Reporte de Análisis sintáctico. Ejercicios

ANÁLISIS SINTÁCTICO

ALUMNO: Ulises Herrera Rodríguez

Introducción

La presente práctica se centra en el estudio de los fundamentos que sustentan el proceso de compilación, abarcando las distintas fases involucradas en la transformación de un programa escrito en un lenguaje de alto nivel a un formato que pueda ser ejecutado por una computadora. Se analiza el funcionamiento de los compiladores desde una perspectiva teórica y práctica, explorando temas clave como el análisis léxico, sintáctico y semántico.

En el **análisis léxico** se examinan las técnicas utilizadas para identificar y clasificar los componentes básicos de un programa, tales como palabras reservadas, identificadores, constantes y operadores, a través del uso de expresiones regulares y otras herramientas de reconocimiento de patrones. Este paso es crucial para preparar la entrada del programa para etapas posteriores.

El **análisis sintáctico** se encarga de verificar que la secuencia de tokens cumpla con la estructura gramatical del lenguaje. Mediante la construcción de árboles sintácticos y la aplicación de métodos de análisis descendente o ascendente, se determina si el programa presenta una estructura coherente y se identifican posibles errores de sintaxis.

Finalmente, el **análisis semántico** revisa que, además de tener una estructura correcta, el programa posea un sentido lógico en cuanto a la utilización de los tipos de datos y las operaciones definidas en el lenguaje. Este proceso incluye la administración de la tabla de símbolos y la detección de errores que, de no corregirse, podrían afectar la ejecución final del código.

Este documento, basado en el estudio de materiales especializados, tiene como objetivo consolidar los conceptos teóricos y prácticos en torno a los lenguajes formales, autómatas y compiladores, proporcionando una base sólida para comprender el complejo proceso de traducción y ejecución de programas. La integración de ejemplos y ejercicios prácticos facilita la aplicación de estos conocimientos en contextos reales, contribuyendo al desarrollo de habilidades fundamentales para el diseño y optimización de sistemas de compilación.

Marco teórico

Lenguajes Formales

Definición

Los lenguajes formales son conjuntos de cadenas compuestas por símbolos de un alfabeto finito, definidos mediante reglas precisas de formación. Estas reglas establecen la estructura sintáctica de las cadenas y permiten modelar la sintaxis de los lenguajes de programación.

Según la complejidad de las reglas, se pueden clasificar en diferentes niveles, de acuerdo con la jerarquía de Chomsky:

- **Lenguajes Regulares:** Pueden ser definidos mediante expresiones regulares y son reconocidos por autómatas finitos.
- **Lenguajes Libres de Contexto:** Son generados por gramáticas libres de contexto y reconocidos por autómatas de pila, permitiendo representar estructuras jerárquicas más complejas.
- **Lenguajes Sensibles al Contexto y Recursivamente Enumerables:** Permiten definir lenguajes con restricciones sintácticas sofisticadas y están relacionados con modelos computacionales de mayor potencia.

Autómatas

Definición

Un autómata es un modelo matemático utilizado para representar y analizar el comportamiento de sistemas de procesamiento de información.

Dentro de este marco, los **autómatas finitos** son especialmente relevantes:

rojo!15 **Autómatas Finitos Deterministas (AFD)**	**Autómatas Finitos No Deterministas (AFND)**
Cada estado y símbolo de entrada determinan un único estado de transición.	Permiten múltiples transiciones para un mismo estado y símbolo.
Se utilizan principalmente para reconocer lenguajes regulares.	Aunque conceptualmente más potentes, en la práctica se transforman a AFD para su implementación.
Constituyen la base de los analizadores léxicos en compiladores.	Permiten expresar ciertos patrones de forma más natural y compacta.

Gramáticas Formales

Definición

Una gramática formal se define como un conjunto de reglas de producción que determinan cómo se pueden generar las cadenas de un lenguaje.

Una gramática se compone de:

Símbolos Terminales:
Elementos básicos del lenguaje.

Símbolos No Terminales:
Utilizados para definir estructuras intermedias.

Producciones:
Reglas que indican cómo se pueden transformar los símbolos no terminales en secuencias de terminales y no terminales.

