Método para el cálculo de símbolos directores en una gramática de contexto libre

Method for calculating director symbols in a free context grammar

Guillermo Roberto Solarte Martínez[1], Luis Eduardo Muñoz Guerrero[2], Camilo Muñoz Albornoz[3]

1.Guillermo Roberto Solarte Martínez, Doctor en Informática de la Universidad Pontificia de Salamanca con sede Madrid España Suficiencia investigativa, D.E. A Universidad Pontificia de Salamanca con sede Madrid España, Magister en Investigación de Operativa y Estadística de la Universidad Tecnológica de Pereira ORCID: https://orcid.org/0000-0001-5147-7798
roberto@utp.edu.co.
2. Luis Eduardo Muñoz Guerrero, Msc. En Ingeniería de Sistemas Universidad Nacional de Bogotá, profesor Asociado, de planta, del programa en ingeniería de Sistemas y Computación ORCID https://orcid.org/0000-0002-9414-6187
lemunozg@utp.edu.co.
3. Camilo Muñoz Albornoz, estudiante del programa de Ingeniera de Sistemas y Computación 4 semestre ORCID: https://orcid.org/0000-0001-7584-9904
camilo.munoz2@utp.edu.co

Recibido: 23/02/2020

Aceptado: 07/07/2020

Cite this article as: G. R. Solarte Martínez, L. E Muñoz Guerrero, C. Muñoz Albornoz. "Method for calculating director symbols in a free context grammar", Prospectiva, Vol 18, N° 2, 2020.

http:://doi.org/10.15665/rp.v18i2.2289

RESUMEN

El objetivo fundamental de esta investigación es desarrollar un método que permita al creador de lenguajes decidir si la gramática cumple con la condición necesaria y suficiente para que una gramática limpia sea del tipo LL1 especificando que los conjuntos de símbolos directores correspondientes a las diferentes expansiones de cada símbolo no terminal son disjuntos. Para alcanzar dicho objetivo es metodológicamente adecuado avanzar en varios aspectos bien diferenciados. Primero, el análisis y diseño de la aplicación se aborda dentro de las metodologías definidas en el campo de estructuras de datos, lenguajes de programación, inteligencia artificial, y algoritmia. En segunda instancia, es importante resaltar que existe un algoritmo para determinar este tipo de gramáticas. Para implementar dicho algoritmo en este método se sigue secuencialmente una serie de pasos que consumen bastante tiempo de análisis manual; por lo tanto, los pasos y los algoritmos implícitos de cada uno de ellos se sistematizan permitiendo disminuir el tiempo de los análisis, aumentar su eficiencia y al final verificar si la gramática es o no del de tipo LL1. Por último, las pruebas realizadas con diferentes tipos de gramáticas fueron contundentes, satisfactorias y reflejaron los resultados esperados.

Palabras clave: Gramáticas tipo LL1, estructuras de datos, producciones, símbolos terminales, símbolos no terminales.

ABSTRACT

The main objective of this investigation is to develop a method that allows the creator of languages to decide if the grammar fulfills with the necessary and enough condition for that a clean grammar be of the type LL1 specifying that the groups of managing symbols corresponding to the different expansions of each symbol non terminal are disjointed. To reach this objective it is methodologically appropriate to advance in several well differentiated aspects. First, the analysis and design of the application is approached inside the methodologies defined in the field of structures of data, programming languages, artificial intelligence, and algorithm. In second instance, it is important to highlight that an algorithm exists to determine this type of grammarians. To implement this algorithm in this tool a series of steps are followed sequentially that consume enough time of manual analysis; therefore, the steps and the implicit algorithms of each one are systematized allowing to diminish the time of the analyses, to increase their efficiency and at the end to verify if the grammar is or is not of type LL1, finally the tests carried out with different types of grammarians were convincing, satisfactory and they reflected the expected results.

Key Word — Grammatical type LL1, structures of data, productions, symbol's terminal, symbols non terminals.

1. INTRODUCCIÓN

La comunicación es un proceso de intercambio de información entre un emisor y un receptor a través de un canal haciendo uso de un lenguaje, el cual tiene sus propios signos y reglas, podemos encontrar lenguajes simples como la luz roja sobre la puerta de un estudio de revelado o más complejos como los distintos idiomas del mundo.

Por ejemplo, el idioma español tiene diferentes signos lingüísticos con los cuales se forman palabras, con ellas oraciones o mensajes infinitos que deben tener una correcta estructura sintáctica, pues no es lo mismo decir "el perro es de Juan" que decir "el Juan es del perro", su significado es diferente. Así mismo funcionan los lenguajes de programación, estos tienen sus reglas para que puedan ser entendidos por la computadora, para especificar dichas reglas se utilizan unas técnicas de notación.

La Notación de Backus Naur, es un metalenguaje, es decir, un lenguaje usado para describir aspectos propios de una lengua, que especifica la descripción compacta y precisa de la sintaxis de un lenguaje formal con símbolos y reglas [1]. A su vez, permite que no exista confusión en términos de lo que está o no permitido a la hora de su uso. Específicamente, esta notación tiene un planteamiento matemático formal, que describe el orden en el que se debe plantear un lenguaje de programación.

