RISI **2**(2), 60-68 (2005)
Rev. investig. sist. inform.
Facultad de Ingeniería de Sistemas e Informática
Universidad Nacional Mayor de San Marcos
ISSN: 1815-0268 (versión impresa)/ ISSN: 1816-3823 (versión electrónica)

ANÁLISIS COMPARATIVO ENTRE UN ANALIZADOR SINTÁCTICO LL Y UN ANALIZADOR SINTÁCTICO LR PARA UN LENGUAJE FORMAL

COMPARATIVE ANALYSIS BETWEEN A PARSER SINTACTIC LL AND A PARSER SINTACTIC LR FOR A FORMAL LANGUAGE

Augusto Cortez*

RESUMEN El presente trabajo surge como una necesidad de contrastar los métodos de análisis sintácticos ascendente y descendente, que se realizan durante la fases de análisis en la compilación de un programa fuente escrito en un lenguaje formal. Al momento de especificar un lenguaje, se tiene que especificar la sintaxis y la gramática, asimismo para diseñar su compilador, específicamente el módulo de análisis sintáctico, tiene que decidirse que técnica utilizar. Este trabajo pretende servir como un instrumento de decisión.

Palabras clave: Lenguaje Formal, Compilador, Analizador Léxico, Analizador Sintáctico.

ABSTRACT The present work arises like a necessity to resist the methods of ascending and descendent syntactic analysis, that is made during the phases of analysis in the compilation of a source program written in a formal language of high level. At the time of specifying a language, it must specify the syntax and the gramatic. Also to design its compiler, specifically I modulate of syntactic analysis, must be decided that technical to use. This work tries to serve like a decision instrument.

Key word: Formal language, compilers, lexical analyzer, syntactic analyzer.

1. INTRODUCCIÓN

Generalmente los programadores se especializan en programar en uno o dos lenguajes. SAMETT en 1969 elaboró una lista de 120 lenguajes, de uso bastante amplio, y desde entonces se han desarrollado muchos más. Frente a ello muchos se preguntan cuál es la finalidad de estudiar la teoría de lenguajes de programación y de traductores. Existen razones para realizar un estudio sobre la teoría de lenguajes, y la construcción de su traductor, y es que muchas de las técnicas son utilizadas también en otras áreas como son los manejadores de base de datos, reconocedores de patrones, etc.

2. MARCO CONCEPTUAL

2.1. LENGUAJES

Un lenguaje consiste de un número finito o infinito de secuencias o frases. Los lingüistas definen un lenguaje como un sistema de comunicación que está compuesto de símbolos y constructores para describir frases válidas para el lenguaje. Aunque un lenguaje puede describirse enumerando sus elementos, es conveniente definirlo a través de propiedades:

Descripción gramatical: Sea G una gramática, definimos el lenguaje generado por una gramática y lo denotamos L(G) de la siguiente forma:

* Docente Asociado del Departamento de Ciencias de la Computación, Facultad de Ingeniería de Sistemas e Informática de la Universidad Nacional Mayor de San Marcos, Lima-Perú
E-mail: acortezv@unmsm.edu.pe

$L(G) = \{w / S \rightarrow^{*} w,$ y además $w \in V_t^{*} \}$,

G denota la gramática del lenguaje L.

L contiene secuencias de terminales que son generadas a partir del axioma de G.

$L = \{a^m b , m \geq 0\}$ descripción algebraica

Podemos hallar G tal que L=L(G)

$G (V_N, V_T, S, P)$ donde:

$V_N = \{S\}$

$V_T = \{a, b\}$

$P:\{ S \rightarrow aS / b\}$

$S \rightarrow b$

$S \rightarrow aS \rightarrow ab$

$S \rightarrow aS \rightarrow aaS \rightarrow aab$

$S \rightarrow aS \rightarrow aaS \rightarrow aaaS \rightarrow aaab$

luego b, ab, aab, aaab $\in$ L(G).

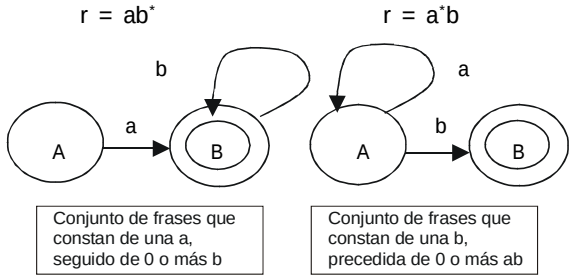
Descripción mediante expresiones regulares

Si r es expresión regular, el conjunto de secuencias generadas por r es denominado lenguaje regular y lo denotamos L(r).

