**'s en cualquier orden.	a b aaaa bb ababaaabbababb
x^*yz^+	Cero o más **x**'s, seguidas de una **y**, seguida de un o más **z**´s.	yz xyz xxxyzzzzzz
$a\|b^*$	Una **a** o cero o más **b**'s.	a b bbbbb ϵ
$z(zz)^*$	Secuencia con un número impar de **z**'s.	z zzz zzzzzzz

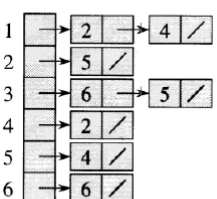
- Dado el siguiente autómata de estados finitos:

$$M = (\{1,2,3,4\}, \{a,b\}, \{\delta(1,a,2), \delta(2,a,1), \delta(2,b,3), \delta(3,b,4), \delta(4,b,3)\}, 1, \{3\})$$

- ¿Es un autómata determinístico (AFD) o no determinístico (AFN)?
- ¿Cuál es la expresión regular?
- ¿Cuál es el diagrama de estados?
- ¿Qué lenguaje describe?

- Bajo la óptica de las **estructura de datos**, un **DFA** es un **grafo**.
- Un grafo se compone de **vértices** y **arcos** que los relacionan. Para un grafo **DFA** los nodos son **estados** y los arcos **transiciones** ponderadas con el símbolo de la transición.

Memoria dinámica
Lista de listas
(de adyacencias o arcos)



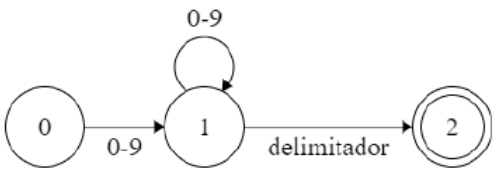
Útil en aplicaciones con altas y bajas de nodos y arcos

Memoria estática
Matriz (de adyacencias o transiciones)

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

Útil en aplicaciones con grafos definidos (constantes)

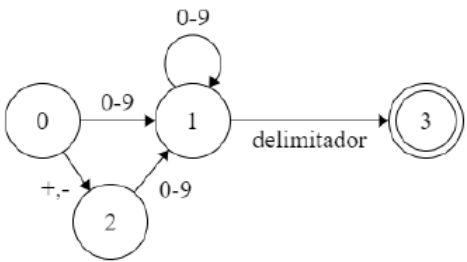
Expresión regular: $[0-9]^+$



Matriz de transiciones

Edo \ simb	0-9	del
0	1	Error
1	1	2

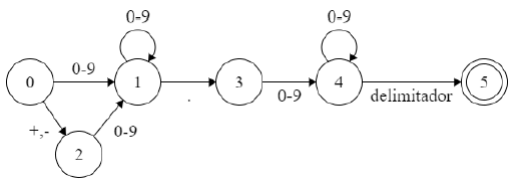
Expresión regular: $(+|-|\epsilon)[0-9]^+$



Matriz de transiciones

Edo \ simb	0-9	+, -	del
0	1	2	Error
1	1	Error	3
2	1	Error	Error

Expresión regular: $(+|-|\epsilon)[0-9]^{+}.)[0-9]^{+}$



Matriz de transiciones

Edo \ simb	0-9	+, -	.	del
0	1	2	Error	Error
1	1	Error	3	Error
2	1	Error	Error	Error
3	4	Error	Error	Error
4	4	Error	Error	5

Implementación de la matriz

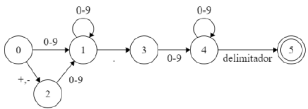
Matriz de transiciones

Edo \ simb	0-9	+, -	.	del
0	1	2	Error	Error
1	1	Error	3	Error
2	1	Error	Error	Error
3	4	Error	Error	Error
4	4	Error	Error	5

```
//Matriz de transiciones
//              dig +,-   .   del  otro
int MT[5][5] = {{1,   2, 200, 200, 200},
                {1, 200,   3, 200, 200},
                {1, 200, 200, 200, 200},
                {4, 200, 200, 200, 200},
                {4, 200, 200, 100, 200}};
```

- La matriz guarda números enteros, por lo que el estado de **ERROR** deberá ser un número designado (por ejemplo, 200). Los estados finales o de aceptación NO tienen renglón en la matriz, pues no tienen transiciones. Se recomienda un identificador con un número clave (por ejemplo, valores mayores a 100 y menores a 200).
- El acceso a las columnas de la matriz, requerirá de una función que convierta los símbolos del alfabeto en número de columna.
- Además, una columna adicional para cualquier otro símbolo ajeno, lo cuál significa que al aparecer un símbolo que no pertenece al alfabeto en la entrada se irá al estado de ERROR.