Historia de la BNF

Noam Chomsky maestro de lingüística en el instituto de tecnología en Massachussets, combinó la lingüística y las matemáticas, tomando esencialmente el formalismo de Axel Thue como la base de su descripción de la sintaxis del lenguaje natural. También introdujo una clara distinción entre reglas generativas (de la gramática libre de contexto) y reglas transformativas (1956).
Jhon Backus adopto las reglas de Chomsky para describir un nuevo lenguaje de programación conocido ahora como ALGOL 58, presentando en el primer Congreso de Computación Mundial (World Computer Congress) el artículo "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference".

Peter Naur, en su reporte sobre ALGOL 60 de 1963, identificó la notación de Backus como la Forma Normal de Backus (Backus Normal Form), y la simplificó para usar un conjunto de símbolos menor, entonces surgió la llamada Backus Nour form donde se remplazó Normal por Nour, como reconocimiento a la contribución que hizo Peter Nour, gracias a sugerencia de Donald Knuth [2]

¿Características de la BNF?

Sus características presentan tres elementos [3]
símbolos no-terminales: Son elementos que habrán de ser definidos por alguna regla, se encuentran entre paréntesis angulares (<>), además se definen usando combinaciones de símbolos terminales, no terminales y metasímbolos, para describir los constructores sintácticos del LP sujeto.
símbolos terminales: son los que se usan en el texto del programa tal como aparecen en la regla (sin las comillas), para describir los símbolos (texto) del LP sujeto
Los metas o símbolos BNF son:

::= de definición (el esquema de la derecha desarrolla el elemento de la izquierda).
| de alternativa (se puede elegir únicamente uno de los elementos que separa).
<> los paréntesis angulares (utilizados para rodear los nombres de las categorías).
Regla de producción para la BNF.

BNF es una especie de juego matemático que inicia con un símbolo llamado símbolo de inicio (por convención, se suele denominar S en los ejemplos) y luego se le dan las reglas por las que se puede sustituir este símbolo. El lenguaje definido por la gramática de BNF [4] es sólo el conjunto de todas las cadenas que puedes producir siguiendo estas reglas.

Las reglas se llaman reglas de producción, y se ven así:

símbolo: = alternativa1 | alternativa2.

la regla de fabricación simplemente establece que el símbolo a la izquierda de: = debe ser reemplazado por una de las alternativas a la derecha. Las alternativas están separadas por |s. (Una variación de esto es usar: = en lugar de: =, pero el significado es el mismo). Las alternativas suelen consistir tanto en símbolos no terminales como en símbolos terminales. Se llaman terminales porque no hay reglas de producción para ellos, terminan el proceso de producción. [5]

Los lenguajes de programación se definen usando una gramática, y esta a su vez se define como un conjunto de producciones, por lo cual se puede afirmar que un lenguaje es precisamente un conjunto de producciones y cada producción está compuesta de símbolos que son los elementos básicos de un lenguaje. La gramática tal y como se conoce en la teoría de lenguajes de programación fue creada por Chomsky en 1959 y está formada por los siguientes elementos: Símbolos terminales, símbolos no terminales y producciones, los cuales están orientados a facilitar el trabajo del desarrollador de lenguajes cuyo campo laboral sea el de crear, diseñar o mejorar lenguajes de programación para las diversas áreas de conocimiento.

En este trabajo investigativo se diseña un método eficiente para el cálculo de símbolos directores de una gramática que se puede aplicar a un contexto libre, y que le permite al creador de lenguajes decidir si la gramática cumple con la condición necesaria y suficiente para que sea una gramática del tipo LL1

La gramática LL1 [1] es el objeto de estudio de este trabajo y se pretende implementar un método para reconocer un tipo de lenguaje específico. La primera L de la abreviatura LL1 indica que la cadena o la instrucción se analiza de izquierda a derecha, la segunda L especifica que se ejecutan derivaciones a la

izquierda, y el número 1 aclara que para analizar la instrucción se necesita leer un solo token de línea de entrada. El método diseñado reconoce como una gramática LL1 aquella cuyos símbolos directores correspondientes a las siguientes expansiones de cada no terminal sean disjuntos.

Construir una gramática requiere un trabajo abstracto que depende del tipo de producciones y de cómo se derivan, aspectos que se tiene en cuenta para diseñar un lenguaje requerido y que permiten diferenciar los tipos de gramáticas [6] existentes.

2. MARCO DE REFERENCIA

Definiciones

SÍMBOLOS ACCESIBLE: es un símbolo que aparece en una cadena derivada del símbolo inicial. Es decir, aquel símbolo que no es inaccesible.
SÍMBOLOS DIRECTORES: son los que integran un conjunto ligado con una producción de un no terminal, los símbolos directores de las diferentes producciones de un mismo terminal son conjuntos disjuntos.
SÍMBOLOS EXTRAÑO: se denomina así a todo muerto o inaccesible.
SÍMBOLOS INACCESIBLE: es un símbolo no terminal al que no se puede llegar por medio de producciones desde el símbolo inicial.
SÍMBOLOS INICIALES: es el conjunto de símbolos terminales que aparecen al principio de las cadenas derivadas.
SÍMBOLOS MUERTOS: es un símbolo no terminal que no genera ninguna cadena de símbolos terminales.
SÍMBOLOS NO TERMINALES: (VN) es un conjunto finito de símbolos. Ellos imponen una estructura jerárquica sobre el lenguaje.
SÍMBOLOS TERMINALES: (VT) es un conjunto finito de símbolos.
SÍMBOLOS VIVOS: es un símbolo no terminal de la cual se puede derivar una cadena de símbolos terminales. Todos los símbolos terminales son vivos.
PRODUCCIONES: en una gramática especifican la manera en la cual los terminales y no terminales pueden ser combinados para formar una cadena de producciones.