Para $r = a^{*}b$, se tiene L(r) = {b, ab, aab, aaab,...}, esto denota el conjunto de frases de cero o más símbolos a, que terminan con símbolo b.

Descripción mediante un autómata

Si M es un autómata finito, al conjunto de secuencias que reconoce M lo denotamos como L(M).



La figura muestra los diagramas de transiciones de dos autómatas que corresponde a la expresión regular **ab***

Sintaxis de un lenguaje

La sintaxis proporciona al programador el formato de un programa, y se describe mediante esquemas sintácticos o mediante diagramas sintácticos:

- Esquemas sintácticos
- Representación gráfica de la sintaxis

Esquemas sintácticos

Un programa tiene la siguiente sintaxis

```
ACCIÓN <Id>
INICIO
        CUERPO
FIN
```

donde:
Id: denota el nombre del programa.
CUERPO: denota el cuerpo del programa.

donde:
Id: denota el nombre del programa.
CUERPO: denota el cuerpo del programa.

```
SD;

SP;
```

El cuerpo consta de dos partes

CUERPO:

donde:
SD: denota la sección declaraciones.
SP: denota la sección proposiciones.

```
Declaración 1
Declaración 2
…
Declaración n
```

La sección de declaraciones consta de una secuencia de declaraciones.

SD se define:

En forma recursiva puede definirse **D / D ; SD**

La sección de declaraciones SD, consta de una declaración D o de una D seguida de SD

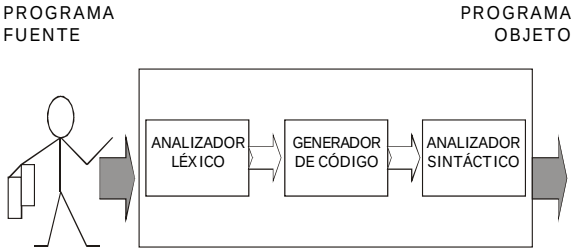
donde:
D: denota una proposición de declaración.

2.2. COMPILADORES

Traductores

Un traductor es, en forma general, un programa que traduce un programa escrito en un lenguaje:

el lenguaje fuente, y lo traduce a un programa equivalente en otro lenguaje: el lenguaje objeto. Durante este proceso de traducción el compilador informa al usuario de la presencia de errores en el programa fuente y presenta técnicas de recuperación [12].



Funciones básicas del compilador

Una proposición está conformada por una cadena de caracteres; sin embargo, por razones de análisis el compilador agrupa los caracteres en secuencias llamadas componentes léxicos (TOKENS). Estos componentes pueden suponerse como los axiomas o fundamentos del lenguaje, pues vienen a ser los átomos o ladrillos con los que se construyen las frases de un lenguaje. Un componente léxico puede ser una palabra clave, un nombre de variable (identificador), una constante numérica entera o real, un operador aritmético etc. Después de analizar los componentes léxicos, cada proposición del programa debe reconocerse como una frase del lenguaje, como una declaración, proposición de asignación etc. Este proceso se denomina análisis sintáctico (PARSING), lo realiza la parte del compilador denominada analizador sintáctico (PARSER). El último paso en el proceso básico de traducción es la generación de código objeto [1], [5].

3. GRAMÁTICA

La gramática de un lenguaje describe las reglas para verificar el orden estructural de las frases.

Definición formal de gramática

Una gramática se define formalmente de la siguiente forma: $\mathbf{G = (V_T, V_N, P, S)}$

donde:

$\mathbf{V_T}$: conjunto finito de símbolos terminales del lenguaje.

$\mathbf{V_N}$: conjunto finito de símbolos no terminales.

P: conjunto finito de reglas de producción.

S: Símbolo distinguido o axioma inicial a partir del cual se reconocerán las secuencias de L, aplicando sucesivamente las reglas de producción.

Ejemplo:

$\boldsymbol{L}$ = { $a^m b$, $m \geq 0$} = { b, ab, aab, aaab,...}
G (V_N, V_T, S, P)

donde:

V_N = {S}
V_T = {a, b}
P: { $S \rightarrow aS / b$}

$S \rightarrow b$
$S \rightarrow aS \rightarrow ab$
$S \rightarrow aS \rightarrow aaS \rightarrow aab$
$S \rightarrow aS \rightarrow aaS \rightarrow aaaS \rightarrow aaab$

luego b, ab, aab, aaab $\in$ L(G).

