RESUMEN CAPITULO 1 - TRADUCTORES/COMPILADORES

LENGUAJES DE PROGRAMACION (Comunicación hombre-maquina)
Las computadoras operan sobre bits (unos y ceros) y las personas nos entendemos por medio de idiomas. El vehículo de comunicación entre una pc y un humano son los lenguajes de programacion.

Lo podemos definir a un lenguaje de programación como un conjunto de símbolos junto a un conjunto de reglas para combinar los símbolos de forma que permita al usuario crear programas que serán entendidos por la computadora para realizar alguna tarea.
Los lenguajes de programación los podemos clasificar en lenguajes de bajo nivel y de alto nivel.

LENGUAJES DE BAJO NIVEL: Son totalmente dependientes de la maquina, por lo tanto no se puede migrar a otras maquinas. Cada instrucción corresponde a una acción ejecutable por la PC

Dentro de estos se encuentra:
- Lenguaje de maquina: Las instrucciones se representan por medio de código binario. Cada cifra del código binario corresponde a un pulso eléctrico. Cada operación requiere una determinada combinación de "señales". Por esto el código binario es el lenguaje que la PC entiende.
- Lenguaje ensamblador: Las instrucciones se presentan por medio de palabras en ves de código binario, es decir las instrucciones se presentan en forma simbólica

LENGUAJE DE ALTO NIVEL: se encuentran mas cercano al lenguaje natural humano que al de maquina. Son independientes de la maquina por lo cual se puede migrar de una a otra.
Cada instrucción corresponde a varias acciones ejecutables por la PC.
Estos lenguajes permiten al programador olvidarse del funcionamiento interno del PC. Estos programas necesitan de un traductor.

TRADUCTOR: Los programas fuente escrito en un lenguaje de alto nivel necesita de un proceso de traducción al lenguaje de maquina. A esto se lo llama proceso de traducción y al programa que realiza el proceso traductor

PROGRAMA FUENTE ———-> **TRADUCTOR** ———-> **PROGRAMA OBJETO**
Programa fuente: programa escrito por el programador
Programa objeto: conjunto de instrucciones que entiende el ordenador

Un lenguaje de programación tiene un léxico (símbolos o vocabulario), una sintaxis (reglas) y una semántica (significado).

Existen 2 tipos de traductores: los compiladores y los interpretes

INTEPRETE: Programa informatico que procesa el código fuente de un programa durante su tiempo de ejecución y actúa como una interfaz entre ese programa y el procesador. Procesa el código fuente linea por linea. El proceso de conversion no finaliza hasta que se ha interpretado todo el código, solo se interrumpe si se produce un fallo durante el procesamiento (error), de esta forma se simplifica la resolución de errores.
Python, Ruby o PHP son algunos de los lenguajes de programación que dependen de un interprete para su traducción a código binario.

COMPILADOR: Es un programa informatico que traduce todo el código fuente antes de su ejecución. Algunos lenguajes compilados son C, C++ o Pascal.

depuestos clave que caracterizan intérpretes y compiladores:

	Intérprete	**Compilador**
Momento en que se traduce el código fuente	Durante el tiempo de ejecución del software	Antes de ejecutar el software
Procedimiento de traducción	Línea por línea	Siempre todo el código
Presentación de errores de código	Después de cada línea	En conjunto, después de toda la compilación
Velocidad de traducción	Alta	Baja
Eficiencia de traducción	Baja	Alta
Lenguajes típicos	Perl, PHP, Python, Ruby	C, C++, Pascal, GO

SOLUCION HIBRIDA DE COMPILADOR E INTERPRETE:
Durante el proceso de compilación tiene lugar un paso intermedio, antes de generar la traducción final en código maquina, se convierte el código fuente en un código intermedio (código objeto) para luego se utilizado por un interprete para ser compatible en diversas plataformas. Un ejemplo es JAVA.

