RESUMEN SSL

## Introducción

### SINTAXIS

* Disciplina lingüística que estudia el orden y la relación de las palabras o sintagmas en la oración, así como las funciones que cumplen.
* Modo de combinarse y ordenarse las palabras y las expresiones dentro del discurso

### SEMÁNTICA

Parte de la lingüística que estudia el significado de las expresiones lingüísticas.

Mientras la sintaxis de un lenguaje se refiere a la estructura de las oraciones, la semántica se asocia al significado de las expresiones.

### LÉXICO

Conjunto de las palabras de una lengua.

## Lenguajes

CARÁCTER: Entidad básica para constituir un lenguaje. Es fundamental, indivisible, puede ser múltiple.

ALFABETO Σ: Conjunto finito de caracteres. Con sus caracteres constituyen un lenguaje formal.

CADENA: Secuencia finita de caracteres tomados de cierto alfabeto. Se construye concatenando caracteres.

* Longitud de una cadena: Cantidad de caracteres que la componen. Se representa |C|
* Cadena vacía: No tiene caracteres. Se representa con ε (épsilon) o λ (lambda)
* Potenciación de una cadena: Simplificación de un carácter que se repite u concatena una o más veces.
* Concatenación de una cadena: Operación que, tomando dos cadenas, genera otra cadena formada por los caracteres de la primera, seguido de los caracteres de la segunda. Se representa S1.S2. La concatenación NO es conmutativa.

PALABRA: Es aquella cadena que pertenece a un lenguaje. Es palabra de ese lenguaje. La concatenación de dos palabras no siempre es una palabra del lenguaje.

* Longitud de una palabra: Cantidad de caracteres que la componen. Se representa |abc|
* Palabra vacía: No tiene caracteres. Se representa con ε (épsilon) o λ (lambda)
* Potenciación de una palabra: Simplificación de un carácter que se repite u concatena una o más veces.
* Concatenación de una palabra: Operación que, tomando dos palabras, genera otra palabra formada por los caracteres de la primera, seguido de los caracteres de la segunda. La concatenación puede producir una cadena que no sea palabra del lenguaje.

LENGUAJE NATURAL: Lenguaje hablado y/o escrito que es utilizado para comunicarse.

* Evolucionan con el tiempo
* Las reglas gramaticales surgen después del desarrollo del lenguaje
* El significado (semántica) es más importante que como se escribe (sintaxis)
* Son ambiguos (dependen de la interpretación)

### LENGUAJE FORMAL

Conjunto de cadenas formadas con caracteres de un alfabeto dado.

* Tienen únicamente sintaxis dada (no tienen semántica asociada)
* Nunca es ambiguo
* Las reglas gramáticas están preestablecidas
* No evolucionan
* No son ambiguos.
* Cardinalidad: cantidad de palabras que lo componen
* Si un Lenguaje Finito es finito, siempre será un Lenguaje Regular
* Puede ser descrito por:
* Extensión: L:{101,1001,10001,100001}
* Comprensión: L = {10n1 / 1 < n < 4}
* Mediante una frase en un lenguaje natural: “L es un lenguaje de cuatro palabras…”
* Etc

SUBLENGUAJE: es un subconjunto de un Lenguaje formal

* Sublenguaje vacío: Sublenguaje de cualquier Lenguaje formal. Se representa con Ø y tiene cardinalidad 0

LENGUAJE FORMAL INFINITO: Lenguaje con cantidad infinita de palabras. Cada palabra debe tener longitud finita.

### LENGUAJE REGULAR

* Es un lenguaje regular:
  + Si existe una ER que lo represente.
  + Si existe una GR que lo genere.
  + Si puede ser reconocido por un AF
  + Si es finito
* **Es cerrado bajo la unión:** La unión de dos LR, es otro LR
* **Es cerrado bajo la concatenación:** La concatenación de dos LR, es otro LR
* **Es cerrado bajo las dos clausuras:** La clausura de kleene y la clausura positiva es un LR
* **Es cerrado bajo el complemento del Lenguaje Universal:** el complemento de un LR con respecto al LU es un LR
* **Es cerrado bajo la intersección:** La intersección de dos LR es un LR.

### LENGUAJE UNIVERSAL Σ\*

Lenguaje Formal Infinito que contiene todas las palabras que se pueden formar con los caracteres del alfabeto más la palabra vacía. El lenguaje universal es cerrado bajo concatenación.

### CLAUSURA DE KLEENE

Operador unario que se representa con un asterisco o estrella \*.

* Si L es un LR, su clausura de Kleene es un LR infinito salvo en la excepción en donde el LR contenga únicamente a la palabra vacía.

CLAUSURA POSITIVA: Operador unario que se representa con un “+”.

### LENGUAJE INDEPENDIENTE DE CONTEXTO(LIC)

Es un Lenguaje Formal que es generado por una GIC.

## Gramáticas Formales y Jerarquía de Chomsky

### GRAMÁTICA FORMAL (GF)

Conjunto de producciones (reglas de reescritura) que se aplican para obtener las palabras del Lenguaje Formal. Una GF genera todas las palabras de un LF pero a su vez no genera cadenas que estén fuera de ese LF. Dos gramáticas son equivalentes si generan el mismo lenguaje formal.

Se representa: (VN1, VT1, P, S)

* VN: Vocabulario de noterminales. Conjunto finito de “productores”
* VT: Vocabulario de terminales. Conjunto finito de caracteres del alfabeto sobre el cual se construyen las palabras del LF que es generado por la gramática descripta.
* P: Conjunto finitos de producciones.
* S: Noterminal llamado axioma. A partir de este, se deben comenzar a aplicarse las producciones que generan las palabras de un LF.

