ANÁLISIS SINTÁCTICO

José Sánchez Juárez

Septiembre, 2018

II

Índice general

Índice de figuras

Índice de cuadros

Construcción de analizadores sintácticos

> Para ser justos es necesario ser libres. Los sentimientos de justicia son hijos de la libertad, pues nunca, siendo esclavos, podremos ser justos.
>
> Carlos Pellicer.
> 1899−1977

El análisis sintáctico es menos local que el análisis léxico, donde se requieren métodos más avanzados. Se usa la misma estrategia: una notación más fácil para el entendimiento humano es transformada en una máquina, semejante a una más sencilla de bajo nivel para una ejecución más eficiente. Este proceso es llamado generación del analizador.

Analizadores Sintácticos

En la etapa de análisis sintáctico de un compilador se verifican los tokens reconocidos por el analizador léxico, los cuales son agrupados de acuerdo a reglas sintácticas del lenguaje. Si los tokens en una cadena son agrupados de acuerdo a reglas de sintaxis del lenguaje, entonces las cadenas de tokens entregados por el analizador léxico, son aceptadas como construcción válida del lenguaje; de otra manera un manejo de errores se aplica. Se deben tomar en cuenta dos casos al diseñar la fase de análisis sintáctico en un proceso de compilación;

Primero todas las construcciones válidas de un lenguaje de programación deben ser especificadas: y para usar estas especificaciones un programa válido se forma. Aquellas, que nosotros formamos en una especificación de aquellos tokens que el analizador léxico regresará, y que nosotros especificamos de que manera estos tokens son agrupados, de modo que lo que resulte del agrupamiento sea una construcción válida del lenguaje.

Segundo debe ser usada una notación apropiada para especificar la construcción de un lenguaje. La notación para especificar la construcción debe ser compacta, precisa y fácil de entender. La especificación de la estructura sintáctica para el lenguaje de programación (es decir, la construcción válida del lenguaje) usa gramáticas libres de contexto (GLC), porque para ciertas cla-

ses de gramáticas, nosotros podemos automáticamente construir un analizador sintáctico que determine si un programa fuente es sintácticamente correcto.

Para encontrar la estructura del texto de entrada, el análisis sintáctico debe rechazar el texto invalido para reportar los errores de sintaxis. A partir de un reconocedor (analizador léxico) adecuado que será diseñado para reconocer si una cadena de tokens es una construcción válida o no.

Formalización de gramáticas

Formalizar quiere decir ser preciso o determinado en la forma de representar las gramáticas, por lo que para cumplir con esto una gramática se expresa por medio de un cuarteto, como se muestra en la definición siguiente:

DEFINICIÓN 1 (Formalización de gramáticas.)

$$G(N, T, P, S)$$

Cada una de las partes que componen al cuarteto, son:

1. $\boldsymbol{N}$ *es el conjunto de los no terminales.*
2. $\boldsymbol{T}$ *es el conjunto de los terminales.*
3. $\boldsymbol{P}$ *es el conjunto de producciones.*
4. $\boldsymbol{S}$ *es el símbolo inicial.*

De esta manera la gramática queda expresada de manera formal, ya que la precisión radica en la forma de las producciones, basadas en la jerarquía de Chomsky.

Para ejemplificar el uso de la representación formal. Se presenta el siguiente ejemplo:

EJEMPLO 1 (Gramática formal.) *Una gramática que expresa la sintaxis para declarar los datos en C++:*

$$G(N, T, P, S)$$

Cada una de las partes que componen al cuarteto, son:

1. $\boldsymbol{N} = \{S, T, A\}$ *es el conjunto de los no terminales.*
2. $\boldsymbol{T} = \{i, f, d, id\}$ *es el conjunto de los terminales.*
3. $\boldsymbol{P} = \{S \rightarrow Tid; , T \rightarrow i, f, d\}$ *es el conjunto de producciones.*
4. $\boldsymbol{S} = \{S\}$ *es el símbolo inicial.*

Notación de las producciones

Cada regla en el conjunto **P** de una gramática se llama producción. Para representar el conjunto de producciones se usa la notación **BNF** (Forma de Backus Naur), que se aplica de la manera siguiente:

$$\langle N \rangle \rightarrow \langle X_1 \rangle \; x_1 \cdots \langle X_n \rangle \; x_n$$

donde $\langle N \rangle$ representa el conjunto de los no terminales y que se llama la parte izquierda de la producción, y $\langle X_1 \rangle \; x_1 \; \cdots \langle X_n \rangle \; x_n$ es la parte derecha de la producción y representan cero o más símbolos, cada uno de los símbolos $x_1 \cdots \mathrm{x}_n$ son terminales y cada uno de los símbolos $\langle X_1 \rangle \cdots \langle X_n \rangle$ son no terminales.

Pero la representación de las producciones para mayor facilidad, se hace sin usar los paréntesis triangulares y se usa el símbolo $\rightarrow$ en lugar del símbolo ::=.

Las gramáticas libres de contexto sirven para representar reglas sintácticas pero también pueden expresar algunos sonidos de máquinas, como se muestra en el ejemplo siguiente:

EJEMPLO 2 (Uso de las producciones.) *El sonido de un tren se representa por medio de producciones de la siguiente manera:*

$$SonidoTren \rightarrow baa \;\; SonidoTren$$
$$\mid baa$$

La barra vertical es la separación de dos partes izquierdas con la misma parte derecha de la producción y que se representa en forma expandida como sigue:

$$SonidoTren \rightarrow baa \;\; SonidoTren$$
$$SonidoTren \rightarrow baa$$

*La misma gramática del sonido de un tren en notación **BNF** (BACKUS-NAUR FORM) es la siguiente:*

$$< SonidoTren > \;\; ::= \;\; baa \;\; < SonidoTren >$$
$$\mid baa$$

Jerarquía de Chomsky

Las gramáticas se clasifican según la forma de la producción en la jerarquía de Chomsky, donde se establecen cuatro clases de gramáticas, las cuales se enumeran del 0 al 3 [6]:

DEFINICIÓN 2 (Jerarquía de Chomsky.) ***La gramática 0*** *o gramática sin restricciones tiene la forma:*

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

, dónde $A \in N, \alpha, \beta, \gamma \in V^*$ *y* V *es el alfabeto.*
La gramática 1 *o gramática sensible al contexto tiene la forma:*

$$\alpha \rightarrow \beta$$

donde $|\alpha| \leq |\beta|$.
La gramática 2 *o gramática libre de contexto tiene la forma:*

$$A \rightarrow \alpha$$

donde $A \in N$ *y* $\alpha \in V^*$.
La gramática 3 *o gramática regular tiene la forma:*

$$A \rightarrow a$$

o

$$A \rightarrow aB$$

que es la forma lineal derecha o bien.

$$A \rightarrow a$$

o

$$A \rightarrow Ba$$

que es la forma lineal izquierda. Donde $A, B \in N$, $a \in T^*$.

Ejemplo de gramáticas formales

Un ejemplo de gramática es la siguiente $G_1 = (\{S\}, \{a, b\}, P_1, S)$:

EJEMPLO 3 (Gramática formal G_1.) *Donde el símbolo inicial es* S, *el conjunto de no terminales* $T = \{a, b\}$, *el conjunto de no terminales* $N = \{S\}$ *y el conjunto de producciones* P_1, *donde los elementos del conjunto* P_1 *son los siguientes,*

$$P_1 = \{S \rightarrow aSa,$$

$$S \rightarrow bSb,$$

$$S \rightarrow \epsilon\}$$

Ya que cada producción de la gramática G_1 tiene la forma de la gramática

2 de Chomsky, como se muestra en el cuadro 2 , esta es una gramática libre de contexto:

Número	Producción	Forma de Chomsky
1	$S \to aSa$	$A \to \alpha$
2	$S \to bSb$	$A \to \alpha$
3	$S \to \epsilon$	$A \to \alpha$

Cuadro 1: Producciones de la gramática G_1, comparadas con la manera formal de la jerarquía de Chomsky.

Derivaciones

Sean δ_1 y δ_2 dos cadenas de caracteres, si la cadena δ_1 se transforma en la cadena δ_2 utilizando el conjunto de producciones de la gramática, es a lo que se llaman derivación y se representan de la siguiente forma:

$$\delta_1 \Rightarrow \delta_2$$

Obtención de derivaciones Para hacer las derivaciones se utilizan las definiciones de forma sentencial y desde luego la definición de derivación.

DEFINICIÓN 3 (Forma sentencial.) *La forma sentencial es una cadena de símbolos que ocurre como un paso en una derivación válida [5].*

En cada paso de las derivaciones se obtienen las formas sentenciales:

DEFINICIÓN 4 (Derivación.) *Derivación es una secuencia de reescribir los pasos que comienzan con el símbolo inicial de la gramática y que finaliza con la sentencia en el lenguaje [5].*

En cada paso se aplican las producciones de la gramática para producir la forma sentencial.

Tipos de derivaciones Existen dos formas de derivar las cadenas de caracteres, la derivación por la derecha y la derivación por la izquierda.

DEFINICIÓN 5 (Derivación derecha.) *La derivación derecha es la que reescribe en cada paso el símbolo no terminal de más a la derecha [5]. Y se representa como* $\Rightarrow^D$.

DEFINICIÓN 6 (Derivación izquierda.) *La derivación izquierda es la que reescribe en cada paso el símbolo no terminal de más a la izquierda [5]. Y se representa como* $\Rightarrow^I$.

Ejemplo de derivaciones

Para poder acoplar las producciones de la gramática G_1 con las cadenas de caracteres que se presentan en el código fuente, se aplican las derivaciones de la siguiente forma: si se tiene la cadena **abba**, la que se deriva en:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow ab\epsilon ba \Rightarrow abba$$

Lo que se puede expresar como:

$$S \Rightarrow^* abba$$

Otro ejemplo de derivaciones Se tiene la gramática $G_2 = (N_2, T_2, P_2, S_2)$ con el conjunto de producciones P_2:

$$P_2 = \{E \rightarrow E + E,$$

$$E \rightarrow E * E,$$

$$E \rightarrow (E),$$

$$E \rightarrow id\}$$

Desarrollar por medio de derivaciones izquierdas y derivaciones derechas la siguiente cadena, **id + id * id**.

MÉTODO 1 (Desarrollo de derivaciones.) *Se comienza con el símbolo inicial, se selecciona el no terminal de la izquierda o derecha de acuerdo al tipo de derivación, este se substituye por la producción que produzca una forma sentencial y así sucesivamente hasta obtener la cadena que se considere.*

Derivación derecha:

$$E \Rightarrow^D E * E \Rightarrow^D E * id \Rightarrow^D E + E * id \Rightarrow^D E + id * id \Rightarrow^D id + id * id$$

Derivación izquierda:

$$E \Rightarrow^I E * E \Rightarrow^I E + E * E \Rightarrow^I id + E * E \Rightarrow^I id + id * E \Rightarrow^I id + id * id$$

Árboles de análisis sintáctico

Los tipos de árboles que hay, son: los árboles de análisis sintácticos y los árboles de derivación.

DEFINICIÓN 7 (Árbol de análisis sintáctico.) *Este árbol presenta en cada uno de sus nodos el símbolo no terminal, de tal manera que se presenta la sintaxis de la gramática.*

Si tenemos la siguiente gramática, el árbol de análisis sintáctico es el que se muestra en la figura :

EJEMPLO 4 (Árbol de análisis sintáctico.) *Figura que muestra el árbol de análisis sintáctico.*

El árbol de derivaciones se define:

DEFINICIÓN 8 (Árbol de derivación.) *Este árbol presenta la forma de obtener las formas sentenciales aplicando las producciones de la gramática.*

Si tenemos a la gramática $G_3 = (N, T, P_3, S)$ con el siguiente conjunto de producciones P_3:

$$P_3 = \{E \to E + E$$

$$E \to E * E,$$

$$E \to id\}$$

EJEMPLO 5 (Árbol de derivación.) *Y se requiere obtener la siguiente cadena: $id + id * id$. Un primer árbol de derivaciones, obtenido de las derivaciones:*

$$E \Rightarrow E + E \Rightarrow id + E \Rightarrow id + E * E \Rightarrow id + id * E \Rightarrow id + id * id$$

, es el que se muestra en la figura 1 :

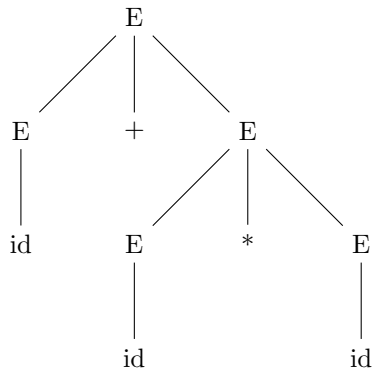

Figura 1: Árbol de derivaciones de la gramática G_3.

Otro ejemplo de árbol de derivaciones para la cadena de caracteres **abba** obtenida por derivaciones, que se desarrolla aplicando el conjunto de producciones P_1 de la gramática G_1, como se muestra a continuación:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow ab\epsilon ba \Rightarrow abba$$

EJEMPLO 6 (Árbol de derivación.) *Donde su árbol de derivaciones es el que se muestra en la figura 2 :*

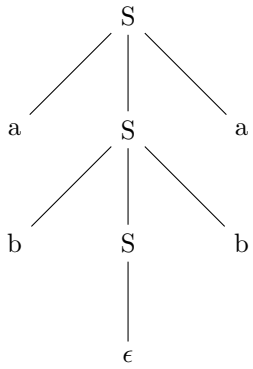

Figura 2: Árbol de derivaciones de la gramática G_1.

El libro [5] cita que el árbol sintáctico es el mismo que el árbol de derivación. Así que consideraremos en estos apuntes que el árbol de derivación y el árbol sintáctico son lo mismo.

Diferencia entre gramáticas regulares y gramáticas libres de contexto

El listado de palabras con una cantidad de **a's** emparejadas, es el siguiente $\{aba, aabaa, aaabaaa, \cdots\}$, que se representa por el conjunto de producciones:

1. $$S \to aCa$$
2. $$C \to aCa$$
3. $$C \to b$$

que son parte del cuarteto de la gramática formal del tipo libre de contexto.

Aplicando la gramática anterior y las derivaciones se obtiene la cadena **aaabaaa**:

$$S \Rightarrow^D aCa \Rightarrow^D aaCaa \Rightarrow^D aaaCaaa \Rightarrow^D aaabaaa$$

Sin embargo esta cadena también se puede obtener por medio de la gramática formal que se enlista a continuación:

1. $$S \to aS$$
2. $$S \to aB$$
3. $$B \to bC$$
4. $$C \to aC$$
5. $$C \to a$$

De la derivación se obtiene aplicando la gramática anterior, la cadena **aaabaaa**:

$$S \Rightarrow^D aS \Rightarrow^D aaS \Rightarrow^D aaaB \Rightarrow^D aaabC \Rightarrow^D aaabaC$$

$$\Rightarrow^D aaabaaC \Rightarrow^D aaabaaa$$

La diferencia esta en que con la gramática libre de contexto no se puede obtener la cadena **aabaaaa** y con la gramática regular si se puede obtener:

$$S \Rightarrow^D aS \Rightarrow^D aaB \Rightarrow^D aabC \Rightarrow^D aabaC \Rightarrow^D aabaaC$$

$$\Rightarrow^D aabaaaC \Rightarrow^D aabaaaa$$

Esto quiere decir que no se puede obtener una cadena emparejada, por ejemplo los paréntesis emparejados que son esenciales en análisis sintáctico.

EJEMPLO 7 (Diferencia entre una GR y una GLC.) *La gramática regular se abrevia* ***GR*** *y la gramática libre de contexto se abrevia* ***GLC****.*

Con la siguiente expresión regular no se pueden expresar operaciones con paréntesis:

Con una gramática libre de contexto si se pueden representar operaciones con paréntesis. Como se muestra en la sección ***"Representación de gramáticas libres de contexto"****.*

Representación de gramáticas libres de contexto

Las gramáticas libres de contexto se representan como ya se dijo por medio de cuartetos, pero también se representan por medio de la teoría de conjuntos:

$$\{cadena|logica\}$$

ya que se requiere representar las estructuras por medio de un listado de la teoría de conjuntos, con la finalidad de expresar características que no se pueden observar por medio de la representación formal de las gramáticas libres de contexto. Como se muestra con el siguiente ejemplo:

Gramática libre de contexto La gramática con conjunto de producciones:

$$P = \{S \to aCa,$$
$$C \to aCa,$$
$$C \to b\}$$

Se representa por medio de la teoría de conjuntos de la siguiente forma:

$$L(G) = \{a^n ba^n | n \geq 1\}$$

Gramática regular La gramática con conjunto de producciones:

$$P = \{S \to aS,$$
$$S \to aB,$$
$$B \to bC,$$
$$C \to aC,$$
$$C \to a\}$$

Se representa por medio de la teoría de conjuntos de la siguiente forma:

$$L(G) = \{a^n b a^m | n, m \geq 1\}$$

EJEMPLO 8 (Resaltar la característica con la teoría de conjuntos.) *Se requiere exaltar las características de emparejamiento. Las cuales se hacen por medio de las reglas de la teoría de conjuntos.*

$$L(G) = \{(^n b)^m | n, m \geq 1\}$$

Se nota claramente el emparejamiento de los paréntesis, cuando $\boldsymbol{n}$ *y* $\boldsymbol{m}$ *son iguales y en este caso se convierte en la siguiente gramática:*

$$L(G) = \{(^n b)^n | n \geq 1\}$$

Gramáticas ambiguas

Para la construcción de analizadores sintácticos se usan las gramáticas libres de contexto, las cuales no todas son ideales. Existe el problema de que pueden ser ambiguas. Una gramática es ambigua si existe alguna cadena de terminales que pueda obtenerse mediante árboles de derivación distintos (dos árboles distintos dan la misma cadena). El problema de la ambigüedad es que esta genera retroceso en la programación.

DEFINICIÓN 9 (Gramática ambigua.) *Si aplicando las producciones de una gramática se obtienen para una cadena por derivación dos árboles sintácticos diferentes.*

Para la gramática G_3, la cadena $id + id * id$ se obtiene por dos árboles de derivación diferentes. El segundo árbol de derivación se obtiene de la serie de derivaciones $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow id + E * E \Rightarrow id + id * E \Rightarrow id + id * id$, y el árbol de análisis sintáctico es el siguiente:

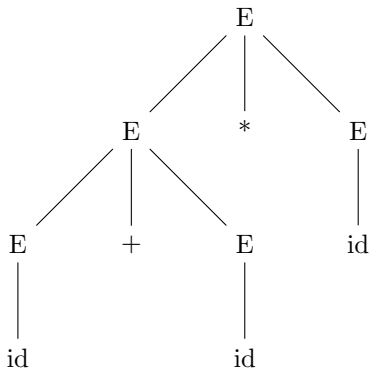

Figura 3: Otro árbol de análisis sintáctico de la gramática G_3.

Como esta derivación tiene dos árboles de análisis sintácticos diferentes, el árbol de análisis sintáctico de la figura 1 y el árbol de análisis sintáctico de la figura 3, entonces la gramática G_3 es una gramática ambigua.