Algoritmos para procesar entradas con un DFA



- Estado = 0
- Mientras el Estado no sea un estado de aceptación o de error:
  - Leer el siguiente símbolo de la entrada.
  - Estado = MATRIZ[Estado, símbolo]
  - Si el Estado es de aceptación, indicar la aceptación del lexema.
  - Si el Estado es de ERROR, indicar el error que corresponde.

```
char entrada[80];
cout << "Dame la entrada a evaluar: ";
cin >> entrada;

int i = 0;
char c;
int estado = 0;
while (estado < 100)
{ c = entrada[i++];
  estado = MT[estado][filtro(c)];
  if (estado == 100)
    cout << "Entrada aceptada.\n";
  else
    if (estado == 200)
      cout << "ERROR en la entrada.\n";
}
```

Actividad de programación 1

1. Baja el archivo dfa_executor.cpp del Google Drive y pruébalo en tu computadora.
2. Modifica el DFA de las constantes de punto flotante para que también reconozca a las constantes enteras. Utiliza 2 estados de aceptación, uno para cada tipo de constante.
3. Haz los cambios correspondientes en el código del programa para que ahora indique cuándo se reconoce cada tipo de constante.
4. Modifica el programa para que sirva para reconocer entradas, hasta que se dé una entrada nula.

- El **análisis léxico o escaneo del código fuente**, es el primer proceso que realiza un **traductor** para identificar que se estén usando **vocablo válidos** del lenguaje.
- El léxico de un lenguaje de programación se compone de **palabras reservadas**, **identificadores**, **valores constantes numéricos y no númericos**, **símbolos (operadores y delimitadores)**, **comentarios**.
- El léxico de un lenguaje de programación cumple las características de un **lenguaje regular**.
- Las **expresiones regulares** y los **autómatas finitos** nos sirven para describir formalmente el **léxico** de un lenguaje de programación.

Tareas que realiza un analizador léxico

- **Lee** carácter por carácter el texto de entrada, y los agrupa tratando de reconocer elementos válidos del lenguaje (lexemas).
- **Elimina** espacios y líneas en blanco, tabuladores y los comentarios del programa.
- **Asocia** a cada elemento reconocido con una clave llamada **token**:
  - Los símbolos y palabras con significado único y útil para análisis sintáctico posterior, reciben una clave de token particular.
  - Los lexemas variables pero que son del mismo tipo, reciben una clave de token genérico, y se guardan en memoria (tabla de símbolos) para su posterior referencia.
- **Genera** mensajes de error cuando no reconoce elementos válidos.

Ejemplo

```
main()
{ //programa que no hace nada
   int a;
   a = a * 2;
}
```

Lista de Tokens:

PR_main (5)
Paréntesis_abierto (12)
Paréntesis_cerrado (13)
Llave_abierta (18)
PR_int(4)
identificador (20, dirX)
Puntoycoma (33)
identificador (20, dirX)
Op= (67)
identificador (20,dirX)
Op* (58)
Cte_entera (25, dirZ)
Puntoycoma (33)
Llave_cerrada (19)

- Diseñar un **autómata de estados finitos determinísticos** que reconozca a todos los elementos del lenguaje.
- Representar el autómata en memoria a través de una matriz de transiciones.
- Implementar el algoritmo general que ejecuta el proceso de análisis.

- Agregar al autómata de las constantes numéricas la identificación de los identificadores de variables. Los identificadores de variables se forman con una o más vocales mayúsculas. Ejemplos: A, AEIOU, AA, OUI.
- Agregar los operadores aritméticos básicos.

