ANEXO B

ESTRUCTURAS DE LOS LENGUAJES DE PROGRAMACION

Según los autores Terrence W. Pratt y Marvin V. Zelkowitz en el libro Lenguaje de programación. Diseño e implementación. 3ª Edición. 2001, se debe tener en cuenta las siguientes elementos para trabajar con un lenguaje de programación:

Elementos sintácticos que el lenguaje va a manejar:

1. Conjunto de caracteres

- Es lo primero que se hace al proyectar una sintaxis de lenguaje.
- El código ASCII es un conjunto de caracteres especiales.

2. identificadores

- Mediante las tres primeras letras, identificar cada uno de ellos.
- Restringir la longitud

3. Símbolos de operadores

- Debemos emplear caracteres especiales para representar las operaciones básicas.

4. Palabras claves y palabras reservadas

- Palabra clave, es un identificador que se usa como parte de la sintaxis
- Una palabra clave es una palabra reservada, si no se puede usar también como un identificador.
- El manejo de palabras reservadas facilita la etapa de traducción. No pueden utilizarse como nombres de variables.

5. Palabras pregonadas

- Son palabras opcionales que se insertan en los enunciados para mejorar la legibilidad -

- Algunas son opcionales

6. Comentarios

- Son muy importantes para la documentación

7. Espacios en blanco

- Esta regla varia dependiendo del lenguaje.
- En algunos casos, no son significativos. En otros casos si.
- Algunos son separadores, concatenadores...etc

8. Delimitadores y corchetes

- El delimitador se utiliza para señalar el principio o el fin de una unidad sintáctica.

9. Formatos de campo libres y fijos

- Cuando los enunciados se pueden escribir en cualquier parte de un renglón sin que importe la posición sobre el renglón se les denomina sintaxis de campo libre.
- Cuando la sintaxis es de campo fijo se utiliza una posición fija del renglón para colocar los elementos de un enunciado.

Atributos de un buen lenguaje

1. Claridad, sencillez y unidad,

- Un lenguaje nos debe ayudar a poder expresar nuestros algoritmos
- Mucho antes de codificar, el lenguaje le debe ayudar al programador
- Los conceptos que tiene, deben ser claros y sencillos para que se nos facilite la implementación de los algoritmos.
- La sintaxis nos permite escribir, entender y modificar con mayor facilidad o dificultad nuestros programas

-Una sintaxis "hermética" le facilita el trabajo al programador pero dificulta la labor de tratar de entender (lectura).

- Algunos manejan una sintaxis errónea donde lo que se lee no es igual a lo que significa

2. Ortogonalidad

- Es el atributo de ser capaz de combinar varias características de un leguaje en todas las maneras posibles.
- No se debe perder el significado.
- Si son ortogonales, tendremos menos excepciones y casos que recordar.
- Los programas suelen compilar sin errores (esto causa molestia en algunos casos).

3. Naturalidad para la aplicación

- El lenguaje nos debe proporcionar estructuras de datos, operaciones, estructuras de control, y una sintaxis apropiada para resolver el problema.
- Los lenguajes deberían estar orientados a una clase de aplicaciones, simplificando la creación de programas.

4. Apoyo para la abstracción

- El programador debe estar en capacidad de hacer una abstracción para la solución del problema y luego implementarla empleando lo que le brinda el lenguaje.
- El programador debe conocer las propiedades básicas de lo que le proporciona el lenguaje sin preocuparse por su implementación.

5. Facilidad para verificar programas

- El lenguaje nos debe permitir verificar la validez de un programa.

6. Entorno de programación

- La presencia de un entorno adecuado, nos facilita el trabajo
- Disponibilidad de una implementación confiable
- Editores especiales
- Paquetes de pruebas
- Recursos de mantenimiento

7. Portabilidad de programas

- Si un programa es ampliamente disponible e independiente de la maquina, se podrá

decir que es portable.

8. Costo de uso

- El costo es un elemento importante en la evaluación de cualquier lenguaje y se debe tener en cuenta:

a. Costo de ejecución del programa,

En comienzo, las cuestiones de costo se referían exclusivamente a la ejecución de los programas.

Era importante el diseño de los compiladores para ser mas eficientes.

Este es muy importante para grandes programas de producción que se van a ejecutar con frecuencia.

Pero, para máquinas de "escritorio" no es de mucha preocupación.

b. Costo de traducción de programas

Para lenguajes de enseñanza, la cuestión de traducción es de mucha mas importancia.

Estos se compilan muchísimas veces pero se ejecutan solo unas pocas veces.

c. Costo de creación, prueba y uso de programas

Se deben buscar programas que minimicen el tiempo y esfuerzo total que se invierte en la solución de un problema en la computadora.

d. Costo de mantenimiento de los programas

En esta parte esta el mayor costo que interviene en cualquier programa. En el mantenimiento esta: Reparación de errores (después de usar el programa), cambios en el programa, mejoras y extensiones.

Un programa que facilite todo esto puede ser mucho menos costoso

Sintaxis de lenguajes de programación

- La sintaxis describe la serie de símbolos que constituyen programas válidos.