Otro ejemplo de gramática ambigua. Es la gramática G_4 con las siguientes producciones:

$$S \to aSbS|bSaS|\epsilon$$

Para obtener la siguiente cadena de caracteres *abab*, se obtiene por medio de las siguientes derivaciones:

$$S \Rightarrow aSbS \Rightarrow abSaSbS \Rightarrow ab\epsilon aSbS \Rightarrow aba\epsilon bS \Rightarrow abab\epsilon \Rightarrow abab$$

El árbol de análisis sintáctico es el siguiente:

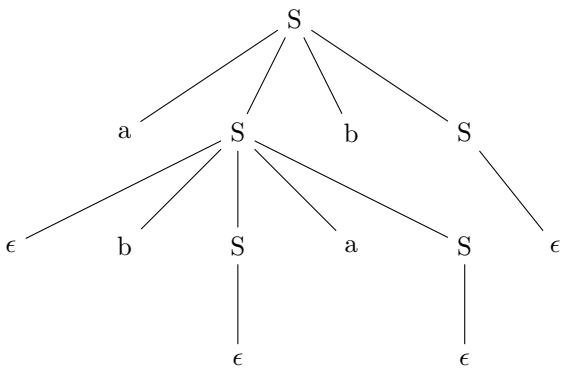

Figura 4: Árbol de análisis sintáctico de la gramática G_4.

Para la gramática G_4 se puede obtener otro árbol de análisis sintáctico. Por lo que esta gramática es ambigua.

Eliminación de la ambigüedad

Si para una cadena de caracteres terminales con las producciones de una gramática **no** se obtienen dos árboles de derivación diferentes, entonces se dice que una gramática **no** es ambigua. Y se puede decir que una gramática ambigua al aplicarle las transformaciones, se demuestra que no cumple con el concepto de una gramática ambigua, **entonces se afirma que se ha eliminado la ambigüedad**.

La eliminación de la ambigüedad en las gramáticas libres de contexto se hace de diferentes maneras:

Uno, reduciendo la gramática. Esto se refiere a identificar aquellos símbolos gramaticales (llamados símbolos gramaticales no usuales) y desde luego a las producciones que no juegan un papel en la derivación, la cual podemos eliminar de la gramática.

De una gramática un símbolo usual X es aquel que cumple lo siguiente:

1. Si este deriva en una cadena de terminales.
2. Es usado en al menos en una derivación de w.

Por lo que:

1. $X \Rightarrow^* w$, dónde w esta en T^*.
2. $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$ en L(G).

EJEMPLO 9 *Un ejemplo es el que se basa en el conjunto de producciones siguiente:*

$$P_5 = \{S \to AC|SB,$$
$$A \to bASC|a,$$
$$B \to aSB|bbC,$$
$$C \to Bc|ad\}$$

Aplicando el método de la reducción de la gramática, el conjunto de producciones queda de la siguiente manera:

$$S \to AC$$
$$A \to bASC|a$$
$$C \to ad$$

EJERCICIO 1 *Determinar si es ambigua y corregirla en caso de ser ambigua, a la siguiente gramática libre de contexto:*

$$T \to R$$
$$T \to aTa$$
$$R \to b$$
$$R \to bR$$

EJEMPLO 10 *Si tenemos la siguiente gramática:*

$$E \to E + E$$
$$E \to E - E$$
$$E \to E * E$$
$$E \to E/E$$
$$E \to (E)$$
$$E \to id$$

Y se requiere obtener la siguiente cadena de caracteres $(id + id)/(id - id)$ esto se hace obteniendo la derivación por la izquierda:

$$E \Rightarrow E/E \Rightarrow (E)/E \Rightarrow (E + E)/E \Rightarrow (id + E)/E \Rightarrow (id + id)/E \Rightarrow$$
$$(id + id)/(E) \Rightarrow (id + id)/(E - E) \Rightarrow (id + id)/(id - E) \Rightarrow (id + id)/(id - id)$$

Esto se puede hacer con derivaciones por la derecha:

$$E \Rightarrow E/E \Rightarrow E/(E) \Rightarrow E/(E - E) \Rightarrow E/(E - id) \Rightarrow E/(id - id) \Rightarrow$$
$$(E)/(id - id) \Rightarrow (E + E)/(id - id) \Rightarrow (E + id)/(id - id) \Rightarrow (id + id)/(id - id)$$

Se ve que la gramática anterior es ambigua ya que se obtienen dos árboles de derivación diferentes.

Dos, reescribiendo la gramática. Para eliminar la ambigüedad se hace también reescribiendo la gramática:

Para reescribir la gramática de tal manera que se elimine la ambigüedad se hace de la siguiente manera:

$$E \to E + T|E - T|T$$
$$T \to T * F|T/F|F$$
$$F \to (E)|id$$

De la siguiente gramática obtener la cadena de caracteres *aabbbcc*:

$$T \to R,$$
$$T \to aTc,$$

$$R \to \epsilon,$$

$$R \to RbR$$

Se hacen las siguientes derivaciones:

$T \Rightarrow aTc \Rightarrow aaTcc \Rightarrow aaRcc \Rightarrow aaRbRcc \Rightarrow aaRbcc \Rightarrow aaRbRbcc \Rightarrow aaRbRbRbcc \Rightarrow aaRbbRbcc \Rightarrow aabbRbcc \Rightarrow aabbbcc$

Otra derivación es la siguiente:

$T \Rightarrow aTc \Rightarrow aaTcc \Rightarrow aaRcc \Rightarrow aaRbRcc \Rightarrow aaRbRbRcc \Rightarrow aabRbRcc \Rightarrow aabRbRbRcc \Rightarrow aabbRbRcc \Rightarrow aabbbRcc \Rightarrow aabbbcc$

Ahora la gramática reescrita es la siguiente:

$$T \to R,$$

$$T \to aTc,$$

$$R \to \epsilon,$$

$$R \to bR$$

La siguiente gramática es ambigua:

$$E \to E\ Op\ E,$$

$$E \to num$$

Ahora se reescribe la gramática de la siguiente manera:

$$E \to E\ Op\ E',$$

$$E \to E'$$

$$E' \to num$$

Pero ahora se tiene el problema de la recursividad izquierda.

Tres, recursividad izquierda. En el reconocimiento de tokens de izquierda a derecha se presenta un problema. Consideremos la siguiente gramática G_4:

$$E \to E + T|T$$

$$T \to T * F|F$$

$$F \to i|(E)$$

Y analicemos la expresión $i + i + i$. Se aplican las siguientes derivaciones:

$$E \Rightarrow E + T \Rightarrow E + T + T \Rightarrow T + T + T \Rightarrow F + T + T \Rightarrow i + T + T$$

$$\Rightarrow i + F + T \Rightarrow i + i + T \Rightarrow i + i + F \Rightarrow i + i + i$$

árbol de análisis sintáctico es el siguiente:

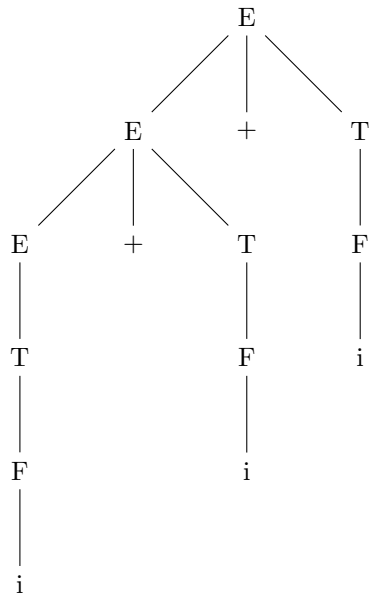

Figura 5: Otro árbol de análisis sintáctico de la gramática G_4.

$$E \Rightarrow E + T \Rightarrow E + T + T \Rightarrow E + T + T + T \Rightarrow \ldots$$

El problema que se presenta en las producciones es la recursividad izquierda. Ya que la producción $E \to E + T$ tiene la siguiente forma:

$$A \to A\alpha$$

O lo que es lo mismo $A = E$ y $\alpha = +T$, por lo que la gramática G_4 tiene recursividad izquierda.

Un análisis descendente para la gramática G_4 comenzará por expandir E con la producción $E \Rightarrow E + T$. Se tratará de expandir de nuevo E de la misma forma. Esto da el siguiente árbol de análisis sintáctico:

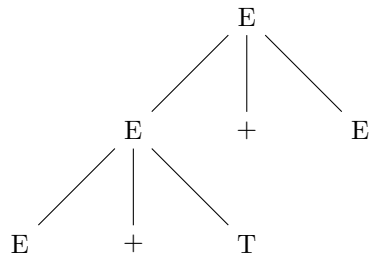

Figura 6: Árbol de análisis sintáctico que muestra que E es el que se debe expandir, en la gramática G_4.

pero no sabemos cuando se debe expandir con $E \to T$ en lugar de otra vez usar $E \Rightarrow E + T$. Esto se soluciona eliminando la recursividad izquierda.

Eliminación de la recursión izquierda en las gramáticas

Las producciones que tienen recursividad izquierda, se igualan carácter a carácter con las siguientes producciones:

$$A \to A\alpha|\beta$$

Aunque también se pueden usar las siguientes producciones:

$$A \to B\alpha| \dots$$
$$B \to A\beta| \dots$$

Para después hacer la siguiente transformación:

$$A \to \beta A'$$
$$A' \to \alpha A'|\epsilon$$

EJEMPLO 11 *La siguiente gramática tiene recursividad izquierda:*

$$S \to Sa|b$$

De las siguientes derivaciones:

$$S \Rightarrow Sa \Rightarrow Saa \Rightarrow Saaa \Rightarrow \cdots \Rightarrow ba*$$

Donde el proceso termina cuando se usa b. Para eliminar la recursividad izquierda, se aplica:
La igualación:
$A = S, \alpha = a, \beta = b$ así que la transformación queda como sigue:

$$S \to bS'$$
$$S' \to aS'|\epsilon$$

Entonces las derivaciones se convierten en:

$$S \Rightarrow bS' \Rightarrow baS' \Rightarrow baaS' \Rightarrow baaaS' \Rightarrow \cdots \Rightarrow ba*$$

EJEMPLO 12 *Un ejemplo de gramática recursiva es la siguiente:*

$$A \to Ac|Ad|e|f$$

Donde las producciones recursivas izquierdas son:

$$A \to Ac|Ad$$

Y las producciones no recursivas son:

$$A \to e|f$$

Las producciones se factorizan:

$$A \to A(c|d)|(e|f)$$

Se hace la igualación carácter a carácter con la formula y se obtiene lo siguiente: $A = A$, $\alpha = (c|d)$ y $\beta = (e|f)$, para obtenerse la transformación que a continuación se muestra:

$$A' \to eA'|fA'$$

y

$$A' \to cA'|dA'|\epsilon$$

EJEMPLO 13 *Si tenemos otro ejemplo:*

$$S \to aA|b|cS$$
$$A \to Sd|e$$

Se adaptan las producciones y se obtiene la siguiente producción:

$$A \to aAd|bd|cSd|e$$

Así que las derivaciones tienen tres alternativas, pero no hay producciones con recursividad izquierda.

EJEMPLO 14 *El siguiente ejemplo:*

$$S \to A|B|Sc|dS$$
$$A \to Bd|cA|f$$
$$B \to Se|Ad|g$$

Estas producciones se corrigen, para la primera producción donde esta presente el problema de la recursión izquierda en Sc. Se corrige de la siguiente manera:

$$S \to AP|BP|dSP$$
$$P \to \epsilon|cP$$

Para la segunda producción la recursividad izquierda no esta presente.
Para la tercera producción:

$$B \to APe|BPe|dSPe|Ad|g$$

Para la cuarta nos da las producciones APe y Ad:

$$B \to BdPe|cAPe|fPe|BPe$$
$$|dSPe|Bdd|cAd|fd|g$$

Como $BdPe$, BPe y Bdd tienen recursión izquierda inmediata, se debe corregir y que queda de la siguiente manera:

$$B \to cAPeQ|fPeQ|dSPeQ|cAdQ|fdQ|gQ$$
$$Q \to \epsilon|dPeQ|PeQ|ddQ$$

Así que finalmente la gramática queda de la siguiente forma:

$$S \to AP|BP|dSP$$

$$P \to \epsilon|cP$$

$$A \to Bd|cA|f$$

$$B \to cAPeQ|fPeQ|dSPeQ|cAdQ|fdQ|gQ$$

$$Q \to \epsilon|dPeQ|PeQ|ddQ$$

EJEMPLO 15 *Otro ejemplo. Eliminar la recursión izquierda de la siguiente gramática:*

$$E \to E + T|T$$

$$T \to T * F|F$$

$$F \to i|(E)$$

Así que queda de la siguiente manera se introduce un nuevo no terminal Q:

$$E \to TQ$$

$$Q \to +TQ| - TQ|\epsilon$$

Se crea otro no terminal R:

$$T \to FR$$

$$R \to *FR|/FR|\epsilon$$

Por lo que la gramática completa queda de la siguiente forma:

$$E \to TQ$$

$$Q \to +TQ| - TQ|\epsilon$$

$$T \to FR$$

$$R \to *FR|/FR|\epsilon$$

$$F \to i|(E)$$

Árboles de sintaxis

El árbol de sintaxis abstracta (**ASA**) o árbol de análisis sintáctico es el resultado de simplificar el árbol sintáctico concreto (**ASC**) hasta llegar a obtener lo que se necesita para representar el significado del programa. El **ASA** es un árbol que se define con mayor simplicidad y por lo tanto es más fácil de procesar en las etapas de ejecución.

Para representar a los árboles existe una notación que se escribe como a continuación se plantea:

$$\#(ABC)$$

donde el símbolo # indica que la primera letra dentro del paréntesis es un nodo y los otros caracteres representan a los hijos del nodo, como se muestra en la figura 7 :

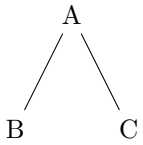

Figura 7: Árbol para la notación $\#(ABC)$.

Definición de lenguajes libres de contexto

Una gramática es un conjunto finito de reglas para generar un conjunto infinito de sentencias. En lenguaje natural, las sentencias están hechas de palabras y en el lenguaje de programación las sentencias están hechas de tokens.

Las gramáticas libres de contexto son una notación recursiva para describir conjuntos de cadenas e imponer una estructura en cada una de las cadenas. Esta notación puede en algunos casos ser traducida casi directamente en programas recursivos, pero es mejor generar un autómata de pila. Esto es similar al autómata finito del análisis léxico pero estos pueden usar adicionalmente una pila, lo cual permite contar y acoplar no localmente de símbolos. Hay dos formas de generar este autómata: LL y LR.

DEFINICIÓN 10 (Partes de la gramática LL.) *En una gramática LL(n). La primera L quiere decir lectura de los símbolos de izquierda a derecha y la segunda L quiere decir derivaciones por la izquierda. Lo que se coloca entre paréntesis es un número que indica la cantidad símbolos que se utilizan para hacer el análisis de las cadenas.*

DEFINICIÓN 11 (Partes de una gramática LR.) *En una gramática LR(n). La L quiere decir lectura de símbolos de izquierda a derecha y la R representa las derivaciones por la derecha. El número que se coloca entre paréntesis indica la cantidad de símbolos que se utilizan para hacer el análisis de las cadena.*

Se presentó la definición de estas gramáticas por medio de la jerarquía de Chomsky. La **gramática libre de contexto o gramática 2** tiene sus producciones de la forma:

$$A \to \alpha$$

donde $A \in N$ y $\alpha \in V^*$.

La gramática libre de contexto se expresa como:

$$G(N, T, P, S)$$

y se dice que es una gramática libre de contexto, debido a que sus producciones cumplen con la **forma 2** de la jerarquía de Chomsky.

DEFINICIÓN 12 (Lenguaje libre de contexto.) *Definir un lenguaje de programación significa describir la sintaxis y la semántica del mismo. Para la parte de sintaxis, un lenguaje (formal) L se caracterizará con referencia a una gramática G :*

$$L(G) = L(T, N, P, S)$$

por lo que si la gramática que define al lenguaje es libre de contexto, el lenguaje también será un lenguaje libre de contexto [6].

Tipos de analizadores sintácticos

Existen dos tipos de analizadores sintácticos:

1. Analizador sintáctico descendente.
2. Analizador sintáctico ascendente.

Análisis sintáctico descendente

El analizador sintáctico descendente comienza desde la raíz del árbol de análisis sintáctico trata de reconstruir el crecimiento del árbol que nos lleva a la cadena de tokens. De modo que haciendo la derivación izquierda, en efecto los pasos resultan en la forma sentencial de derivación, en orden inverso.

El análisis descendente se basa en gramáticas LL que permiten analizar una frase de entrada sin que existan bloqueos mutuos [6].

Análisis sintáctico ascendente

Este analizador también se conoce como analizador por desplazamiento y reducción. Este analizador intenta construir un árbol de análisis sintáctico para una cadena de entrada que comienza por las hojas (el fondo) y avanza hacia la raíz (la cima) [3].

Primeros y siguientes

Estos dos conjuntos de símbolos son importantes debido a que muestran los caracteres más importantes de la cadena y nos auxiliarán a plantear reglas que facilitarán la clasificación de las gramáticas LL(1).

Para poder calcular los primeros y siguientes de una cadena de caracteres, se basa en la utilización de las producciones.

DEFINICIÓN 13 (PRIMERO.) *Sea N un no terminal, el conjunto PRIMERO(N) son todos los símbolos terminales que pueden aparecer al principio de una frase que puede derivarse de una secuencia arbitraria de símbolos.*
Sean G(N, T, P, S) una gramática y α una secuencia arbitraria de símbolos, es decir, $\alpha \in (N \cup T)^*$. Definimos entonces el conjunto de terminales que pueden aparecer al principio de cualquier frase derivable de α como:

$$PRIMERO(\alpha) = \{t | t \in T_\epsilon \wedge \alpha \Rightarrow t\alpha'\}$$

donde $T_\epsilon = T \cup \{\epsilon\}$. [6].

EJEMPLO 16 *Se muestra como se calculan los PRIMEROS de la siguiente gramática:*

$$E \rightarrow TE'$$
$$E' \rightarrow +TE|\epsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT'|\epsilon$$
$$F \rightarrow (E)|id$$

Se calculan los PRIMEROS de los no terminales (E, E', T, T', F) de la gramática. Para el no terminal E, se usan las producciones 1, 3 y 5. Como la producción 1 es $E \rightarrow TE'$ por lo que se tiene que PRIMERO(TE') = PRIMERO(T), ya que T se deriva por la producción 4 en $*, \epsilon$, entonces se tiene que:

$$PRIMERO(TE') = PRIMERO(T) = \{*, \epsilon\}$$

Para la producción 3 es $T \rightarrow FT'$ por lo que se tiene que PRIMERO(FT') = PRIMERO(F), ya que F se deriva por la producción 5 en $(, id$, entonces se tiene que:

$$PRIMERO(FE') = PRIMERO(F) = \{(, id\}$$

Así que los PRIMERO(E), son:

$$PRIMERO(E) = \{PRIMERO(TE'), PRIMERO(FT')\} = \{*, (, id\}$$

Los PRIMERO(E') se obtienen de la producción 2 es $E' \rightarrow +TE|\epsilon$, por lo que:

$$PRIMERO(E') = \{+, \epsilon\}$$

Los PRIMERO(T') se obtienen de la producción 4 es $T' \rightarrow *FT'|\epsilon$, por lo que:

$$PRIMERO(T') = \{*, \epsilon\}$$

DEFINICIÓN 14 (SIGUIENTE) *Sea X un no terminal, el conjunto SIGUIENTE(X) son todos los símbolos terminales que pueden aparecer después de el no terminal X*

$$SIGUIENTE(X) = \{t | t \in T_\epsilon \wedge S \Rightarrow \alpha X t \beta\}$$

[6].