Consideraciones para el diseño del AFD y la implementación del algoritmo para un analizador léxico:

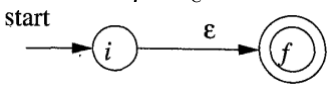
- Minimizar la cantidad de estados intermedio ayuda a que la matriz no sea tan grande y sea más eficiente.
- Evitar reconocer directamente en el autómata a las palabras reservadas. Esto, debido a que cumplen el patrón de los identificadores, el estado de aceptación de este patrón distinguirá a las palabras reservadas a través de una búsqueda en una tabla predefinida de palabras reservadas.

¿Es posible automatizar el proceso de la generación de un analizador léxico?

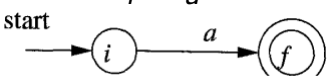
1. Construir, a partir de una expresión regular, un autómata finito no determinístico (NFA).
2. Convertir el NFA en un autómata finito determinístico.
3. Implementar el algoritmo general que ejecuta el proceso de análisis

Construcción de un AFN a partir de una expresión regular

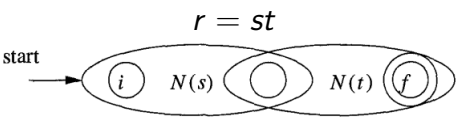
$r = \epsilon$



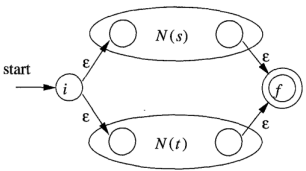
$r = a$



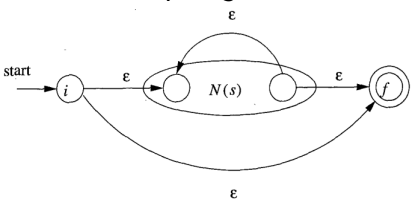
$r = st$



$r = s|t$



$r = s^*$



Construye el AFN de la siguiente expresión:

$$r = (a|b)^* abb$$

OPERATION	DESCRIPTION
ϵ-*closure*(s)	Set of NFA states reachable from NFA state s on ϵ-transitions alone.
ϵ-*closure*(T)	Set of NFA states reachable from some NFA state s in set T on ϵ-transitions alone; $= \cup_{s \text{ in } T}$ ϵ-*closure*(s).
move(T, a)	Set of NFA states to which there is a transition on input symbol a from some state s in T.

Figure 3.31: Operations on NFA states

```
while ( there is an unmarked state T in Dstates ) {
        mark T;
        for ( each input symbol a ) {
                U = ε-closure(move(T, a));
                if ( U is not in Dstates )
                        add U as an unmarked state to Dstates;
                Dtran[T, a] = U;
        }
}
```

Figure 3.32: The subset construction

```
push all states of T onto stack;
initialize ε-closure(T) to T;
while ( stack is not empty ) {
        pop t, the top element, off stack;
        for ( each state u with an edge from t to u labeled ε )
                if ( u is not in ε-closure(T) ) {
                        add u to ε-closure(T);
                        push u onto stack;
                }
}
```

Figure 3.33: Computing ϵ-*closure*(T)

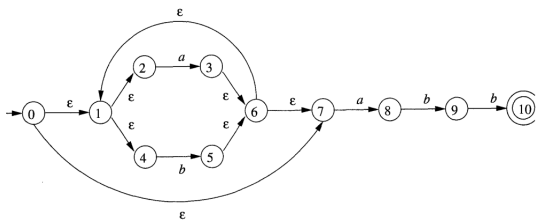
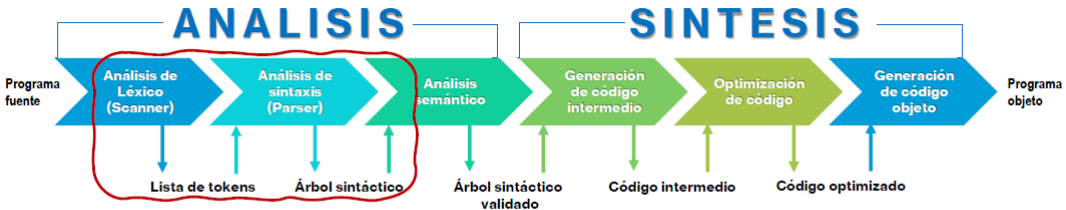


Figure 3.34: NFA N for (**a**|**b**)***abb**

Ejercicio 5

1. Usando el algoritmo visto en clase, construye el autómata fínito no determinístico para la expresión regular: $a^*b(a|b)^*$
2. Usando el algoritmo visto en clase, convierte en NFA a DFA.