- La sintaxis nos da información que se necesita para entender un programa
- Nos permite tener información para hacer la traducción del programa fuente a un programa objeto.
- El solo desarrollo de una sintaxis no es suficiente para especificar sin ambigüedad la estructura de un enunciado.
- Se necesita algo más que la estructura sintáctica para la plena. descripción de un lenguaje de programación.
- El termino de semántica incluye, uso de declaraciones, operaciones, control de secuencia, entornos de refinamiento... afectan a una variable y no siempre están determinados por reglas de sintaxis.

Criterios generales de sintaxis

- El propósito de la sintaxis es proveer una notación para la comunicación entre el programador y el procesador de lenguaje de programación.
- Otro es que se puedan clasificar para hacer que los programas sean fáciles de leer, escribir, traducir y no ambiguos

1. legibilidad, Que sea entendible sin documentación independiente
Esta se mejora con:
- formatos naturales de enunciados
- enunciados estructurados
- palabras clave
- palabras pregonadas
- comentarios incrustados

la legibilidad no se puede garantizar mediante el diseño de un lenguaje.
Ya que hasta el mejor se daña con una programación deficiente.

- las construcciones que son similares se ven parecidas, las que hacen cosas diferentes se ven distintas
- Entre mas construcciones sintácticas mas fácil es la estructura del programa.
- Entre menos tenga, esto conduce a programas menos legibles

2. facilidad de escritura,

- Las características que hacen que un lenguaje sea fácil de escribir se encuentran en conflicto con las que mejoran su lectura
- Mejora a través del uso de estructuras sintácticas concisas y regulares
- Algunas son implícitas permitiendo así que los programas sean mas cortos pero difíciles de entender.
- Una sintaxis es redundante si comunica el mismo elemento de información en más de una forma.

Esto hace los programas mas elocuentes, dificultando su escritura (colocando una palabra especial al comienzo de una variable)

3. facilidad de verificación

- Este aspecto tiene relación con la legibilidad y facilidad de escritura
- Se necesitan técnicas que permitan probar la validez de un programa ya que es muy difícil hacer programas correctos.

4. facilidad de traducción

- Estos deben ser fáciles de traducir a una forma ejecutable
- Se relaciona con las necesidades del traductor para procesar el programa escrito
- La clave es la regularidad de la estructura
- LISP nos proporciona una estructura fácil de traducir

5. Carencia de ambigüedad

- Este es un problema en todo diseño del lenguaje

- Una construcción ambigua permite dos o mas interpretaciones distintas.

"Realmente el programador es quien determina si el lenguaje vive o muere".

2. Análisis Léxico

La función principal de este análisis es leer los caracteres de la frase (o del código fuente de un lenguaje) y elaborar como salida una secuencia de **componentes léxicos,** que utiliza el analizador sintáctico para hacer el análisis. Las técnicas de análisis léxico se utilizan en otras áreas como: Lenguajes consulta y sistemas de recuperación de información.

Puede realizar funciones secundarias como eliminar del programa fuente caracteres en blanco, comentarios, caracteres TAB y relaciona los mensajes de error.

2.1 Componentes Léxicos

Componente léxico: en la mayoría de los lenguajes de programación se consideran como componentes léxicos: palabras clave, operadores, identificadores, constantes, cadenas literales y signos de puntuación: paréntesis **)**, coma **,** y el punto y coma**;.**

3. Análisis Sintáctico

Introducción

Una de las situaciones más complejas en el desarrollo de un lenguaje es el establecimiento de las reglas gramaticales que éste debe cumplir. La bondad o tortura que puede ser el emplear un lenguaje determinado se debe, en parte, a la facilidad de

describir situaciones por medio del lenguaje que representen un conjunto de acciones específicas. Contrariamente a los lenguajes naturales, los lenguajes computacionales son más restrictivos en cuanto a su capacidad de expresión ya que, a diferencia de un idioma, una computadora no debe entender más que de una sola manera lo que un programador escribe. Se dice que las gramáticas que soportan a los lenguajes de cómputo deben ser independientes al contexto o forma en que se emiten sus enunciados (no importa quien lo emita, en qué momento o con qué entonación: debe significar siempre lo mismo).
Como diseñadores de lenguajes nos encontramos con seguir el paradigma del lenguaje que usamos más o con el que sentimos mayor agrado; la forma de describir una gramática está dada por la notación BNF, que trata primordialmente la construcción de gramáticas libres al contexto. Esta notación está basada en trabajos de **Noam Chomsky** y se ha usado extensivamente para declarar las gramáticas de la mayoría de los lenguajes que empleamos hoy en día.

A partir de unas clases de gramáticas se puede construir automáticamente un analizador sintáctico eficiente que determine si un programa fuente está sintácticamente bien formado

Lenguajes formales y gramáticas formales.

Un lenguaje se puede definir como un conjunto de secuencias aceptado por alguna clase de reconocedor, tales como la máquina de estado finito o las máquinas de pila. El término gramática formal es aplicado a cualquier especificación de lenguaje formal basado en reglas gramaticales a partir de las cuales la secuencias pueden ser generadas o analizadas. Una gramática da una especificación sintáctica precisa y fácil de entender un lenguaje de programación.