Lenguajes definidos por una gramática

Si G es una gramática, genera un lenguaje $L(G) \subseteq V_T^*$

$$L = \{w / S \rightarrow^* w \text{ , y además } w \in V_T^* \},$$

es decir son todas las secuencias que están en la cerradura de V_T, que pueden obtenerse a partir del axioma de la gramática aplicando las reglas de producción.

4. AUTÓMATAS FINITOS

Un autómata finito es un modelo matemático que sirve como reconocedor de frases de un lenguaje, es representado como un diagrama de transiciones –grafo dirigido–. en el que los nodos son los estados y las aristas etiquetadas representan las funciones de transición [3].

Formalmente un autómata se define como un modelo matemático, el cual denotaremos:

$M = (E, \Sigma, \delta, \varepsilon_0, F)$

E: conjunto de estados

Σ: alfabeto de entrada (alfabeto)

δ: función de transición, δ: E x $\Sigma \rightarrow$ E

ε_0: estado inicial

F: conjunto de estados de aceptación

El autómata M puede ser **determinista (AFD)** o **no determinista (AFND)**. M es determinista cuando a partir de un estado puede pasarse a un solo estado con cada entrada del alfabeto, además no tiene ninguna transición con λ para cada estado. Un autómata es no determinista cuando se da el caso que existe más de una transición desde un estado con la misma entrada [7], [9].

Ejemplo: sea r = (a/b)*ab expresión regular, construimos M(r) autómata no determinista, a partir de la expresión regular.

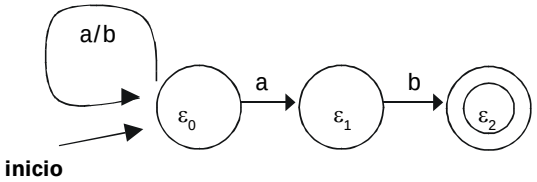


Figura 1. Autómata finito no determinista

M(E, Σ, δ, ε_0, F) **E={ε_0, ε_1, ε_2}** **Σ = { a, b}** **F= {ε_2}**

δ	a	b
ε_0	{ε_0, ε_1}	{ε_0}
ε_1	-	{ε_2}
ε_2	-	-

El siguiente autómata determinista es equivalente al autómata de la figura 1.

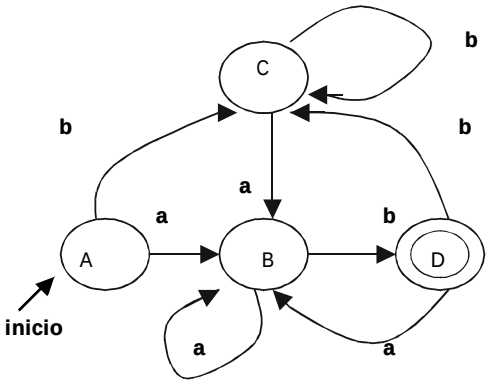


Figura 2. Autómata finito determinista

M(E, A, δ, ε_0, F) **E={A, B, C, D}** **A = { a, b}** **F = {D}**

δ	a	b
A	B	C
B	B	D
C	B	C
D	B	C

Existe un método llamado «de particiones», el cual no detallaremos, que permite reducir el número de estados.

La máquina simplificada es la siguiente:

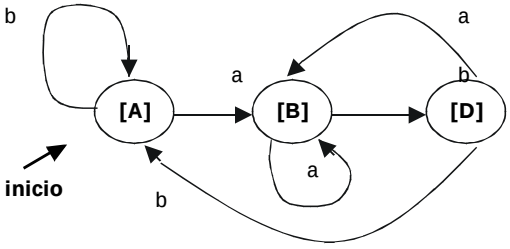


Figura 3. Autómata finito determinista simplificado

M(E, Σ, δ, ε_0, F) **E={[A], [B], [D]}**
Σ = { a, b} **F = {[D]}**

δ	a	b
[A]	[B]	[A]
[B]	[B]	[D]
[D]	[B]	[A]

5. AUTÓMATA DE PILA

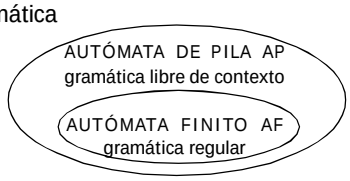
Un autómata de pila es esencialmente un autómata finito al que se le ha añadido una memoria (pila). Así como los lenguajes regulares pueden definirse a través de un autómata finito, un lenguaje libre de contexto puede definirse a través de un autómata de pila [3], [7], [5].