### PRODUCCIÓN

Toda producción está formada por tres partes:

* Lado Izquierdo: Símbolos Productores
* Lado Derecho: Símbolos que forman las palabras (caracteres)
* Medio: Operador de producción Metasímbolos (“→” o “|”)
* Ejemplo:

S → ab  
S → ac

* Producción Epsilon: Producción que se reemplaza por nada. Se escribe: S → ε

### JERARQUÍA DE CHOMSKY

Establece una clasificación de cuatro tipo

* Gramáticas Regulares o de Tipo 3
* Gramáticas Independientes del Contexto o de Tipo 2
* Gramáticas Sensibles al Contexto o de Tipo 1
* Gramáticas Irrestrictas o de Tipo 0

### GRAMÁTICA REGULAR (GR)

Sus producciones tienen las siguientes restricciones:

* El lado izquierdo debe tener un solo noterminal
* El lado derecho debe estar formado por:
  + Un solo terminal
  + Un terminal seguido por un noterminal
  + ε (épsilon)
* Ejemplo:   
  S → a | bN  
  N → c

### GRAMÁTICA REGULAR INFINITA

Utilizan producciones recursivas.

* Ejemplo:   
  T → aT

### GRAMÁTICA QUASI-REGULAR (GQR)

Similares a una GR en donde un conjunto de terminales es reemplazado por un no terminal en una o varias producciones. Es una abreviatura de una GR y siempre puede ser re-escrita en una GR.

* Ejemplo:   
  S → N | NS  
  N → a | b | c

### GRAMÁTICA INDEPENDIENTE DEL CONTEXTO (GIC)

No tienen restricciones respecto a la forma del lado derecho de sus producciones, aunque sí requiere que el lado izquierdo de cada producción siga siendo un único no terminal. Toda GR puede ser escrita como una GIC pero no a la inversa.

* Ejemplos:  
  S → ba  
  G → ZXZ  
  E → aGb

GRAMÁTICA INDEPENDIENTE DE CONTEXTO INFINITA: Utilizan producciones recursivas.

* Ejemplo:   
  T → aTb | ac

### GRAMÁTICA IRRESTRICTA

Son las GF más amplias. Tanto en la izquierda como en la derecha pueden haber terminales como no terminales. Sus producciones tienen la forma α → β

* Ejemplo:  
  aD → aEC

### GRAMÁTICA SENSIBLE AL CONTEXTO

Es una gramática irrestricta con la siguiente restricción en las longitudes: |α| ≤ |β|

* Ejemplo:  
  aD → aEC SI
* aDb → a NO

### PROCESO DE DERIVACIÓN

Proceso que permite obtener cada una de las palabras de un LF a partir del axioma de una GF. También permite descubrir si una cadena no es palabra de un lenguaje. Existen 3 tipos de derivación:

* Horizontal
* Vertical
* En forma de árbol

DERIVACIÓN VERTICAL:

* Derivación vertical izquierda
* Derivación vertical derecha

Si una palabra se puede obtener mediante una derivación vertical a izquierda, entonces se puede obtener también mediante una derivación vertical a derecha.

Ejemplo:

|  |  |
| --- | --- |
| Derivación Vertical | Comentario |
| S | Aplico la primera producción |
| aSb | Aplico nuevamente la primera producción |
| aaSbb | Aplico la segunda producción |
| aaabbb | Resultado |

## Expresiones Regulares

### EXPRESIÓN REGULAR

Es una expresión que describe un conjunto de cadenas. Aquellas cadenas son palabras del LR que la ER representa.

* Un ER representa un único LR
* Dos ER son equivalentes si representan al mismo LR.
* ø es una ER que representa al LR vacío (sin palabras).
* Se obtiene mediante la utilización de:
  + Los caracteres del alfabeto
  + ε
  + El operador concatenación “.”
  + El operador union “+” o “|”
* *No todos los LRs finitos se representan mejor con ERs.*

### OPERADORES

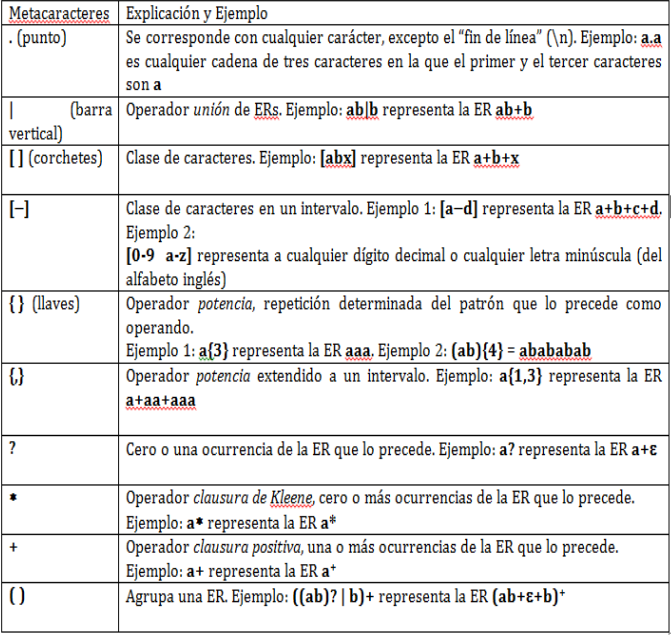
* Unión: “+” o “|”. Se utiliza para indicar una unión. Es conmutativa (prioridad baja)
* Concatenación: “.”. Se utiliza para indicar una concatenación. No es conmutativa (prioridad intermedia)
* Potenciación: Se utiliza para simplificar la lectura/escritura de una ER (prioridad máxima)
* Clausura de Kleene: “\*”. Se utiliza para representar repeticiones indeterminadas. Representa la palabra vacía y todas las palabras que resulten de la concatenación. (prioridad máxima)
  + aa\* = a+
  + a\*a\*= a\*
  + (R\*)\* = R\*
  + (R\* + S\*)\* = (R\*S\*)\* = (R + S)\*
* Clausura Positiva: “+”Se utiliza para representar repeticiones indeterminadas. Representa todas las palabras que resulten de la concatenación (prioridad máxima)

### EXPRESIÓN REGULAR UNIVERSAL

Es aquella que representa al Lenguaje Universal

### METALENGUAJE

Es un lenguaje que se utiliza para describir otro lenguaje.