Ejemplo de gramática formal:

¿Qué es una oración? ¿Qué elementos posee una oración?

<oración>

<sujeto>

<verbo>

<complemento>

<nombre>

<artículo>

¿Cómo se construye una oración en inglés?

The boy sees girl

artículo nombre verbo nombre

La gramática formal sería:

1. <oración> -> <sujeto> <verbo> <complemento>.
2. <sujeto> -> <artículo> <nombre>
3. <complemento> -> <artículo> <nombre>
4. <verbo> -> see
5. 5. <artículo> -> the
6. 6. <nombre> -> boy
7. 7. <nombre> -> girl

De ahí se derivan oraciones. Ejemplo. Derivar la frase "The boy see the girl.".

<oracion> -> <sujeto><verbo><complemento>.

(2)

<oracion> -> <articulo><nombre><verbo><complemento>.

(3)

<oracion> -> <articulo><nombre><verbo><articulo><nombre>.

(5) (5)

<oracion> -> <the><nombre><verbo>the<nombre>.

(6) (4) (7)

<oracion> -> the boy see the girl

Se debe aplicar las reglas una a una, ya que esto implica manejo de pilas, árboles y colas. Esta derivación se muestra en el siguiente árbol sintáctico:

El árbol sintáctico demuestra que la secuencia obtenida es independiente del orden en la cual las sustituciones son realizadas para entidades intermedias.

Ejemplo. Elaborar una gramática formal de una frase en español.

1. <frase> -> <sujeto><predicado><punto>
2. <sujeto> -> <sustantivo>
3. <sustantativo> -> María
4. <sutantivo> -> Juan
5. <predicado> -> <verbo intransitivo>
6. <predicado> -> <verbo transitivo><objeto>
7. <verbo intransitivo> -> patinar
8. <verbo intransitivo> -> golpear
9. <verbo transitivo> -> quiere
10. <objeto> ->a <sustantivo>
11. <punto> -> **.**

3.1. Gramática Independiente de Contexto

Una gramática independiente de contexto es un método independiente de contexto bastante poderoso para describir casi todas las características sintácticas de la programación de lenguajes. A diferencia de las gramáticas regulares, estas gramáticas no tienen restricciones con respecto a la forma del lado derecho de sus reglas de reescritura, aunque aun se requiere que el lado izquierdo de cada regla sea un solo no terminal.

Especificaciones:

Esta gramática se especifica como:

1. Un conjunto finito de no terminales, las cuales son variables sintácticas que denotan conjuntos de cadenas. (son los que se encuentran entre los siguientes símbolos < >)

Estos no terminales definen conjuntos de cadenas que ayudan a determinar el lenguaje generado por la gramática. Además imponen una estructura jerárquica sobre el lenguaje que es útil tanto para el análisis sintáctico como para la traducción

1. Un conjunto finito de terminales a los cuales se llega desde los no terminales. Son los símbolos que forman las cadenas. Los componentes léxicos son sinónimos de terminales cuando se habla de lenguajes de programación.

Ejemplo: Mario, Quiere, If, Else

1. Un conjunto finito de producciones de la forma:

<A> ->μ A: no terminal

μ: secuencia de terminales y no terminales, incluyendo nulo.

Los cuales se combinan para formar cadenas.

Ejemplo: a <sustantivo> Quiere

μ μ

1. Un símbolo inicial el cual es uno de los no terminales. Este caso es <frase> (a partir de esta inicia toda la gramática)

La gramática independientemente del contexto no tiene restricciones con respecto a la forma del lado derecho de las reglas de reescritura.

Se requiere que el lado izquierdo de cada regla o producción sea un solo no terminal.

¿Por qué se llama independiente de contexto? El hecho que el lado izquierdo de una producción contenga un solo no terminal , permite que la producción pueda aplicarse sin importar el contexto donde se encuentre dicho no terminal. Es aplicar una regla libremente en cualquier parte.

Otra forma de representación:

1. S -> aAbS
2. S -> b
3. A -> SAs
4. A -> €

Mayúsculas: no terminales Minúsculas: terminales

Nota: si <A> -> µ1, <A> -> µ2, ... , <A>->µn

Se puede representar como: <A>-> µ1| µ2| ...|µn ,

Las reglas de una gramática son usadas para definir una clase especial de sustituciones de cadenas. En este punto se considera una producción como una regla de reescritura donde el no terminal de la izquierda es sustituido por la cadena del lado derecho de la producción.

1. <S> -> a <A> <B> c
2. <S> -> € (cadena nula)
3. <A> -> c <S> <B>

4. <A> -> <A> b
5. <B> -> b <B>
6. <B> -> a

Generalmente 1 es la producción inicial.
<S> -> a <A> <B> c

La derivación depende de lo que se va a reconocer.

Una serie de situaciones se denominan una derivación. Para llegar a la cadena a c a b a c ¿Cuál es el conjunto de situaciones que se obtienen?

Ejemplo: Qué derivaciones se debería hacer para obtener: a c a b a c

3.2. Árbol de Análisis Sintáctico

Un árbol se puede considerar como una representación gráfica de una derivación de una cadena dentro del lenguaje libre de contexto. También se denomina árbol de derivación.