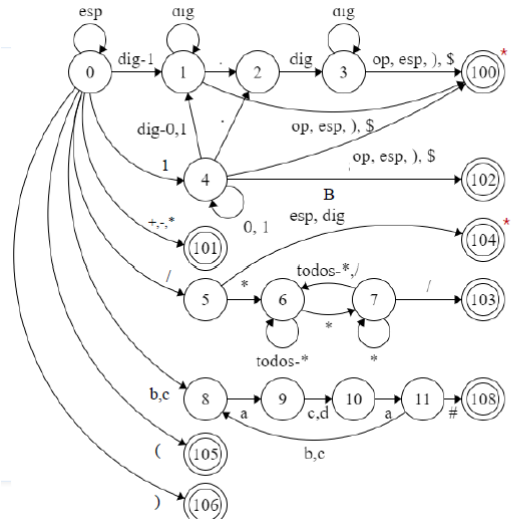
Estructura de un compilador



Descompone el código fuente para verificar el uso correcto del lenguaje

Transforma a los elementos reconocidos del lenguaje en el código objeto

Diseño de un analizador léxico



Constante decimal

Constante binaria

Operadores (101)
Operador / (104)

Comentarios

Identificador abecedario
((b|c)a(c|d)a)+

Paréntesis

Diagramas de sintaxis

- Un diagrama de sintaxis es una herramienta **gráfica** que permite representar el **flujo** (**orden**) en que los lexemas de un lenguaje se combinan para formar **frases válidas** (instrucciones de programación).
- Los diagramas utilizan **flechas** y **modularizan** las estructuras sintácticas, incluyendo **recursividad** cuando es necesario.
- Es posible representar las 3 estructuras procesales básicas: **secuencia**, **alternativa** y **repetición**.

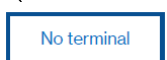
Flujo

Nombre del diagrama (identificador NO TERMINAL)

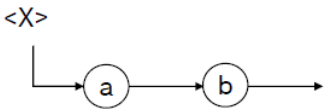
< Nombre >

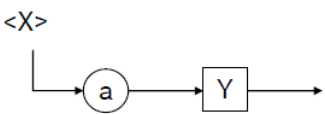
Terminales (o lexemas)

NO TERMINALES (o llamadas a otros diagramas)

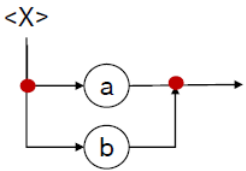


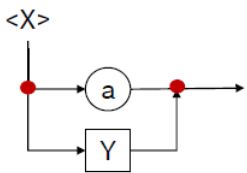
Secuencia



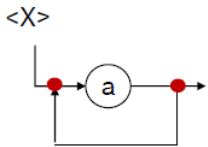


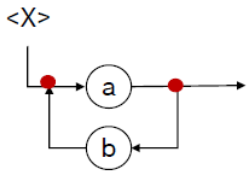
Alternativa



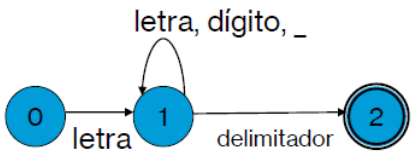


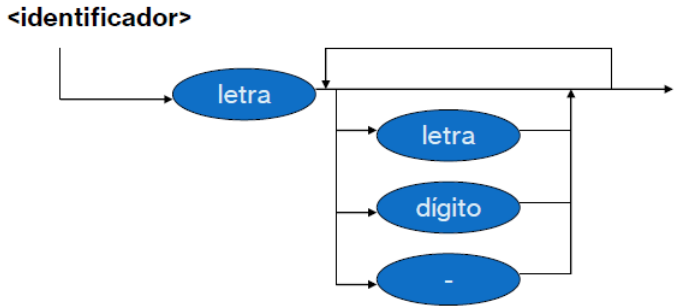
Repetición





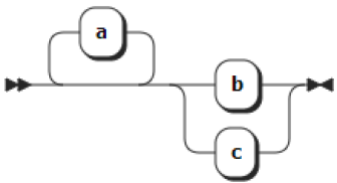
$letter(letter|digit|\_)^*$





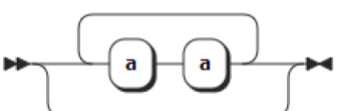
- Lenguaje que acepta cero o más **a**'s que terminan con una **b** o una **c**.