EJEMPLO 17 *Los SIGUIENTE de los no terminales de la gramática, se calculan de la siguiente manera:*

Como E es el símbolo inicial el SIGUIENTE(E) es $ (es eof). Para E basado en la producción 5, también el SIGUIENTE(E) es). Así que:

$$SIGUIENTE(E) = \{\$,)\}$$

Analizadores sintácticos descendentes

Para construir los analizadores sintácticos descendentes se considera la formación del árbol de análisis sintáctico y existen dos casos:

Uno, las hojas del árbol de análisis sintáctico contiene solo símbolos terminales, y la cadena de entrada se ha agotado. Es claro que el análisis tiene éxito.

Dos, un claro acoplamiento ocurre entre las hojas del árbol de análisis sintáctico parcialmente construido y la cadena de entrada. Para este caso existen dos situaciones:

Una primer situación es, que el analizador pudo haber seleccionado la producción errónea en un paso anterior del proceso, en este caso se puede retroceder,

sistemáticamente reconsiderando decisiones anteriores. Para una cadena que es una sentencia válida, el retroceso llevará al analizador a una secuencia correcta de selecciones y permitirá la construcción de un árbol de análisis sintáctico correcto. La segunda situación es, que la cadena de entrada no es una sentencia válida, el retroceso fallará y el analizador deberá reportar el error al usuario.

Un gran subconjunto de gramáticas libres de contexto pueden ser analizadas sin retroceso [5].

Los analizadores sintácticos descendentes son: el analizador sintáctico por descenso recursivo y el analizador sintáctico LL(1).

Construcción de analizadores sintácticos por descenso recursivo

Para construir el analizador sintáctico de descenso recursivo, conviene que la gramática sea LL(1). Ya que algunas de las gramáticas libres de contexto (GLC) tienen problemas, como:

1. Recursión izquierda o recursividad izquierda.

2. Ambigüedad.

3. Retroceso (backtracking).

Y aunque la mayoría de las gramáticas libres de contexto pueden ser analizadas sin retroceso. Es necesario comprobar que la GLC sea LL(1). Para saber que una GLC es LL(1) debe cumplir con los siguientes puntos:

PRIMERO, si las producciones tienen la forma $A \to \sigma_1|\sigma_2|\sigma_3$ deben cumplir con la regla $PRIMERO(\sigma_1) \cap PRIMERO(\sigma_2) \cap PRIMERO(\sigma_3) = \phi$. Llamada **regla UNO**.

SEGUNDO, si las producciones tienen la forma $X \to \sigma|\epsilon$ deben cumplir con la rgla $PRIMERO(X) \cap SIGUIENTE(X) = \phi$. Llamada **regla DOS**

MÉTODO 2 *Forma del esqueleto de programa para construir el analizador de descenso recursivo.*

Se tienen los siguientes no terminales $N_0, N_1, \cdots, N_m$ y el símbolo inicial $S = N_0$. Así que el programa consistirá que por cada no terminal, se creará un procedimiento y también un procedimiento de error y otro de lectura de carac-

teres, como se muestra a continuación:

Algorithm 0.1: Analizador sintáctico por descenso recursivo patrón

```
1 PROGRAMA
2 Pro: Error();
3 Pro: N_0;
4 Pro: N_1;
  ⋮
5 Pro: N_m;
6 Iniciar
7 SigPal();
8 Pro: N_0;
9 Fin
```

El árbol sintáctico se da implícitamente por la secuencia de llamadas a procedimientos. Una gramática libre de backtrack (retroceso) se puede tratar con descenso recursivo. Una analizador con descenso recursivo se estructura como un conjunto de procedimientos mutuamente recursivos, uno para cada no terminal de la gramática. El procedimiento que corresponde a un no terminal A, que reconoce un valor de A en la cadena de entrada. Para reconocer un no terminal B en el lado derecho de A ($A \rightarrow \alpha A \beta$), el analizador invoca el procedimiento correspondiente a B. Así la gramática por si misma sirve como una guía para la implementación del analizador.

EJEMPLO 18 *Construir el analizador sintáctico por descenso recursivo de la gramática G_6, que se muestra en el cuadro 2 .*

Número	Producción
1	$T \rightarrow R$
2	$T \rightarrow aTc$
3	$R \rightarrow bR$
4	$R \rightarrow \epsilon$

Cuadro 2: Gramática G_6, que se implementará con un analizador de descenso recursivo.

Las producciones 1 y 2 se escriben como sigue:

$$T \rightarrow R|aTc$$

A esta producción se le aplica la **regra UNO**:

$$PRIMERO(R) \cap PRIMERO(aTc) =$$
$$\{b\} \cap \{a\} = \phi$$

Por lo que esta producción cumple con la **regla UNO**.

Las producciones 3 y 4 se escriben como sigue:

$$R \rightarrow bR|\epsilon$$

Por lo que a esta producción se le debe aplicar la **regla DOS**:

$$PRIMERO(R) \cap SIGUIENTE(R) =$$

$$\{b\} \cap \{\$, c\} = \phi$$

La producción $T \rightarrow R|aTc$ cumple con la **regla UNO** y la producción $R \rightarrow bR|\epsilon$ cumple la **regla DOS**, entonces la gramática G_6 es una gramática LL(1). Por lo que es más fácil de aplicarle el procedimiento para construir el analizador sintáctico por descenso recursivo.

Gramática aumentada

Se agrega la producción $T' \rightarrow T$ al inicio del conjunto de producciones, con la finalidad de asegurar la aceptación de las cadenas de entrada que reconoce la gramática G_6. La gramática G_6 aumentada se muestra en el cuadro 3 .

Número	Producción
0	$T' \rightarrow T$
1	$T \rightarrow R$
2	$T \rightarrow aTc$
3	$R \rightarrow bR$
4	$R \rightarrow \epsilon$

Cuadro 3: Gramática aumentada G_6, que se implementará con un analizador de descenso recursivo.

Obtención de los primeros y siguientes

Se deben obtener los primeros y siguientes de la gramática G_6, los cuales se concentran en el cuadro 4 .

	T'	T	R
PRIMEROS	a, b	a, b	b, ϵ
SIGUIENTES	\$	\$, c	\$, c

Cuadro 4: Tablas de primero y siguientes de la gramática G_6.

Obtención de los procedimientos del análisis por descenso recursivo

Se debe obtener un procedimiento para cada no terminal utilizando los primeros y siguientes del cuadro 4 . Para el símbolo no terminal T', que tiene como

producción $T' \rightarrow T$:

Algorithm 0.2: Procedimiento del símbolo inicial T'()

```
1 Palabra ← SigPal();
2 if Palabra == PRIMERO(T') then
3 |   PRO T();
4 |   if Palabra == eof then
5 |   |   EsEnPan(SentenciaAceptada);
6 |   else
7 |   |   EsEnPan(Error);
8 else
9 EsEnPan(Error);
```

Para el símbolo no terminal T, que tiene como producción $T \rightarrow R|aTc$:

Algorithm 0.3: Procedimiento del símbolo T()

```
1 if Palabra == PRIMERO(T) then
2 |   PRO R();
3 else if Palabra == a then
4 |   Palabra := SigPal(a);
5 |   PRO T() ;
6 |   Palabra := SigPal(c) ;
7 else
8 |   EsEnPan(Error);
```

Para el símbolo no terminal R, que tiene como producción $R \rightarrow bR|\lambda$:

Algorithm 0.4: Procedimiento del símbolo R()

```
1 if Palabra == b then
2 |   Palabra := SigPal(b);
3 |   PRO R() ;
4 else if Palabra == SIGUIENTE(R) then
5 |   RETURN() ;
6 else
7 |   EsEnPan(Error);
```

Análisis de cadenas de entrada con el analizador de descenso recursivo

Para analizar la cadena de entrada **abbc**, se van utilizando los procedimientos de manera recursiva que se muestran en el cuadro 5 .

ENTRADA	PROCEDIMIENTO
$\uparrow abbc\$$	pal(a), PRO T'()
a $\uparrow bbc\$$	PRO T'(a)
a $\uparrow bbc\$$	PRO T(a), Acep
a $\uparrow bbc\$$	Pal(a),PRO T(), Pal(c), Acep
ab $\uparrow bc\$$	PRO T(b), Pal(c), Acep
ab $\uparrow bc\$$	PRO R(b), Pal(c), Acep
ab $\uparrow bc\$$	Pal(b), PRO R(), pal(c), Acep
abb $\uparrow c\$$	PRO R(b), Pal(c), Acep
abb $\uparrow c\$$	Pal(b), PRO R(), Pal(c), Acep
abb $\uparrow c\$$	PRO R(c), Pal(c), Acep
abb $\uparrow c\$$	Pal(c), Acep
abbc $\uparrow \$$	Acep

Cuadro 5: Tabla de análisis de la cadena de entrada **abba** con el analizador de descenso recursivo para la gramática G_6.

EJERCICIO 2 *Construir el analizador por descenso recursivo de la siguiente gramática;*

$$E' \rightarrow +TE' \mid -TE' \mid \lambda$$

$$T \rightarrow id$$

Para reconocer los valores de E', se crea una rutina E'(). Esta sigue un esquema simple: Seleccionar el número de producción basada en el conjunto de primeros de sus lados derechos. Para cada lado derecho, el código prueba directamente para cualquier símbolo futuro.

EJERCICIO 3 *Completar la construcción del analizador sintáctico de descenso recursivo de la siguiente gramática de expresiones G_7 del cuadro 6:*

Número	Producción
1	$E \rightarrow E + T \mid T$
2	$T \rightarrow T * F \mid F$
3	$F \rightarrow (E) \mid id$

Cuadro 6: Gramática de expresiones G_7.

Para aplicar el descenso recursivo se debe corregir la recursividad izquierda, por lo que la gramática queda de la siguiente forma:

1. $E \rightarrow TE'$
2. $E' \rightarrow +TE' \mid \lambda$
3. $T \rightarrow FT'$
4. $T' \rightarrow *FT' \mid \lambda$

5. $F \rightarrow (E) \mid id$

La gramática se escribe producción por producción y se aumenta como se muestra en el cuadro 7 :

Número	Producción
0	$G \rightarrow E$
1	$E \rightarrow TE'$
2	$E' \rightarrow +TE'$
3	$E' \rightarrow \lambda$
4	$T \rightarrow FT'$
5	$T' \rightarrow *FT'$
6	$T' \rightarrow \lambda$
7	$F \rightarrow (E)$
8	$F \rightarrow id$

Cuadro 7: Gramática de expresiones corregida para construir el analizador de descenso recursivo.

Se calculan los primeros:

1. Primero(G) = {(, id }
2. Primero(E) = {(,id }
3. Primero(E') = {+ }
4. Primero(T) = {(,id}
5. Primero(T') = {* }
6. Primero(F) = {(,id }

Se calculan los siguientes:

1. Siguiente(G) = { eof }
2. Siguiente(E) = { eof,) }
3. Siguiente(E') = { eof,) }
4. Siguiente(T) = { +, eof,) }
5. Siguiente(T') = { +, eof,) }
6. Siguiente(F) = { *, eof,)}

Los primeros y siguientes de la gramática G_7 se muestran en el cuafro 8 :

	G	E	E'	T	T'	F
PRIMEROS	(, id	(, id	+	(, id	*	(, id
SIGUIENTES	$	$,)	$,)	+, $	+,), $	*,), $

Cuadro 8: Tablas de primero y siguientes de la gramática G_7.

Se comienzan a plantear los procedimientos, el primer procedimiento es para el símbolo inicial:

Algorithm 0.5: Procedimiento del analizador de descenso recursivo

```
1  Pro G()
2  {
3  Pro E()
4  if Palabra == eof then
5  |  EsEnPan(SentenciaAceptada);
6  else
7  |  EsEnPan(Error);
8  }
9  Pro E()
10 {
11 Pro T();
12 Pro E'();
13 }
14 Pro E'()
15 {
16 if Palabra == + then
17 |  Palabra ← SigPal();
18 |  Pro T();
19 |  Pro E'();
20 }
21 Pro T()
22 {
23 Pro F();
24 Pro T'();
25 }
26 Pro T'()
27 {
28 if Palabra == * then
29 |  Palabra ← SigPal();
30 |  Pro F();
31 |  Pro T'();
32 }
33 Pro F()
34 {
35 if Palabra == id ∧ Palabra == ) then
36 |  Palabra ← SigPal();
37 |  Pro E();
38 |  Palabra ← SigPal();
39 }
```

Para comenzar se debe llamar el primer procedimiento que corresponde al

símbolo inicial, con producción $G \rightarrow E$ de la siguiente manera:

Algorithm 0.6: Procedimiento del símbolo inicial G()

```
1  Palabra ← SigPal();
2  Pro G();
3  if Palabra == PRIMERO(G) then
4  |  PRO E();
5  |  if Palabra == eof then
6  |  |  EsEnPan(SentenciaAceptada);
7  |  else
8  |  |  EsEnPan(Error);
9  else
10 EsEnPan(Error);
```

El siguiente procedimiento es para el no terminal E con producción $E \rightarrow TE'$, así que el procedimiento E() es como sigue:

Algorithm 0.7: Procedimiento del símbolo E()

```
1  Pro E();
2  if Palabra == PRIMERO(E) then
3  |  PRO T();
4  |  PRO E'() ;
5  else
6  EsEnPan(Error);
```

El siguiente procedimiento es para el no terminal E' con producción $E' \rightarrow +TE'|\lambda$, así que el procedimiento E'() es como sigue:

Algorithm 0.8: Procedimiento del símbolo E'()

```
1  Pro E'();
2  if Palabra == PRIMERO(E') then
3  |  Palabra := SigPal(+) ;
4  |  PRO T();
5  |  PRO E'() ;
6  else if Palabra == SIGUIENTE(E') then
7  |  RETURN() ;
8  else
9  |  EsEnPan(Error);
```

El siguiente procedimiento es para el no terminal T con producción $T \rightarrow FT'$, así que el procedimiento T() es como sigue:

Algorithm 0.9: Procedimiento del símbolo T()

```
1  Pro T();
2  if Palabra == PRIMERO(T) then
3  |  PRO F();
4  |  PRO T'() ;
5  else
6  EsEnPan(Error);
```

El siguiente procedimiento es para el no terminal T' con producción $T' \rightarrow$

$*FT'|\lambda$, así que el procedimiento T'() es como sigue:

Algorithm 0.10: Procedimiento del símbolo T'()

```
1  Pro T'();
2  if Palabra == PRIMERO(T') then
3  |  Palabra := SigPal(*) ;
4  |  PRO F();
5  |  PRO T'() ;
6  else if Palabra == SIGUIENTE(T') then
7  |  RETURN () ;
8  else
9  |  EsEnPan(Error);
```

El siguiente procedimiento es para el no terminal F con producción $F \rightarrow (E)|id$, así que el procedimiento F() es como sigue:

Algorithm 0.11: Procedimiento del símbolo F()

```
1  Pro F();
2  if Palabra == "(" then
3  |  Palabra := SigPal(() ;
4  |  PRO E();
5  |  Palabra := SigPal()) ;
6  else if Palabra == "id" then
7  |  Palabra := SigPal(id);
8  else if Palabra == SIGUIENTE(F) then
9  |  RETURN () ;
10 else
11 |  EsEnPan(Error);
```

Análisis de cadenas en un analizador de descenso recursivo. El análisis de la cadena **id * id + id**, se muestra en el cuadro 9 :

ENTRADA	PROCEDIMIENTO
$\uparrow id * id + id\$$	pal(id)
$\uparrow id * id + id\$$	PRO G(id)
$\uparrow id * id + id\$$	PRO E(a), Acep
$\uparrow id * id + id\$$	Pro T(), PRO E'(), Acep
$\uparrow id * id + id\$$	PRO T(id), PRO E'(), Acep
$\uparrow id * id + id\$$	PRO T(id), PRO E'(), Acep
$\uparrow id * id + id\$$	PRO F(id), PRO T'(), PRO E'(), Acep
$\uparrow id * id + id\$$	Pal(id), PRO T'(), PRO E'(), Acep
id $\uparrow *id + id\$$	PRO T'(*), PRO E'(), Acep
id * $\uparrow id + id\$$	Pal(*), PRO F(), PRO T'(), PRO E'(), Acep
id * $\uparrow id + id\$$	PRO F(id), PRO T'(), PRO E'(), Acep
id * id $\uparrow +id\$$	Pal(id), PRO T'(), PRO E'(), Acep
id * id $\uparrow +id\$$	PRO T'(+), PRO E'(), Acep
id * id $\uparrow +id\$$	PRO E'(+), Acep
id * id $\uparrow +id\$$	Pal(+), PRO T(), PRO E'(), Acep
id * id + $\uparrow id\$$	PRO T(id), PRO E'(), Acep
id * id + $\uparrow id\$$	PRO F(id), PRO T'(), PRO E'(), Acep
id * id + $\uparrow id\$$	Pal(id), PRO T'(), PRO E'(), Acep
id * id + id $\uparrow$ \$	PRO T'($), PRO E'(), Acep
id * id + id $\uparrow$ \$	PRO E'($), Acep
id * id + id $\uparrow$ \$	Acep
id * id + id $\uparrow$ \$	SentenciaAceptada

Cuadro 9: Tabla de análisis de la cadena de entrada **id * id + id** con el analizador de descenso recursivo para la gramática G_7 de expresiones corregida.

Se mete en una lista enlazada la cadena de entrada y se va leyendo carácter por carácter por medio de la función de lectura **SigCar()**, en la columna **PROCEDIMIENTO** del cuadro 9 se escribe la función o procedimiento del no terminal hasta consumirse la función o procedimiento. La aceptación es cuando coincide que en la lista de entrada aparece el símbolo \$ y en la columna **PROCEDIMIENTO** aparece **Acep**.

Comprobación por derivación del analizador por descenso recursivo. La derivación de la cadena con el uso de las producciones de la gramática de expresiones corregida:

$$G \Rightarrow E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow idT'E' \Rightarrow id * FT'E'$$

$$\Rightarrow id * idT'E' \Rightarrow id * id\lambda E' \Rightarrow id * idE'$$

$$\Rightarrow id * id + TE' \Rightarrow id * id + FT'E' \Rightarrow id * id + idT'E'$$

$$\Rightarrow id * id + id\lambda E' \Rightarrow id * id + id\lambda\lambda \Rightarrow id * id + id$$

Construcción de analizadores sintácticos LL(1)

Para estudiar este analizador se usan las gramáticas libres de contexto, pero no todas las gramáticas libres de contexto son ideales. Las gramáticas libres de contexto que no tengan esa ventaja, deben ser adaptadas para poder programar sin retroceso.