2.1 Formulación del problema

Primero se define la condición necesaria y suficiente para que una gramática sea de tipo LL [7] es que los símbolos directores correspondientes a las diferentes expansiones de cada no terminal sean conjuntos disjuntos.

La condición es necesaria, puesto que si un símbolo aparece en dos conjuntos de símbolos directores no se sabrá qué tipo de analizador utilizar, en ese caso si definimos que tipo de gramática es de acuerdo con el método que se desarrolla permitiendo definir si pertenece al caso del tipo LL se puede escoger el tipo de analizador a utilizar.

El método que permite definir la solución del problema se implementa basándose en un algoritmo diseñado, que permite decidir si la gramática es o no LL1; la base consiste en determinar sencillamente los símbolos directores para cada expansión de un no terminal, de esta manera se puede definir si sus conjuntos son disjuntos y por tanto el tipo de gramática.

Para construir los conjuntos citados se ejecutan los siguientes pasos:

1. Encontrar los no terminales y las producciones que sean anulables.

$\xrightarrow{*}$

2.Construir la Relación "Empieza directamente con".
3.Calcular la Relación " Empieza con "
4.Calcular los iniciales de cada no terminal.
5. Calcular los iniciales de cada producción.
6.Construcción "Esta directamente seguido por ".
7.Construcción de la relación "Es fin directo de ".
8. Construcción de la relación "Es fin de ".
9.Construcción de la relación "Está seguido por ".
10. Anexar finalización "Está seguido por".
11.Seguidores de cada no terminal anulable,
12.Calcular los conjuntos de símbolos directores.

Estos pasos se realizan manualmente, por lo tanto, al diseñador de la gramática se le dificulta su construcción, le ocasiona problemas de tiempo, confusiones por la construcción manual de las diferentes tablas, produciendo posibles errores causados por la gran cantidad de información manejada.

La propuesta además de sistematizar los diferentes pasos mediante una herramienta muy eficiente ahorra tiempo en el desarrollo manual de los pasos definidos con anterioridad, al ser sistematizados se garantiza que cada uno de los pasos de respuestas libre posibles errores.

2.2 Conceptos básicos

2.2.1 Gramáticas LL (1)

Un analizador para una gramática LL(l) puede decidir en todo momento, qué acción o alternativa debe ejecutar a partir del token o símbolo más a la izquierda del resto del texto fuente. Es decir, un símbolo terminal debe corresponder únicamente a una producción de un determinado no terminal, y, por lo tanto, a cada producción le corresponde un único conjunto de símbolos terminales, los cuales se denominan **símbolos directores** de una producción.

INICIALES

Considerando las siguientes producciones para un símbolo

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots \mid \alpha_n \mid$$

Los símbolos directores para α_i estarán compuestos, entre otros, por todos los símbolos terminales que pueden aparecer a la izquierda de cualquier cadena generada por α_i. A este subconjunto de símbolos los denominan **símbolos iniciales,** el concepto de iniciales se aplica a una cadena α de una gramática libre de contexto y se define INICIALES (α) como el conjunto de símbolos terminales que aparecen al principio de las cadenas derivadas de α. Es decir,

INICIALES $(\alpha) = [\alpha \in VT \mid \alpha \rightarrow a\beta, \beta\epsilon (VT \cup VN)]$

Donde α es cualquier cadena, lo que quiere decir

$\alpha \epsilon (VT \cup VN)^*$

Tomando como ejemplo la siguiente gramática

<PROG> ::= module<DECLS>;<PROCS>end
<DECLS>::= d;<A>
<A> ::= <vacio> | <DECLS>
<PROCS>::= p <B>
<B> ::= <vacio> | ; <PROCS>

Observé que los iniciales de cada una de las cadenas que son las partes derechas de las producciones.
INICIALES
(module<DECLS>;<procs>end)= {module }
(d ; <A>) = { d }
(<vacio>) = { }
(<DECLS>) ={ d }
(p <B>) ={ p }
(<vacio>) = { }
(; <PROCS>) ={ ; }

Seguidores

Para seguir con este estudio, se introduce el concepto de **símbolos seguidores**. Dada una gramática libre de contexto con un símbolo inicial S, para un símbolo no terminal A, se dice que SEGUIDORES (A) es el conjunto de símbolos terminales que le pueden seguir inmediatamente en una cadena derivada de S. La definición formal sería:

SEGUIDORES (A)= [a|S→ βAΓ, βΓ ϵ (VT U VN)* ,
A ϵ VN, a ϵ INICIALES(Γ)]

El conjunto de símbolos seguidores del no terminal A de la gramática anterior es:
SEGUIDORES (<A>) = {p}
SEGUIDORES (<B>) = {end}

Cadena anulable

Antes de definir formalmente los símbolos directores, es preciso introducir el concepto de cadena anulable. Se dice que es una cadena **o** símbolos es una cadena anulable, si y sólo si

$\overset{*}{\rightarrow}$ a → <vacio>

Asimismo, se dice que una producción de una gramática es anulable, si y sólo si, su parte derecha es anulable.