## Autómatas Finitos

### AUTÓMATA FINITO

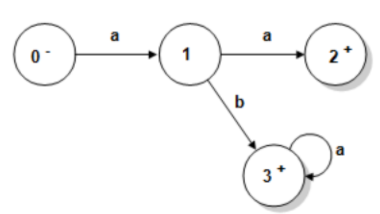
Es un mecanismo que reconoce a un determinado LR. Es decir, acepta cada cadena que pertenece al lenguaje y rechaza toda cadena que no pertenece al lenguaje.

### AUTÓMATA FINITO DETERMINÍSTICO (AFD)

* Cuando un carácter es leído por el autómata, éste transita desde un estado de partida a un estado de llegada preciso.
* Un AFD es una colección de tres conjuntos:
* Conjunto finito de estados. (único estado inicial y uno o más estados finales)
* Alfabeto de caracteres
* Conjunto finito de transiciones
* Se representa formalmente como una 5-upla (Q,Σ ,T, q0, F)
  + Q: Conjunto finito no vacío de estados
  + Σ: alfabeto de caracteres reconocidos por el autómata
  + q0: Estado inicial. q0 ϵ Q
  + F: Conjunto finito no vacío de estados finales. F⊆Q
  + T: Conjunto de transiciones
* Dos AFD son equivalentes si reconocen el mismo LR

Ejemplo:

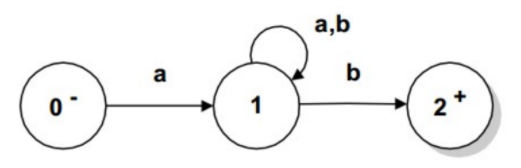
* + Q: {0,1,2,3};
  + Σ: {a,b};
  + q0: 0;
  + F: {2,3};
  + T: {0 => a => 1, 1 => a => 2, 1 => b => 3, 3 => a => 3}



### AUTÓMATA FINITO NO DETERMINÍSTICO (AFN)

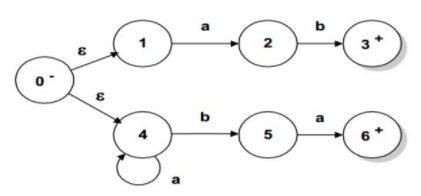
* Es un AFN cuando existe una o más transiciones para cada estado.
* Se representa formalmente como una 5-upla (Q,Σ ,T, q0, F)
  + Q: Conjunto finito no vacío de estados
  + Σ: alfabeto de caracteres reconocidos por el autómata
  + q0: Estado inicial. q0 ϵ Q
  + F: Conjunto finito no vacío de estados finales. F⊆Q
  + T: Conjunto de transiciones

Ejemplo:



### AFN CON TRANSICIONES ε

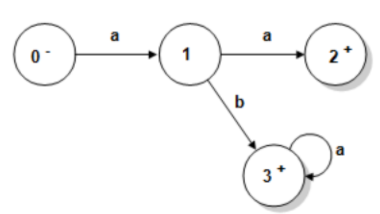
AFN que contiene transiciones ε (cambio de estado repentino sin que intervenga ningún carácter)



### DIAGRAMA DE TRANSICIONES (DT)

Permite visualizar con mayor claridad el funcionamiento del AF.

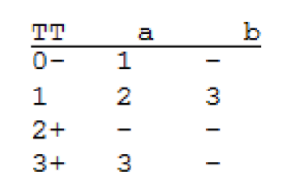
* Sus transiciones están etiquetadas con símbolos del alfabeto
* Sus estados estan representados habitualmente con un número entero no negativo o una letra.
* El estado inicial contiene un supraíndice “-”
* Los estados finales contiene un supraíndice “+”



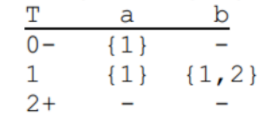
### TABLA DE TRANSICIÓN (TT)

Tabla que representa las transiciones de un AFD.

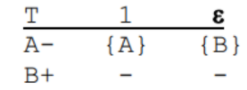
* Tabla de AFD



* Tabla de AFN

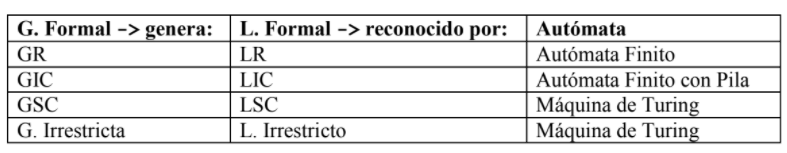


* Tabla de AFN con ε debe tener una columna extra con dicho símbolo



### AFD COMPLETO

* Un AF es completo cuando cada estado tiene exactamente una transición por cada carácter del alfabeto (su tabla no tiene “huecos”)
* Para completarlo, se deben seguir los siguientes pasos:
  + Agregar un nuevo estado llamado “Estado de Rechazo” o “Estado de no aceptación”
  + Reemplazar cada “hueco” con una transición al nuevo estado
  + Se incorpora una nueva entrada para el estado de rechazo



### AUTÓMATA FINITO CON PILA (AFP)

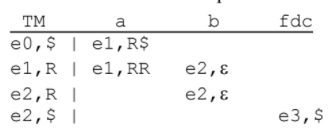
Reconoce los LR y además reconocen los LIC. Un AF solo tiene control sobre los caracteres que forman la cadena a analizar, mientras que un AFP controla los caracteres a analizar y la pila.