El analizador sintáctico LL(1) es descendente. Para poder determinar si la gramática es del tipo LL(1), debe cumplir con dos reglas:

1. **Regla UNO**. Si la producción es de la forma $A \to \sigma_1 \mid \sigma_2 \mid \cdots \mid \sigma_n$, se debe cumplir con la siguiente condición: $PRIMERO(\sigma_1) \cap PRIMERO(\sigma_2) \cap \cdots \cap PRIMERO(\sigma_n) = \Phi$.
2. **Regla DOS**. Si la producción es de la siguiente forma $X \to \sigma \mid \lambda$, se aplica la siguiente condición $PRIMERO(X) \cap SIGUIENTE(X) = \Phi$

Cuando una gramática no es LL(1), para convertirla en LL(1) se debe seguir el siguiente procedimiento:

1. Se aplican las reglas UNO y DOS, para determinar si es LL(1).
2. Se debe determinar si las producciones de la gramática tienen recursividad izquierda. Esto se comprueba si las producciones tienen la siguiente forma $A \to A\alpha \mid \beta$.
3. Se debe eliminar la recursividad izquierda, haciendo la siguiente transformación: $A \to \beta A'$, $A' \to \alpha A' \mid \lambda$.

El siguiente ejemplo muestra el procedimiento para construir el analizador LL(1).

EJEMPLO 19 *Construir el analizador sintáctico LL(1) de la gramática G_7:*

1. $S \to uBDz$
2. $B \to w|Bv$
3. $D \to EF$
4. $E \to y|\epsilon$
5. $F \to x|\epsilon$

Calcular los primeros y siguientes de los no terminales

Se obtienen los primeros y siguientes de los no terminales de la gramática, los cuales se muestran en el cuadro 10 .

	S	B	B'	D	E	F
PRIMEROS	u	w,PRIM(B)	v, ϵ	y, x	y, ϵ	x, ϵ
SIGUIENTES	$	y, x, z	y, x, z	z	x, z	z

Cuadro 10: Tablas de primero y siguientes de la gramática G_7.

Comprobar si la gramática es LL(1)

Se comprueba si la gramática G_7 es del tipo LL(1). Para esto se aplican las reglas UNO y DOS, como a continuación se muestra. A la producción 2 se le aplica la regla UNO.

$$PRIMERO(w) \cap PRIMERO(Bv) =$$
$$\{w\} \cap \{w\} \neq \Phi$$

A la producción 4 se aplica la regla DOS:

$$PRIMERO(E) \cap SIGUIENTE(E) =$$
$$\{y\} \cap \{x, z\} = \phi$$

A la producción 5 se le aplica la regla DOS.

$$PRIMERO(F) \cap SIGUIENTE(F) =$$
$$\{x\} \cap \{z\} = \phi$$

La producción 2 no cumple la regla UNO, debido a que tiene recursividad izquierda. La forma de la producción 2, es como sigue:

$$A \rightarrow A\alpha|\beta$$

La producción 2 se escribe a continuación:

$$B \rightarrow Bv|w$$

Convertir la gramática en LL(1)

Para hacer que la producción 2 cumpla con la regla UNO, se elimina la recursividad izquierda haciendo la siguiente transformación:

$$A = B, \quad A' = B', \quad \alpha = v, \quad \beta = w$$

Estos valores se substituyen en las siguientes producciones:

$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A'|\epsilon$$

La producción 2 se cambia por la producción 2 sin recursividad en la gramática. La producción 2 sin recursividad izquierda es la siguiente:

$$B \rightarrow wB'$$
$$B' \rightarrow vB'|\epsilon$$

La gramática G_7 y sin recursividad izquierda, se presenta a continuación:

1. $S \rightarrow uBDz$
2. $B \rightarrow wB'$
3. $B' \rightarrow vB'|\epsilon$

4. $D \rightarrow EF$
5. $E \rightarrow y$
6. $E \rightarrow \epsilon$
7. $F \rightarrow x$
8. $F \rightarrow \epsilon$

A la producción 3 se le aplica la regla DOS:

$$PRIMERO(B') \cap SIGUIENTE(B') =$$

$$\{v\} \cap \{y, z\} = \phi$$

Como la producción 3 cumple con la regla DOS, la gramática G_7 es ahora una gramática LL(1). La gramática G_7 expandida es:

1. $S \rightarrow uBDz$
2. $B \rightarrow wB'$
3. $B' \rightarrow vB'$
4. $B' \rightarrow \epsilon$
5. $D \rightarrow EF$
6. $E \rightarrow y$
7. $E \rightarrow \epsilon$
8. $F \rightarrow x$
9. $F \rightarrow \epsilon$

La nueva tabla de primeros y siguientes es la que se muestra en el cuadro 11 :

	S	B	B'	D	E	F
PRIMEROS	u	w	v, ϵ	y, x	y, ϵ	x, ϵ
SIGUIENTES	$	y, x, z	y, x, z	z	x, z	z

Cuadro 11: Tablas de primero y siguientes de la gramática G_7 convertida a LL(1).

Construcción de la tabla LL(1)

Para llenar el cuadro 12 , se utilizan los primeros y siguientes del cuadro 10 . Para las producciones que no generan la cadena vacía de la gramática G_7 expandida se usan los primeros. Para las producciones que generan la cadena vacía se usan los siguientes.

NO TERMINALES	u	w	v	y	x	z	$
S	1	e	e	e	e	e	e
B	e	2	e	e	e	e	e
B'	e	e	3	4	4	4	e
D	e	e	e	5	5	e	e
E	e	e	e	6	7	7	e
F	e	e	e	e	8	9	e

Cuadro 12: Tabla LL(1) de la gramática G_7.

Análisis de cadenas de entrada con un analizador LL(1)

Se analiza la cadena de entrada **uwvyxz** y el análisis se muestra en el cuadro 13 :

ENTRADA	PILA	NUM. PRODUCCIÓN
$\uparrow uwvyxz\$$	\$ S	1
$\uparrow uwvyxz\$$	\$ zDBu	$\leftarrow$
u $\uparrow wvyxz\$$	\$ zDB	2
u $\uparrow wvyxz\$$	\$ zDB'w	$\leftarrow$
u w $\uparrow vyxz\$$	\$ zDB'	3
u w $\uparrow vyxz\$$	\$ zDB'v	$\leftarrow$
u wv $\uparrow yxz\$$	\$ zDB'	4
u wv $\uparrow yxz\$$	\$ zD	5
u wv $\uparrow yxz\$$	\$ zFE	6
u wv $\uparrow yxz\$$	\$ zFy	$\leftarrow$
u wvy $\uparrow xz\$$	\$ zF	8
u wvy $\uparrow xz\$$	\$ zx	$\leftarrow$
u wvyx $\uparrow z\$$	\$ z	$\leftarrow$
u wvyxz $\uparrow$ \$	\$	Aceptación

Cuadro 13: Tabla de análisis de la cadena de entrada $uwvyxz$ con la tabla LL(1) de la gramática G_7.

La cadena de entrada se mete a una lista enlazada y de esta lista se va leyendo carácter por carácter por medio de la función **SigCar()**. En la pila se mete el símbolo \$ y el símbolo inicial de la gramática y en la columna del cuadro 13 se escribe el número de producción utilizada. La flecha indica que se aplica un pop() a la pila. Cuando aparece la producción indicada por el número que se escribe en la columna **NUM PRODUCCIÓN**, se cambia el símbolo no

terminal por su equivalencia que se expresa en la parte derecha del número de producción que se indica con el número.

Programación del analizador LL(1). La forma de programar una gramática LL(1), se hace de la siguiente forma:

Algorithm 0.12: Análisis descendente LL(1)

```
1  Palabra ← SigPal();
2  PUSH(eof,Pila);
3  PUSH(SimboloInicial,Pila);
4  Cima ← Cima.valor;
5  while True do
6      if Cima == eof ∧ Palabra == eof then
7          EsEnPan(Sentencia Aceptada);
8          Salir();
9      if Cima ∈ T ∨ Cima == eof then
10         if Cima == Palabra then
11             POP(Pila);
12             Palabra ← SigPal() ;
13         else
14             EsEnPan(Sentencia No Aceptada);
15     else
16         if Cima ∈ N then
17             A ← Leer(Tabla[Cima, Palabra]);
18             k ←| A |;
19             POP(Pila);
20             for i ← k hasta 1 do
21                 if B_i <> λ then
22                     PUSH(B_i,Pila);
23         else
24             EsEnPan(No se puede expandir B_i);
25     Cima ← Cima.valor;
```

Ejemplo de construcción de un analizador sintáctico descendente

Se muestra la gramática libre de contexto G_8 en el cuadro 14 :

NÚMERO	PRODUCCIÓN
1	$S \to AB$
2	$A \to aa \vert a$
3	$B \to b \vert ac$

Cuadro 14: Gramática G_8.

Basados en la gramática G_8 construir los analizadores LL(1) y el descenso

recursivo. Primero se comprueba que la gramática es LL(1), como se muestra a continuación:

Se aplican la regla UNO a las producciones 2 y 3. Para la producción 2:

$$PRIMERO(aa) \cap PRIMERO(a) \neq \phi$$

$$\{a\} \cap \{a\} \neq \phi$$

Se aplica la regla UNO a la producción 3:

$$PRIMERO(b) \cap PRIMERO(ac) = \phi$$

$$\{b\} \cap \{a\} = \phi$$

Como la producción 2 no cumple con la regla UNO. Entonces la gramática se debe transformar en LL(1). Para cumplir con la característica de gramática LL(1), se transforma la producción 2 como sigue:

$$A \rightarrow aa|a$$

Se factoriza como se muestra a continuación:

$$A \rightarrow a(a|\lambda)$$

$$A \rightarrow aC$$

$$C \rightarrow a|\lambda$$

Ahora, a la producción $C \rightarrow a|\lambda$, se la aplica la regla DOS:

$$PRIMERO(C) \ \cap \ SIGUIENTE(C)$$

$$\{a\} \cap \{a, b\} \neq \phi$$

Como no se puedo corregir, entonces se procede a reescribir la gramática. Se comienza con la producción 1, como se muestra en seguida:

$$S \rightarrow AB$$

En la producción 1 se substituyen las producciones 2 y 3 como se muestra:

$$S \rightarrow (aa|a)(b|ac)$$

Haciendo las operaciones en la parte derecha de la producción queda de la siguiente forma:

$$S \rightarrow aab|aaac|ab|aac$$

Se factoriza la producción del no terminal S y queda como sigue:

$$S \rightarrow a(ab|aac|b|ac)$$

$$S \rightarrow aD$$

$$D \rightarrow ab|aac|b|ac$$

Se factoriza la producción del no terminal D y queda como sigue:

$$D \rightarrow a(ac|b|c)|b$$

$$D \rightarrow aE|b$$

$$E \rightarrow ac|b|c$$

Por lo que la gramática convertida a LL(1) se muestra en el cuadro 15 :

NÚMERO	PRODUCCIÓN
1	$S \rightarrow aD$
2	$D \rightarrow aE$
3	$D \rightarrow b$
4	$E \rightarrow ac$
5	$E \rightarrow b$
6	$E \rightarrow c$

Cuadro 15: Gramática G_8 convertida a LL(1).

Analizador LL(1). Primero se obtiene la tabla de primeros y siguientes que se muestra en 16 :

	S	D	E
PRIMEROS	a	a, b	a, b, c
SIGUIENTES	$	$	$

Cuadro 16: Tabla de primeros y siguientes de la gramática G_8 convertida a LL(1).

Con el apoyo de la tabla 16 se obtiene la tabla LL(1) que se muestra en la tabla 17 :

	a	b	c	$
S	1	e	e	e
D	2	3	e	e
E	4	5	6	e

Cuadro 17: Tabla LL(1) de la gramática G_8 convertida a LL(1).

Se analiza la cadena **aab** usando el cuadro 17 de donde se obtiene el cuadro de análisis 18 :

ENTRADA	PILA	NUM. PRODUCCIÓN
$\uparrow aab\$$	\$ S	1
$\uparrow aab\$$	\$ D a	$\leftarrow$
a $\uparrow ab\$$	\$ D	2
a $\uparrow ab\$$	\$ E a	$\leftarrow$
a a $\uparrow b\$$	\$ E	4
a a $\uparrow b\$$	\$ b	$\leftarrow$
a a b $\uparrow$ \$	\$	Acep

Cuadro 18: Tabla de análisis de la cadena de entrada **aab** con la tabla LL(1) de la gramática G_8.

Analizador de descenso recursivo. Lo primero que se hace es aumentar la gramática la cual se muestra en el cuadro 19 :

NÚMERO	PRODUCCIÓN
0	$G \to S$
1	$S \to aD$
2	$D \to aE$
3	$D \to b$
4	$E \to ac$
5	$E \to b$
6	$E \to c$

Cuadro 19: Gramática aumentada de G_8 convertida a LL(1).

Se obtiene la tabla de primeros y siguientes que se muestra en el cuadro 20 :

	G	S	D	E
PRIMEROS	a	a	a	a, b, c
SIGUIENTES	\$	\$	\$	\$

Cuadro 20: Tabla de primeros y siguientes de la gramática aumentada G_8 convertida a LL(1).

Apoyados en la gramática G_8 y la tabla de primeros y siguientes del cuadro 20 , se obtienen las funcionas para cada uno de los no terminales G, S, D, E de

la gramática. Para la producción $G \rightarrow S$:

Algorithm 0.13: Función G() de la gramática G_8 convertida a LL(1).

```
1  Palabra ← SigPal();
2  Fun G()
3  if Pal == PRIMERO(G)) then
4  |  Fun S();
5  |  if Pal == eof then
6  |  |  EsEnPan(Sentencia Aceptada);
7  |  else
8  |  |  EsEnPan(ERROR);
9  else
10 |  EsEnPan(Sentencia NO aceptada) ;
```

Para la producción $S \rightarrow aD$ la función es la siguiente:

Algorithm 0.14: Función S() de la gramática G_8 convertida a LL(1).

```
1  Fun S()
2  if Pal == PRIMERO(S) then
3  |  Palabra ← SigPal(a);
4  |  Fun D();
5  else
6  |  EsEnPan(ERROR) ;
```

Para las producciones $D \rightarrow aE$ y $D \rightarrow b$ la función es la siguiente:

Algorithm 0.15: Función D() de la gramática G_8 convertida a LL(1).

```
1  Fun D()
2  if Pal == a then
3  |  Palabra ← SigPal(a);
4  |  Fun E();
5  else if pal == b then
6  |  Palabra ← SigPal(b);
7  else
8  |  EsEnPan(ERROR);
```

Para las producciones $E \rightarrow ac$, $E \rightarrow b$, $E \rightarrow c$ la función es la siguiente:

Algorithm 0.16: Función E() de la gramática G_8 convertida a LL(1).

```
1  Fun E()
2  if Pal == a then
3  |  Palabra ← SigPal(a);
4  |  Palabra ← SigPal(c);
5  else if pal == b then
6  |  Palabra ← SigPal(b);
7  else if pal == c then
8  |  Palabra ← SigPal(c);
9  else
10 |  EsEnPan(ERROR);
```

Analizar con las funciones la cadena **aab**, el análisis se muestra en el cuadro 21 :

ENTRADA	FUNCIÓN
$\uparrow aab\$$	Pal(a)
$\uparrow aab\$$	Fun G(a)
$\uparrow aab\$$	Fun S(a), Acep
a $\uparrow ab\$$	Pal(a), Fun D(), Acep
a $\uparrow ab\$$	Fun D(a), Acep
a $\uparrow ab\$$	Pal(a), Fun E(), Acep
a a $\uparrow b\$$	Fun E(b), Acep
a a $\uparrow b\$$	Pal(b), Acep
a a b $\uparrow \$$	Acep
a a b $\uparrow \$$	Sentencia aceptada

Cuadro 21: Tabla de análisis de la cadena de entrada **aab** con las funciones de la gramática G_8.

Ejercicios propuestos de analizadores sintácticos descendentes

Resolver los siguientes ejercicios por medio de la metodología de LL(1) y de descenso recursivo:

Ejercicio 1. Se tiene la siguiente gramática que se presenta en la tabla 22 :

NÚMERO	PRODUCCIÓN
1	$S \rightarrow aS\|bA$
2	$A \rightarrow d\|ccA$

Cuadro 22: Gramática del ejercicio 1.

Ejercicio 2 (si). Se tiene la siguiente gramática que se presenta en la tabla 23 :

NÚMERO	PRODUCCIÓN
1	$S \rightarrow aA\|cB\|b\|d$
2	$A \rightarrow aA\|b$
3	$B \rightarrow cB\|d$

Cuadro 23: Gramática del ejercicio 2.

Ejercicio 3. Se tiene la siguiente gramática que se presenta en la tabla 24 :

NÚMERO	PRODUCCIÓN
1	$S \rightarrow aAd \vert aB$
2	$A \rightarrow b \vert c$
3	$B \rightarrow ccd \vert ddc$

Cuadro 24: Gramática del ejercicio 3.

Ejercicio 4 (si). Se tiene la siguiente gramática que se presenta en el cuadro 25 :

NÚMERO	PRODUCCIÓN
1	$Z \rightarrow E$
2	$E \rightarrow T \vert EMT$
3	$M \rightarrow * \vert \lambda$
4	$T \rightarrow i \vert n$

Cuadro 25: Gramática del ejercicio 4.

Construcción del analizador LL(1) de la gramática del ejercicio 4. Se comprueba que la gramática sea LL(1), aplicando las reglas UNO y DOS. Solamente se aplican para las producciones que tienen alternativas, es el caso de las producciones 2, 3 y 4. Para la **producción 2** se aplica la regla UNO:

$$PRIMERO(T) \cap PRIMERO(EMT) =$$
$$\{PRIMERO(T)\} \cap \{PRIMERO(T), PRIMERO(E)\} \neq \phi$$

La producción 2 no cumple la regla UNO, se debe corregir. Para la **producción 3** se aplica la regla DOS ya que el no terminal M genera la cadena vacia:

$$PRIMERO(M) \cap SIGUIENTE(M) =$$
$$\{*\} \cap \{PRIMERO(T)\}$$
$$\{*\} \cap \{i, n\} = \phi$$

La producción 3 si cumple con la regla DOS. Para la **producción 4** se aplica la regla UNO:

$$PRIMERO(i) \cap PRIMERO(n) =$$
$$\{i\} \cap \{n\} = \phi$$

La producción 4 si cumple con la regla UNO. Así que se procede a corregir a la **producción 2**, la cual presenta la recursividad izquierda. Por lo que para corregirla se aplica la eliminación de la recursividad izquierda.