L1:

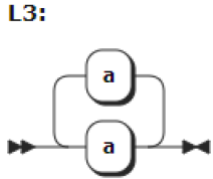


- Lenguaje que acepta un número par de **a**'s.

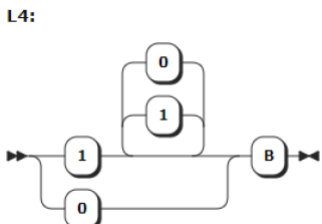
L2:



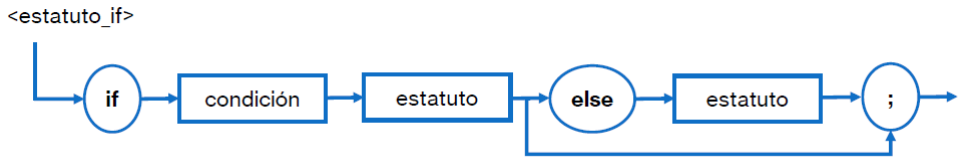
- Lenguaje que acepta un número impar de **a**'s.



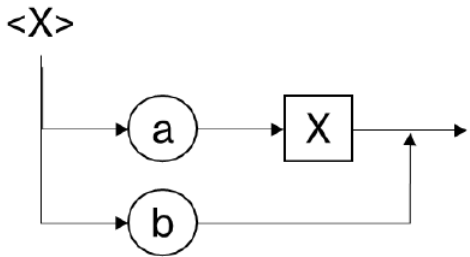
- Lenguaje que reconoce a las constantes binarias: secuencia de **0**'s y **1**'s que terminan con **B** asegurando que no hay ceros a la izquierda.



- El estatuto `if` en C/C++.



¿Qué lenguaje representa el siguiente diagrama de sintaxis?



- Decimos que algo es recursivo cuando se requiere así mismo para definirse.
- La recursividad es una forma de expresar un ciclo.
- Para que la recursividad no sea infinita se requiere:
  - Una definición que se llame a si misma con el mismo caso pero más pequeño.
  - Una definición que representa el caso más pequeño que ya no es recursivo.
- Una estructura recursiva permite controlar lo que hay antes y después de la llamada recursiva.
- Los lenguajes libres de contexto que NO son regulares se definen con **estructuras inherentemente recursivas**.
- La **sintaxis** de la mayoría de los lenguajes de programación tiene las características de un **lenguaje libre de contexto**.

Ejemplo

La expresión $a^n b^n$ para $n > 0$ define el lenguaje de las cadenas que se forman con una secuencia de una o más **a**'s, seguidas por la **misma cantidad** de **b**'s.

- No hay una expresión regular o un autómata finitos que puede representar este lenguaje, dado que **no es un lenguaje regular**.
- Es un ejemplo perfecto de un **lenguaje libre de contexto** con una estructura **inherentemente recursiva**.

- Un valor constante es una expresión.
- Un operador unario aplicado a una expresión es una expresión.
- Una expresión unida con otra expresión por medio de un operador binario es una expresión.
- Una expresión entre paréntesis es una expresión.