Un AFP está constituido por:

* Un flujo de entrada, infinito en una dirección, que contiene la secuencia de caracteres que debe analizar
* Un control finito formado por estados y transiciones etiquetadas
* Una pila abstracta
* Se representa formalmente como una 7-upla M = (E, A, A’, T, e0, p0, F)
  + E es un conjunto finito de estados
  + A es el alfabeto de entrada, cuyos caracteres se utilizan para formar la cadena a analizar
  + A' es el alfabeto de la pila
  + e0 es el estado inicial
  + p0 es el símbolo inicial de la pila, el que indica que la pila no tiene símbolos
  + F es el conjunto de estados finales
  + T es la función T: E x (A ∪ {ε}) x A' -> conjuntos finitos de E x A'\*
* Un AFP puede reconocer un LIC de dos maneras:
  + Por estado final, como en los AFs;
  + Por pila vacía

### AUTÓMATA FINITO CON PILA DETERMINÍSTICO (AFPD) FALTA

TABLA DE MOVIMIENTOS (TM): s



## ER a AF

### ALGORITMO DE THOMPSON

Al aplicar el algoritmo, se parte de una ER y se termina con un AFN con transiciones ε.

Pasos a seguir:

* Desmembrar la ER en componentes básicos
  + Caracteres
  + Operadores
  + ε
* Generar un AF básico por cada carácter o símbolo ε que forme parte de la ER
* Componer esos autómatas básicos según los operadores existentes en la ER hasta lograr el AF
* El resultado es un AFN con transiciones ε

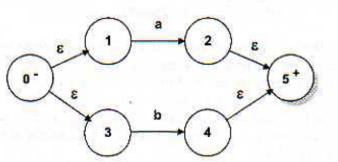
### Características generales de un AFN por Thompson:

* Al estado inicial no llegan transiciones
* El estado final es único
* Del estado final no parten transiciones
* De cualquier estado no final pueden partir:
  + Una sola transición etiquetada con un caracter del alfabeto
  + Una sola transición-ε
  + Dos transiciones-ε

#### AUTÓMATA PARA LA UNIÓN

* Para realizar la unión se debe arrancar con transiciones ε.
* El algoritmo de Thompson exige que el autómata construido tenga un único estado final por lo que se debe terminar con una transición ε.

Ej: a+ b

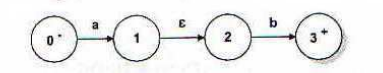


Nota: Este algoritmo trabaja solamente con los tres operadores oficiales (unión, concatenación y clausura de Kleene)

#### AUTÓMATA PARA LA CONCATENACIÓN

* Para realizar la concatenación se debe agregar una transición ε entre cada carácter a concatenar.

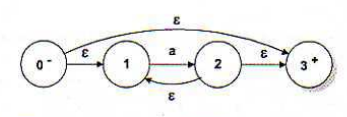
Ej: ab



#### AUTÓMATA PARA LA CLAUSURA DE KLEENE

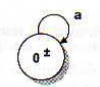
* Se construye el autómata básico
* Se agrega una transición ε al principio del autómata
* Se agrega una transició ε al final del autómata
* Se agrega una transición ε desde el estado inicial hasta el final
* Se agrega una transición ε desde el ex estado final al ex estado final (2 a 1)

Ej: a\*

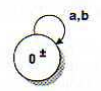


### MECANISMO MANUAL

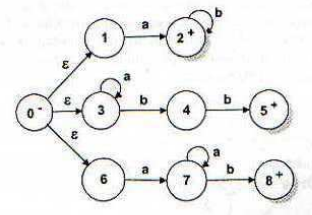
* a\*



* (a+b)\*



* ab\*+a\*bb+a+b



## AFN a AFD

Es un AFN si:

* Tiene transiciones ε
* Hay algún estado del que parten dos o más transiciones etiquetadas con un mismo carácter del alfabeto
* Ambas cosas

Para todo AFN, existe un AFD equivalente que reconoce el mismo lenguaje.

### CLAUSURA-ε DE UN ESTADO

Es el conjunto de estados formado por **q** y todos aquellos estados a los cuales se llega desde **q**, utilizando solo transiciones ε. El conjunto Clausura-ε nunca puede estar vacío porque contiene como mínimo a su propio estado

Mirando la Tabla de transiciones:

* Para los estados que no tienen transiciones ε.
  + El conjunto Clausura-ε está compuesto por dicho estado
* Para los estados que tienen transiciones ε el conjunto está compuesto de:
  + Dicho estado
  + Los estados a los que tiene transiciones
  + Si dichos estados también tienen transiciones hay que agregarlos

Ejemplo:



Clausura-ε (0) = {0,3,4,6}

Clausura-ε (1) = {1,2,4}

Clausura-ε (2) = {2,4}

Clausura-ε (3) = {3,4}

Clausura-ε (4) = {4}

Clausura-ε (5) = {5}

Clausura-ε (6) = {6}

### CLAUSURA-ε DE UN CONJUNTO DE ESTADOS

Sea R un conjunto de estados, la clausura-ε(R) es la unión de las clausuras-ε de los estados que componen el conjunto R.

Ejemplo:

R = {1,3,6}

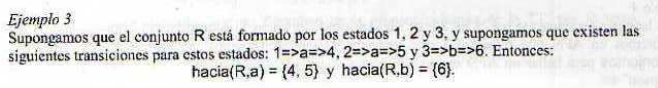
Clausura-ε(R) = clausura-ε(1) U clausura-ε(3) U clausura-ε(6) =>

clausura-ε(R) = {1,2,4} U {3,4} U {6} => {1,2,3,4,6}

### CONJUNTO HACIA

Sea R un conjunto de estados y x un símbolo del alfabeto.

Entonces hacia (R,x) es el conjunto de estados a los cuales se transita por el símbolo x desde los estados R que tengan esa transición.



### ALGORITMO DE CLAUSURA-**ε**

Realiza la construcción de un AFD a partir de un AFN.