Consideremos el lenguaje L(G) = {$a^n b^n$ / n ∈ N} generado por la gramática G

P: { 1 S → ab,
2 S → aSb,
}

esta gramática no es regular, es decir no existe una expresión regular que la genere y por ello tampoco existe un autómata finito que reconozca L(G). Para este tipo de lenguajes es necesario adicionar al autómata finito una pila que sirva para memorizar algunos símbolos de referencia.

Ejemplo: Sea la gramática



G (V_N, V_T, S, P)

donde:

V_N = {S} ,

V_T= {a, b, c}

P: { 1 S → aSa,
2 S → bSb,
3 S → c
}

L(G) = {wcw^R / w ∈ {a, b}* }

No existe M autómata finito tal que L(M) = L(G), puesto que los autómatas de pila sólo aceptan gramáticas libres de contexto. Para reconocer la frase «aabcbaa» deberá empilarse un símbolo asociado a cada símbolo de la entrada antes que aparezca el símbolo c, luego de la cual por cada símbolo que aparezca después se desempilará un símbolo de la pila que se compara con el símbolo de la entrada.

Sea ***P*** autómata de pila , definido formalmente como:

(E, Σ, Σ_p, δ, ε_o , p_0, F)

E: conjunto finito de estados.

Σ: conjunto finito de entradas (alfabeto del sistema).

Σ_p : conjunto finito de la pila (alfabeto de la pila).

δ: Función de transición.

$\varepsilon_o \in E$: estado inicial.

$P_0 \in \Sigma_p$: estado inicial de la pila.

$F \subseteq E$: conjunto de estados finales.

E, Σ, ε_o y F se definen igual que los autómatas finitos.

Ejemplo:

G = (V_N , V_T , P , S) Donde V_N = {S , L} V_T ={if , else , e , while, ; , { , } , p}

P ={ S→ if e S; / if e S else S;
S→ while e S;
S→ { L } / p
L→ L ; S / S
}

A partir de G se construye P = ($E , \Sigma , \Sigma_p , \delta , \varepsilon_i , \varepsilon_\phi$, F)

Donde : E = {ε_i , ε_p , ε_q , ε_f } Σ= {if , else , e , while, ; , { , } , p}

Σ_P = {S , L} U {if, else ,e, while, ; , { , } , p} U { # }

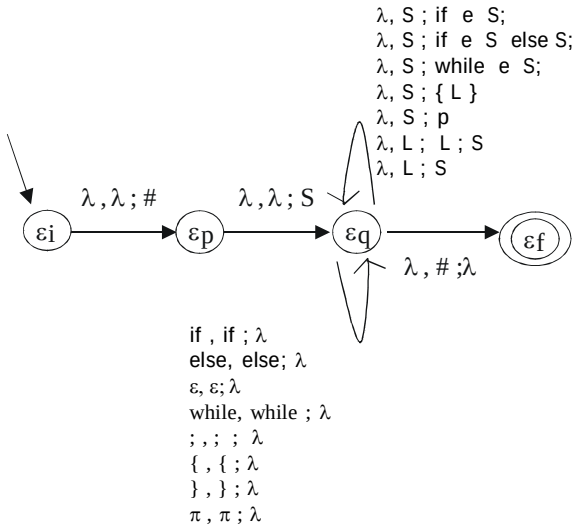


Figura 4.

6. ANÁLISIS LÉXICO

La tarea de examinar la proposición con la finalidad de reconocer y clasificar los distintos componentes léxicos se conoce como análisis léxico y la parte del compilador que realiza esa función analítica se denomina analizador léxico (SCANNER).

7. ANÁLISIS SINTÁCTICO

El proceso de análisis sintáctico puede describirse como el intento de construir un árbol de análisis sintáctico para la frase que se quiere reconocer. Aunque en la práctica los árboles sintácticos no se implementan porque ocuparían mucho espacio. La sintaxis de las construcciones de los lenguajes de programación pueden describirse por medios de gramáticas libres del contexto [1], [11].

Los analizadores sintácticos pueden ser de dos tipos:

Analizador sintáctico ascendente: Intenta construir un árbol de análisis sintáctico, empezando desde la raíz y descendiendo hacia las hojas. Lo que es lo mismo que intentar obtener una derivación por la izquierda para una cadena de entrada, comenzando desde la raíz y creando los nodos del árbol en orden previo.