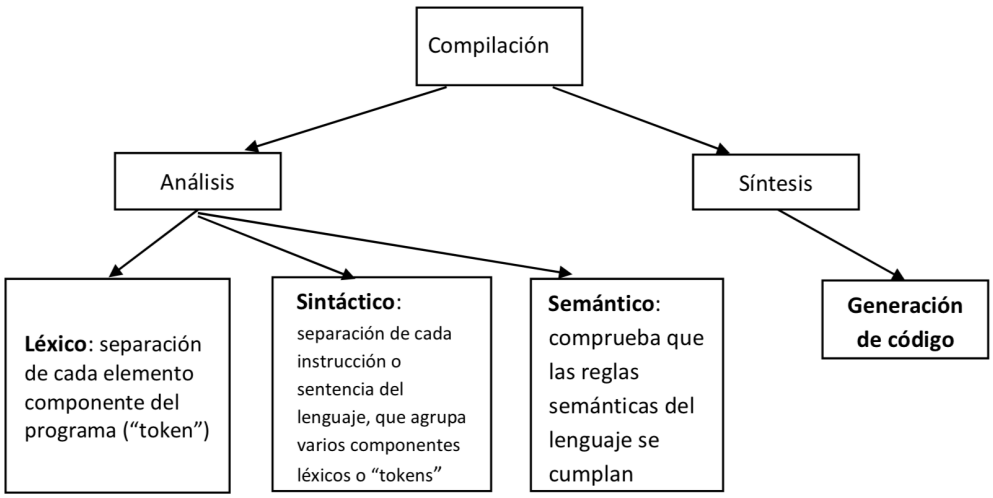
Una vez finalizada la compilación y obtenido el programa objeto, es sometido a otro proceso para llegar al ejecutable. Por lo tanto podemos decir que un compilador no es un programa que funciona de manera aislada, si no que necesita de otros programas, algunos son:

- preprocesador: se ocupa de incluir archivo, eliminar comentarios, y otras tareas similares
- Enlazador: o linker, se encarga de construir el archivo ejecutable
- Depurador o debugger: permite seguir pasar a paso la ejecución de un programa
- Ensamblador: convierte el programa objeto en ejecutable mediante un ensamblador.

CONCEPTOS:
TOKEN: es una categoria a la que pertenece cada cadena o palabra del código fuente. También se llaman categorías léxicas. Los tokens pueden variar de un programa a otro pero en general son palabras reservadas, identificadores, constantes, operadores de asignación, operadores aritméticos, etc.
<categoria lexica o token, valor lexico>
Valor léxico es el dato relaciónado con el token, es decir el valor del token. También se lo llama lexema.

PATRON: es la Telga de generación de caracteres que podría representar a un determinado token. La descripción formal de un patron ser realiza mediante expresiones regulares

ATRIBUTO: caracteristica propia de cada tipo de token



LAS FACES DE UN COMPILADOR:
Tiene dos etapas: una inicial en la que el proceso depende del código fuente y es independiente de PC. Y otra etapa final donde el proceso se hace dependiente de la maquina y indpendiente del lenguaje fuente.

La etapa de análisis o etapa inicial tiene los procesos de Análisis léxico, Análisis sintactico y Análisis semántico.

Todos los errores se informan al final de la etapa de análisis y recién ai se detiene el proceso de compilación.

ANALISIS LEXICO: El AL reconoce para cada cadena del código fuente si concuerda o no con algún token y lo clasifica por su categoría

- **ENTRADA**: cadena de caracteres (código fuente)
- **PROCESO**: se agrupan los caracteres, se separan los token
- **SALIDA**: secuencia de <Tokens, valor> correspondientes

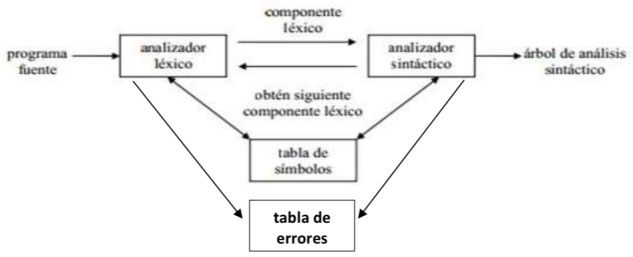
Cuando el AL detecta un identificador o constance almacena los token con sus atributos en la tabla de símbolos.

El aAL opera bajo petición del AS, devolviendo un token cuando el AS lo va necesitando
ERRORES: detecta un error cuando los caracteres que toma el buffer no corresponden a ningún token valido del lenguaje.

ANALISIS SINTACTICO: La escritura del código fuente se rige por un conjunto de reglas gramáticas. Esto se denomina sintaxis del lenguaje y permite determinar matemáticamente si un programa es correcto o no desde la sintaxis.
La tarea del AS es determinar si la estructura sintáctica del del programa es correcta o no.