Se deben determinar las siguientes cosas:

* Estado inicial del AFD
* Sus estados finales
* Su tabla de transiciones

1. Se obtiene el estado inicial del AFD siendo este la clausura ε del estado inicial del AFN.
2. Se agrega este estado a la primera columna de la tabla
3. Para cada símbolo del alfabeto se calcula el conjunto hacia del estado que se acaba de agregar a la primera columna de la tabla
4. Se determinan nuevos estados del AFD por medio de la clausura ε de cada conjunto hacia recién obtenido. Estos estados (conjuntos) se incorporan a la tabla, en las columnas que corresponden
5. Si un nuevo estado obtenido en el punto anterior no existe todavía en la primera columna de la TT, se lo agrega
6. Se repiten los pasos 3 al 5 hasta que no surjan nuevos estados.

## AFD

El algoritmo de Thompson no puede resolver:

* Complemento
* Intersección

Nota: Estas dos operaciones sólo se pueden aplicar a AFD.

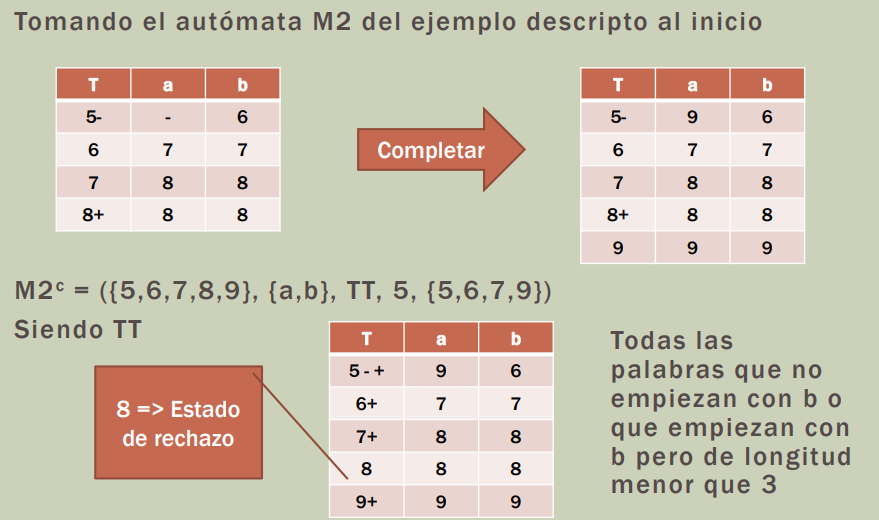
### COMPLEMENTO DE UN AFD

El complemento de un AFD se obtiene invirtiendo los estados finales y no finales:

* Todo estado “No final” pasa a ser un estado “Final” del AFD
* Todo estado “Final” pasa a ser un estado “No final” del AFD
* Todos estados de “Rechazo” se convierten en estado final

Formalmente se define:

AFD M1 = {Q1,Σ,T1, q1,F1} el complemento se define como M = {Q1,Σ,T1, q1, Q1 - F1}

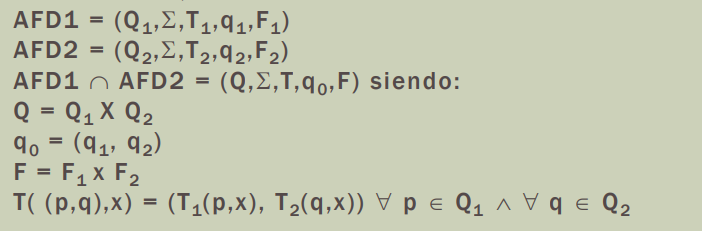


Nota: Es importante que el AFD esté completo para poder realizar el complemento del mismo.

### INTERSECCIÓN DE UN AFD

La intersección de dos AFDs es otro AFD y reconoce las palabras comunes a los LRs reconocidos por los dos AFDs. Los estados de los AFD intersección son pares ordenados de estados,uno de cada AFD dato.

Formalmente se define:

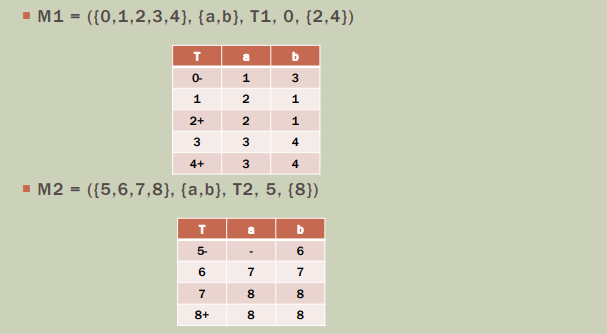


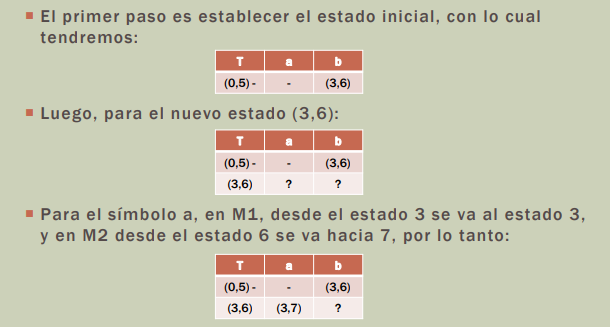
Se deben tener en cuenta las siguientes consideraciones:

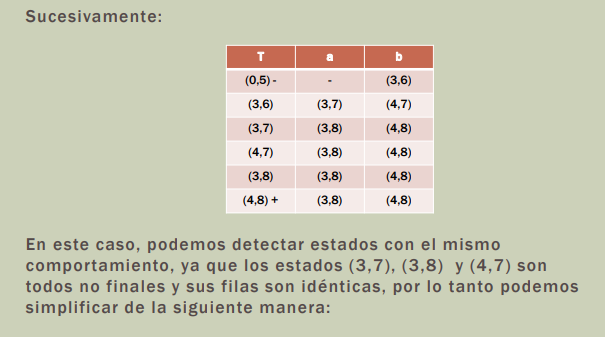
* Todos los estados son pares ordenados.
  + Si uno de estos estados es un estado erróneo, entonces el par ordenado representará un estado erróneo.
    - Estado inalcanzable desde el estado inicial
    - Estado desde el cual no se puede llegar a un estado final
* Los estados erróneos de la intersección no deben ser considerados en la TT final