Características:

- Cada nodo interior de un árbol se etiqueta con un no terminal. Los hijos de ese nodo se etiquetan de izquierda a derecha, con los símbolos del lado derecho de la producción.

- Las hojas del árbol de análisis sintáctico se etiquetan con terminales y no terminales y leídas de izquierda a derecha constituyen una forma de frase que también se la conoce como frontera de árbol.

- Ningún símbolo terminal puede ser un nodo interior del árbol y ningún símbolo no terminal puede ser una hoja cuando se ha concluido el reconocimiento de toda la cadena.

- La raíz del árbol es el símbolo inicial no terminal.

1. <S>-> a <A> <B> c
2. <S>->€
3. <A>-> c <S> <B>
4. <A>-> <A>b
5. <B>-> b <B>
6. <B>-> a

El árbol no suministra información a cerca del orden en el cual las producciones o reglas fueron aplicadas.

Tipos de derivación

Existen diferentes formas de realizar la derivación de una cadena entre las más conocidas están, la derivación por la izquierda y la derivación por la derecha porque facilitan el manejo sistemático de una gramática.

Derivación por la izquierda: consiste en sustituir el símbolo no terminal de más a la izquierda de la cadena resultante.

Ejemplo:

<S>-> a <A> <B> c

4

a <A> b <B> c

3

a c <S> <B> b<B> c

2

a c <B> b <B> c

6

a c a b <B> c

6

a c a b c

Derivación por la derecha: se sustituye siempre al no terminal de más a la derecha de la cadena resultante.

Ejemplo:

<S>->a <A> <B> c

6

-> a <A> a c

4

a <A> b a c

3

-> a c <S><B> b a c

6

a c <S> a b a c

2

a c a b a c

Conclusiones de la derivación

1. Para cada cadena derivada desde una gramática de contexto dada, hay uno o más árboles de derivación.

1. Para cada árbol de derivación hay una o más derivaciones.

Ejercicio: hacer derivación por la izquierda y derecha para obtener z a z a b z b z

1. <S>-> z <M><N>z
2. <M>-> a <M> a
3. <M>-> z
4. <N>-> b <N> b
5. <N>-> z

1. Para cada árbol de derivación hay una sola derivación por la derecha y una sola derivación por la izquierda.

1. Si cada cadena derivada de una gramática de contexto tiene sólo un árbol de derivación se dice que la gramática no es ambigua, de lo contrario, la gramática es ambigua.

Ejercicio1. Obtener la gramática que genere la secuencia $\{X^nY^n, n \in N\}$

Método 1:

$\{X^nY^n, n > 0\}$

1. <S> -> x <S> y
2. <S> -> €

$\{X^nY^n, n > 0\}$

Método 2:

<S> -> x <B> y

<B> -> €

<B> -> x <B> y

Ejercicio 2. Crear la gramática que genere un número entero sin signo.

d es cualquier dígito de 0...9

Método 1:

<S> -> d <B>

<B> -> €

<B> -> d <B>

Método 2:

<S> -> d <S>

<B> -> d

Ejercicio 3. Realizar una gramática que permita definir constantes numéricas de la forma: 3.21E-21,

3.21, 3. , .3, 4 , 3.21 E21, 3.21E+21, es decir, números decimales sin E y con E.

1. <número decimal> -> <constante decimal>
2. <número decimal> -> <constante decimal> E <entero>
3. <entero> -> + <entero sin signo>
4. <entero> -> - <entero sin signo>
5. <entero> -> <entero sin signo>
6. <entero sin signo> -> d <entero sin signo>
7. <entero sin signo> -> d
8. <constante decimal> -> <entero sin signo> . <entero sin signo>
9. <constante decimal> -> <entero sin signo>.
10. <constante decimal> -> .<entero sin signo>
11. <constante decimal> -> <entero sin signo>
12. <constante decimal> -> - <constante decimal>

donde d es un dígito arbitrario.

Ejercicio 4. Crear una gramática que genere instrucciones de la forma:
(átomo, ((átomo, átomo), átomo)) ó atomo, es decir, parejas de átomo entre paréntesis, separadas por coma (se pueden subagrupar parejas).

1. <S> -> átomo
2. <S> -> (<S>. <S>)

Derivación por la izquierda:

<S> -> (<S>. <S>)
1
<S> -> (átomo . <S>)
2
-> (átomo . ((<S>.<S>)
2
-> (átomo ((<S>.<S>) . <S>)
1
-> (átomo . ((átomo . átomo) . <S>))
1
-> (átomo . ((átomo . átomo) . átomo))

Ejercicio 5. Gramática que permita definir expresiones aritméticas con operadores +, * , y utilice ( ). Ejemplo: 1 + 2 * 3 + (5 + 6) * 7

Método 1:

1. <E> -> <E> + <T>
2. <E> -> <T>
3. <T> -> <T>*<P>
4. <T> -> <P>

5. <P> -> <P><F>
6. <F> -> <P>
7. <P> -> (<E>)
8. <P> -> I

Método 2:

<E> -> <E> <OP> <E>

<E> -> (<E>)