Se usa la producción de la forma general de la recursividad izquierda, se compara para determinar las igualdades y usar la transformación para eliminar la recursividad izquierda, de la siguiente manera:

$$A \to A\alpha|\beta$$

$$E \to EMT|T$$

Las igualdades que se obtienen son :

$$A = E, A' = E', \alpha = MT, \beta = T$$

Se aplica la transformación:

$$A \to \beta A'$$

$$A' \to \alpha A'|\lambda$$

Y se obtienen las producciones ya corregidas:

$$E \to TE'$$

$$E' \to MTE'|\lambda$$

Así que la gramática queda como se muestra en el cuadro 26 :

NÚMERO	PRODUCCIÓN
1	$Z \to E$
2	$E \to TE'$
3	$E' \to MTE'\|\lambda$
4	$M \to *\|\lambda$
5	$T \to i\|n$

Cuadro 26: Gramática corregida del ejercicio 4.

Se procede a aplicar la regla DOS a la producción 3 de la gramática del cuadro 26 :

$$PRIMERO(E') \cap SIGUIENTE(E') =$$

$$\{i, n\} \cap \{\$\} = \phi$$

Ya que la producción 3 cumple con la regla DOS, la gramática del cuadro 26 es LL(1). Por lo que para obtener su tabla LL(1). Se lista la gramática expandida:

NÚMERO	PRODUCCIÓN
0	$S \to Z$
1	$Z \to E$
2	$E \to TE'$
3	$E' \to MTE'$
4	$E' \to \lambda$
5	$M \to *$
6	$M \to \lambda$
7	$T \to i$
8	$T \to n$

Cuadro 27: Gramática corregida y expandida del ejercicio 4.

De la gramática del cuadro 27 se obtienen los primeros y siguientes de los no terminales que se muestran en el cuadro 28 :

	Z	E	E'	M	T
PRIMEROS	i, n	i, n	*, i, n	*	i, n
SIGUIENTES	$	$	$	i, n	*, i, n, $

Cuadro 28: Tabla de primeros y siguientes de la gramática corregida y expandida del ejercicio 4.

La tabla LL(1) se obtiene basándose en el cuadro 27 de las producciones y en el cuadro 28 de los primeros y siguientes, como se muestra en el cuadro 29 , donde se escriben las producciones a utilizar de acuerdo a los primeros y siguientes de cada no terminal de la gramática corregida del ejercicio 4:

NÚMERO	i	n	*	$
Z	1	1	e	e
E	2	2	e	e
E'	3	3	3	4
M	6	6	5	e
T	7	8	e	e

Cuadro 29: Tabla LL(1) de la gramática corregida del ejercicio 4.

Con el uso del cuadro 29 se analiza la cadena **i*n*i**, el análisis se describe en el cuadro 30 :

ENTRADA	PILA	NUM. PRODUCCIÓN
$\uparrow i * n * i\$$	\$ Z	1
$\uparrow i * n * i\$$	\$ E	2
$\uparrow i * n * i\$$	\$ E' T	7
$\uparrow i * n * i\$$	\$ E' i	$\leftarrow$
i $\uparrow * n * i\$$	\$ E'	3
i $\uparrow * n * i\$$	\$ E' T M	5
i $\uparrow * n * i\$$	\$ E' T *	$\leftarrow$
i * $\uparrow n * i\$$	\$ E' T	8
i * $\uparrow n * i\$$	\$ E' n	$\leftarrow$
i * n $\uparrow * i\$$	\$ E'	3
i * n $\uparrow * i\$$	\$ E' T M	5
i * n $\uparrow * i\$$	\$ E' T *	$\leftarrow$
i * n * $\uparrow i\$$	\$ E' T	7
i * n * $\uparrow i\$$	\$ E' i	$\leftarrow$
i * n * i $\uparrow$ \$	\$ E'	4
i * n * i $\uparrow$ \$	\$ λ	Acep

Cuadro 30: Tabla de análisis de la cadena de entrada **i*n*i** con la tabla LL(1) de la gramática corregida del ejercicio 4.

Construcción del analizador por descenso recursivo de la gramática corregida del ejercicio 4. La construcción del analizador sintáctico de descenso recursivo se inicia con el aumento de la gramática, la que se muestra en el cuadro 31 :

NÚMERO	PRODUCCIÓN
1	$Z \rightarrow E$
2	$E \rightarrow TE'$
3	$E' \rightarrow MTE'$
4	$E' \rightarrow \lambda$
5	$M \rightarrow *$
6	$M \rightarrow \lambda$
7	$T \rightarrow i$
8	$T \rightarrow n$

Cuadro 31: Gramática corregida, expandida y aumentada del ejercicio 4.

La gramática del cuadro 27 es una gramática aumentada, por lo que solo se escribe otra vez la gramática en el cuadro 31 . La tabla de primeros y siguientes es la que se muestra en el cuadro 28 . Las funciones se presentan a continuación,

para la producción $S \rightarrow Z$:

Algorithm 0.17: Función S() de la gramática corregida del ejercicio 4.

```
1  Palabra ← SigPal();
2  Fun S()
3  if Palabra == PRIMERO(S) then
4  |  Fun Z();
5  |  if Palabra == eof then
6  |  |  EsEnPan(Sentencia Aceptada);
7  |  else
8  |  |  EsEnPan(ERROR);
9  else
10 EsEnPan(Sentencia NO aceptada);
```

Ahora para la producción $Z \rightarrow E$:

Algorithm 0.18: Función Z() de la gramática corregida del ejercicio 4.

```
1  Palabra ← SigPal();
2  Fun Z()
3  if Palabra == PRIMERO(Z) then
4  |  Fun E();
5  |  if Palabra == eof then
6  |  |  EsEnPan(Sentencia Aceptada);
7  |  else
8  |  |  EsEnPan(ERROR);
9  else
10 EsEnPan(Sentencia NO aceptada);
```

La función para la producción E considerando la producción $E \rightarrow TE'$ es la siguiente:

Algorithm 0.19: Función E() de la gramática corregida del ejercicio 4.

```
1  Fun E()
2  if Palabra == PRIMERO(E) then
3  |  Fun T();
4  |  Fun E'();
5  else
6  EsEnPan(ERROR);
```

La función para el no terminal E' considerando la producción $E' \rightarrow MTE'|\lambda$

es la siguiente:

Algorithm 0.20: Función E'() de la gramática corregida del ejercicio 4.

```
1 Fun E'()
2 if Palabra == PRIMERO(E') then
3 |   Fun M();
4 |   Fun T();
5 |   Fun E'();
6 else if Palabra == SIGUIENTE(E') then
7 |   RETURN();
8 else
9 |   EsEnPan(ERROR);
```

La función para el no terminal M considerando la producción $M \rightarrow *|\lambda$ es la siguiente:

Algorithm 0.21: Función M() de la gramática corregida del ejercicio 4.

```
1 Fun M()
2 if Palabra == PRIMERO(M) then
3 |   Palabra ← SigPal(*);
4 else if Palabra == SIGUIENTE(M) then
5 |   RETURN();
6 else
7 |   EsEnPan(ERROR);
```

La función para el no terminal T considerando la producción $T \rightarrow i|n$ es la siguiente:

Algorithm 0.22: Función T() de la gramática corregida del ejercicio 4.

```
1 Fun T()
2 if Palabra == i then
3 |   Palabra ← SigPal(i);
4 else if Palabra == n then
5 |   Palabra ← SigPal(n);
6 else
7 |   EsEnPan(ERROR);
```

El siguiente paso es el análisis de cadenas utilizando las funciones ya construidas anteriormente. Se analiza la cadena **i*n*i**, el análisis se muestra en el cuadro 32 :

ENTRADA	ACCIÓN
$\uparrow i * n * i\$$	Pal(i)
$\uparrow i * n * i\$$	Fun Z(i)
$\uparrow i * n * i\$$	Fun E(i), Acep
$\uparrow i * n * i\$$	Fun T(i), Fun E'(), Acep
$\uparrow i * n * i\$$	Palabra(i), Fun E'(), Acep
i $\uparrow *n * i\$$	Fun E'(*), Acep
i $\uparrow *n * i\$$	Fun M(*), Fun T(), Fun E'(), Acep
i $\uparrow *n * i\$$	Palabra(*), Fun T(), Fun E'(), Acep
i * $\uparrow n * i\$$	Fun T(n), Fun E'(), Acep
i * $\uparrow n * i\$$	Palabra(n), Fun E'(), Acep
i * n $\uparrow *i\$$	Fun E'(*), Acep
i * n $\uparrow *i\$$	Fun M(*), Fun T(), Fun E'(), Acep
i * n $\uparrow *i\$$	Palabra(*), Fun T(), Fun E'(), Acep
i * n * $\uparrow i\$$	Fun T(i), Fun E'(), Acep
i * n * $\uparrow i\$$	Palabra(i), Fun E'(), Acep
i * n * i $\uparrow$ \$	Fun E'($), Acep
i * n * i $\uparrow$ \$	Acep
i * n * i $\uparrow$ \$	Sentencia Aceptada

Cuadro 32: Tabla de análisis de la cadena de entrada **i*n*i** con las funciones de la gramática corregida del ejercicio 4.

Ejercicio 5 (si). Se tiene la siguiente gramática que se presenta en la tabla 33 :

NÚMERO	PRODUCCIÓN
1	$S \to aAc$
2	$A \to Ab \vert \lambda$

Cuadro 33: Gramática del ejercicio 5.

Ejercicio 6. Se tiene la siguiente gramática que se presenta en la tabla 34 :

NÚMERO	PRODUCCIÓN
1	$S \to AB$
2	$A \to aa \vert a$
3	$B \to b \vert ac$

Cuadro 34: Gramática del ejercicio 6.

Ejercicio 7. Se tiene la siguiente gramática que se presenta en la tabla 35 :

NÚMERO	PRODUCCIÓN
1	$A \to aB \mid aC$
2	$B \to bB \mid b$
3	$C \to bC \mid c$

Cuadro 35: Gramática del ejercicio 7.

Ejercicio 8. Se tiene la siguiente gramática que se presenta en la tabla 36 :

NÚMERO	PRODUCCIÓN
1	$S \to A \mid xb$
2	$A \to aAb \mid B$
3	$B \to x$

Cuadro 36: Gramática del ejercicio 8.

Ejercicio 9. Se tiene la siguiente gramática que se presenta en la tabla 37 :

NÚMERO	PRODUCCIÓN
1	$S \to LB$
2	$B \to ;; S; L \mid := L$
3	$L \to (EJ \mid, EJ \mid)$
4	$E \to a \mid L$

Cuadro 37: Gramática del ejercicio 9.

Ejercicio 10. Se tiene la siguiente gramática que se presenta en la tabla 38 :

NÚMERO	PRODUCCIÓN
1	$S \to bAe$
2	$S \to \lambda$
3	$A \to [S]$

Cuadro 38: Gramática del ejercicio 10.

Construcción de tablas

Descubriendo la derivación de una cadena de entrada

El compilador debe inferir la derivación para una cadena de entrada. El proceso de construir una derivación de una sentencia de entrada se llama análisis. Un analizador toma como entrada un presunto programa escrito en algún lenguaje fuente. El analizador ve el programa como este emerge del escaner: un flujo de palabras acompañadas de su categoría sintáctica. Y la salida es una derivación o un mensaje de error para el programa de entrada. Visualizar el

análisis para construir un árbol de análisis para el programa de entrada.

El analizador descendente se construye mediante un algoritmo que se implementa con una derivación más a la izquierda, este árbol de análisis anclado a la variable raíz, usa una pila, con funciones de acceso push() y pop() para seguir a la parte no acoplada de la periferia del árbol.

Algorithm 0.23: Análisis descendente más a la izquierda

```
1  raiz ← node para el simbolo inicial, S;
2  foco ← raiz;
3  push(null) ;
4  palabra ← SigPal();
5  while true do
6      if foco es un no terminal then
7          siguiente regla a expandir foco (A → β1 , β2 , ...,βn ) ;
8          construir nodos para β1 , β2 ...,βn como hijos de foco ;
9          push(βn , βn−1 , ..., β2 ) ;
10         foco ← β1;
11     if igualar palabra con foco then
12         palabra ← SigPal();
13         foco ← pop();
14     if palabra == eof  and  foco == null then
15         aceptar la entrada y regresar raiz ;
16     else
17         retroceso ;
```

Construcción de tablas:

Algorithm 0.24: Construcción de una tabla

```
1  switch valores de T do
2      case valor uno de T do
3          Hacer eso;
4          hacer aquello;
5      case valor dos de T do
6          Hacer eso dos;
7          hacer aquello dos;
8      case otro valor de T do
9          una sola linea;
10     case último valor de T do
11         Hacer esto;
12         break;
13     otherwise do
14         Para los otros valores de T;
15         Hacer aquello;
```

Aplicar la construcción de tablas para construir el analizador sintáctico, se

hace de la siguiente forma:

Algorithm 0.25: Analizador LL()

```
switch cadena == leerCadenaEntrada() do
    case cadena == a do
        usar produccion 1 ;
        substituir() ;
    case cadena == b do
        usar producción 2 ;
        substituir() ;
    case cadena == ϵ do
        no_substituir();
    case cadena == c do
        usar producción 3 ;
        Break ;
    otherwise do
        cadena == ϵ ;
        dejar igual ;
```

Analizadores sintácticos ascendentes

Analizadores LR(0)

Este tipo de analizador se basa en las derivaciones por la derecha y la lectura por la izquierda, lo que se expresa como LR(). Para este tipo de analizador se deben obtener los elementos LR(0), los cuales se representan usando un punto que indica la posición del análisis en las cadenas de las producciones. Si se tiene la siguiente gramática:

$$E \to E + T|T$$

$$T \to T * F|F$$

$$F \to (E)|id$$

Aumentar la gramática

Se aumenta la gramática y se enumera.

DEFINICIÓN 15 (Gramática aumentada.) *Una gramática aumentada es a la que se le agrega una producción al inicio para asegurar la aceptación.*

EJEMPLO 20 *La gramática de expresiones anterior no es una gramática aumentada. La gramática aumentada es la siguiente:*

Por lo que se expanden y se enumeran las producciones del 0 al 6, que se muestran en el cuadro 39 :

Número	Producción	S1	S2	S3
0	$E' \rightarrow$	E		
1	$E \rightarrow$	E	+	T
2	$E \rightarrow$	T		
3	$T \rightarrow$	T	*	F
4	$T \rightarrow$	F		
5	$F \rightarrow$	(	E	)
6	$F \rightarrow$	id		

Cuadro 39: Gramática aumentada que se implementará con un analizador LR(0).

Analizadores LR(0)

Para comenzar a calcular los elementos LR(0) se hacen las siguientes definiciones y se muestra el procedimiento.

DEFINICIÓN 16 (Cerradura.) *La cerradura. Donde I es un conjunto de elementos para una gramática G, entonces cerradura(I) es el conjunto de elementos construido a partir de I por las dos reglas:*

1. *Todo elemento de I se añade a cerradura(I).*
2. *Si $A \rightarrow \alpha \bullet B\beta$ está en cerradura(I) y $B \rightarrow \gamma$ es una producción, entonces añádase el elemento $B \rightarrow \bullet\gamma$ a cerradura(I), si todavía no esta ahí. Se aplica esta regla hasta que no se puedan añadir más elementos a cerradura(I) [3].*

DEFINICIÓN 17 (Función Ir_a.) *La función Ir_a (I, β) se define como: $Ir\_a(I, \beta) = cerraduta(I')$ donde $I' = \{N \rightarrow \alpha\beta \bullet \gamma | (N \rightarrow \alpha \bullet \beta\gamma) \in I\}$ [6].*

Los elementos LR(0) se calculan de la siguiente manera. Se obtiene el estado I_0 aplicando la operación de cerradura a la producción aumentada:

$$I_0 = cerradura(E' \rightarrow \bullet E) = \{ \quad E' \rightarrow \bullet E \quad , \quad E \rightarrow \bullet E + T \quad , \quad E \rightarrow \bullet T \quad ,$$

$$T \rightarrow \bullet T * F \quad , \quad T \rightarrow \bullet F \quad , \quad F \bullet (E) \quad , \quad F \rightarrow \bullet id \quad \}$$

Se obtiene el estado I_1 aplicando la función de movimiento Ir_a de I_0 con E:

$$I_1 = Ir\_a(I_0, E) = cerradura(E' \to E\bullet \quad , \quad E \to E\bullet + T) = \{ \quad E' \to E\bullet \quad ,$$
$$E \to E\bullet + T \quad \}$$

Se obtiene el estado I_2 aplicando la función de movimiento Ir_a de I_0 con T:

$$I_2 = Ir\_a(I_0, T) = cerradura(E \to T\bullet \quad , \quad T \to T\bullet * F) = \{ \quad E \to T\bullet \quad ,$$
$$T \to T\bullet * F \quad \}$$

Se obtiene el estado I_3 aplicando la función de movimiento Ir_a de I_0 con F:

$$I_3 = Ir\_a(I_0, F) = cerradura(T \to F\bullet) = \{ \quad T \to F\bullet \quad \}$$

Se obtiene el estado I_4 aplicando la función de movimiento Ir_a de I_0 con (:

$$I_4 = Ir\_a(I_0, () = cerradura(F \to (\bullet E)) = \{ \ F \to (\bullet E) \ , \ E \to \bullet E + T \ , \ E \to \bullet T \ ,$$
$$T \to \bullet T * F \quad , \quad T \to \bullet F \quad , \quad F \bullet (E) \quad , \quad F \to \bullet id \quad \}$$

Se obtiene el estado I_5 aplicando la función de movimiento de Ir_a de I_0 con id:

$$I_5 = Ir\_a(I_0, id) = cerradura(F \to id\bullet) = \{ \quad F \to id\bullet \quad \}$$

Se obtiene el estado I_6 aplicando la función de movimiento de Ir_a de I_1 con +:

$$I_6 = Ir\_a(I_1, +) = cerradura(F \to E + \bullet T) = \{ \ E \to E + \bullet T \ , \ T \to \bullet T * F \ , \ T \to \bullet F \ ,$$
$$F \to \bullet(E) \ \}$$

Se obtiene el estado I_7 aplicando la función de movimiento de Ir_a de I_2 con *:

$$I_7 = Ir\_a(I_2, *) = cerradura(T \to T * \bullet F) = \{ \ T \to T * \bullet F \ , \ F \to \bullet(E) \ , \ F \to \bullet id \ \}$$

Se obtiene el estado I_8 aplicando la función de movimiento de Ir_a de I_4 con E:

$$I_8 = Ir_a(I_4, E) = cerradura(F \to (E\bullet) \quad , \quad E \to E\bullet + T) = \{ \quad F \to (E\bullet) \quad ,$$
$$E \to E\bullet + T \quad \}$$

Se obtiene el estado I_9 aplicando la función de movimiento de Ir_a de I_6 con T:

$$I_9 = Ir\_a(I_6, T) = cerradura(E \to E+T\bullet \ , \ T \to T\bullet *F) = \{ \ E \to E+T\bullet \ , \ T \to T \bullet *F \ \}$$

Se obtiene el estado I_{10} aplicando la función de movimiento de Ir_a de I_7 con F:

$$I_{10} = Ir\_a(I_7, F) = cerradura(T \to T * F\bullet) = \{ \ T \to T * F \bullet \ \}$$

Se obtiene el estado I_{11} aplicando la función de movimiento de Ir_a de I_8 con):

$$I_{11} = Ir\_a(I_8,)) = cerradura(F \to (E)\bullet) = \{ \ F \to (E) \bullet \ \}$$

Cálculo de los primeros y siguientes

Se obtienen los PRIMEROS y SIGUIENTES de todos los no terminales, los cuales se muestran en el cuadro 40 :

	E'	E	T	F
PRIMEROS	id, (	id, (	id, (	id, (
SIGUIENTES	$	$, +,)	$, +,), *	$, +,), *

Cuadro 40: Tabla de primeros y siguientes de la gramática G para el analizador LR(0).