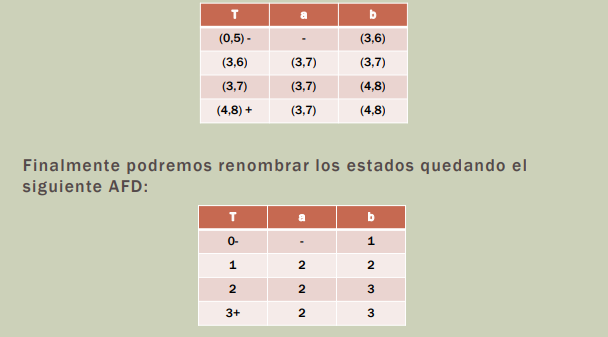
Para hallar la intersección se deben completar los siguientes pasos:

1. Obtener el estado inicial y colocarlo en la primera fila de la TT
2. Determinar las transiciones desde el estado inicial y agregar nuevas entradas en la tabla a medida que surjan nuevos estados, siempre que éstos no sean estados erróneos
3. Repetir el proceso para cada nuevo estado, hasta que no se puedan crear más estados.
4. Tener en cuenta que los únicos estados finales son aquellos en los que ambos estados del par ordenado eran estados finales originalmente en sus AFDs respectivos
5. Finalmente detectar estados con el mismo comportamiento para realizar simplificación. Dos o más estados tienen el mismo comportamiento:
   1. Si todos son estados no finales o
   2. Todos son estados finales y además sus filas son idénticas.









## AF a ER

### 1. DEPURACIÓN DEL AF

La depuración es el primer paso para pasar de un Autómata finito a una expresión regular y consiste en eliminar los estados erróneos de la tabla de transiciones que representa al AF.

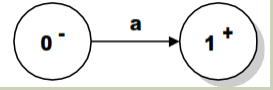
* Estados Inalcanzables: Estados a los que no se pueden llegar desde el estado inicial
* Estados de Rechazo: Estados que no conducen a un estado final

### 2. SISTEMA DE ECUACIONES

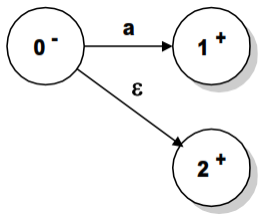
Se parte de la TT depurada y:

* Se crearán tantas ecuaciones como estados tenga el AF
* En el lado izquierdo estará el estado
* En el lado derecho, la unión de términos que representan las transiciones
* La ecuación de un estado final se agrega un término Ԑ.

Ejemplos:



* 0 = a1
* 1 = Ԑ



* 0 = a1 + 2
* 1 = Ԑ
* 2 = Ԑ

### 3. PROCESO DE REDUCCIÓN

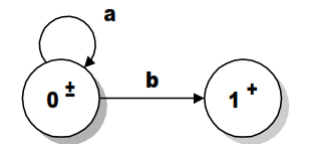
Se deben resolver las ecuaciones. Siempre se debe resolver última la ecuación que tenga el estado inicial.

Para aquellas ecuaciones recursivas que tienen uno o más ciclos con formato:

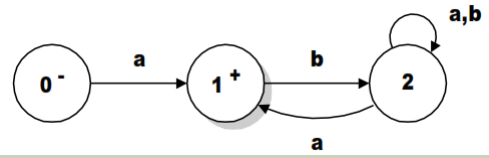
e = αe + β => e = α\*β

* e: estado
* α: es una ER
* β: expresión que puede estar formada por caracteres del alfabeto, Ԑ y estados

Ejemplos:



* 0 = a0 + b1 + Ԑ => Reducción => 0 = a\*(b1 + Ԑ)
* 1 = Ԑ



* 0 = a1
* 1 = b2 + Ԑ
* 2 = (a + b)2 + a1 => Reducción => 2 = (a + b)\*a1

## AFD Mínimo

### AFD MÍNIMO

Es aquel que reconoce un LR con la mínima cantidad de estados posibles. Se lo conoce como el único autómata óptimo asociado a la aceptación de determinado LR.

Tiene 3 aplicaciones:

* Determinar si dos o más AFDs son equivalentes
* Probar la equivalencia de dos o más ER
* El AFD mínimo es el que tiene la TT más reducida con menor cantidad de filas y esto beneficia la implementación de un AFD mediante un programa de computadora.

**AFD EQUIVALENTES**: Dos AFD son equivalentes si el AFD mínimo se obtiene a partir de ellos es el mismo (solo puede diferir los nombres de los estados)

**ER EQUIVALENTES**: Dos ER son equivalentes sin son reconocidas por el mismo AFD mínimo.

### ALGORITMO

Se divide en una serie de pasos

1. Verificar que la TT esté completa, de no estarlo, deberá completarse (agregar un estado erroneo).
2. Particionar el conjunto de estados en dos clases:
   1. La clase de los estados no finales
   2. La clase de los estados finales
3. Detectar estados equivalentes por clase => mismo comportamiento y misma clase (mismo comportamiento es que tienen la misma fila)
4. Reducir el autómata => eliminar los estados de equivalencia inmediata
5. Construir tabla de transiciones por clase
6. Búsqueda de estados equivalentes por clase.
7. Actualizar la tabla

El **COMPORTAMIENTO** de un estado es la sucesión de estados de llegada de las transiciones que parten de ese estado, para un ordenamiento establecido de los caracteres del alfabeto.