Dada las producciones:
<A>::= <B> | c
<B> ::= <vacío> | <D> a b
<D> ::= d e

Las cadenas <A> y <B> son anulables, mientras que <D> y <D> a b son no anulables.

Símbolos directores

En este momento ya podemos definir el conjunto de símbolos directores. Dada una producción A → α donde α es una cadena de terminales y no terminales, se define como un conjunto de símbolos directores (SD):

$$SD(A, \alpha) = INICIALES(\alpha)$$

Si α es no anulable, y

$$SD(A, \alpha) = INICIALES(\alpha) \cup SEGUIDORES(A)$$

Si a es anulable. Veamos la definición formal:

$$SD(A, \alpha) = [a \mid a \in INICIALES(\alpha) \text{ OR } (A \rightarrow <vacío> \text{ AND } a \in SEGUIDORES(A)\ )\ ]$$

En la gramática anterior el conjunto de símbolos directores para la producción A → vacío es:

SD(<A>,<vació>) =INICIALES(<vacío>) ∪
SEGUIDORES(<A>) = { p }

Y los directores de la producción A → DECLS,

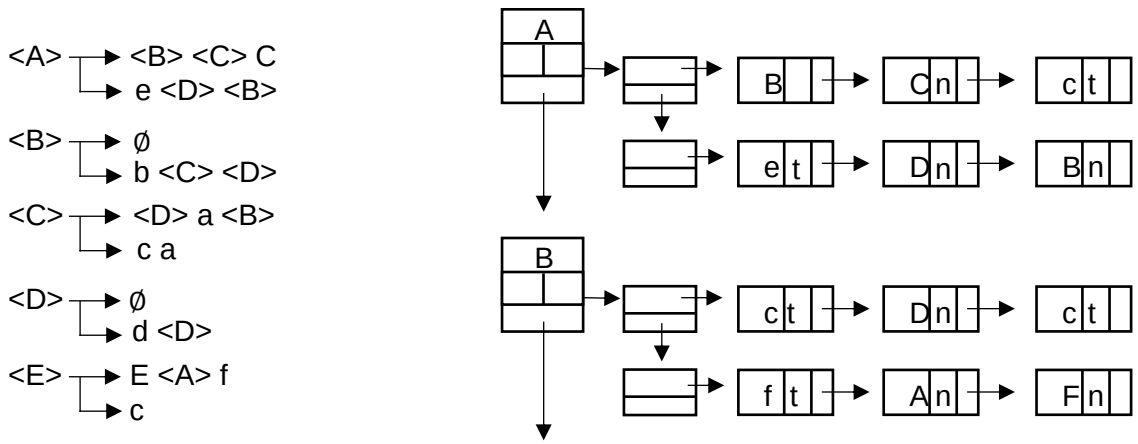
SD(<A>,<DECLS>)=INICIALES(<DECLS>) = {d}

Puesto que DECLS no es anulable.

3. METODOLOGÍA

Procedimiento para implementar el cálculo de símbolos directores estructura. de la gramática
La representación interna de una gramática con sus símbolos terminales, no terminales y las producciones está constituida mediante una multilista como se observa en la figura 1.

Figura 1. Estructura de la gramática



Fuente: Autores.

En la siguiente gramática realizaremos los doce pasos para el algoritmo de decisión.

Paso 1: Encontrar la lista de símbolos no terminales anulables.

Se obtienen sus símbolos no terminales anulables, que son <B> y <D>. Con

Paso 2: construir la relación "empieza directamente con"

Este paso consiste en la construcción de una matriz que refleja las relaciones "EMPIEZA DIRECTAMENTE CON" ver figura 2. Pero nos damos cuenta de que solo existe un 10 % de relaciones.

Figura 2. Empieza directamente con.

	<A>	<B>	<C>	<D>	<E>	a	b	c	d	e	f
<A>	1	1	1	1		1	1	1	1	1	
<B>		1					1				
<C>			1	1		1		1	1		
<D>				1					1		
<E>					1			1		1	
a						1					
b							1				
c								1			
d									1		
e										1	
f											1

Fuente: Autores.

Paso 3: Construcción de la relación "empieza con"

La relación EMPlEZA_CON es el cierre reflexivo transitivo de la relación EMPIEZA_ DIRECTAMENTE_CON, para lo cual se emplea el algoritmo de Warshall corno lo indica la figura 3:

Figura 3. Empieza con.

	<A>	<B>	<C>	<D>	<E>	a	b	c	d	e	f
<A>		1	1							1	
<B>							1				
<C>				1		1		1			
<D>									1		
<E>								1		1	
a											
b											
c											
d											
e											
f											

Fuente: Autores.

Paso 4: Construcción de los conjuntos iniciales de cada símbolo no terminal

INICIALES(<A>) = {a,b,c,d,e}
INICIALES(<B>) = {b}

INICIALES(<C>) = {a,c,d}
INICIALES(<D>) = {d}
INICIALES(<E>) = {c,e}
pero podemos representar

Paso 5: Construcción de los conjuntos de iniciales para cada regla de producción
El conjunto de iniciales de cada regla de producción está dado por el conjunto de iniciales del primer símbolo de cada regla, y su unión con los conjuntos de iniciales de los símbolos adyacentes en la regla, siempre y cuando todos los símbolos anteriores al adyacente sean anulables.