Analizador sintáctico descendente: Intenta construir un árbol de análisis sintáctico, empezando desde las hojas (la cadena) y ascendiendo hacia la raíz. Lo que es lo mismo que intentar obtener una reducción desde una cadena hasta llegar al axioma.

El analizador sintáctico tanto ascendente como descendente puede representarse de dos formas: mediante tabla de análisis sintáctico o mediante autómata de pilas.

7.1. ANÁLISIS SINTÁCTICO DESCENDENTE LL BASADO EN TABLA DE ANÁLISIS (TAS)

Intenta construir un árbol de análisis sintáctico, empezando desde la raíz y descendiendo hacia las hojas. Lo que es lo mismo que intentar obtener una derivación por la izquierda para una cadena de entrada, comenzando desde la raíz y creando los nodos del árbol en orden previo.

L: left to right: lee la entrada de izquierda a derecha.

L: left derivation: produce una derivación por la izquierda.

Ejemplo:

Consideremos la gramática

A	→	A+B /B	eliminamos	A	→ BA'
B	→	B*C/C	la recursividad	A'	→ +BA'/λ
C	→	(A)/Id		B	→ CB'
				B'	→ *CB'/λ
				C	→ (A)/Id

Sea la Tabla de análisis sintáctico T

	Id	+	*	(	)	$
A	A → BA'	-	-	A → BA'	-	-
A'	-	A' → +BA'	-	-	A' → λ	A' → λ
B	B → CB'			B → CB'		
B'	-	B' → λ	B' → *CB'	-	B' → λ	B' → λ
C	C → Id			C → (A)		

Paso	PILA	ENTRADA	SALIDA
1	$A	Id+Id*Id$	
2	$A'B	Id+Id*Id$	A → BA'
3	$A'B'C	Id+Id*Id$	B → CB'
4	$A'B'Id	Id+Id*Id$	C → Id
5	$A'B'	+Id*Id$	
6	$A'	+Id*Id$	B' → λ
7	$A'B+	+Id*Id$	A' → +BA'
8	$A'B	Id*Id$	
9	$A'B'C	Id*Id$	B → CB'
10	$A'B'Id	Id*Id$	C → Id
11	$A'B'	*Id$	
12	$A'B'C*	*Id$	B' → *CB'
13	$A'B'C	Id$	
14	$A'B'Id	Id$	C → Id
15	$A'B'	$	
16	$A'	$	B' → λ
17	$	$	A' → λ

CONSTRUCCIÓN DE LA TABLA

```
Acción  CONSTRUYE_TABLA()
Inicio
∀ A ——> α    //regla de derivación
             ∀ α ∈ PRIMERO(α)
                    Incluir A  a en T[A , a]
             Si λ ∈ PRIMERO (α)
                    Incluir A ——> a en T[A , a]
                           ∀ b ∈ SIGUIENTE(A)
             Fin si
             Si λ ∈ PRIMERO (α) y $ ∈ SIGUIENTE(A)
                    Incluir A ——> α en T[A , $]
                    Fin si
Fin
```

En el ejemplo anterior, aplicando el algoritmo, se obtiene la tabla T mostrada anteriormente.

ANÁLISIS SINTÁCTICO ASCENDENTE LR BASADO EN AUTÓMATA DE PILA

Construye un autómata de pila que reconoce mediante transiciones una derivación por la derecha. A partir de la gramática G es posible construir un analizador LR basado en autómata de pila [3] [4].

ANÁLISIS SINTÁCTICO ASCENDENTE LR

L: left to right: lee la entrada de izquierda a derecha.

R: right derivation: produce una derivación por la derecha.

Si G es una gramática libre de contexto entonces, existe analizador LR basado en Pila P, tal que L(G) = L(**P**).

Sea $G = (V_N, V_T, S, P)$
Construimos $P = (E, \Sigma, \Sigma_p, \delta, \varepsilon_i, R, F)$

Hacemos:
1 $\Sigma = V_T$
2 $\Sigma_p = V_T \cup V_N \cup \{\#\}$
no está en V_T ni V_N
3 $E = \{\varepsilon_i, \varepsilon_p, \varepsilon_q, \varepsilon_f\}$
ε_i: estado inicial ε_f: estado final
4 Introducir las transiciones (operación de reducción)

$(\varepsilon_i, \lambda, \lambda ; \varepsilon_p, \#)$
$(\varepsilon_p, \lambda, S ; \varepsilon_q, \lambda)$
$(\varepsilon_p, \lambda, W ; \varepsilon_p, N)$