- **ENTRADA**: tokens producidos por el AL
- **PROCESO**: examina los tokens analizando la categoria lexica del mismo y comprueba que van llegando en el orden especificado por la gramática.
- **SALIDA**: Arbol sintactico

Los dos analizadores (AL y AS) trabajan juntos, de hecho el AL es una subrutina del AS
El accionar de estos analizadores apunta a comprobar si el programa esta bien construido respecto a las reglas del lenguaje.



ADMINISTRACION DE LA TABLA DE SIMBOLOS Y TIPOS

TABLA DE TIPOS: es esencial para que el Ase pueda trabajar. Se inicializan todos los tipos primitivos y se agregan los tipos definidos por el programador.

TABLA DE SIMBOLOS: es una estructura de datos contenedora de l información que el programa procesa. A lo largo de la ejecución de un programa algunos de estos símbolos cambian de valor pero su dirección de memoria es fija.

TABLA DE TIPOS

Código del tipo	nombre	dimensión
0	*char*	*1*
1	*int*	*1*
2	*Ar*	*10*

TABLA DE SIMBOLOS

Código del símb.	nombre	categoria	tipo	dirección
0	*x*	*variable*	*2*	*9000*
1	*car*	*variable*	*0*	*9020*
2	*z*	*variable*	*1*	*9021*

ANALIZADOR SEMANTICO: es la fase mas compleja, comprueba que exista coherencia en el programa. El AS invoca a los procesos del Ase. Para validar el significado de los tokens el Ase recurre a sus atributos.

- **ENTRADA**: árbol jerárquico resultado del AS
- **PROCESO**: comprueba el significado de lo que va leyendo sea valido.

- **SALIDA**: árbol semantico.

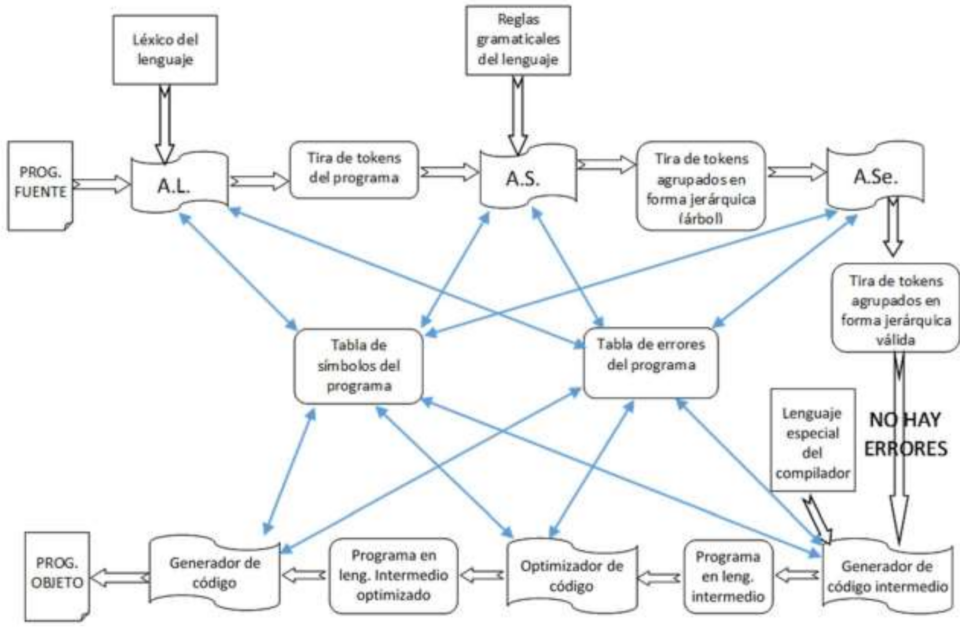
ERRORES: detecta un error cuando una construcción corresponde a una operatoria no valida (aunque fuera sintácticamente correcta).

GENERACION DE CODIGO INTERMEDIO: Después del AS y Ase algunos compiladores generaron un representación intermedia del programa fuente,
Este código intermedio es de facil traducción al lenguaje de maquina al que responde un determinado procesador.

GENERACION DE CODIGO OBJETO: Es la fase final, cada una de las instrucciones se traduce a una secuencia de instrucciones de maquina.