Paso 6: Construir la relación "es seguido directamente por"
Un símbolo s1 ES_SEGUIDO _DIRECTAMENTE_ POR. s2, si en la parte derecha de una producción, s2 se encuentra
inmediatamente después de s1 (de izquierda a derecha), o si entre s1 y s2 solo existen símbolos no terminales anulables. (Para saber cómo encontrar los símbolos no terminales anulables de la gramática, remítase al paso 1.)

Figura 4. Es seguido directamente por.

	<A>	<B>	<C>	<D>	<E>	a	b	c	d	e	f
<A>											
<B>			1								
<C>				1	1			1			
<D>		1			1	1					
<E>											
a		1									
b			1								
c						1					
d				1							
e	1	1		1							
f											

Fuente: Autores.

Paso 7: Construir la relación "es fin directo de"

Este paso es la contraparte del paso 2 (Construir la relación "Empieza directamente con") y es la construcción de la relación "Es fin directo de" que se define de la siguiente manera ver figura 5:

Figura 5. Es fin directo de.

	<A>	<B>	<C>	<D>	<E>	a	b	c	d	e	f
<A>											
<B>	1		1								
<C>											
<D>	1			1							
<E>		1									
a			1								
b											
c	1				1						

d				1							
e	1										
f					1						

Fuente: Autores.

Paso 8: Construir la relación "es fin de"

Este paso es la contraparte del paso 3 (Construir la relación "Empieza con") y es la construcción de la relación "Es fin de" que no es más que el cierre reflexivo-transitivo de la relación "Es fin directo de" que se calculó en el paso 7. Este cierre reflexivo-transitivo se realiza mediante la aplicación del algoritmo de Warshall sobre la matriz de la relación "Es fin directo de" como lo indica la figura 6:

Figura 6. Es fin de

	<A>	<B>	<C>	<D>	<E>	a	b	c	d	e	f
<A>	1										
<B>	1	1	1								
<C>			1								
<D>	1			1							
<E>	1	1	1		1						
a			1			1					
b							1				
c	1	1	1		1			1			
d	1			1					1		
e	1									1	
f	1	1	1		1						1

Paso 9: Construir la relación "es seguido por"

Se puede calcular la relación "Es seguido por", que está compuesta de las relaciones anteriores, si se realiza el producto de las matrices que representan a esas relaciones, de la siguiente manera como lo indica la figura 7:

ES FIN DE * (ES_SEGUIDO_DIRECTAMENTE_POR * EMPIEZA_CON)

Figura 7. Es seguido por

	<A>	<B>	<C>	<D>	<E>	a	b	c	d	e	f
<A>											1
<B>			1	1	1	1		1	1	1	1
<C>				1	1			1	1	1	
<D>		1			1	1	1	1		1	1
<E>			1	1	1	1		1	1	1	1
a		1		1	1		1	1	1	1	
b			1	1		1		1	1		
c			1	1	1	1		1	1	1	1
d		1		1	1	1	1	1	1	1	1
e	1	1	1	1		1	1	1	1	1	1
f			1	1	1	1		1	1	1	1

Fuente: Autores.

Paso 10: Anexar el símbolo de finalización

En esta etapa se desea agregar, en la matriz "Es seguido por", una marca a los símbolos de la gramática para indicar cuáles de ellos finalizan alguna cadena derivada del símbolo inicial <S>, de forma que se obtengan las cadenas de la forma <S>¬, donde "¬" es el símbolo de finalización como lo indica la figura 8:

Figura 8. Símbolo de finalización

	<A>	<B>	<C>	<D>	<E>	a	b	c	d	e	f	¬
<A>											1	1
<B>			1	1	1	1		1	1	1	1	1
<C>				1	1			1	1	1		
<D>		1			1	1	1	1		1	1	1
<E>			1	1	1	1		1	1	1	1	1
a		1		1	1		1	1	1	1		
b			1	1		1		1	1			
c			1	1	1	1		1	1	1	1	1
d		1		1	1	1	1	1	1	1	1	1
e	1	1	1	1		1	1	1	1	1	1	1
f			1	1	1	1		1	1	1	1	1

Fuente: Autores.

Paso 11.: Calcular el conjunto de seguidores de cada símbolo no terminal anulable

Este paso es el cálculo de los conjuntos de símbolos SEGUIDORES de cada símbolo no terminal anulable de la gramática. Este paso se lleva a cabo fácilmente una vez se ha calculado la relación "Es seguido por" y se ha agregado el símbolo de finalización. Los símbolos SEGUIDORES de un determinado símbolo no terminal anulable son aquellos símbolos terminales cuyas columnas tienen el número uno (1) en la fila del símbolo no terminal anulable, en la matriz que se obtiene en el paso 10 como se indica en la figura 9:

Figura 9. Seguidores

	<A>	<B>	<C>	<D>	<E>	a	B	c	d	e	f	¬
<A>											1	1
<B>			1	1	1	1		1	1	1	1	1
<C>				1	1			1	1	1		
<D>		1			1	1	1	1		1	1	1
<E>			1	1	1	1		1	1	1	1	1
a		1		1	1		1	1	1	1		
b			1	1		1		1	1			
c			1	1	1	1		1	1	1	1	1
d		1		1	1	1	1	1	1	1	1	1
e	1	1	1	1		1	1	1	1	1	1	1
f			1	1	1	1		1	1	1	1	1

Fuente: Autores.