(Reducción)
$\leftrightarrow N \quad W$,
$W \in (V_N \cup V_T)^*$

$(\varepsilon_p, a, \lambda; \varepsilon_p, a) \quad \forall a \in V_T$ **(desplazamiento)**
$(\varepsilon_q, \lambda, \# ; \varepsilon_F, \lambda)$

COMPARACIÓN ENTRE ANÁLISIS SINTÁCTICO LL Y ANÁLISIS SINTÁCTICO LR

Cuando se utiliza un autómata de pila surge un problema al interrogar qué símbolos existen en la cima, de la pila, pues sólo se conoce el símbolo en la cima, y en caso se quiera conocer varios símbolos, tendrían que hacerse varias desempilaciones, en caso de que los símbolos sacados no sean los que se esperaban tienen que regresarse a la pila. Esta desventaja es subsanada por la técnica mediante tabla de análisis sintáctico, que integra en cada entrada la información que se requiere para hacer una derivación sin necesidad de usar pila.

Problemas al implantar LL

1 ¿Cómo determinar si debemos aplicar una operación de desplazamiento o una operación de reducción?

2 Si reducimos, puede existir más de una alternativa a elegir. Esto se resuelve con el principio de preanálisis.

3 Cómo manejar la pila. Si se tiene el movimiento

$(\varepsilon_i, \lambda, aPa ; \varepsilon_p, P)$ significa desempilar **aPa** y empilar **P**.

Pero cómo saber si se tiene la tira **aPa** en el extremo de la pila si sólo conocemos la cima. Una solución es implementar una pila híbrida en la que se pueda acceder no sólo a la cima de la pila sino a todos sus elementos.

Ejemplo

Consideremos al lenguaje

$L(G) = \{ca^n ca^n b^m cb^m c \quad / \ m, n \in Z^+\}$ cuya gramática es $G = (V_N, V_T, S, P)$ libre del contexto

donde $V_N = \{S, A, B\}$, $V_T = \{a, b, c\}$

P: { 1 S ⟶ cABc
2 A ⟶ aAa/c
3 B ⟶ bBb/c }

Construimos $\boldsymbol{P} = (E, \Sigma, \Sigma_p, \delta, \varepsilon_0, R, F)$ a partir de G

1	$\delta(\varepsilon_0, \lambda, \lambda) = (\varepsilon_p, \#)$	**7**	$\delta(\varepsilon_q, a, \lambda) = (\varepsilon_q, a)$
2	$\delta(\varepsilon_p, \lambda, cABc) = (\varepsilon_q, S)$	**8**	$\delta(\varepsilon_q, b, \lambda) = (\varepsilon_q, b)$
3	$\delta(\varepsilon_q, \lambda, aAa) = (\varepsilon_q, A)$	**9**	$\delta(\varepsilon_q, c, \lambda) = (\varepsilon_q, c)$
4	$\delta(\varepsilon_q, \lambda, c) = (\varepsilon_q, A)$	**10**	$\delta(\varepsilon_p, \lambda, S) = (\varepsilon_q, \lambda)$
5	$\delta(\varepsilon_q, \lambda, bBb) = (\varepsilon_q, B)$	**11**	$\delta(\varepsilon_q, \lambda, \#) = (\varepsilon_f, \lambda)$
6	$\delta(\varepsilon_q, \lambda, c) = (\varepsilon_q, B)$		

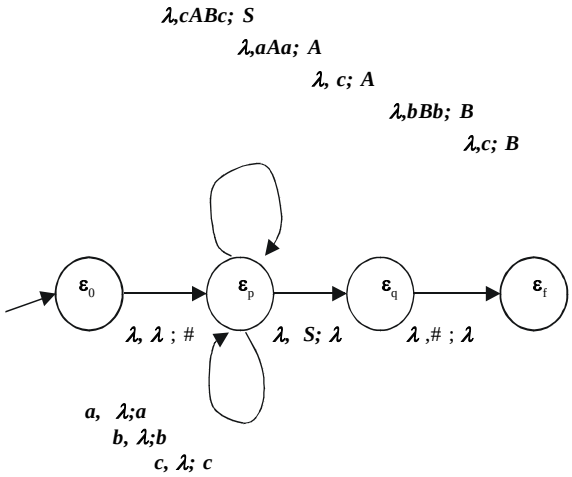


Figura 5.