Símbolo Inicial:
El punto de partida para la generación de cadenas.

Compiladores: Fases de Análisis

El proceso de compilación se divide en varias fases, cada una de las cuales transforma la representación del código fuente hacia una forma ejecutable:

Análisis Léxico:
Lee el código fuente de forma lineal y agrupa los caracteres en lexemas, generando tokens mediante el uso de expresiones regulares. Además, elimina espacios en blanco y comentarios.

Análisis Sintáctico:
Verifica que la secuencia de tokens cumpla con la gramática del lenguaje. Se construyen árboles sintácticos que representan la estructura jerárquica del programa. Se emplean métodos descendentes o ascendentes para la generación de dichos árboles.

Análisis Semántico:
Valida que, además de tener una estructura correcta, el programa tenga sentido en términos de tipos de datos y lógica. Este proceso incluye la verificación de tipos, el manejo de la tabla de símbolos y la detección de errores semánticos.

Relevancia y Aplicaciones

El estudio combinado de lenguajes formales, autómatas y compiladores es fundamental para el diseño de sistemas que traducen y ejecutan programas de manera eficiente. Entre las aplicaciones prácticas se encuentran:

- La implementación de analizadores léxicos y sintácticos en compiladores modernos.
- El desarrollo de herramientas de procesamiento de texto y reconocimiento de patrones.
- La optimización y verificación de código en entornos de desarrollo profesional.

Herramientas empleadas

Para la elaboración de este documento se emplearon las siguientes herramientas:

LaTeX

El sistema de composición tipográfica LaTeX fue utilizado para la redacción y estructuración del documento debido a sus ventajas en la organización de contenido, manejo de referencias bibliográficas y generación de ecuaciones matemáticas. Su uso permitió obtener un formato profesional y estandarizado en la presentación de la información.

Documento PDF de Referencia

El contenido del presente documento se basa en el análisis y estudio del archivo PDF proporcionado, el cual contiene información relevante para el desarrollo del tema. Dicho material sirvió como fuente principal para la redacción de los apartados y la fundamentación teórica.

Ejercicios

Ejercicio 1: Gramáticas y árboles sintácticos básicos

a) Gramática para el conjunto {s, s;s, s;s;s, ...}

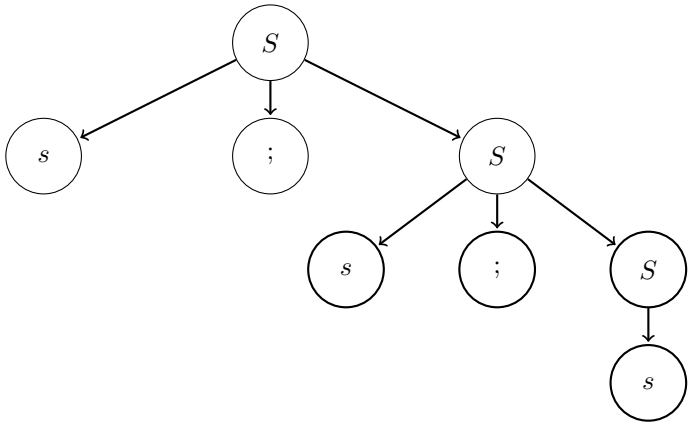
La siguiente gramática genera el conjunto de cadenas requerido:

$$S \rightarrow s \mid s; S \tag{1}$$

Explicación:

- $S \rightarrow s$ genera la cadena base "s"
- $S \rightarrow s; S$ permite generar cadenas con múltiples "s" separadas por punto y coma

b) Árbol sintáctico para la cadena s;s;s

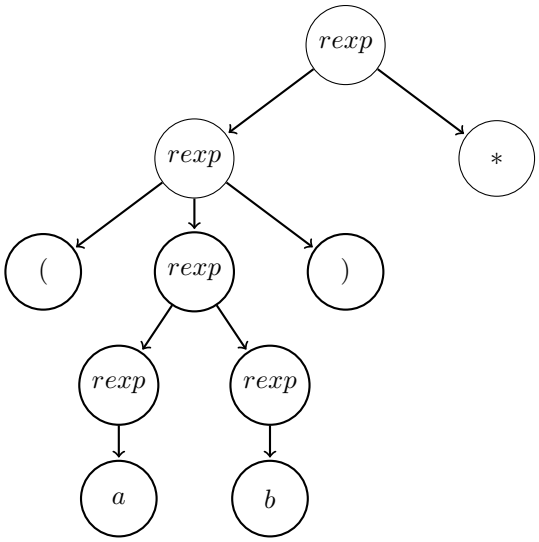


Ejercicio 2: Expresiones regulares

*a) Árbol sintáctico para la expresión regular (ab)**

Considerando la gramática:

$$\begin{aligned} rexp &\rightarrow rexp \text{“|”} rexp && (2)\\ &\mid rexp\ rexp && (3)\\ &\mid rexp\ \text{“*”} && (4)\\ &\mid \text{“(”}\ rexp\ \text{“)”} && (5)\\ &\mid letra && (6) \end{aligned}$$



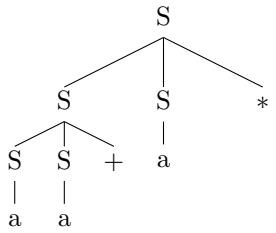
Ejercicio 3: Gramáticas y notaciones

Descripción de gramáticas

- **Gramática A:** $S \to SS+ \mid SS* \mid a$ genera expresiones en notación postfija
- **Gramática B:** $S \to 0S1 \mid 01$ genera cadenas con igual número de 0s y 1s
- **Gramática C:** $S \to +SS \mid *SS \mid a$ genera expresiones en notación prefija

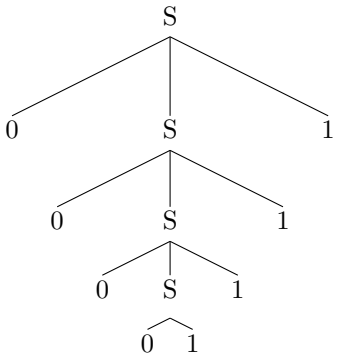
a) Gramática para notación postfija

$S \to SS+ \mid SS* \mid a$ con la cadena $aa + a*$
Lenguaje generado: Esta gramática genera expresiones en notación postfija (donde los operadores siguen a sus operandos). El lenguaje consiste en expresiones aritméticas con el terminal 'a' como operando y los símbolos '+' y '*' como operadores.



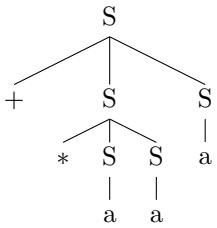
b) Gramática para cadenas 01

$S \to 0S1 \mid 01$ con la cadena 000111
Lenguaje generado: Esta gramática genera el lenguaje $L = \{0^n 1^n \mid n \geq 1\}$, es decir, cadenas con igual número de 0s seguidos por igual número de 1s, con al menos un par.



c) Gramática para notación prefija

$S \to +SS \mid *SS \mid a$ con la cadena $+ * aaa$
Lenguaje generado: Esta gramática genera expresiones en notación prefija (donde los operadores preceden a sus operandos). El lenguaje consiste en expresiones aritméticas con el terminal 'a' como operando y los símbolos '+' y '∗' como operadores.



Ejercicio 4: Análisis del lenguaje xy

Gramática: $S \to xSy \mid \varepsilon$
Lenguaje generado: $L = \{x^n y^n \mid n \geq 0\}$, es decir, cadenas formadas por n repeticiones de 'x' seguidas por n repeticiones de 'y', incluyendo la cadena vacía (cuando $n = 0$).