<E> -> - <E>

<OP> -> +

<OP> -> -

<OP> ->

<OP> -> -

<E> -> I

Donde I representa un entero

Método 3:

1. <E> -> <T> <E-lista>
2. <E-Lista> -> + <T> <E-lista>
3. <E-Lista> -> €
4. <T> -> <F> <T-lista>
5. <T-Lista> -> * <F> <T-lista>
6. <T-Lista> -> €
7. <F> -> <P> <F-lista>
8. <F-Lista >-> <P> <F-lista>
9. <F-Lista> -> €
10. <P> -> (<E>)
11. <P> -> I

4. Conjuntos Regulares como Gramáticas Libres de Contexto

Toda construcción que se pueda escribir mediante una expresión regular, también se puede describir por medio de una gramática. Una expresión regular es aquella que se puede reconocer por un autómata de estado finito.

Una gramática libre de contexto se puede obtener a partir de un autómata de estado finito (o máquina de estado finito) de la siguiente forma:

1. El conjunto de símbolos de entrada de la máquina corresponde al conjunto de terminales de la gramática.
2. El conjunto de estados de la máquina corresponde al conjunto de no terminales de la gramática y el estado de inicio al no terminal inicial.
3. Si la máquina tiene una transición desde el estado A al estado B con una entrada x, la gramática genera la siguiente producción:

   Transición (A , x) -> B
   Producción: A -> x B

4. Si A es una estado de aceptación de la máquina, entonces se genera la regla o producción:

A -> €

Ejemplo. Dada la siguiente tabla de transición generar la gramática correspondiente.

	A	b	
S	A	B	1 : Aceptación
A	B	A	0 : No aceptación

B S A 1 : Aceptación

1. S -> a A
2. S -> bB
3. A -> aB
4. A -> bA
5. B -> aS
6. B -> bA
7. S -> €
8. B -> €

4.1 Gramáticas Especiales

Las **gramáticas especiales** tienen la forma:
1. <A> -> x <B> (el lado derecho tiene un terminal y un no terminal)
2. <A> -> € (Debe haber una producción que alcance a la lista vacía.)
Las **gramáticas regulares** sirven para definir expresiones regulares y tienen la forma:
<A> -> x (la producción genera un no terminal)

Una gramática regular se puede convertir en gramática especial de la siguiente forma:

<A> -> x Es regular y se reemplaza por:

<A> -> x <Z>
<Z> -> €

Las gramáticas especiales se definen con el fin de establecer a partir de la gramática un reconocedor de estado finito, permitiendo establecer un procedimiento para definir esta máquina de estado finito. El procedimiento para convertir una gramática especial en una máquina de estado finito es el siguiente:

1. El conjunto de terminales de la gramática corresponde al conjunto de entradas de la máquina.
2. El conjunto de no terminales corresponden al conjunto de estados de la máquina y el no terminal inicial al estado inicial.
3. Si existe la producción de esta forma: <A> -> xB , entonces la máquina establece una transición de A a B con una entrada x
4. Si existe una producción A -> €, entonces A es un estado de aceptación.

Ejemplo: Un identificador en un lenguaje de programación debe iniciar con una letra y seguir con un conjunto de números o números. Dada la siguiente gramática elaborar la máquina de estado finito.

<identificador> -> l <letras o dígitos> (l : letras, d :dígitos)
<letras o dígitos> -> l <letras o dígitos>
<letras o dígitos> -> d <letras o dígitos>
<letras o dígitos> -> €

	l		d	
<Identificador>	<Letras dígitos>	o		0
<Letras dígitos>	o <Letras dígitos>	o	<Letras dígitos>	o 1

4.2 Gramáticas con no Terminales Extraños

Existen gramáticas que exigen ser depuradas. Son gramáticas con no terminales extraños cuando existen no terminales muertos o también no terminales inalcanzables.

No Terminales muertos: son aquellos que no pueden generar una cadena terminal (con símbolos terminales). Si en una cadena existen terminales muertos hay que eliminarlos.

Los no terminales inalcanzables: son aquellos que no se pueden alcanzar ya que algunos elementos terminales no se generan por elementos no terminales. Si hay elementos no terminales inalcanzables se los puede eliminar.

Los no terminales vivos: son los que si permiten generar cadenas terminales.

PROPIEDAD A

Si todos los símbolos del lado derecho de una producción son vivos, entonces también los son los del lado izquierdo.

Pasos:

1. Mirar qué elementos no terminales pueden alcanzar en forma directa elementos terminales (no terminales vivos).
2. Aplicar la propiedad A, es decir, si se encuentra una producción en donde todos los no terminales del lado derecho están en la lista, entonces, el lo terminal del lado izquierdo se adiciona a dicha lista.
3. Cuando no se puede adicionar nuevos terminales a la lista, entonces, los elementos de la lista son los no terminales vivos, y los que no estén en la lista son los no terminales muertos. Se procede a eliminar los elementos no terminales muertos de la gramática.

PROPIEDAD B

Si el no terminal del lado izquierdo de una producción es alcanzable, entonces, también lo son los del lado derecho.