SEGUIDORES(<B>) = {a, c, d, e, f, ¬}
SEGUIDORES(<D>) = {a, b, c, e, f, ¬}

Paso 12: construir los conjuntos de selección

Una vez se han obtenido los conjuntos de INICIALES de cada regla y los conjuntos de SEGUIDORES para cada símbolo no terminal anulable, se procede, con esa información, a construir los conjuntos de selección para cada regla de producción de la gramática, aplicando los siguientes criterios.

Iniciales de cada regla

INICIALES(<B> <C> c) = {a,b,c,d}
INICIALES(e <D> <B>) = {e}
INICIALES(∅) = {}
INICIALES(b <C> <D> <E>) = {b}
INICIALES(<D> a <B>) ={a,d}
INICIALES(c a) = {c}
INICIALES(∅) = {}
INICIALES(d <D>) = {d}
INICIALES(e <A> f) = {e}
INICIALES(c) = {c}

Seguidores de los no terminales anulables

SEGUIDORES(<B>) = {a, c, d, e, f, ¬}
SEGUIDORES(<D>) = {a, b, c, e, f, ¬}

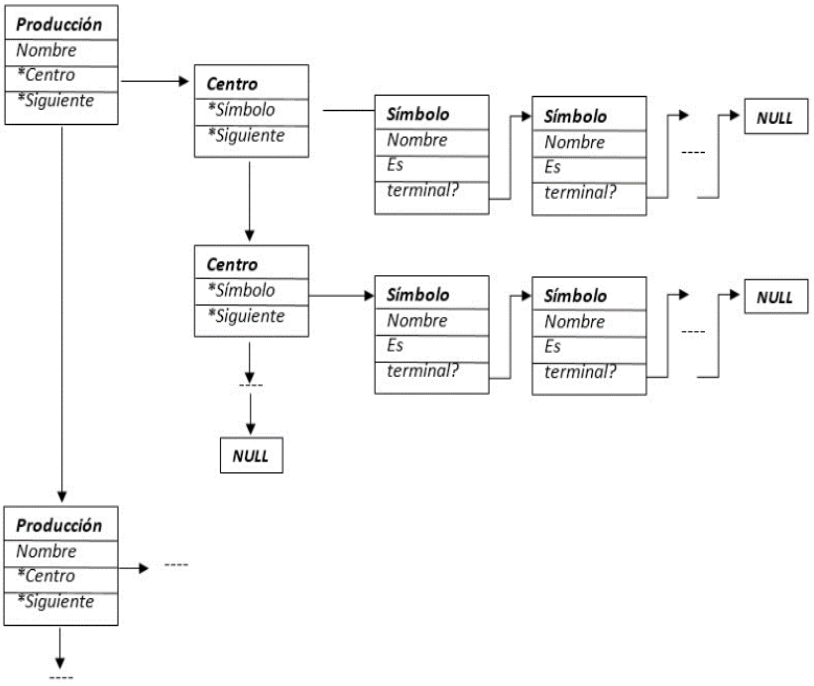
Para cada no terminal, los conjuntos de selección de las reglas de producción de un símbolo no terminal, son disjuntos, y esto sucede para todos los no terminales que tienen reglas de producción, entonces se dice que la gramática es de tipo LL1(1)

4. RESULTADOS

Los Metalenguajes BNF y EBNF [1] y [8] son útiles para orientar cómo se debe escribir en un lenguaje de programación, se constituyen en una especie de manual con sus reglas, apareciendo en la documentación de estos lenguajes. Por ello se debe tener claridad acerca de su manejo, pues un desconocimiento de la sintaxis de un lenguaje no permitiría utilizarlo de manera correcta.

Se desarrolló un método eficiente que permite calcular los símbolos directores para una gramática de contexto libre del tipo LL1, además se diseñó e implementó una nueva herramienta novedosa. Ver figura 10.

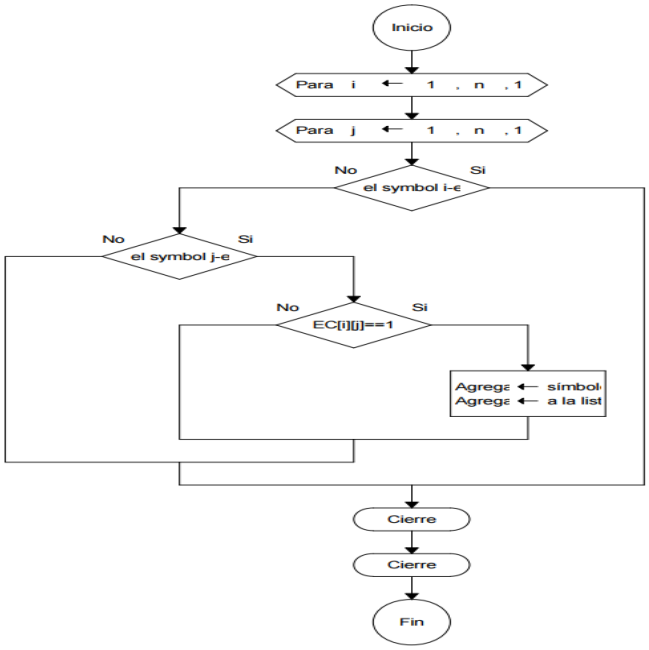
Figura 10. Símbolo de finalización



Fuente: Autores.