El programa objeto puede ser :

- Lenguaje de maquina absoluto: el programa se coloca en una posición fija de memoria y se genera el .exe para ejecutarlo.
- Lenguaje de maquina redireccionable: las posiciones de memorias no son fija. El .obj se debe someter al proceso de linkedicion para generar el .exe
- Lenguaje simbólico: es util en arquitecturas con poca memoria



RESUMEN CAPITULO 2 - INTRODUCCION A LOS LENGUAJES FORMALES

Lenguaje Natural -> a todo lenguaje hablado o escrito que es utilizado por los humanos para comunicarse entre si. Estos lenguajes tienen características

- Evolucionan con el paso del tiempo agregando nuevos términos y reglas gramaticales
- Cada palabra tiene una semantica (significado)
- Los lenguajes naturales presentan ambigüedades.

Lenguaje Formal -> están conformados por un conjunto de cadenas y estas cadenas están constituidas por símbolos de un alfabeto en particular. A diferencia de un lenguaje natural, los lenguajes formales tienen estas características

- Las cadenas que constituyen el lenguaje no tienen una semántica asociada
- Nunca es ambiguo
- Esta definido por reglas gramaticales prestablecidas

SIMBOLO o CARACTER -> entidad indivisible, los símbolos son letras (a,b,..,z), dígitos (1,2,3….9) y otros caracteres (+,-,/…..)
ALFABETO o VOCABULARIO -> conjunto finito de símbolos. Se definen como una enumeración de los símbolos. Se referencia con la letra V

V1 = {a, b, c, d, ….., z} Alfabeto castellano V2 = {0,1} alfabeto binario

CADENA SOBRE ALFABETO -> toda secuencia finita de símbolos de un determinado alfabeto. Un símbolo puede aparecer varias veces en una cadena. También se lo llama string, palabra. Se referencia con la letra s

s1 = abcsca -> cadena sobre el alfabeto V1
s2 = 01111011 -> cadena sobre el alfabeto V2

LONGITUD DE CADENA -> cantidad de símbolos de s. Se indica |s| ——> |s1| = 6 ; |s1| = 8

CADENA VACIA -> cadena de longitud cero (no pose simbolos). Se referencia con e |e| = 0

PARTES DE UNA CADENA

PREFIJO -> se define como prefijo a aquella cadena q no es ni s ni e.
ejemplo -> s = sintaxis ————> sin

SUFIJO -> se define como aquella cadena que ni es ni s ni e.
ejemplo -> s = sintaxis ————> taxis

SUBCADENA -> toda cadena que se obtiene de eliminar un prefijo y un sufijo sin ser s ni e.
ejemplo -> s = sintaxis ————> inda

SUBSECUENCIA -> cualquier cadena que se forme eliminado 0 o mas símbolos de s
ejemplo -> s = sintaxis ————> nax

Con un alfabeto finito puedo hacer infinitas cadenas.

OPERACIONES ENTRE CADENAS

CONCATENACION DE CADENAS -> Sean x e y dos cadenas sobre un alfabeto V, la concatenación de x e y se representa como x.y a una nueva cadena que se obtiene agregando y a x.

x = vice ; y = director ————> x.y = vicerrector && y.x = directorvice
Por lo tanto x.y != y.x

POTENCIA ENTERA n >= -> estamos en presencia de una definición por recurrencia
S = ja
S^o = e
S^1 = ja ————————- S^2 = S.S = jaja ———————— S^3 = S.S.S = jajaja

LENGUAJE -> conjunto (finito o infinito) de cadenas que se forman con un alfabeto V. Se representa con la letra L.

V = {0,1,2,3,4,5,6,7,8,9}
L1(V) = {n de 2 cifras}
L2(V) = {todos los numeros}
L3(V) = {nos binarios}

LENGUAJE VACIO -> es un conjunto vacío y que se denota con O o { }. No debe confundirse con un lenguaje que conténga una sola cadena y esta sea la cadena vacía.