Dos estados son equivalentes si:

* Pertenecen a la misma clase
* Tienen un mismo comportamiento.

## BNF

**BNF ( Backus Naur Form)**: Notaciones que tienen propiedades similares a las de las gramáticas formales. Se utilizan para describir los lenguajes de programación en dos aspectos: Léxico y Sintáctico

* Los no terminales se escriben entre < y >
* El meta símbolo de producción es ::= (operador ES)
* Se mantiene el meta símbolo | (o)

BNF en ALGOL:

Cada regla BNF se forma con elementos provenientes de tres conjuntos disjuntos:

* Metavariables o noterminales: son palabras o frases encerradas entre corchetes angulares (ejemplo: <símbolo básico>, <identificador>).
* Terminales: son los caracteres del alfabeto o palabras del lenguaje sobre los cuales se construye el LP descripto. Se los llama terminales porque no existen reglas de producción para ellos (ejemplo: una palabra reservada es un terminal).
* Metasímbolos: son caracteres o grupos de caracteres que ayudan a representar estas reglas (ejemplo: <>, ::=, |).

En toda regla BNF, encontramos, al igual que en toda producción de una GIC, tres componentes:

* El lado izquierdo, formado por un solo noterminal
* El lado derecho, formado por terminales y noterminales o, eventualmente, vacío
* Un metasímbolo, el operador es, que vincula el lado izquierdo con el lado derecho.

### CATEGORÍAS SINTÁCTICAS

Son expresiones y sentencias de un LP y generalmente son LICs. Estos lenguajes no pueden ser generador por GRs ni GQRs sino que requieren ser generador por GICs.

* Expresiones
* Sentencias
* Estructuras de control

Notas:

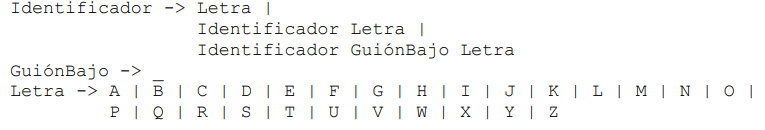
* Todos estos tipos de gramáticas se caracterizan porque sus producciones tienen un único noterminal en el lado izquierdo
* Una Gramática Formal no solo genera un Lenguaje Formal, sino que también se la puede utilizar para describir la sintaxis del lenguaje que genera

### CATEGORÍAS LÉXICAS (TOKENS)

Son los identificadores, los números enteros, los números reales, los caracteres, las cadenas constantes, los operadores y los caracteres de puntuación- constituyen diferentes LRs

* identificador
* palabraReservada
* constante
* literalCadena
* operador
* caracterPuntuación

**IDENTIFICADOR**: una secuencia de uno o más caracteres que nombra diferentes entidades de un LP (variables, funciones, procedimientos, constantes, tipos, etc.). Los “identificadores” constituyen un LR infinito. El identificador es el axioma.



Nota: La gramática anterior es recursiva a la izquierda ya que el identificador se va formando de derecha a izquierda.

**PALABRAS RESERVADAS**: están compuestas por una secuencia de caracteres,

forman un LR finito, independiente de los identificadores

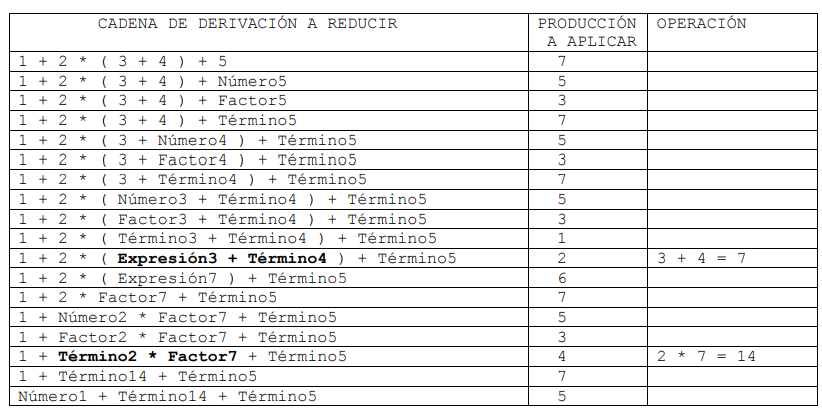
**EXPRESIONES**: un LP está representado por las expresiones que se pueden escribir en ese LP. Una GIC para expresiones no solo debe producir todas las expresiones válidas, sino que también debe preocuparse por la precedencia (o prioridad) y la asociatividad (evaluación de izquierda a derecha o de derecha a izquierda para operadores con igual precedencia) de cada uno de los operadores existentes en ese LP.

Prioridades de los operandos en una GIC: Cuánto más cerca del axioma, menor es la prioridad de un operador

### REDUCCIÓN

Es el proceso inverso al de la derivación que se realiza para llegar al axioma (se reduce la tabla de derivación). Se parte de la secuencia de terminales que forman la expresión y se finaliza cuando se llega al axioma de la GIC.

Tabla de evaluación: Tabla que junto a cada noterminal se agrega el número terminal del cual proviene, para tenerlo presente al realizar la correspondiente evaluación.



Notas:

* Si se intenta “nivelar” precedencias puedo terminar con una gramática ambigua
* Si hay 3 operandos vinculados con operadores de igual precedencia, el operando central operará primero con el operando de la izquierda si la operación es asociativa a izquierda pero si la operación es asociativa a derecha, el operando central operará primero con el operando de la derecha.

### GRAMÁTICA AMBIGUA

Una gramática es ambigua si hay dos posibles derivaciones del mismo tipo (izquierda o derecha) para la misma cadena.