Pasos:

1. Realizar un lista que empiece con el no terminal inicial.
2. Aplicar la propieda B, es decir, si se encuentra una producción en donde el lado izquierdo está en la lista, entonces todos los no terminales del lado derecho de adicionan a éste.
3. Esta es la lista de todos los no terminales alcanzables. Los demás son los inalcanzables. Las producciones que involucren no terminales inalcanzables son eliminadas.

4.3. Gramáticas Alineadas por la Derecha

Son gramáticas utilizadas para generar conjuntos regulares y se denominan **alineadas por la derecha** porque el lado derecho de cada producción tiene a lo sumo una ocurrencia de un no terminal, el cual está ubicado al final de ésta.

Son de la forma:
A -> w B
ó
A -> w

donde w es secuencia de terminales

Ejemplo:

A -> a b <B>

A -> a b c

La importancia que adquieren estas gramáticas es que después de ciertos pasos de reacomodo fácilmente se puede convertir en una gramática especial.

Ejemplo:

A -> a b c <S>

No es especial porque tiene tres elementos terminales

Esta producción se puede convertir en gramática especial de la siguiente forma:

A -> a < bcS >

<bcS> -> b<cS>

<cS> -> c<S>

Ejemplo: Convertir la siguiente gramática alineada por la derecha en gramática a especial.

1. <S> -> ab<A>
2. <A> -> bb<A>
3. <A> -> b<S>
4. <S> -> €

Gramática especial:

1a. <S> -> a <bA>

1b. <bA> -> bb <A>

2a. <A> -> b <bA>

2b. <bA> -> b <A> (se eliminan porque es la misma 1b)

3. <A> -> b <S>

4. <S> -> €

4.4 Procesamiento Descendente

El Método Parsing (análisis gramátical) se divide en descendente y ascendente. Cada uno de ellos se caracteriza por el orden en el cual las producciones en un árbol de derivación son reconocidas.

En el **descendente** se reconocen las producciones que están en la parte superior del árbol antes que las producciones que están en la parte baja.

En el **ascendente** primero se reconocen las producciones que están en la parte baja para llegar a las producciones en la parte alta.

Tanto para el procesamiento descendente como para el ascendente se utilizan las máquinas de pila. El procesamiento impone restricciones en la forma de la gramática de entrada y la gramática de traducción o traslación.

Un caso particular de este tipo de gramáticas descendentes son las **gramáticas-s.**

GRAMATICA-S

Posee las siguientes restricciones:

- El lado derecho de cada producción inicia con un símbolo terminal.
- Si dos producciones tienen el mismo elemento no terminal izquierdo, entonces sus lados derechos deben iniciar con diferente símbolo terminal.

Ejemplo:

1. <S> -> ab<R>
2. <S> -> b<R>b<S>
3. <R> -> a
4. <R> -> b <R>

Para una gramática–s se puede aplicar una máquina de pila y establecer un diseño de la tabla de control aplicando las siguientes reglas:

1. El conjunto de símbolos de entrada es el conjunto de terminales de la gramática, al cual se le adiciona la marca de fin de secuencia.
2. El conjunto de símbolos de la pila consiste de:

- Marca de fondo de pila
- Símbolos no terminales de la gramática
- Los símbolos terminales que aparecen en el lado derecho de la producción después del primer elemento terminal.

3. La pila inicia con la marca de fondo de pila • y el no terminal inicial: —|<S>
4. El control está descrito por una tabla de control de un estado cuyas filas etiquetadas por los símbolos de pila y las columnas con los símbolos de entrada.
5. Una entrada en la tabla se determina de la siguiente forma:

- Si la producción tiene la forma:

<A> -> b μ

Donde μ es una cadena de no terminales y terminales, donde b es un terminal y donde <A> es un no terminal. Para este caso la transición corresponde a las operaciónes: **Reemplace (μ ^{r}), avance (donde r: inverso)**

- Si μ es vacio tenemos producciones de la forma:

<S> -> b

Para este caso la transición corresponde a las operaciones: **pop, avance**. (donde pop es desapile)

6. Si el terminal b es un símbolo de pila, la entrada en la tabla para la fila b es: **pop, avance**.

7. La entrada en la tabla para la marca de fondo de pila y la marca de fin de entrada es **acepte.**

7. Toda la entrada no descrita por los pasos 5,6 y 7 es **rechace**.

Las dos condiciones que definen la gramática-S aseguran que las reglas anteriores trabajen correctamente.

Ejemplo: Realizar la tabla de control para la siguiente gramática-s:

1. <S> -> d <S> <A>
2. <S> -> b <A>c
3. <A> -> d <A>
4. <A> -> c

	b	c	d	->
<S>	Reemplace (c,<A>) Avance	R	Reemplace (<A><S>) Avance	R
<A>	R	Pop Avance	Reemplace (<A>) Avance	R
c	R	Pop Avance	R	R
•	R	R	R	Acepte

También se puede simplificar la tabla utilizando una nomenclatura de la siguiente forma:

	b	c	d	—|

<S>	#2	R	#1	R
<A>	R	#4	#3	R
c	R	Pop Avance	R	R
•	R	R	R	Acepte

#1: Reemplace (<A><S>) Avance

#2: Reemplace (c<A>) Avance

#3: Reemplace (<A>) Avance

#4: Pop, Avance

Probar el funcionamiento de esta máquina de pila para reconocer la secuencia de entrada **dbccdc.**