El método desarrollado se basa en un algoritmo de decisión para definir si una gramática es o no es LL1, mediante el cálculo de los símbolos directores. Ver figura 13.

Figura 11. Símbolo de finalización



Fuente: Autores.

Paso 1: Encontrar la lista de símbolos no terminales anulables.

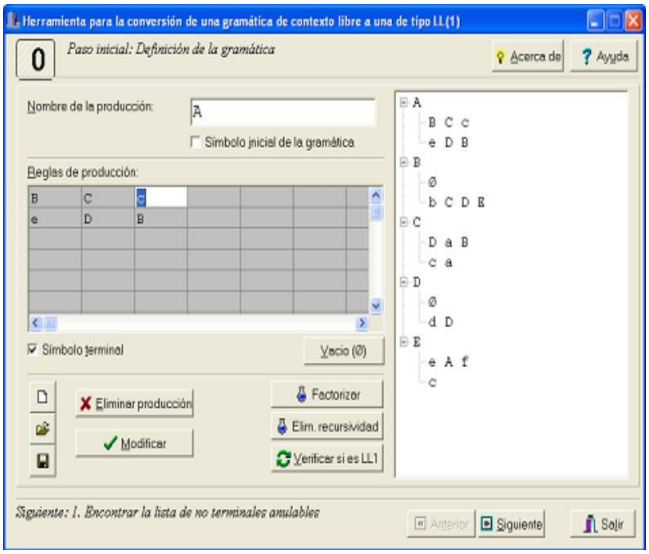
El algoritmo para determinar la lista de símbolos no terminales anulables es el siguiente:

1. Inspeccionar cada símbolo de cada regla de producción, de cada producción, empleando la siguiente convención:
   - Si se encuentra un Ø (vacío), marcar la producción con un 0 y verificar la regla de producción siguiente.
   - Si solo se encontraron no terminales en la parte derecha, marcar con 1, pero si todos los símbolos de la regla están en la lista de anulables que se está construyendo, entonces esta producción también es anulable y se debe marcar con 0.
   - Si se encuentra por lo menos un símbolo terminal, y la producción no ha sido marcada con 0 o 1, marcar con 2.
2. Si después de revisar todas las reglas de la producción, la producción está marcada con 0, agregarla a lista de anulables.

3. Volver al paso 1 si en esta revisión de la gramática se agregó una producción a la lista de anulables, ya que esto podría hacer que una de las producciones de la gramática se convierta en anulable, si la producción recientemente agregada forma parte de una de sus reglas.

El resultado final (método y software) es una herramienta que permite gran interactividad con el usuario por su fácil manejo, debido a que cada paso se presenta en forma didáctica, es decir se usan diagramas y tablas que muestran claramente los resultados obtenidos como se muestran en las figuras anteriores. Ver figura 12.

Figura 12. Símbolo de finalización



Fuente: Autores.

El método es relevante ya que coloca en práctica los conceptos teóricos de compiladores (lenguajes formales), aportando así al desarrollo y aplicación del conocimiento en este campo de la ingeniería de sistemas.

Comparación con otras herramientas

- Existen diferentes herramientas creadas para ayudar al aprendizaje de la materia de compiladores. A continuación, se presenta un análisis comparativo de las características propias del software creado denominado Gramática LL1 y las características de cada herramienta encontrada. Ver Tabla 1.

Tabla 1. Comparación de herramientas

Características	LL1	CSF	LEX	BISON	YACC	FLEX	LALR	JFLAP
Interfaz Grafica	S	S	N	N	N	S	S	S
Lectura Gramática	S	S	S	N	S	S	S	S
Lectura de Cadena	S	S	S	S	S	S	S	S
Usabilidad	S	S	N	N	N	S	S	S
Identificación de gramática LL1	S	N	N	N	N	N	N	N
símbolos directores	S	N	N	N	N	S	S	S
Metalenguajes BNF e BNF	S	N	S	N	S	S	S	S

Fuente: Autores

Lectura gramática. El programa es capaz de leer un archivo externo que contenga la gramática a analizar.

Lectura cadena. La herramienta está capacitada para leer un archivo externo que contenga las distintas cadenas que se van a analizar.

Interfaz gráfica: modo de visualización y manejo de herramienta

Usabilidad: facilidad de entender y uso de la aplicación

Identificación de gramática tipo LL1: Reconocimiento de un tipo de lenguaje específico Indica que la cadena o la instrucción se analiza de izquierda a derecha.
Identificación de signos directores: son los que integran un conjunto ligado con una producción de un no terminal, los símbolos directores de las diferentes producciones de un mismo terminal son conjuntos disjuntos.

Metalenguajes BNF y EBNF: Son útiles para orientar cómo se debe escribir en un lenguaje de programación, se constituyen en una especie de manual con sus reglas, apareciendo en la documentación de estos lenguajes.

Como se puede observar entre las aplicaciones que más se asemejan a la desarrollada en este artículo denominada Gramática LL1 son las herramientas FLEX [9], LALR [10] ,JFLAP [11], derivaciones de las herramientas de compilación LEX, BISON e YACC [12]

Se puede evidenciar que las herramientas que se asemejan a la aplicación Gramática LL1 carecen de identificadores de tipo LL1, característica que permite catalogarla como una herramienta novedosa en estos campos.