DESCRIPCION DE UN LENGUAJE -> Los lenguajes formales puede ser descriptos por:

ENUMERACION o EXTENSION -> L1 = {0,1} ; M = {b, bb, bbbb, bbbbbb}

COMPRENSION -> M = {cadenasque contengan entre 1b y 5b inclusive}
N = {Nombres de los colores primarios}

EXPRESIONES REGULARES o GRAMATICAS FORMALES

OPERACIONES SOBRE LENGUAJES:

UNION -> L U M = {s / s esta en en L o esta en M}
CONCATENACION -> L . M = {s / s = r . T con r en L y t en M}

ejemplo R . S = {no, si} . {33, 9a} = {no33, no9a, si33, si9a}

POTENCIA -> $L^0 = \{e\}$ ——— $L^i = L^{i-1} . L$

CERRADURA DE KLEENE -> lenguaje infinito que contiene al propio lenguaje y que contiene a la cadena vacía

$L^* = \{e\} \cup L^1 \cup L^2 \cup \ldots\ldots L^n$
$V^* = \{e\} \cup V^1 \cup V^2 \cup \ldots\ldots V^n$
{e} -> cadenas de longitud 0
V -> cadenas de longitud 1
V . V -> cadena de longitud 2
V . V . V -> cadenas de longitud 3 y así sucesivamente

CERRADURA POSITIVA DE L^+

$L^+ = L^1 \cup L^2 \cup \ldots\ldots L^n$

RESUMEN CAPITULO 4 - GRAMATICAS FORMALES

GRAMATICA DE UN LENGUAJE NATURAL -> En un LN la estructura de las frases se describen por medio de una gramática que agrupa las palabras en categorías sintácticas tales como sujetos, predicados, frases preposicionales etc.

GRAMATICA EN UN LENGUAJE FORMAL -> Un lenguaje formal surge a partir de su gramática y no presenta excepciones en su definición. Esto es así ya que los lenguajes formales se usan para comunicar a los hombres con las maquinas. Por lo tanto a partir de las **gramaticas formales** surgen los lenguajes de programación.
Los objetivos de una gramatica son:

- Describir estructuralmente las cadenas del lenguaje formal
- Definir si una cadena pertenece o no al lenguaje formal

Por lo que podemos decir que las gramáticas formales son descripciones estructurales de las sentencias de los lenguajes.
Con las gramaticas podemos definir cualquier lenguaje formal

- El ALFABETO con el que se construyen sus palabras, es un conjunto de símbolos denominados terminales. Se representa como V_T
- El alfabeto que sera utilizado como auxiliar en el proceso de formación de cadenas (derivacion) pero no forman parte de las cadenas del lenguaje. Se denominan no terminales. Se representan con V_N
- Un simbolo inicial o start, es un símbolo especial que pertenece a los no terminales.
- Un conjunto de producciones que representa como se realiza la transformación desde los símbolos no terminales a las palabras del lenguaje. Una producción es una regla de

definición P, donde cada producción es un par (x, b) tal que su vinculación responde a la forma x -> b donde x y b son cadenas y la -> significa x es sustituida por b.

DEFINICION FORMAL DE GRAMATICA FORMAL:
Es un modelo matematico basado en un conjunto de producciones, que utiliza ademas 2 conjuntos distintos de símbolos (el de los terminales y el de los no terminales), por lo cual se puede representar como una 4-upla

$$G = \{V_T, V_N, P, S\}$$

Una cadena del lenguaje no contendrá símbolos no terminales y el lenguaje generado por una gramática es el conjunto de todas las palabras o cadenas que se pueden obtener a partir del símbolo inicial de la gramática por aplicación de derivaciones. Cada derivación implica la transformación de una cadena en otra, por lo cual podemos decir que la derivación es un proceso que apunta a obtener las palabras del lenguaje.

Se llama forma sentencia de una gramática a una derivación de la misma (cualquier estado intermedio de la formación de las palabras del lenguaje).
Y se llama sentencia o palabra a cualquier cadena formada solo por símbolos terminales.

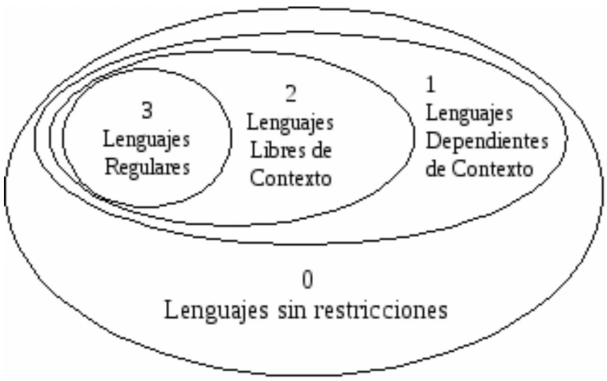
ESTRATEGIAS DE DERIVACION:

- **Derivación por izquierda**: consiste en ir reemplazando sucesivamente el no terminal de mas a la izquierda.
- **Derivación por derecha**: consiste en ir reemplazando sucesivamente el no terminal de mas a la derecha.