Simbolo de pila	**Secuencia de entrada**	**Producción #**
• <S>	d b c c d c —|	1
• <A><S>	b c c d c —|	2
• <A>c<A>	c c d c —|	4
• <A> c	c d c —|	Pop, avance
• <A>	d c —|	3
• <A>	c —|	4
•	—|	Acepte

4.5 Siguiente de un No Terminal

El siguiente (<X>), es el conjunto de símbolos de entrada que pueden seguir inmediatamente a <X> dentro de una cadena intermedia derivada desde el símbolo inicial. Es el conjunto de terminales que aparecen a la derecha de <X> en alguna forma de frase de las siguientes características.

<S> -> μ <X> **a** <B>

donde **a** es un terminal siguiente de <X>

El propósito del siguiente de <X> es saber en qué momento aplica la producción nula de una gramática:
<A> -> €.

Ejemplo. Dada la producción y suponiendo que se tiene la siguiente derivación veriguar el siguiente del no terminal <A>.

<S> -> a <A><S>
-> a<A>**a**<A><S>
-> a<A>**a**<A>**b**

El siguiente de (<A>) es {**a**, **b**}

En una derivación si <X> puede ser el símbolo más a la derecha, entonces la marca de fin de entrada —| también es un elemento perteneciente al siguiente de <X>. De lo anterior concluimos que es inútil aplicar la producción vacía 4 (<A>-> €) si la entrada no está en el conjunto siguiente de <A>. (NOTA: Esto se aplica en las gramáticas LLL(3))

5. Conjunto de Selección de una Producción

Entrega las entradas para el cual un control de pila debe aplicar una producción.

- Si la producción es de tipo:

<A> -> b μ (μ puede ser nula)

entonces, la selección de esta producción es **{b}**

- Si la producción es:

<A> -> €**,** la selección es el **siguiente (<A>)**

Ejemplo: Averiguar el conjunto de selección para cada producción de la siguiente gramática:

1. <S> -> a <A> <S> Selección (1) = {a}
2. <S> -> b Selección (2) = {b}
3. <A>-> c <A> <S> Selección (3) = {c}
4. <A>-> € Selección (4), siguiente de <A> = {a,b}

El siguiente de <A> es:

<S> -> a <A> <S>

(2)

-> a <A> **b**

Otra derivación:

<S> -> a <A> <S>

(3)

-> a c <A> <S><S>

(1)

-> a c <A> **a** <A> <S>

GRAMÁTICAS – Q

La gramática –q es una gramática libre de contexto que tienen las siguientes condiciones:

1. El lado derecho de cada producción puede ser nulo o iniciar con un símbolo terminal.
2. Producciones con el mismo lado izquierdo tienen un conjunto de selección diferente. Es decir, la intersección entre los conjuntos de selección debe ser vacía:

{c} Intersección {a, b}= {}

Una gramática–Q también puede ser representada por una máquina de pila.

¿Cómo se llena la tabla de control?

Se utilizan las reglas 1,2,3 y 4 definidas en la gramática–S, además de la regla 5:

1. El conjunto de símbolos de entrada es el conjunto de terminales de la gramática, al cual se le adiciona la marca de fin de secuencia.
2. El conjunto de símbolos de la pila consiste de:
   - Marca de fondo de pila
   - Símbolos no terminales de la gramática
   - Los símbolos terminales que aparecen en el lado derecho de la producción después del primer elemento terminal.
3. La pila inicia con la marca de fondo de pila Ñ y el no terminal inicial: Ñ<S>
4. El control está descrito por una tabla de control de un estado cuyas filas etiquetadas por los símbolos de pila y las columnas con los símbolos de entrada.
5. Esta regla está conformada a su vez por dos subrreglas:

1. Una producción es aplicada siempre que el símbolo de la pila esté en el lado izquierdo y la entrada esté en el conjunto de selección de esta producción.

- Para aplicar una producción de la forma: <A>-> bμ**,**
  reemplace (μ ′), avance
- Para aplicar a una producción de la forma <A> -> b,
  pop, avance
- Si la producción es de la forma <A> -> €
  pop, retiene

1. Si hay una producción nula para un no terminal <A> y no se presentan entradas en la tabla de control para el símbolo de la pila y el elemento de entrada después de aplicar la regla 5.1, entonces la entrada en la tabla puede ser aplicar la producción nula o rechace.

	A	b	C	—|
<S>	#1	#2		R
<A>	#4	#4	#3	R o #4 (posterga el rechace)
•	R	R	R	A

#1: reemplace (<S>**,** <A>)**,** avance (forma : <A> -> bμ)

#2: pop, avance

#3: reemplace (<S>**,** <A>) **,** avance

#4: pop **,** retiene (forma : <A> -> €)

De acuerdo a la tabla de control se puede hacer un manejo de errores.