Construcción del grafo

Para obtener el grafo de los estados. Primero se observan los movimientos: el movimiento de I_0 con E se va a I_1. El movimiento de I_0 con T se va a I_2. El movimiento de I_0 con F se va a I_3. El movimiento de I_0 con (se va a I_4. El movimiento de I_0 con id se va a I_5. El movimiento de I_1 con + se va a I_6. El movimiento de I_2 con * se va a I_7. El movimiento de I_4 con E se va a I_8. El movimiento de I_6 con T se va a I_9. El movimiento de I_7 con F se va a I_{10}. El movimiento de I_8 con) se va a I_{11}.

DEFINICIÓN 18 (Continuidad.) *Continuidad es que todos los elementos LR(0) en todos los estados, deben tener transición dentro del grafo. De no haber esa transición dentro del grafo, se debe crear.*

Segundo se aplica la continuidad en cada uno de los 12 estados, de la siguiente forma: En el estado I_0 los elementos LR(0) tienen transiciones representadas en el grafo. En el estado I_1 los elementos LR(0) tienen transiciones representadas en el grafo. En el estado I_2 los elementos LR(0) tienen transiciones representadas en el grafo. En el estado I_3 los elementos LR(0) tienen transiciones representadas en el grafo. En el estado I_4 los elementos que no tienen transiciones representadas

en el grafo son: $E \to \bullet E + T$ se debe crear la transición con el símbolo + del estado I_8 a I_6. Para el elemento LR(0) $E \to \bullet T$ se crea la transición con T de I_4 a I_2, por lo que con esto se crea la continuidad para el elemento LR(0) $T \to \bullet T * F$. Para el elemento LR(0) $T \to \bullet F$ se crea la transición con F de I_4 a I_3. Para el elemento LR(0) $F \to \bullet(E)$ por lo que se crea la transición con (de I_4 a I_4, esto es un ciclo. Para el elemento LR(0) $F \to \bullet id$ se crea la transición con id de I_4 a I_5. En el estado I_5 los elementos LR(0) tienen transiciones representadas en el grafo. En el estado I_6 los elementos LR(0) que faltan de representación en el grafo para $F \to \bullet T * F$ la transición con * falta, así que se crea la transición con este símbolo entre I_9 e I_7. En el estado I_7 los elementos LR(0) $F \to \bullet(E)$ el símbolo (no tiene transición por lo que se crea de I_7 a I_4, para el elemento LR(0) $F \to \bullet id$ no hay transición para el símbolo id entonces se crea de I_7 a I_5. Para el estado I_8 para el elemento LR(0) $E \to E \bullet + T$ falta la transición con el símbolo + por lo que se crea del estado I_8 al estado I_6. Para el estado I_9 se crea la transición con el símbolo * del estado I_9 al estado I_7. Los estados I_{10} e I_{11} ya no tienen transiciones. Con esto se cumple con el concepto de continuidad. Todo esto se muestra en el grafo de la figura 8 .

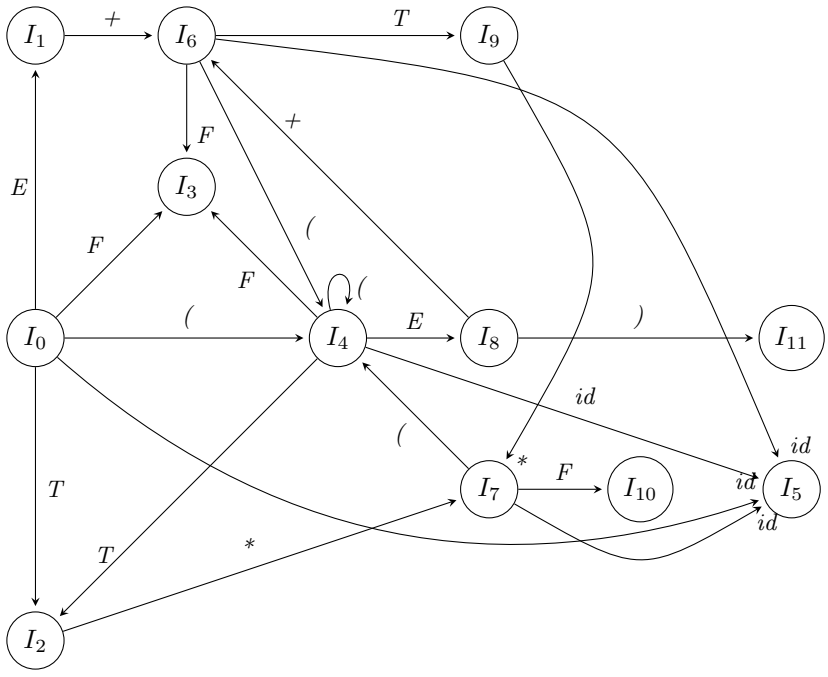

Figura 8: Grafo del analizador sintáctico LR(0).

Obtención de las tablas ACCIÓN y de IR_A

DEFINICIÓN 19 (Reducción.) *La acción de reducción es la transformación de una cadena de entrada en un no terminal de la parte izquierda de la producción.*

DEFINICIÓN 20 (Mando o mango.) *Un mando o mango es cuando una cadena de entrada se transforma en el no terminal de la parte izquierda de la producción y este a su vez debe llegar a transformarse en el símbolo inicial.*

DEFINICIÓN 21 (Desplazamiento) *La acción de desplazamiento indica la acción de pasar de un estado a otro. O de otra forma, un símbolo es leído a la entrada y metido en la pila [4].*

DEFINICIÓN 22 (Aceptación.) *La acción de aceptación es cuando la gramática se cumple.*

DEFINICIÓN 23 (Error.) *Cuando no se aplica ninguna de las acciones de reducción, desplazamiento o aceptación. Entonces la acción es un error.*

Las acciones que se reportan en la tabla de acciones, son la reducción que se representa con una r y como subíndice el número de producción, el desplazamiento que se representa por una d y como subíndice el estado al cuál se desplaza, la aceptación que se representa por **acp** y el error que se representa con **e**. Se obtienen las tablas de Acción y de IR_A que se muestran en el cuadro 41 :

	ACCIÓN						IR_A		
ESTADOS	id	+	*	(	)	$	E	T	F
I_0	d_5	e	e	d_4	e	e	1	2	3
I_1	e	d_6	e	e	e	acp	e	e	e
I_2	e	r_2	d_7	e	r_2	r_2	e	e	e
I_3	e	r_4	r_4	e	r_4	r_4	e	e	e
I_4	d_5	e	e	d_4	e	e	8	2	3
I_5	e	r_6	r_6	e	r_6	r_6	e	e	e
I_6	d_5	e	e	d_4	e	e	e	9	3
I_7	d_5	e	e	d_4	e	e	e	e	10
I_8	e	d_6	e	e	d_{11}	e	e	e	e
I_9	e	r_1	d_7	e	r_1	r_1	e	e	e
I_{10}	e	r_3	r_3	e	r_3	r_3	e	e	e
I_{11}	e	r_5	r_5	e	r_5	r_5	e	e	e

Cuadro 41: Tablas de Acciones y de Ir_a para el analizador LR(0).

Análisis de cadenas de entrada

Se hace el análisis de la cadena de entrada $id * id + id$: la cadena de entrada se coloca en la columna ENTRADA del cuadro 42, la flecha vertical indica que se esta leyendo el símbolo que esta a la izquierda de dicha flecha. En la pila se coloca el $ (eof) y el estado I_0 queda en la cima de la pila, que esta representada en la columna PILA. Para determinar la acción que se escribirá en la columna ACCIÓN, se consulta el cuadro de Acción: se comienza en el estado I_0 y con el símbolo de entrada id concurre la acción d_5, por lo que en la columna PILA se escribe el símbolo id y el subíndice 5 quiere decir que en la cima de la pila se escribe el estado I_5, después se avanza la flecha vertical y queda a su izquierda el símbolo *. Ahora en el estado I_5 y la concurrencia con el símbolo * se tiene la acción r_6, que quiere decir que el símbolo id se transforma en F. Se consulta la tabla IR_A, por lo que un símbolo F en el estado I_0 se va al estado I_3. En el cuadro 42 se tiene todo el recorrido de la cadena de entrada hasta llegar a $, donde se muestra como va cambiando el contenido de la pila.

ESTADOS	PILA	ACCIÓN	ENTRADA
-	\$ I_0	-	$\uparrow id * id + id\$$
I_0	\$ I_0	d_5	$\uparrow id * id + id\$$
I_5	\$ I_0 id I_5	r_6	id$\uparrow * id + id\$$
I_3	\$ I_0 F I_3	r_4	id$\uparrow * id + id\$$
I_2	\$ I_0 T I_2	d_7	id$\uparrow * id + id\$$
I_7	\$ I_0 T I_2 * I_7	d_5	id*$\uparrow id + id\$$
I_5	\$ I_0 T I_2 * I_7 id I_5	r_6	id*id $\uparrow + id\$$
I_{10}	\$ I_0 T I_2 * I_7 F I_{10}	IR_A 10	id*id $\uparrow + id\$$
I_2	\$ I_0 T I_2	r_3	id*id $\uparrow + id\$$
I_9	\$ I_0 E I_1	r_2	id*id $\uparrow + id\$$
I_6	\$ I_0 E I_1 + I_6	d_6	id*id + $\uparrow id\$$
I_5	\$ I_0 E I_1 + I_6 id I_5	d_5	id*id+id $\uparrow$ \$
I_3	\$ I_0 E I_1 + I_6 F I_3	r_6	id*id+id $\uparrow$ \$
I_9	\$ I_0 E I_1 + I_6 T I_9	r_4	id*id+id $\uparrow$ \$
I_1	\$ I_0 E I_1	r_1	id*id+id $\uparrow$ \$
I_1	\$ I_0 E I_1	acp	id*id+id $\uparrow$ \$

Cuadro 42: Tabla del algoritmo de análisis de una cadena de entrada para el analizador LR(0).

Ejercicios de analizadores LR(0)

Construir el analizador LR(0) de las siguientes gramáticas:

EJERCICIO 4 *De la gramática:*

1. $T \to R$
2. $T \to aTc$
3. $R \to \lambda$
4. $R \to bR$

EJERCICIO 5 *La siguiente gramática:*

1. $E \to T + E$
2. $E \to T$
3. $T \to x$

EJERCICIO 6 *La siguiente gramática:*

1. $S \to L = R$
2. $S \to R$
3. $L \to *R$
4. $L \to id$

5. $R \to L$

EJERCICIO 7 *De la siguiente gramática:*

1. $S \to V = E$
2. $S \to E$
3. $V \to *E$
4. $V \to x$
5. $E \to V$

Solución a ejercicio 7. Se comienza aumentando la gramática y dividiéndola en capas, como se muestra en el cuadro 43 :

Número	Producción
0	$G \to S$
1	$S \to V = E$
2	$S \to E$
3	$V \to *E$
4	$V \to x$
5	$E \to V$

Cuadro 43: Gramática aumentada que se implementará con un analizador LR(0), ejercicio 7.

Se comienza a calcular los elementos LR(0) utilizando las operaciones de **cerradura** y de **Ir_a**, que se muestran en el cuadro 44 :

Estado	Ir_a	Conjunto cerradura
I_0	$G \to \bullet S$	
I_1	$S \to V = E$	
I_2	$S \to E$	
I_3	$V \to *E$	
I_4	$V \to x$	
I_5	$E \to V$	

Cuadro 44: Estados y sus elementos LR(0) de la gramática aumentada que se implementará con un analizador LR(0), ejercicio 7.

Del conjunto de estados se obtiene el grafo que se muestra en la figura 9 .

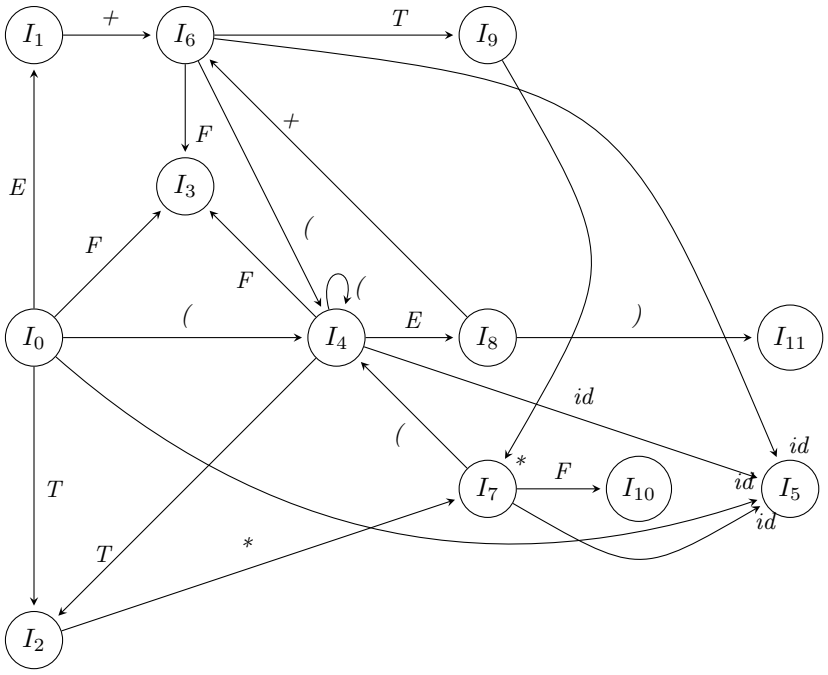

Figura 9: Grafo parcial del analizador sintáctico LR(0), del ejercicio 7.

Aplicando la continuidad se obtiene el grafo completo que se muestra en la figura 10 :

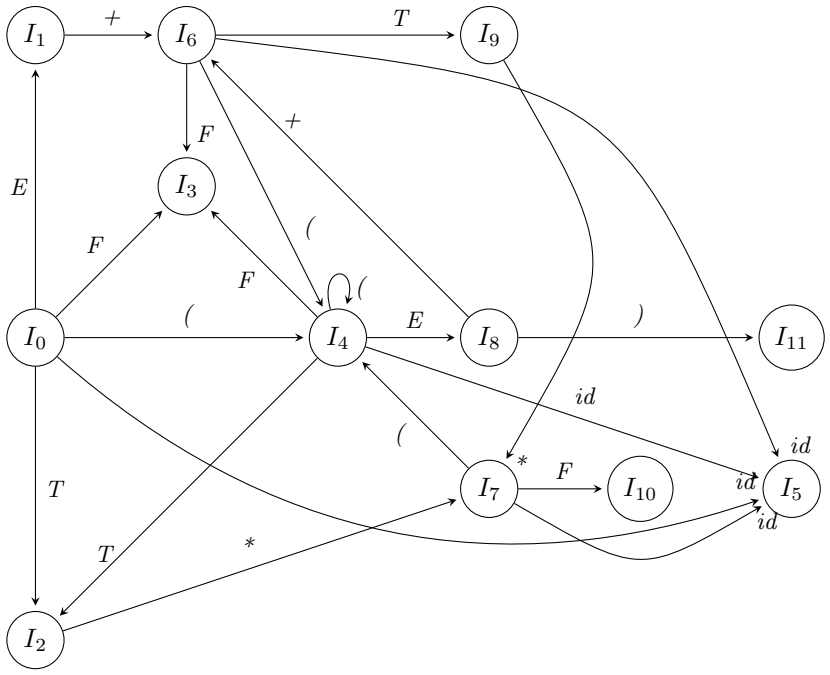

Figura 10: Grafo completo del analizador sintáctico LR(0), del ejercicio 7.

Del grafo completo de la figura 10 se obtienen las tablas de acción y de Ir_a que se muestra en el cuadro 45 :

Estado	Ir_a	Conjunto cerradura
I_0	$G \to \bullet S$	
I_1	$S \to V = E$	
I_2	$S \to V$	
I_3	$V \to *E$	
I_4	$V \to x$	

Cuadro 45: Tablas de acción e Ir_a de la gramática aumentada del ejercicio 7.

Con la tabla de acción y de Ir_a se hace el análisis de cadenas de entrada, como se muestra con la cadena **x = * x**. Lo que se muestra en la tabla **??** :

EJERCICIO 8 *La siguiente gramática:*

1. $N \to SL$
2. $S \to +$
3. $S \to -$
4. $L \to LB$

5. $L \rightarrow B$
6. $B \rightarrow 0$
7. $B \rightarrow 1$

EJERCICIO 9 *La siguiente gramática:*

1. $S \rightarrow LV : i$
2. $LV \rightarrow LV, V \mid V$
3. $V \rightarrow id$

EJERCICIO 10 *Comprobar que la siguiente gramática es LR(0):*

1. $S \rightarrow bBS$
2. $S \rightarrow b$
3. $B \rightarrow aSB$
4. $B \rightarrow a$

Analizadores LR(1)

La construcción del analizador sintáctico LR(1) al igual que el analizador sintáctico LR(0) es el mismo procedimiento, lo que se muestra a continuación. Se tiene la siguiente gramática:

1. $S \rightarrow E = E$
2. $S \rightarrow f$
3. $E \rightarrow T$
4. $E \rightarrow E + T$
5. $T \rightarrow f$
6. $T \rightarrow T * f$

Aumentar la gramática

El siguiente paso es aumentar la gramática y enumerar las producciones de 0 a 6, como se muestra en el cuadro 46 :

Número	Producción	S1	S2	S3
0	$G \rightarrow$	S		
1	$S \rightarrow$	E	=	E
2	$S \rightarrow$	f		
3	$E \rightarrow$	T		
4	$E \rightarrow$	E	+	T
5	$T \rightarrow$	f		
6	$T \rightarrow$	T	*	f

Cuadro 46: Gramática aumentada que se implementará con un analizador LR(1).

Calcular los primeros y siguientes

Se debe obtener la tabla de primeros y siguientes, los que se muestran en el cuadro 47 :

	G	S	E	T
Primeros	f	f	f	f
Siguientes	$	$	$, =, +	$, =, +, *

Cuadro 47: Tabla de primeros y siguientes del analizador LR(1).