CLASIFICACION DE LAS GRAMATICAS:

Un lenguaje L puede ser generado por varias gramáticas pero si una gramática G genera a L no puede generar a L1

Toda gramatica genera un lenguaje único pero distintas pueden generar el mismo lenguaje.
Por lo tanto se clasifican aumentando las restricciones sobre la forma de las producciones.



Tipo	Gramática Formal	Lenguaje Formal
0	Gramática sin Restricciones	Lenguaje Irrestricto
1	Gramática Sensible o Dependiente del Contexto	Lenguaje Dependiente del Contexto
2	Gramática Independiente del Contexto	Lenguaje Independiente del Contexto
3	Gramática Regular	Lenguaje Regular

GRAMATICA TIPO 3 o GRAMATICA REGULAR:

Cada producción es de la forma A -> xB y A -> x (lineal por derecha)

o

A -> Bx y A -> x (lineal por izquierda)

Donde A, B pertenecen a los no terminales y x pertenece a los terminales

La caracteristica de una gramatica de tipo 3 es:

- El miembro izquierdo de las producciones es un solo símbolo no terminal.
- La cadena sustituta, que esta en el miembro derecho de las producciones nunca sera hacia (puede contener la cadena vacia)

Y la restricción es que la cantidad de símbolos que forman la cadena de la izquierda es 1

Los lenguajes generados por gramaticas de tipo 3 se denominan lenguajes regulares. Para la representación de los patrones de los tokens del AL de un compilador es conveniente utilizar expresiones regulares ya que son menos complejas de implementar.

GRAMATICA TIPO 2 o GRAMATICA INDEPENDIENTE DEL CONTEXTO

Cada producción es de la forma A -> y

A pertenece al conjunto de los noterminales

Y y pertenece a la union de los conjuntos terminales y no terminales y no puede ser vacía.

La caracteristica de una gramatica de tipo 2 es:

- el miembro izquierdo de las producciones es un solo símbolo no terminal.
- la cadena sustituta, a la derecha de las producciones nunca estara hacia (puede contener la cadena vacia)

Y la restricción es que la cantidad de símbolos del lado izquierdo siempre es uno.

La mayor parte de los lenguajes de programación pueden describirse mediante gramáticas de este tipo, x lo tanto son lenguajes independientes del contexto. Un ejemplo tipo del uso de una gramática libre de contexto en los lenguajes de programación es la descripción de expresiones aritméticas

ASOCIATIVIDAD DE OPERADORES: los operadores aritméticos como + o - son asociativos por izquierda y los operadores de exponenciacion y asignación por derecha.

GRAMATICA TIPO 0 o GRAMATICA SIN RESTRICCIONES

Es la gramatica formal mas amplia de todas y la mas compleja de implementar. Es de la forma:

$\alpha \rightarrow \beta$ donde:

$\alpha \in (V_T \cup V_N)^+ \Rightarrow$ no habrá producción de la forma $\varepsilon \rightarrow \beta$

$\beta \in (V_T \cup V_N)^*$

GRAMATICA TIPO 1 o GRAMATICA DEPENDIENTE DEL CONTEXTO

En este caso cada producción es de la forma xAy -> xby

Donde A pertenece al conjunto de los no terminales

Y xy pertenecen a la union de los terminales y no terminales. Lo que significa que x e y contienen a cualquier símbolo de la gramática sea terminal o no, incluso la cadena vacía.

La caracteristica de este tipo de gramática es que:

- el miembro izquierdo de las producciones contiene al menos un símbolo no terminal

- la cadena sustituta nunca estara vacía por lo cual es una secuencia de símbolos terminales y/o no terminales.
Y la restricción es que la cantidad de símbolos que forma la cadena del lado izquierdo es menor o igual a la cantidad de símbolos que forma la cadena del lado derecho.