Veamos el comportamiento del autómata de la figura 5, para la secuencia ***aacbb***

Orden	Pila	Entrada	Transición	
1	λ	***aacbb***	$(\varepsilon_i, \lambda, \lambda ; \varepsilon_p, \#)$	
2	*#*	***aacbb***	$(\varepsilon_p, a, \lambda ; \varepsilon_p, a)$	**desp**
3	*#a*	***acbb***	$(\varepsilon_p, a, \lambda ; \varepsilon_p, a)$	**desp**
4	*#aa*	***cbb***	$(\varepsilon_p, c, \lambda ; \varepsilon_p, c)$	**desp**
5	*#aac*	***bb***	$(\varepsilon_p, \lambda, c ; \varepsilon_p, S)$	**reduc**
6	*#aaS*	***bb***	$(\varepsilon_p, b, \lambda ; \varepsilon_p, b)$	**desp**
7	*#aaSb*	***b***	$(\varepsilon_p, \lambda, aSb ; \varepsilon_p, S)$	**reduc**
8	*#aS*	***b***	$(\varepsilon_p, b, \lambda ; \varepsilon_p, b)$	**desp**
9	*#aSb*	λ	$(\varepsilon_p, \lambda, aSb ; \varepsilon_p, S)$	**reduc**
10	*#S*	λ	$(\varepsilon_p, \lambda, S ; \varepsilon_q, \lambda)$	
11	*#*	λ	$(\varepsilon_q, \lambda, \# ; \varepsilon_f, \lambda)$	
12	λ	λ		

Veamos el comportamiento del autómata para la secuencia ***cacabcbc***

Orden	Pila	entrada	transición	
1	λ	***cacabcbc***	$(\varepsilon_i, \lambda, \lambda; \varepsilon_p, \#)$	
2	*#*	***cacabcbc***	$(\varepsilon_p, c, \lambda; \varepsilon_p, c)$	**desp**
3	*#c*	***acabcbc***	$(\varepsilon_p, a, \lambda; \varepsilon_p, a)$	**desp**
4	*#ca*	***cabcbc***	$(\varepsilon_p, c, \lambda; \varepsilon_p, c)$	**desp**
5	*#cac*	***abcbc***	$(\varepsilon_p, \lambda, c; \varepsilon_p, A)$	**reducc.**
6	*#caA*	***abcbc***	$(\varepsilon_p, a, \lambda; \varepsilon_p, a)$	**desp**
7	*#caAa*	***bcbc***	$(\varepsilon_p, \lambda, aAa; \varepsilon_p, A)$	**reduc.**
8	*#cA*	***bcbc***	$(\varepsilon_p, b, \lambda; \varepsilon_p, b)$	**desp**
9	*#cAb*	***cbc***	$(\varepsilon_p, c, \lambda; \varepsilon_p, c)$	**desp.**
10	*#cAbc*	***bc***	$(\varepsilon_p, \lambda, c; \varepsilon_p, A)$	**reduc.**
11	*#cAbA*	***bc***	$(\varepsilon_p, b, \lambda; \varepsilon_p, b)$	**desp.**
12	*#cAbAb*	***c***	$(\varepsilon_p, \lambda, bAb; \varepsilon_p, B)$	**reducc.**
13	*#cAB*	***c***	$(\varepsilon_p, , c, \lambda, \varepsilon_p, c)$	**desp.**
14	*#cABc*	λ	$(\varepsilon_p, \lambda, cABc; \varepsilon_p, S)$	**reducc.**
15	*#S*	λ	$(\varepsilon_p, \lambda, S, \varepsilon_q, \lambda)$	
16	*#*	λ	$(\varepsilon_q, \lambda, \#, \varepsilon_f, \lambda)$	
17	λ	λ		

8. DISCUSIÓN DE RESULTADOS

– **Razones para estudiar los lenguajes de programación**

- Facilitar el aprendizaje de un nuevo lenguaje.
- Hacer posible una mejor elección del lenguaje de programación a utilizar.
- Mejorar la habilidad para desarrollar algoritmos eficaces.
- Facilitar el diseño de un nuevo lenguaje.
- Mejorar el uso del lenguaje de programación disponible.
- Acrecentar el propio vocabulario con construcciones útiles sobre programación.