Calcular los elementos LR(1)

Los elementos LR(1) tienen la forma $G \to S\bullet, \$$, donde este consta de un núcleo que es la producción, se separa con una coma el núcleo y los símbolos de análisis, entre cada uno de los símbolos de análisis se separan por dos puntos como sigue: $T \to \bullet T * f, =: + : *$. Los elementos LR(1) se calculan de la siguiente manera. Se obtiene el estado I_0 aplicando la operación de cerradura a la producción aumentada:

$$I_0 = Cerr(G \to \bullet S, \$) = \{\ \ G \to \bullet S, \$\ \ ,\ \ S \to \bullet E = E, \$\ \ ,\ \ S \to \bullet f, \$\ \ ,$$

$$E \to \bullet T, =: +\ \ ,\ \ E \to \bullet E + T, =: +\ \ ,\ \ T \to \bullet f, =: + : *\ \ ,$$

$$T \to \bullet T * f, =: + : *\ \ \}$$

Se obtiene el estado I_1 aplicando la función de movimiento Ir_a de I_0 con S:

$$I_1 = Ir\_a(I_0, S) = Cerr(G \to S\bullet, \$) = \{\ \ G \to S\bullet, \$\ \ \}$$

Se obtiene el estado I_2 aplicando la función de movimiento Ir_a de I_0 con E:

$$I_2 = Ir\_a(I_0, E) = Cerr(E \to E\bullet = E, \$\ \ ,\ \ E \to E \bullet + T, =: +)$$

$$= \{\ \ E \to E\bullet = E, \$\ \ ,\ \ E \to E \bullet + T, =: +\ \ \}$$

Se obtiene el estado I_3 aplicando la función de movimiento Ir_a de I_0 con f:

$$I_3 = Ir\_a(I_0, f) = Cerr(S \to \bullet f, \$\ \ ,\ \ T \to f\bullet, =: + : *)$$

$$= \{\ \ S \to f\bullet, \$\ \ ,\ \ T \to f\bullet, =: + : *\ \ \}$$

Se obtiene el estado I_4 aplicando la función de movimiento Ir_a de I_0 con T:

$$I_4 = Ir\_a(I_0, T) = Cerr(E \to T\bullet, =: +\ \ ,\ \ T \to T \bullet * f, =: + : *)$$

$$= \{\ \ E \to T\bullet, =: +\ \ ,\ \ T \to T \bullet * f, =: + : *\ \ \}$$

Se obtiene el estado I_5 aplicando la función de movimiento Ir_a de I_2 con =:

$$I_5 = Cerr(\ \ Ir - a(I_2, =)) = Cerr(S \to E = \bullet E, \$\ \ ) = \{\ \ S \to E = \bullet E, \$\ \ ,$$
$$E \to \bullet T, \$: +\ \ ,\ \ T \to \bullet f, \$: + : *\ \ ,\ \ T \to \bullet T * f, \$: + : *\ \ ,$$
$$E \to \bullet E + T, \$: +\ \ \}$$

Se obtiene el estado I_6 aplicando la función de movimiento Ir_a de I_2 con +:

$$I_6 = Ir\_a(I_2, +) = Cerr(\ \ E \to E + \bullet T, =: \$\ \ ) = \{\ \ E \to E + \bullet T, =: \$\ \ ,$$
$$T \to \bullet f, =: + : *\ \ ,\ \ T \to \bullet T * f, =: + : *\ \ \}$$

Se obtiene el estado I_7 aplicando la función de movimiento Ir_a de I_4 con *:

$$I_7 = Ir\_a(I_4, *) = Cerr(\ \ T \to T * \bullet f, =: + : *\ \ ) =$$
$$\{\ \ T \to T * \bullet f, =: + : *\ \ \}$$

Se obtiene el estado I_8 aplicando la función de movimiento Ir_a de I_5 con E:

$$I_8 = Ir\_a(I_5, E) = Cerr(\ \ S \to E = E\bullet, \$\ \ ,\ \ E \to E \bullet + T, \$: +\ \ ) =$$
$$\{\ \ S \to E = E\bullet, \$\ \ ,\ \ E \to E \bullet + T, \$: +\ \ \}$$

Se obtiene el estado I_9 aplicando la función de movimiento Ir_a de I_6 con T:

$$I_9 = Ir\_a(I_6, T) = Cerr(\ E \to E + T \bullet\ , =\ :\ +\ ;\ T \to T \bullet * f\ ,\ =\ :\ +\ :\ *\ ) =$$
$$\{\ E \to E + T \bullet\ ,\ =\ :\ +\ ;\ T \to T \bullet * f\ ,\ =\ :\ +\ :\ *\ \}$$

Se obtiene el estado I_{10} aplicando la función de movimiento Ir_a de I_7 con f:

$$I_{10} = Ir\_a(I_7, f) = Cerr(\ \ T \to T * f \bullet\ , =\ :\ +\ :\ *\ \ ) =$$
$$\{\ \ T \to T * f \bullet\ ,\ =\ :\ +\ :\ *\ \ \}$$

Se obtiene el estado I_{11} aplicando la función de movimiento Ir_a de I_5 con T:

$$I_{11} = Ir\_a(I_5, T) = Cerr(\ \ E \to T \bullet\ ,\ \$\ :\ +\ ;\ T \to T \bullet * f\ ,\ \$\ :\ +\ :\ *\ \ ) =$$
$$\{\ \ E \to T \bullet\ ,\ \$\ :\ +\ ;\ T \to T \bullet * f\ ,\ \$\ :\ +\ :\ *\ \ \}$$

Se obtiene el estado I_{12} aplicando la función de movimiento Ir_a de I_5 con T:

$$I_{12} = Ir\_a(I_5, f) = Cerr(\ \ T \to f \bullet\ ,\ \$\ :\ +\ :\ *\ ) =$$

$$\{\ \ T \to f \bullet\ ,\ \$\ :\ +\ :\ *\ \}$$

Se obtiene el estado I_{13} aplicando la función de movimiento Ir_a de I_{11} con *:

$$I_{13} = Ir\_a(I_{11}, *) = Cerr(\ \ T \to T * \bullet f\ ,\ \$\ :\ +\ :\ *\ ) =$$

$$\{\ \ T \to T * \bullet f\ ,\ \$\ :\ +\ :\ *\ \}$$

Se obtiene el estado I_{14} aplicando la función de movimiento Ir_a de I_{13} con f:

$$I_{14} = Ir\_a(I_{13}, f) = Cerr(\ \ T \to T * f \bullet\ ,\ \$\ :\ +\ :\ *\ ) =$$

$$\{\ \ T \to T * f \bullet\ ,\ \$\ :\ +\ :\ *\ \}$$

Se obtiene el estado I_{15} aplicando la función de movimiento Ir_a de I_8 con +:

$$I_{15} = Ir\_a(I_8, +) = Cerr(\ \ E \to E + \bullet T\ ,\ \$\ :\ +\ ) =$$

$$\{\ \ E \to E + \bullet T\ ,\ \$\ :\ +\ ;\ T \to \bullet f\ ,\ \$\ :\ +\ :\ *\ ;\ T \to \bullet T * f\ ,\ \$\ :\ +\ :\ *\ \}$$

Se obtiene el estado I_{16} aplicando la función de movimiento Ir_a de I_{15} con T:

$$I_{16} = Ir\_a(I_{15}, T) = Cerr(\ \ E \to E + T \bullet\ ,\ \$\ :\ +\ ;\ T \to T \bullet * f\ ,\ \$\ :\ +\ :\ *\ ) =$$

$$\{\ \ E \to E + T \bullet\ ,\ \$\ :\ +\ ;\ T \to T \bullet * f\ ,\ \$\ :\ +\ :\ *\ \}$$

Construir el grafo del analizador LR(1)

Ahora se comienza a revisar la continuidad desde el estado I_0. Si el elemento LR(1) no se acopla a un estado, se le debe crear un estado nuevo:

El estado I_0 es continuo. Ya que para cada símbolo que esta a la derecha del punto de análisis existe un arco en el grafo. El estado I_1 también es continuo. Para el estado I_2 también es continuo. También el estado I_3 es continuo. La diferencia en este paso de crear el grafo con el analizador LR(0) es que aquí se crean estados y arcos, por lo que siguiendo el proceso, se obtiene el grafo de la figura 11.

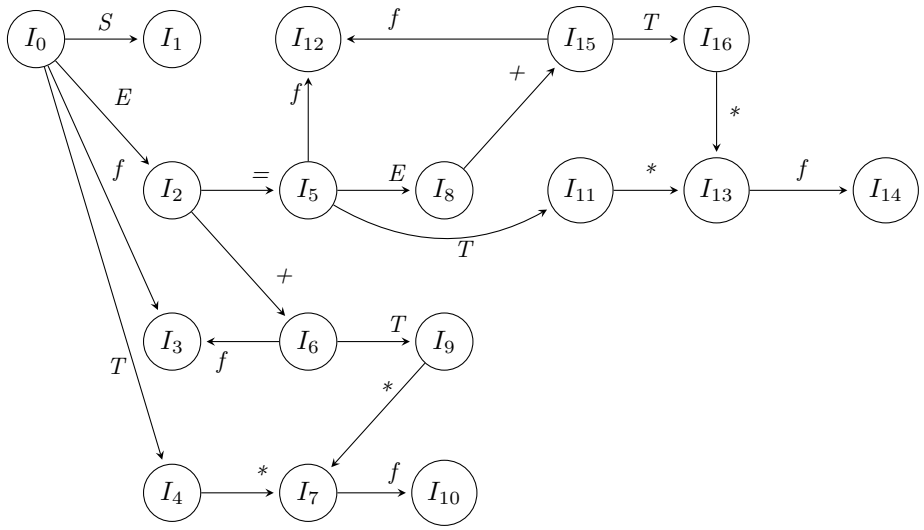

Figura 11: Grafo del analizador sintáctico LR(1).

Obtención de las tablas ACCIÓN y de IR_A del analizador LR(1)

Del grafo de la figura 11, se obtiene la tabla del analizador LR(1) que se muestra en el cuadro 48:

	ACCIÓN					IR_A		
ESTADOS	f	+	*	=	$	S	E	T
I_0	d_3	e	e	e	e	1	2	4
I_1	e	e	e	e	acp	e	e	e
I_2	e	d_6	e	d_5	e	e	e	e
I_3	e	r_5	r_5	r_5	r_2	e	e	e
I_4	e	r_3	d_7	r_3	e	e	e	e
I_5	d_{12}	e	e	e	e	e	8	11
I_6	d_3	e	e	e	e	e	e	9
I_7	d_{10}	e	e	e	e	e	e	e
I_8	e	d_{15}	e	e	r_1	e	e	e
I_9	e	r_4	d_7	r_4	e	e	e	e
I_{10}	e	r_6	r_6	r_6	e	e	e	e
I_{11}	e	r_3	d_{13}	r_3	e	e	e	e
I_{12}	e	r_5	r_5	e	r_5	e	e	e
I_{13}	d_{14}	e	e	e	e	e	e	e
I_{14}	e	r_6	r_6	e	r_6	e	e	e
I_{15}	d_{12}	e	e	e	e	e	e	16
I_{16}	e	r_4	d_{13}	e	r_4	e	e	e

Cuadro 48: Tablas de Acciones y de Ir_a para el analizador LR(1).

Análisis de cadenas de entrada con el analizador LR(1)

Se hace el análisis de la cadena de entrada $f = f*f+f$: la cadena de entrada se coloca en la columna ENTRADA del cuadro 49, la flecha vertical indica que se esta leyendo el símbolo que esta a la derecha de dicha flecha. En la pila se coloca el \$ (eof) y el estado I_0 queda en la cima de la pila, que esta representada en la columna PILA. Para determinar la acción que se escribirá en la columna ACCIÓN, se consulta el cuadro de Acción: se comienza en el estado I_0 y con el símbolo de entrada f concurre la acción d_3, por lo que en la columna PILA se escribe el símbolo f y el subíndice 3 quiere decir que en la cima de la pila se escribe el estado I_3, después se avanza la flecha vertical y queda a su izquierda el símbolo f. Ahora en el estado I_3 y la concurrencia con el símbolo f se tiene la acción r_5, que quiere decir que el símbolo f se transforma en T. Se consulta la tabla IR_A, por lo que un símbolo T en el estado I_0 se va al estado I_4. En el cuadro 49 se tiene todo el recorrido de la cadena de entrada hasta llegar al símbolo \$, donde se muestra como va cambiando el contenido de la pila.

ESTADOS	PILA	ACCIÓN	ENTRADA
-	\$ I_0	-	$\uparrow f = f * f + f\$$
I_0	\$ I_0	d_3	$\uparrow f = f * f + f\$$
I_3	\$ I_0 f I_3	r_5	f $\uparrow = f * f + f\$$
I_4	\$ I_0 T I_4	r_3	f $\uparrow = f * f + f\$$
I_2	\$ I_0 E I_2	d_5	f $\uparrow = f * f + f\$$
I_5	\$ I_0 E $I_2 = I_5$	d_5	f = $\uparrow f * f + f\$$
I_{12}	\$ I_0 E $I_2 = I_5$ f I_{12}	d_{12}	f = f $\uparrow * f + f\$$
I_{11}	\$ I_0 E $I_2 = I_5$ T I_{11}	r_5	f = f $\uparrow * f + f\$$
I_{13}	\$ I_0 E $I_2 = I_5$ T I_{11} * I_{13}	d_{13}	f = f * $\uparrow$ f + $f\$$
I_{14}	\$ I_0 E $I_2 = I_5$ T I_{11} * I_{13} f I_{14}	d_{14}	f = f * f $\uparrow + f\$$
I_{11}	\$ I_0 E $I_2 = I_5$ T I_{11}	r_6	f = f * f $\uparrow + f\$$
I_8	\$ I_0 E $I_2 = I_5$ E I_8	r_3	f = f * f $\uparrow + f\$$
I_{15}	\$ I_0 E $I_2 = I_5$ E I_8 + I_{15}	d_{15}	f = f * f + $\uparrow f\$$
I_{12}	\$ I_0 E $I_2 = I_5$ E I_8 + I_{15} f I_{12}	d_{12}	f = f * f + f $\uparrow$ \$
I_{16}	\$ I_0 E $I_2 = I_5$ E I_8 + I_{15} T I_{16}	r_5	f = f * f + f $\uparrow$ \$
I_8	\$ I_0 E $I_2 = I_5$ E I_8	r_4	f = f * f + f $\uparrow$ \$
I_1	\$ I_0 S I_1	r_1	f = f * f + f $\uparrow$ \$
I_1	\$ I_0 S I_1	acp	f = f * f + f $\uparrow$ \$

Cuadro 49: Tabla del algoritmo de análisis de una cadena de entrada para un analizador LR(1).

Ejercicios LR(1)

EJERCICIO 11 *Construir el analizador sintáctico LR(1), de la siguiente gramática:*

1. $S \rightarrow Aa$
2. $A \rightarrow BC|BCf$
3. $B \rightarrow b$

4. $C \rightarrow c$

EJERCICIO 12 *Construir el analizador sintáctico LR(1), de la siguiente gramática:*

1. $A \rightarrow bB$
2. $B \rightarrow cC$
3. $B \rightarrow cCe$
4. $C \rightarrow dA$
5. $A \rightarrow a$

Analizadores LALR

Este método de analizar es más poderoso pero similar que un analizador SLR(1), porque usa un análisis por adelantado el cual puede ser obtenido del proceso de construcción LR(1). Aunque el LALR(1) no es más poderoso que un LR(1), la ventaja que tiene es que requiere menos espacio para su tabla de análisis que el mismo LR(1) [7].

La construcción del analizador sintáctico LALR al igual que el analizador sintáctico LR(1) tiene el mismo procedimiento. El procedimiento se muestra a continuación con la misma gramática que LR(1) que ya se analizó:

1. $S \rightarrow E = E$
2. $S \rightarrow f$
3. $E \rightarrow T$
4. $E \rightarrow E + T$
5. $T \rightarrow f$
6. $T \rightarrow T * f$

Aumentar la gramática

El siguiente paso es aumentar la gramática y enumerar las producciones de 0 a 6, que se muestran en el cuadro 50 :

Número	Producción	S1	S2	S3
0	$G \rightarrow$	S		
1	$S \rightarrow$	E	=	E
2	$S \rightarrow$	f		
3	$E \rightarrow$	T		
4	$E \rightarrow$	E	+	T
5	$T \rightarrow$	f		
6	$T \rightarrow$	T	*	f

Cuadro 50: Gramática aumentada que se implementará con un analizador LALR.

Tabla de primeros y siguientes

Se debe obtener la tabla de primeros y siguientes, que se muestran en el cuadro 51 :

	G	S	E	T
Primeros	f	f	f	f
Siguientes	$	$	$, =, +	$, =, +, *

Cuadro 51: Tabla de primeros y siguientes del analizador LALR.

Elementos LR(1)

Se obtienen los elementos LR(1), los cuales ya se calcularon en la parte del analizador LR(1) y se presentan a continuación otra vez:

El estado I_0:

$$I_0 = \{\ G \to \bullet S, \$\ ,\ S \to \bullet E = E, \$\ ,\ S \to \bullet f, \$\ ,$$
$$E \to \bullet T, =:+\ ,\ E \to \bullet E + T, =:+\ ,\ T \to \bullet f, =:+:*\ ,$$
$$T \to \bullet T * f, =:+:*\ \}$$

El estado I_1:

$$I_1 = \{\ G \to S\bullet, \$\ \}$$

El estado I_2:

$$I_2 = \{\ E \to E\bullet = E, \$\ ,\ E \to E \bullet + E, =:+\ \}$$

El estado I_3:

$$I_3 = \{\ S \to f\bullet, \$\ ,\ T \to f\bullet, =:+:*\ \}$$

El estad I_4:

$$I_4 = \{\ E \to T\bullet, =:+\ ,\ T \to T \bullet * T, =:+:*\ \}$$

El estado I_5:

$$I_5 = \{\ S \to E = \bullet E, \$\ ,$$
$$E \to \bullet T, \$:+\ ,\ T \to \bullet f, \$:+:*\ ,\ T \to \bullet T * f, \$:+:*\ ,$$
$$E \to \bullet E + T, \$:+\ \}$$

El estado I_6:

$$I_6 = \{\ E \to E + \bullet T, =:\$\ ,$$
$$T \to \bullet f, =:+:*\ ,\ T \to \bullet T * f, =:+:*\ \}$$

El estado I_7:

$$I_7 = \{\ \ T \to T * \bullet f, =: + : *\ \ \}$$

El estado I_8:

$$I_8 = \{\ \ S \to E = E\bullet, \$\ \ ,\ \ E \to E \bullet + T, \$: +\ \ \}$$

El estado I_9:

$$I_9 = \{\ \ E \to E + E \bullet\ \ ,\ \ =\ \ :\ \ +\ \ ;\ T \to T \bullet * f\ \ ,\ \ =\ \ :\ \ +\ \ :\ \ *\ \ \}$$

El estado I_{10}

$$I_{10} = \{\ \ T \to T * f \bullet\ \ ,\ \ =\ \ :\ \ +\ \ :\ \ *\ \ \}$$

El estado I_{11}:

$$I_{11} = \{\ \ E \to T \bullet\ \ ,\ \$\ :\ \ +\ \ ;\ T \to T \bullet * f\ \ ,\ \$\ :\ \ +\ \ :\ \ *\ \ \}$$

El estado I_{12}:

$$I_{12} = \{\ \ T \to f \bullet\ \ ,\ \$\ :\ \ +\ \ :\ \ *\ \}$$

El estado I_{13}:

$$I_{13} = \{\ \ T \to T * \bullet f\ \ ,\ \$\ :\ \ +\ \ :\ \ *\ \}$$

El estado I_{14}:

$$I_{14} = \{\ \ T \to T * f \bullet\ \ ,\ \$\ :\ \ +\ \ :\ \ *\ \}$$

El estado I_{15}:

$$I_{15} = \{\ E \to E + \bullet T\ ,\ \$\ :\ +\ ;\ T \to \bullet f\ ,\ \$\ :\ +\ :\ *\ ;\ T \to \bullet T * f\ ,\ \$\ :\ +\ :\ *\ \}$$