Todo lenguaje formal generado por una gramática dependiente del contexto y que no pueda ser generado por otra de menor jerarquía se denomina Lenguaje dependiente del contexto

RESUMEN:

Gramática	Lenguaje	Reglas de Producción	Solución / Máquina /Autómata
Tipo-0	Irrestricto	Sin restricciones. $\alpha \rightarrow \beta$ donde: $\boldsymbol{\alpha} \in (V_T \cup V_N)^+$ $\boldsymbol{\beta} \in (V_T \cup V_N)^*$	Máquinas de Turing
Tipo-1	Dependiente del contexto	$\alpha A \beta \rightarrow \alpha \gamma \beta$ donde: $\mathbf{A} \in V_N$ $\boldsymbol{\alpha}, \boldsymbol{\beta} \in (V_T \cup V_N)^*$ $\boldsymbol{\gamma} \in (V_T \cup V_N)^+$ y $\lvert \alpha A \beta \rvert <= \lvert \alpha \gamma \beta \rvert$	Autómatas lineales acotados (ALA)
Tipo-2	Independiente del contexto	$A \rightarrow \gamma$ donde: $\mathbf{A} \in V_N$ $\boldsymbol{\gamma} \in (V_T \cup V_N)^+$	Autómatas de pila (Push Down)
Tipo-3	Regular *	$A \rightarrow aB$ $A \rightarrow a$ $S \rightarrow \varepsilon$ Lineales a la Derecha $A \rightarrow Ba$ $A \rightarrow a$ $S \rightarrow \varepsilon$ Lineales a la Izquierda dónde: $\mathbf{A, B, S} \in V_N$ $\quad a, \varepsilon \in V_T$	Autómatas finitos

**hay una equivalencia entre gramáticas de tipo 3 o regular con las expresiones regulares.*

RESUMEN CAPITULO 5 - AUTOMATAS FINITOS

AUTOMATAS FINITOS -> Los AF se pueden representar gráficamente como si fuese una cinta y un control de estados. La cinta contiene símbolos de un determinado vocabulario y se mueve en un solo sentido (nunca vuelve atras). Puede ser infinita. Por lo tanto podemos decir que un AF es una maquina reconocedora ya que, a partir de una cadena solo dice si o no indicando si esta cadena pertenece o no al lenguaje.

Un Automata reconoce un lenguaje L si es capaz de reconocer todas las cadenas pertenecientes a L y de no reconocer ninguna cadena que no pertenece a L. Los lenguajes regulares pueden describirse por medio de un AF

El AF representa una expresión regular y reconoce gramática de tipo 3.
Un AF tiene un conjunto de estados y su control se mueve de estado en estado en respuesta a entradas externas. Estas entradas son las cadenas a ser analizadas.

Un AF tiene 3 tipos de estados, un estado inicial, un estado final o de aceptación y estados intermedios que permiten pasar del estado inicial al final.
Los AF se clasifican dependiendo de si s control es determinista o no. Por lo tanto se clasifican en Autómatas Finitos Deterministicos y Autómatas finitos no deterministicos

1 →Expresión regular **r** que representa el lenguaje $\mathbf{L_{(r)}}$
2 →Método por el cual a partir de una expresión regular dada se puede obtener un **AFN**
3 →Proceso para construir un reconocedor de un lenguaje regular (**AFD).**
4 →Método para construir un **AFD óptimo** (Nota: este método se encuentra fuera del alcance de la materia)

PASOS PARA PASAR DE UNA EXPRESION REGULAR A UN AFDO
AUTOMATAS FINITOS NO DETERMINISTAS (AFN)
Tiene la capacidad de estar en varios estados simultáneamente. Lo cual significa que para cada entrada devuelve un conjunto de cero, uno o mas estados.
DEFINICION FORMAL —> Un AFN esta formado por:

1- Conjunto finito de estados S
2- Un alfabeto de símbolos de entrada V
3- Un estado inicial So con So perteneciente a S
4- Un conjunto de estados finales F
5- Una función mueve que transforma pares estado-símbolo

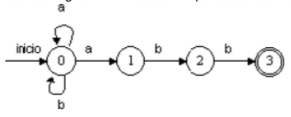
Por lo tanto un AFN = {S, V So, mueve, F}

TABLA DE TRANSICIONES —> Es la manera mas sencilla de implementar esto en una PC es por medio de una matriz en el cual a cada estado le corresponde una fila y a cada símbolo una columna