* Para demostrar que es ambigua basta con encontrar una cadena que admita dos árboles de derivación diferentes
* Para demostrar que es no ambigua se complica más, hay que demostrar que en ningún caso voy a poder armar dos árboles de derivación diferentes

Nota: No hay un algoritmo que tomando cualquier GIC pueda interpretar si la misma es ambigua o no. Sin embargo si es posible determinarlo para algunas clases de gramáticas

## Análisis Léxico, Sintáctico y semántico

Tipos de analisis:

* Léxico: Se puede implementar es con autómatas finitos o implementando una función lexer (o utilizando una herramienta llamada flex o lex que nos genera el scanner)
* Sintáctico: Implementando un parser o una herramienta llamada yacc o vizon
  + Se escriben la GIC en un archivo de texto, se pasa por yacc o vizon y devuelve un programa en c
* Semántico: Forma de implementar el análisis semántico

Análisis (Frontend)

* Scanner: es el analizador lexico, recibe una secuencia de caracteres (el código fuente), reconoce los lexemas de los distintas categoría léxicas (LR) y genera como salida una secuencia de tokens. Este interactua con el parser
* Parser: es el analizador sintáctico, recibe los tokens generados por el scanner y verifica que cumplan la GIC del lenguaje
* Analizador semántico: realiza comprobaciones que incluyen el contexto, como el tipo de las variables, que no hay definiciones duplicadas y otras. A su vez genera un primer código intermedio

Síntesis (Backend)

* Optimización del código intermedio
* Generación del código para la máquina concreta donde debe ejecutar

### SCANNER

Implementaciones directas

* Un proceso por cada LR
  + La primer parte detecta la categoría léxica, luego reconoce el lexema en particular.
  + Típicamente un switch sobre el primer carácter leído y subsecuentes aperturas por if o switch
* Tabla de transición por switch
  + Un loop mientras haya caracteres y un switch en base al estado. En cada uno discrimina según el carácter (o clase de carateres) leído.
  + En alguna época se usaban goto para programarlos, en lugar de variable estado y switch, se saltaba directo desde el estado anterior
* Ventajas
  + El código sigue casi “directamente” la definición del AF
  + Fácil de escribir
  + Rápido al ejecutar
* Contras
  + Mucho código similar
  + Aburrido de escribir si no es un ejemplo simple
  + Casi nada es reutilizable para otro Scanner
  + Se pueden escapar diferencias sutiles con respecto a la definición y se hacen difíciles de detectar y corregir

Implementaciones con tabla de transición

* Sería la implementación que vimos de autómatas finitos.
* Puede ser
  + A mano con implementaciones de AFDs
  + Con una herramienta como lex
* Filas para los estados y columnas para los caracteres o grupo de caracteres
* Al final dependiendo de si acepta (y cuál fue el estado aceptor) o no ejecutará código que maneje la situación
* Ventajas
  + Más reutilizable
  + Menos trabajo para implementarlo
  + Si lo hicimos mediante herramientas estamos seguros de cumplir con la definición
* Contras
  + Semi mágico (documentar bien la tabla)
  + Algo más lento que la implementación directa

Consideraciones:

* Los espacios en blanco (espacio, tabulador, nueva línea) suelen ser simplemente ignorados (no siempre, por ejemplo: python)
* En los casos que es necesario un centinela, luego de leerlo se lo devuelve al flujo de entrada (ungetc).
* Los LR infinitos requieren de un centinela. Los finitos pueden requerirlo o no.
* Es común que al detectar el final de archivo se envíe un Token que lo represente

### PARSER

* Hay varios tipos de parsers. Para este ejemplo usaremos el “Análisis Sintáctico Descendente Recursivo” (ASDR)
* Un ASDR es un parser del tipo top-down, es decir, parte del axioma y va armando el Árbol de Análisis Sintáctico (AAS) con derivación a izquierda.
* Otros parsers son bottom-up es decir, a partir de los terminales (hojas del árbol) van reduciendo hacia el axioma.
* El AAS queda de modo tal que en las hojas están los terminales y en los nodos interiores los no terminales.
* Cada nodo tiene como hijos la secuencia de terminales y no terminales que representa una posible producción de ese no terminal.
* El AAS no siempre es una estructura de datos. En el caso del ASDR el árbol es virtualmente armando por los llamados recursivos entre funciones

### ANALIZADOR SEMÁNTICO

El analizador semántico, o generador de código intermedio, se encarga de:

* Hacer los controles que son propios de una gramática sensible al contexto:
  + Comprobar si las variables están definidas
  + Comprobar el tipo de las variables
  + Comprobar tipo y cantidad de parámetros de una función, etc
* Ir generando código de una máquina virtual (MV) que el compilador usa como intermedio antes de generar código para una arquitectura de procesador específica
* El parser es quien va llamando al analizador semántico en momentos determinados, de modo similar a como hace con el escáner
* Para ello primero se modifica la gramática y se agregan símbolos de acción.
* Para cada símbolo de acción se programará su correspondiente rutina semántica.
  + La gramática con los símbolos de acción agregados se conoce como gramática con anotaciones
* Los símbolos se agregan en el punto en el cuál debe llamarse a la rutina semántica. Los indicamos con un # adelante
* Para que las rutinas semánticas puedan trabajar necesitan datos.
  + Desde el punto de vista formal se conocen como atributos
  + Desde el punto de vista práctico se conocen como registros semánticos

Rutinas Semánticas:

Donde ubicar las rutinas semánticas

* Donde haya que hacer chequeos de semántica estática (en nuestro caso solo declarar las variables cuando aparecen por primera vez)
* Donde haya que generar código
* Donde haya que transmitir / recibir / procesar datos (registros semánticos)

Registro semánticos:

Dada la simpleza de micro solo hay dos tipos de registros semánticos

* REG\_OPERACION, que solo contendrá el valor del token SUMA o RESTA.
* REG\_EXPRESION, que contendrá el tipo de expresión y el valor que le corresponde. Este valor puede ser:
  + una cadena (para el caso de un identificador)
  + un número entero (para el caso de una constante)