- Los lenguajes libres del contexto incluyen a los lenguajes regulares. Podemos deducir de ello que todo lenguaje regular es también lenguaje libre de contexto, no se cumple lo contrario pues, existen lenguajes libres de contexto que no son regulares, para este tipo de lenguajes no basta con un autómata finito, deberá utiizarse un autómata de pila, es por ello que un analizador sintáctico puede implementarse a través, de autómatas de pila ya sea en su forma LL, o en su forma LR.

Ventajas de utilizar un analizador sintáctico

- La gramática proporciona una especificación precisa, fácil de entender de un lenguaje de programación.
- A partir de algunas clases de gramáticas se puede construir automáticamente un analizador eficiente que determine si un programa fuente está sintácticamente bien formado.
- Los lenguajes evolucionan con el tiempo, adquiriendo nuevas construcciones y realizando tareas adicionales. Estas nuevas construcciones se pueden añadir con más facilidad a un lenguaje cuando existe una aplicación basada en una descripción gramatical.
- Una gramática imparte una estructura a un lenguaje de programación útil para la traducción de un programa fuente a código objeto correcto y para la detección de errores.
- El proceso de construcción del analizador sintáctico puede revelar ambigüedades sintácticas y otras construcciones difíciles de analizar que podrían pasar sin ser detectadas en la fase inicial del diseño de un lenguaje y de su compilador.

– Aunque falta mucho por analizar y contrastar sobre todo en cuestiones de implementación, podemos deducir que la colección de analizadores sintácticos LR es más poderosa que la de los analizadores LL, puesto que existen lenguajes libre del contexto que no puede ser analizados por un analizador sintáctico LL, aunque sí por un analizador LR.

– Una implantación de un analizador sintáctico utilizando tabla requiere un trabajo de análisis. Primero tiene que eliminarse la recursión por la izquierda si

fuera el caso. Luego tiene que hallarse los conjuntos PRIMERO y SIGUIENTE. Finalmente tiene que construirse la tabla de análisis sintáctico TAS.

- Existen lenguajes libres del contexto para los cuales no existe ningún analizador sintáctico LR.
- Los analizadores LR pueden analizar lenguajes independientes del contexto determinista, ya que está basado en un autómata de pila determinista.
- Resulta más conveniente una implementación mediante tabla de análisis sintáctico que una implantación mediante autómata de pila. La tabla de análisis integra toda la información necesaria que se requiere cuando se utiliza un autómata de pila y tenga que interrogarse qué símbolos existen en el TOP de la pila.
- Existe una familia de analizadores LL, contenidos en la familia de analizadores LL(K), asimismo existe una familia de analizadores LR, contenidos en la familia de analizadores LR(K), la variable K indica el número de símbolos de preanálisis necesarios para la derivación o reducción dependiendo de si es un análisis descendente o ascendente, respectivamente.
- Las gramáticas LL(k) están incluidas en las gramáticas LR(k)

REFERENCIAS BIBLIOGRÁFICAS

[1] Aho A.,Sethi,Ullman *Compiladores, principios, técnicas y herramientas*; Addison-Wesley1990, Wilmington-Delaware EUA.

[2] BBECK Leland *Software de Sistemas* Addisson Wesley iberoamericana Wilmington Delaware 1988

[3] BROOKSHEAR J. Glean *Teoría de la computación* Addisson Wesley iberoamericana Wilmington Delaware 1993.

[4] Cortez Vásquez, Augusto. *Matemáticas Discretas*, UNMSM EAPIS 1999.

[5] Cortez Vásquez, Augusto. *Lenguajes y traductores*, UNMSM FISI 2005.

[6] Deitel Harvey M. *Introducción a los sistemas operativos*; Addison-Wesley, Iberoamenricana 1987 Wilmigton Delaware EUA.

[7] Hopcroft Jhon, Ullman Jeffrey. *Introducción a la teoría de autómatas.* Edit. CECSA 1993.

[8] Johnsonbaugh Richard. *Matemáticas discretas*; Prentice Hall 1999.

[9] Kolman-Busby-Ross. *Matematicas Discretas*; Prentice Hall 1997.

[10] PEÑA MARI, Ricardo. *Diseño de programas*, *Formalismo y abstracción*. Prentice may - Madrid 1998.

[11] Terrence W. Pratt. *Lenguajes de programación, Diseño e implementación*; Prentice Hall Hispanoamericana 1988.

[12] SETHI, Ravi. *Lenguajes de programación, conceptos y Constructores*; Addison-Wesley, 1992.

[13] Teufel-Smithd-Teufel. *Compiladores, Conceptos fundamentales*; Addison-Wesley, 1990.