Símbolos / Estados	a	b
0	{0, 1}	{0}
1	∅	{2}
2	∅	{3}
(3)	∅	∅

GRAFO DE TRANSICIONES —> Un AFN también se puede representar mediante un grafo dirigido, en el que los nodos representan los estados del automata y las aristas representan la función mueve

El siguiente es el grafo de transiciones que reconoce el lenguaje **(a | b) * a b b**



Estados: 0, 1, 2, 3 Estado inicial: 0
Alfabeto: { a, b } Estados finales: { 3 }

ACEPTACION DE UNA CADENA POR PARTE DE UN AFN —> Una cadena de entrada x es aceptada si hay algún camino en el grafo desde el estado de inicio a un estado de aceptación.

AUTOMATAS FINITOS DETERMINISTICOS (AFD) —> Es un caso especial de AFN en el cual:
1- No existen transiciones epsilon
2- Para cada estado S y cada símbolo de entrada a, hay a lo sumo una arista a saliente del estado S

Un AFD siempre esta en un solo estado después de leer cualquier cadena de entrada.
Un AFD es un modelo matemático formado por:

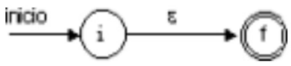
1- Un conjunto finito de estados S
2- Un alfabeto de símbolos de entrada V
3- Un estado So estado inicio
4- Un conjunto de estados de aceptación F
5- Una función de transición mueve

AFD = {S, V, So, mueve, F}

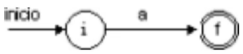
PASO DE UNA EXPRESION REGULAR A UN AFN —> Para el pasaje de una expresión regular a un AFN se utiliza un método llamado método de Thompson. La e.r tiene una estructura en la que aparecen operándoos (epsilon y simbolos) y operadores (* |) por lo tanto se construye un AFN para cada uno de los simbolos básicos de la expresión regular r.

La construcción de los AFN tiene las siguientes reglas:

1- Para cada ocurrencia de E en r, construir el AFN que reconoce el lenguaje {E}

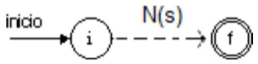


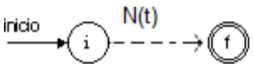
2- Si a en V, x cada ocurrencia de a en r construir el AFN que reconoce {a}

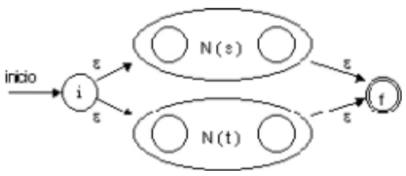


3- Si N(s) y N(t) son los AFN de las e.r s y t entonces:

3.a- Para la e.r s|t construir el AFN

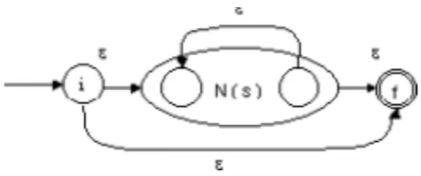






3.b- Para la e.r s.t construir el AFN

3.c- Para la e.r s*



CONVERTIR UN AFN EN UN AFD —> el objetivo de esta conversion es lograr construir un AFD que reconozca el mismo lenguaje que el AFN, para esto usamos el método de los subconjuntos

AUTOMATA LINEALMENTE ACOTADO (ALA) —> Es similar a una maquina de turing. La diferencia con el MT es que en los ALA las cintas no son infinitas. Se utilizan para lenguajes generados por gramática de tipo 1 o GDC

ALA: {E, A, B, &, q0, F, #, S}

E —-> Espacio finito de la cadena

A —-> Alfabeto de entrada

B —-> Alfabeto de la cinta

& —-> Funcion de transición

Q0 —> estado inicial

F ——> conjunto de estados finales

——> Simbolo de inicio de cinta

$ ——-> Simbolo de fin de cinta

AUTOMATA PUSH DOWN —> Es una colección de 8 cosas

1- Un alfabeto de letras de entrada
2- Una cinta de entrada
3- Un alfabeto de caracteres del stack
4- Un stack push down
5- Un estado start
6- Estados de detención (Accept y Reject)
7- Un numero finito de estados PUSH
8- Un numero finito de estados con ramificación de dos clases.