5.1. Gramática de Traducción Utilizando la Gramática –q

Producción de la forma: <A> -> ø**,** donde ø es una cadena de símbolos de acción.
<A> -> €

OUT (ø)**,** POP**,** Retiene
Ejemplo:

<A>->{x} Out (x)**,** pop**,** retiene

GRAMÁTICAS LL (1)

Para producciones donde el lado derecho inicia con un no terminal, se realiza una generalización de la gramática anterior y para ello definimos el concepto del **elemento primero** de una producción.

PRIMERO(μ)

Si tenemos la producción:

<A> -> μ

donde μ es una cadena de no terminales y terminales que inician con un no terminal. Definimos **primero(μ)** como como el conjunto de símbolos terminales que ocurren al inicio de una cadena intermedia derivada desde μ.

Ejemplo 1. Calcular los primeros de la siguiente gramática.

1. <S> -> <A> b <B> primero (<A>b<B>) = {a, c, e, b}

2. <S> -> d primero (d)= {d}
3. <A> -> <C><A> b primero (<C><A>b) = {a, e}
4. <A> -> <B> primero (<B>) = {c}
5. <B> -> c <S>d primero (c<S>d) = {c}
6. <B> -> € primero (€) = { }
7. <C> -> a primero (a)= {a}
8. <C> -> ed primero (ed)= {e} (primero que ocurre)

Las producciones 4, 6 son anulables

Para la producción 1:
<S> -> <A> b <B> primero (<A>b<B>) = {a, c, e, b}

Si hay una <S> en el tope de la pila y la secuencia inicia con a,c,e,b se aplica: reemplace (<B>b<A>) , retiene.

Producción anulable: cuando el lado derecho de la producción es anulable, es decir, se llega a la cadena vacía.

Conjunto de selección de gramáticas LL(1):
Una gramática se denomina LL(1) si y solo si producciones con igual lado izquierdo tienen conjuntos de selección diferentes (no tienen elementos en común). El conjunto de selección para una gramática LL(1) es el siguiente:

μ: es una cadena de no terminales y terminales que inician con un no terminal

Selección (<A>-> μ) = primero(μ) si y solo si μ NO es anulable.
Selección (<A>-> μ) = primero(μ) U siguiente(<A>) si y solo si μ es anulable.
Selección (<B>-> €) = primero(€) U siguiente(<B>).
Selección (<A>-> <B>) = primero(<B>) U siguiente(<A>).

Entonces,

Selección (<B>-> €) = primero(€) U siguiente(<B>) = {} U {—|, b, d} = {—|, b, d}
Selección (<A>-> <B>) = primero(<B>) U siguiente(<A>) = {c} U {b} = {b,c}

Utilizando el conjunto de selección más las reglas de la gramática-Q podemos crear la tabla de control de la gramática LL(1):

	a	b	C	D	e	—|
<S>	#1	#1	#1	#2	#1	R
<A>	#3	#4	#4	R	#3	R
<B>	R	#6	#5	#6	R	#6
<C>	#7	R	R	R	#8	R
B	R	Pop Avance	R	R	R	R
D	R	R	R	Pop Avance	R	R
•	R	R	R	R	R	Acepte

#1: reemplace (<B>b<A>), retiene
#2: pop, avance
#3: reemplace (b<A><C>), retiene
#4: reemplace (<B>), retiene
#5: reemplace (d<S>), avance
#6: pop, retiene
#7: pop, avance
#8: reemplace (d), avance

Ejemplo 2. Elaborar la tabla de control para la siguiente gramática

1. <E> -> <T> <E-L>
2. <E-L> -> + <T> <E-L>
3. <E-L> -> €
4. <T> -> <F> <T-L>
5. <T-L> -> * <F> <T-L>
6. <T-L> -> €
7. <F> -> (<E>)
8. <F> -> Id

Selección (<E>-><T> <E-L>) = primero (<T> <E-L>)= {(, Id}
Selección (+<T> <E-L>) = primero (+<T> <E-L>)= {+}
Selección (<E-L>->€) = primero (€) U siguiente <E-L>)
{} U {), —| }
Selección (<T>-><F><T-L>) = primero (<F><T-L>) = { (, Id }
Selección (*<F><T-L>) = primero (*<F><T-L>)={*}
Selección (<T-L>->€) = primero (€) U siguiente (<T-L>)
{} U {+,), —|}
Selección (<F>->Id) = primero (Id) = {Id}

En resumen:

Selección (1) = {(, Id}
Selección (2 = {+}
Selección (3) = {), —| }
Selección (4) = { (, Id }
Selección (5) = {*}
Selección (6) = {+,), —|}
Selección (7) = { (}
Selección (8) = {Id}

Como el conjunto de selección de las producciones que tienen igual lado izquierdo es diferente, se trata de una gramática LL(1).

	+	*	Id	(	)	—|
<E>			#1	#1		
<E-L>	#2				#3	#3
<T>			#4	#4		
<T-L>	#6	#4			#6	#6
<F>			#8	#7		
)					Pop Avance	
•						Acepte

#1: reemplace (<E-L><T>), retiene

#2: reemplace (<E-L><T>), retiene

#3: pop, avance

#4: reemplace (<T-L><F>), retiene

#5: reemplace (<T-L><F>), avance

#6: pop, retiene

#7: reemplace ( )<E>), avance

#8: pop, avance