El estado I_{16}:

$$I_{16} = \{\ \ E \to E + T \bullet\ \ ,\ \$\ :\ \ +\ \ ;\ T \to T \bullet * f\ \ ,\ \$\ :\ \ +\ \ :\ \ *\ \ \}$$

Unión de estados

El siguiente paso es unir los núcleos de los siguientes estados.
La unión de los estados I_6 e I_{15}, se convierte en el estado I_6

$$I_6 = \{\ E \to E + \bullet T\ ,\ =\ :\ \$\ :\ +\ ;\ T \to \bullet f\ ,\ =\ :\ \$\ :\ +\ :\ *\ ;\ T \to \bullet T * f\ ,\ =\ :\ \$\ :\ +\ :\ *\ \}$$

La unión de los estados I_4 e I_{11}, se convierte en el estado I_4:

$$I_4 = \{\ \ E \to T \bullet\ \ ,\ \ =\ \ :\ \$\ :\ \ +\ \ ;\ T \to T \bullet * f\ \ ,\ \ =\ \ :\ \$\ :\ \ +\ \ :\ \ *\ \ \}$$

La unión de los estados I_{10} e I_{14}, se convierte en el estado I_{10}:

$$I_{10} = \{\ T \to T * f \bullet\ ,\ =\ :\ \$\ :\ +\ :\ *\ \}$$

La unión de los estados I_9 e I_{16}, se convierte en el estado I_9:

$$I_9 = \{\ E \to E + T \bullet\ ,\ =\ :\ \$\ :\ +\ ;\ T \to T \bullet * f\ ,\ =\ :\ \$\ :\ +\ :\ *\ \}$$

La unión de los estado I_7 e I_{13}, se convierte en el estado I_7:

$$I_7 = \{\ T \to T * \bullet f\ ,\ =\ :\ \$\ :\ +\ :\ *\ \}$$

Formación de estados

Así que ahora los estados quedan como a continuación se presentan:

Se obtiene el estado I_0 aplicando la operación de cerradura a la producción aumentada:

$$I_0 = Cerr(G \to \bullet S, \$) = \{\ G \to \bullet S, \$\ \ ,\ \ S \to \bullet E = E, \$\ \ ,\ \ S \to \bullet f, \$\ \ ,$$
$$E \to \bullet T, =: +\ \ ,\ \ E \to \bullet E + T, =: +\ \ ,\ \ T \to \bullet f, =: + : *\ \ ,$$
$$T \to \bullet T * f, =: + : *\ \ \}$$

Se obtiene el estado I_1 aplicando la función de movimiento Ir_a de I_0 con S:

$$I_1 = Ir\_a(I_0, S) = Cerr(G \to S\bullet, \$) = \{\ G \to S\bullet, \$\ \}$$

Se obtiene el estado I_2 aplicando la función de movimiento Ir_a de I_0 con E:

$$I_2 = Ir\_a(I_0, E) = Cerr(E \to E\bullet = E, \$\ \ ,\ \ E \to E \bullet + E, =: +)$$
$$= \{\ E \to E\bullet = E, \$\ \ ,\ \ E \to E \bullet + T, =: +\ \ \}$$

Se obtiene el estado I_3 aplicando la función de movimiento Ir_a de I_0 con f:

$$I_3 = Ir\_a(I_0, f) = Cerr(S \to \bullet f, \$\ \ ,\ \ T \to f\bullet, =: + : *)$$
$$= \{\ S \to f\bullet, \$\ \ ,\ \ T \to f\bullet, =: + : *\ \ \}$$

Se obtiene el estado I_4 aplicando la función de movimiento Ir_a de I_0 con T:

$$I_4 = Ir\_a(I_0, T) = \{\ E \to T\bullet, =: +\ \ ,\ \ T \to T \bullet * T, =: + : *\ \ \}$$

Se obtiene el estado I_5 aplicando la función de movimiento Ir_a de I_2 con =:

$$I_5 = \ Ir\_a(I_2, =) = \{\ S \to E = \bullet E, \$\ \ ,$$
$$E \to \bullet T, \$: +\ \ ,\ \ T \to \bullet f, \$: + : *\ \ ,\ \ T \to \bullet T * f, \$: + : *\ \ ,$$
$$E \to \bullet E + T, \$: +\ \ \}$$

Se obtiene el estado I_6 aplicando la función de movimiento Ir_a de I_2 con +:

$$I_6 = Ir\_a(I_2, +) = \{\ E \to E + \bullet T\ ,\ =\ :\ \$\ :\ +\ ;$$

$$T \to \bullet f\ ,\ =\ :\ \$\ :\ +\ :\ *\ ;\ T \to \bullet T * f\ ,\ =\ :\ \$\ :\ +\ :\ *\ \}$$

Se obtiene el estado I_7 aplicando la función de movimiento Ir_a de I_4 con *:

$$I_7 = Ir\_a(I_4, *) = \{\ T \to T * \bullet f\ ,\ =\ :\ \$\ :\ +\ :\ *\ \}$$

Se obtiene el estado I_8 aplicando la función de movimiento Ir_a de I_5 con E:

$$I_8 = Ir\_a(I_5, E) = Cerr(\ \ S \to E = E\bullet, \$\ \ ,\ \ E \to E \bullet + T, \$: +\ \ ) =$$

$$\{\ \ S \to E = E\bullet, \$\ \ ;\ \ E \to E \bullet + T, \$: +\ \ \}$$

Se obtiene el estado I_9 aplicando la función de movimiento Ir_a de I_6 con T:

$$I_9 = Ir\_a(I_6, T) = \{\ E \to E + T\bullet\ ,\ =\ :\ \$\ :\ +\ ;\ T \to T\bullet * f\ ,\ =\ :\ \$\ :\ +\ :\ *\ \}$$

Se obtiene el estado I_{10} aplicando la función de movimiento Ir_a de I_7 con f:

$$I_{10} = Ir\_a(I_7, f) = \{\ \ T \to T * f \bullet\ \ ,\ \ =\ \ :\ \$\ :\ +\ \ :\ \ *\ \ \}$$

Se obtiene el estado I_{11} aplicando la función de movimiento Ir_a de I_5 con f:

$$I_{11} = Ir\_a(I_5, f) = Cerr(\ \ T \to f \bullet\ ,\ \$\ :\ +\ \ :\ \ *\ \ ) =$$

$$\{\ \ T \to f \bullet\ ,\ \$\ :\ +\ \ :\ \ *\ \ \}$$

Construcción del grafo

Posteriormente se obtiene el grafo que se presenta en la figura 12, aplicándose continuidad:

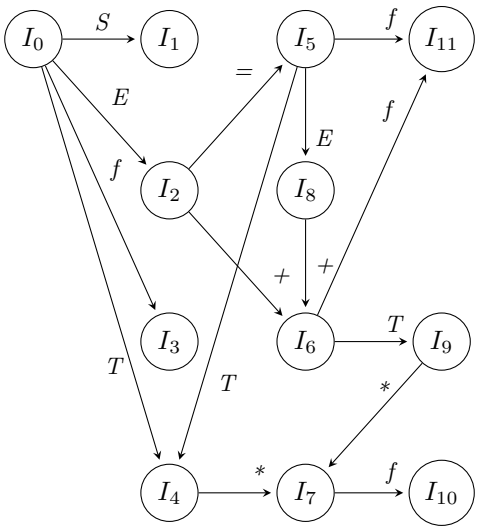

Figura 12: Grafo del analizador sintáctico LALR.

Construcción de las tablas ACCIÓN e IR_A

Del grafo y de los elementos de los estados de I_0 a I_{11} se obtienen las tablas de ACCIÓN y de IR_A, que se presentan en el cuadro 52 .

	ACCIÓN					IR_A		
ESTADOS	f	+	*	=	$	S	E	T
I_0	d_3	e	e	e	e	1	2	4
I_1	e	e	e	e	acp	e	e	e
I_2	e	d_6	e	d_5	e	e	e	e
I_3	e	r_5	r_5	r_5	r_2	e	e	e
I_4	e	r_3/r_6	d_7/r_6	r_3/r_6	e	e	e	e
I_5	d_{11}	e	e	e	e	e	8	4
I_6	d_{11}	e	e	e	e	e	e	9
I_7	d_{10}	e	e	e	e	e	e	e
I_8	e	d_6	e	e	r_1	e	e	e
I_9	e	r_4	d_7	r_4	r_4	e	e	e
I_{10}	e	r_6	r_6	r_6	r_6	e	e	e
I_{11}	e	r_5	r_5	e	r_5	e	e	e

Cuadro 52: Tablas de Acciones y de Ir_a para el analizador LALR.

Análisis de cadenas

La tabla de análisis de la cadena de entrada $f = f + f * f$ es la que se presenta en el cuadro 53.

ESTADOS	PILA	ACCIÓN	ENTRADA
-	\$ I_0	-	$\uparrow f = f * f + f\$$
I_0	\$ I_0	d_3	$\uparrow f = f * f + f\$$
I_3	\$ I_0 f I_3	r_5	f $\uparrow = f * f + f\$$
I_4	\$ I_0 T I_4	r_3	f $\uparrow = f * f + f\$$
I_2	\$ I_0 E I_2	d_5	f $\uparrow = f * f + f\$$
I_5	\$ I_0 E $I_2 = I_5$	d_{11}	f $= \uparrow f * f + f\$$
I_{11}	\$ I_0 E $I_2 = I_5$ f I_{11}	r_5	f = f $\uparrow * f + f\$$
I_4	\$ I_0 E $I_2 = I_5$ T I_4	d_7	f = f $\uparrow * f + f\$$
I_7	\$ I_0 E $I_2 = I_5$ T I_4 * I_7	d_{10}	f = f * $\uparrow f + f\$$
I_{10}	\$ I_0 E $I_2 = I_5$ T I_9 * I_7 f I_{10}	r_6	f = f * f $\uparrow + f\$$
I_4	\$ I_0 E $I_2 = I_5$ T I_4	r_3	f = f * f $\uparrow + f\$$
I_8	\$ I_0 E $I_2 = I_5$ E I_8	d_6	f = f * f $\uparrow + f\$$
I_6	\$ I_0 E $I_2 = I_5$ E $I_8 + I_6$	d_3	f = f * f + $\uparrow f\$$
I_{11}	\$ I_0 E $I_2 + I_6$ T I_9	r_5	f = f * f + f $\uparrow$ \$
I_9	\$ I_0 E $I_2 = I_5$ E $I_8 + I_6$ T I_9	r_4	f = f * f + f $\uparrow$ \$
I_8	\$ I_0 E $I_2 = I_5$ E I_8	r_1	f = f * f + f $\uparrow$ \$
I_1	\$ I_0 S I_1	acp	f = f * f + f $\uparrow$ \$

Cuadro 53: Tabla del algoritmo de análisis de la cadena de entrada $f = f + f * f$ para un analizador LALR.

Conflictos

Existen conflictos de desplazamiento/reducción y conflicto de reducción/-reducción. Cuando surge un conflicto desplazamiento/reducción, el conflicto se resuelve a favor del desplazamiento. En el caso del conflicto reducción/reducción el conflicto se resuelve a favor de la reducción que usa la producción que aparece primero en la gramática.

Ejercicios

Ejercicios LALR

EJERCICIO 13 *Construir el analizador sintáctico LALR de la siguiente gramática:*

1. $S \rightarrow wAz|xBz|wBy|xAy$
2. $A \rightarrow \gamma$
3. $B \rightarrow \gamma$

EJERCICIO 14 *Construir el analizador sintáctico LALR de la siguiente gramática:*

1. $E \rightarrow wEdEz$
2. $E \rightarrow i := E$

3. $E \to E + E$

4. $E \to i$

Proyectos

PROYECTO 1 *Suponer que un elevador es controlado por dos comandos: una comando para subir un piso y un comando para bajar un piso. Considerar que el edificio es alto y que comienza en el piso x. Escribir una gramática LL(1) que ejecute una serie de comandos:*

1. *No cause que el elevador baje al piso x.*

2. *Siempre regresa el elevador al piso x al final de la secuencia.*

Por ejemplo, $\uparrow\uparrow\downarrow\downarrow$ y $\uparrow\downarrow\uparrow\downarrow$ son comandos válidos, pero $\uparrow\downarrow\downarrow\uparrow$ y $\uparrow\downarrow\downarrow$ no. Por conveniencia, una secuencia nula es válida. Probar que la gramática es LL(1).

PROYECTO 2 *Considerar un brazo de robot que acepte dos comandos: $\bigtriangledown$ poner una manzana en una bolsa y $\bigtriangleup$ sacar una manzana de una bolsa. Convenir que el brazo del robot comienza con una bolsa vacía. Una secuencia de comandos válidos para el brazo de robot no debe contener como prefijo que contenga más comandos $\bigtriangleup$ que comandos $\bigtriangledown$. Como ejemplo son secuencias de comandos válidos $\bigtriangledown\bigtriangledown\bigtriangleup\bigtriangleup$ y $\bigtriangledown\bigtriangleup\bigtriangledown$, pero los comandos $\bigtriangledown\bigtriangleup\bigtriangleup\bigtriangledown$ y $\bigtriangledown\bigtriangleup\bigtriangledown\bigtriangleup\bigtriangleup$ no son comandos válidos.*

1. *Escribir una gramática LR(1) que represente todos los comandos.*

2. *Probar que la gramática sea LR(1).*

Generadores de analizadores sintácticos (YACC, JACC, JACCIE, GPPG)

Yacc [9]. Es el que proporciona una herramienta general para describir la entrada para un programa de ordenador. El nombre yacc quiere decir "yet another compiler-compiler". El usuario del yacc especifica las estructuras de su input, junto con el código que se va a llamar a medida que se reconocen cada estructura. Yacc convierte tal especificación en una subrutina que maneja el proceso de input; frecuentemente, es conveniente y apropiado hacer que esta subrutina maneje el flujo de control de la aplicación del usuario.

La subrutina producida por yacc llama a una rutina proporcionada por el usuario para devolver el siguiente item de entrada básico. Por lo tanto, el usuario puede especificar su entrada en términos de caracteres de entrada individuales o en términos de construcciones de nivel más alto tales como nombres y números. La rutina proporcionada por el usuario puede también manejar características idiomáticas tales como convenciones de comentarios y de continuación, las cuales normalmente desafían las especificaciones gramaticales sencillas. La clase de

especificaciones aceptadas es una clase muy general: gramáticas LALR con reglas que evitan la ambigüedad.

Una especificación en Yacc esta dividida en tres secciones separadas por el símbolo %%:

```
declaraciones del analizador
%%
reglas gramaticales
%%
programas
```

Las declaraciones del analizador incluye una lista de símbolos terminales y no terminales.

Las reglas gramaticales son producciones de la forma;

$$exp: \quad exp \quad PLUS \quad exp \quad \{ \, acción \quad semántica \, \}$$

Donde *exp* es un no terminal que tiene como lado derecho $exp \ + \ exp$, y PLUS es un símbolo terminal (token).

Los programas son código C usado como acciones semánticas embebidas.

Ejemplo

Se tiene la siguiente gramática:

1. $P \rightarrow L$
2. $S \rightarrow i \ := \ i$
3. $S \rightarrow while \ i \ do \ S$
4. $S \rightarrow begin \ L \ end$
5. $S \rightarrow if \ i \ then \ S$
6. $S \rightarrow if \ i \ then \ S \ else \ S$
7. $L \rightarrow S$
8. $L \rightarrow L \ ; \ S$

Programa en Yacc:

```
%{
int yylex(void);
void yyerror(char *s) { EM_error(EM_tokPos, "%s", s); }
%}

%token ID WHILE BEGIN END DO IF THEN ELSE SEMI ASSIGN
%start prog
```

```
%%
prog: stmlist
stm : ID ASSIGN ID

| WHILE ID DO stm
| BEGIN stmlist END
| IF ID THEN stm
| IF ID THEN stm ELSE stm

  stmlist : stm
    | stmlist SEMI stm
```

Jacc [8]. Es un generador de analizadores para Java [3] usa un modelado cercano al yacc clásico de Johnson [7]. Es fácil de encontrar un generador de analizador para Java incluyendo el CUP [4], Antlr [11], JavaCC [9], SableCC [13], Coco/R [10], BYACC/Java [5], y el Jikes Parser Generator [12]. Así que, ¿si queremos usar jacc en lugar de una de estas herramientas? En resumen, la diferencia de jacc con otras herramientas, es la combinación de las siguientes características:

1. Compatibilidad sintáctica con el generador de anlizador sintáctico clásico de Johnson para C (de modo que es posible dar que dos herramientas manejen diferentes lenguajes);

2. Compatibilidad semántica con el generador yacc—jacc y el analizador ascendente/desplazamiento-reducción para gramáticas LALR(1) con reglas de desambiguedad;

3. Una implementación pura con Java que es portable y reglas en Java para desarrollar plataformas;

4. Agregados modestos para auxiliar a entender y depurar analizadores generados, incluyendo: una característica para trazar el comportamiento de una cadena de entrada, salida en HTML, y pruebas para conflictos LR(0) y SLR(1);

5. Soporte primario para distribuir gramáticas descritas a lo largo de múltiples archivos para soportar una construcción modular o la extención del analizador;

6. Un mecanismo para generar mensajes de error sintáctico de ejemplos basados en ideas descritas por Jeffery [6];

7. Analizadores generados que usan la técnica descrita por Bhamidipaty y Proebsting [1] para crear fácilmente el analizador yacc-compatible para generar código en lugar de codificar un específico y particular análisis de un conjunto de tablas como en la implementación del yacc clásico.

Bibliografía

[1] Thomas W. Parsons, Introduction to compiler construction, Computer Science Press, 1992.

[2] Ralph Grishman, Computational Linguistics An Introduction, Cambridge University Press, 1986.

[3] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Compiladores Principios, técnicas y herramientas, Addison Wesley, 1990.

[4] Torben Aegidius Mogensen, Basics of compiler design, DEPARTMENT OF COMPUTER SCIENCE UNIVERSITY OF COPENHAGEN, 2009.

[5] Linda Torczon and Keith D. Cooper, Engineering a Compiler, Second Edition, Elsevier 2012.

[6] Bernard Teufel, Stephanie Schmidt, Thomas Teufel, Compiladores, conceptos fundamentales, Addison-Wesley Iberoamericana, 1995.

[7] Jean Paul Tremblay, Paul G. Sorenson, The theory and practice of compiler writing, McGraw Hill Book Company, 1985.

[8] Mark P. Jones, jacc: just another compiler compiler for Java A Reference Manual and User Guide, Department of Computer Science Engineering OGI School of Science Engineering at OHSU 20000 NW Walker Road, Beaverton, OR 97006, USA February 16, 2004.

[9] www.lcc.uma.es/ galvez/ftp/tci/TutorialYacc.pdf