La aplicación gramática LL1 utiliza usabilidad e interfaz gráfica para que el estudiante tenga una forma de interpretación más adecuada, mediante la utilización de diagramas y tablas que muestran claramente los resultados obtenidos Ver figura 12. Además, si medimos el tiempo de ejecución y complejidad algorítmica,

el programa tiene una complejidad computacional n^2, que asegura que entregue resultados en tiempo y consumo de memoria razonables

5. CONCLUSIONES

- Se diseñó un modelo computacional para determinar si una gramática de contexto libre es de tipo LL1.
- Se implementó los 12 pasos del modelo computacional basados en estructura de datos complejas como listas y matrices dispersas para los diferentes algoritmos de decisión.
- Una ventaja de este modelo computacional es que para su implementación se tuvieron en cuenta aspectos como el concepto de calidad de un software (referencia), la usabilidad y accesibilidad, permitiendo así una comunicación amigable entre usuario y la aplicación.
- La utilización de Metalenguajes BNF y EBNF sirve para orientar cómo se debe escribir en un lenguaje de programación, es prácticamente un manual con sus reglas, que ayuda a tener claridad sobre estos metalenguajes, pues el desconocimiento de la sintaxis de un lenguaje no permitiría utilizarlo de manera correcta.
- Se ha aprendido a crear un identificador de gramáticas de tipo LL1, basados en un analizador descendente, proceso que sirvió para para explorar e implementar las fases y los elementos necesarios para analizar una gramática.
- La creación de esta herramienta sirve para los docentes y estudiantes de compiladores de Ingeniería en Sistema, puesto que agiliza y mejora el proceso de aprendizaje y enseñanza de los analizadores sintácticos a través de las TIC. (Tecnologías de la Información y la Comunicación)
- Se realizaron diferentes pruebas con otros tipos de gramáticas, arrojando resultados contundentes, satisfactorios y cumplieron con la complejidad temporal y espacial requerida para este tipo de problema.

AGRADECIMIENTOS

Los autores desean agradecer la colaboración del Ingeniero Luís Roberto Ojeda en la elaboración del método PARA EL CALCULO DE SIMBOLOS DIRECTORES EN UNA GRAMATICA DE CONTEXTO LIBRE, por sus aportes para el diseño de cada uno de los algoritmos que permiten obtener los resultados finales.

Conflicto de intereses

Los autores declaran no tener ningún conflicto de intereses

REFERENCIAS

[1] J. A. Valverde y G. Sánchez Dueñas, “Compiladores e intérpretes un enfoque pragmático,” Madrid, Editorial Díaz Santos, 1988, p. 375.

[2] C. Becerra Santamaría, “Estructura de datos en C++,” de Estructura de datos en C++, Bogota, Kimpres, 2002.

[3] L. M. Garshol, “Gestion de Base de Datos,” 2008. [En línea]. Available: http://www.garshol.priv.no/download/text/bnf.html.

[4] F. Pérez, “Gestión de Bases de Datos,” de bnf notación backus-naur-form para la especificación de sintaxis de lenguajes formales, 2010.

[5] “Aprender la Programación Orientada a Objetos con el lenguaje Java,” de Aprender la Programación Orientada a Objetos con el lenguaje Java, ENI, 2019, p. 387.

[6] J. Vélez Reyes, “Comunidad de Programadores,” 03 03 2014. [En línea]. Available: https://www.lawebdelprogramador.com/pdf/16514-Analisis-sintactico-Gramaticas-libres-de-contexto.html. [Último acceso: 23 febrero 2020].

[7] I. Bosque Muñoz y J. Gutiérrez-Rexach, “Fundamentos de Sintaxis Formal,” de Fundamentos de Sintaxis Formal, Madrid, Akal, 2009, p. 352.

[8] C. M. Zapata Jaramillo, R. E. Arango Sanchez, y L. Jiménez Pinzón, , “Mejoramiento de la consistencia entre la sintaxis textual y gráfica del lenguaje de Semat,” Sistema de Información Científica, vol. 49, nº 12, pp. 83-99, 2014.

[9] F. Tomassetti, “EBNF: How to Describe the Grammar of a Language,” https://tomassetti.me/, agosto 2017. [En línea]. Available: https://tomassetti.me/ebnf/. [Último acceso: 23 3 2020]

[10] L. F. Molina, “Generador de compiladores basado enanalizadores ascendentes,” 18 10 2011.https://ddd.uab.cat/pub/trerecpro/2009/hdl_2072_43845/PFC_LaiaFelipMolina.pdf. [Último acceso: 1 8 2020].

[11] . A. P. Natalia, “Universidad Carlos III de Madrid. Departamento de Informática,” 03 07 2015. [En línea]. Available: http://hdl.handle.net/10016/23149. [Último acceso: 1 8 2020].

[12] . G. R. Sergio y N. M. miguel ángel, Traductores y compiladores con LEX/YACC, JFLEX/CUP Y JAVACC., Madrid, 2005.

[13] J. A. Valverde y G. Sánchez Dueñas, “Compiladores e intérpretes un enfoque pragmático,” Madrid, Editorial Díaz Santos, 1988, p. 375.