12345

-789

123 + 456

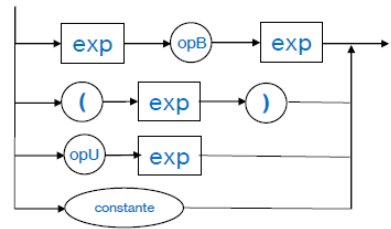
-789 * 456

(123 + 456)

-789 * (123 + 456)

((123 + 456) * (-789 / 123))

<exp>



Una gramática libre de contexto es la herramienta formal para describir a un **lneguaje libre de contexto**. Se define por un cuarteto:

$$G = (T, NT, P, S)$$

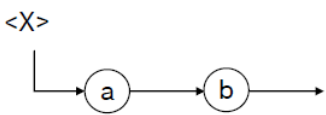
- T es el conjunto de símbolos **terminales (lexemas)**.
- NT es el conjunto de símbolos **NO terminales**.
- P es el conjunto de las **producciones gramaticales** que tiene la forma: $NT \rightarrow$ secuencia de terminales y/n no terminales.
- S es el símbolo No terminal **inicial**.

Por ejemplo,

$$G = (\{a, b\}, \{X\}, \{X \to aXb, X \to \epsilon\}, X)$$

Secuencia

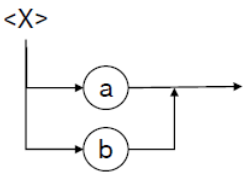
$X \rightarrow ab$



Alternativa

$X \rightarrow a$

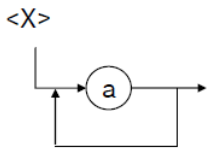
$X \rightarrow b$

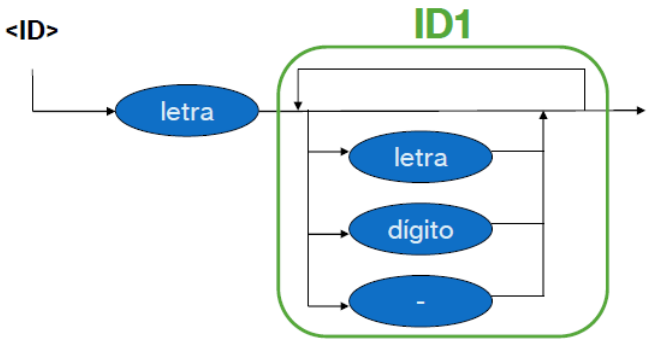


Repetición

$X \rightarrow a$

$X \rightarrow aX$





ID → letra ID1
ID1 → letra ID1
ID1 → dígito ID1
ID1 → _ ID1
ID1 → ε

Notación BNF (Backus-Naur Form)

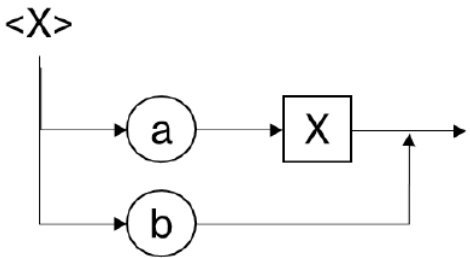
- Es el metalenguaje estándar para definir lenguajes libres de contexto.
- Se define con la siguiente estructura que representa a las producciones gramaticales:

$$\textbf{<símbolo NO terminal>} ::= \textbf{expresión con símbolos}$$

- En donde la expresión con símbolos es una secuencia de símbolos terminales o no terminales que puede utilizar el operador | para representar una alternativa.

$$\textbf{<X>} ::= \textbf{a} \ \textbf{<X>} \ \textbf{b} \mid \epsilon$$

¿Cuál es la gramática BNF equivalente al siguiente diagrama de sintaxis? ¿Cuál es el lenguaje que reconoce?



¿Cuál es la gramática BNF del siguiente diagrama de sintaxis?

Diseña una gramática BNF para los siguientes lenguajes regulares:

- Lenguaje que acepta cero o más **a**'s que terminan con una **b** o con una **c**.
- Lenguaje que acepta un número par de **a**'s.
- Lenguaje que acepta un número impar de **a**'s.
- Lenguaje que reconoce a las constantes binarias, secuencia de **0**'s y **1**'s que terminan con **B** asegurando que no hay ceros a la izquierda.

Ejercicio 6

Diseña una gramática BNF para los siguientes lenguajes regulares:

- Lenguaje que reconoce a las constantes binarias, secuencia de **0**'s y **1**'s que terminan con **B** asegurando que no hay ceros a la izquierda.
- Lenguaje que comience con un número impar de **a**'s, seguidas **de la misma cantidad** impar de **b**'s.
- Obtén la gramática BNF