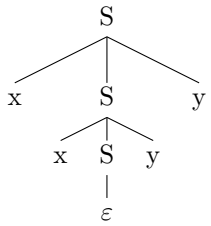
Funcionamiento:

- La regla $S \to \varepsilon$ permite generar la cadena vacía
- La regla $S \to xSy$ permite añadir una 'x' al principio y una 'y' al final de cualquier cadena ya generada por la gramática

Ejemplos:

1. ε (cadena vacía, $n = 0$)
2. xy (aplicando la regla una vez, $n = 1$)
3. $xxyy$ (aplicando la regla dos veces, $n = 2$)
4. $xxxyyyy$ (aplicando la regla tres veces, $n = 3$)

Árbol sintáctico para $xxyy$:

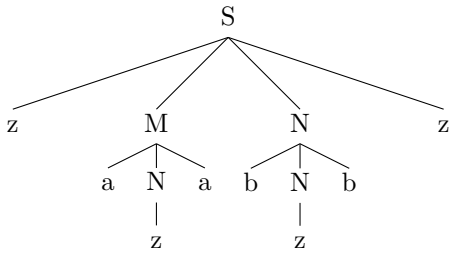


Ejercicio 5: Análisis de la cadena "zazabbz"

Gramática

$$S \to zMNz$$
$$M \to aNa$$
$$N \to bNb$$
$$N \to z$$

Árbol sintáctico para la cadena "zazabbz":



Ejercicio 6: Demostración de gramática ambigua

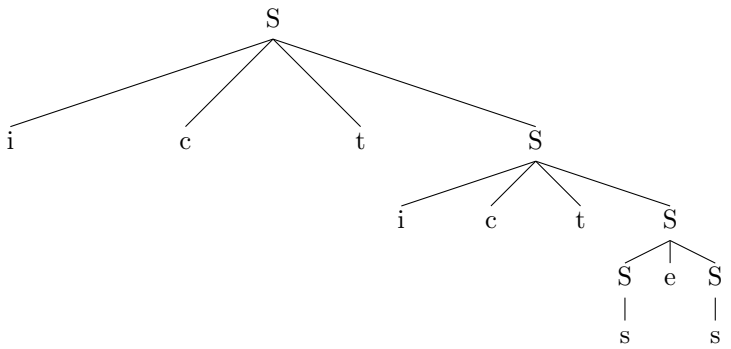
Gramática

$$S \to ictS$$
$$S \to ictSeS$$
$$S \to s$$

Para demostrar que la gramática es ambigua, mostraremos que la cadena "ictictses" tiene dos derivaciones diferentes que producen distintos árboles sintácticos.

Derivación 1

$$
\begin{aligned}
S &\Rightarrow ictS \\
&\Rightarrow ict(ictS) \\
&\Rightarrow ict(ict(SeS)) \\
&\Rightarrow ict(ict(s)eS) \\
&\Rightarrow ict(ictse(s)) \\
&\Rightarrow ictictses
\end{aligned}
$$



Derivación 2

$$
\begin{aligned}
S &\Rightarrow ictSeS \\
&\Rightarrow ict(S)e(S) \\
&\Rightarrow ict(ictS)e(S) \\
&\Rightarrow ict(ict(s))e(S) \\
&\Rightarrow ictictse(s) \\
&\Rightarrow ictictses
\end{aligned}
$$

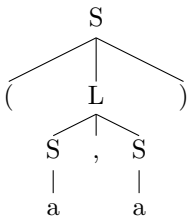
Conclusión: Ambos árboles son diferentes aunque producen la misma cadena “ictictses”, lo que demuestra que la gramática es ambigua.

Ejercicio 7: Árboles de análisis para listas

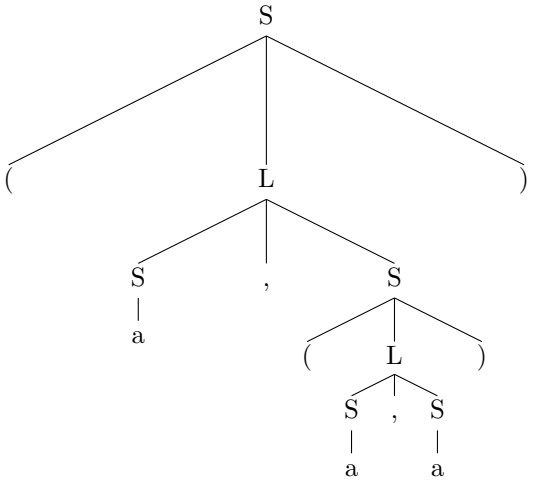
Gramática

$$
\begin{aligned}
S &\to (L) \mid a \\
L &\to L, S \mid S
\end{aligned}
$$

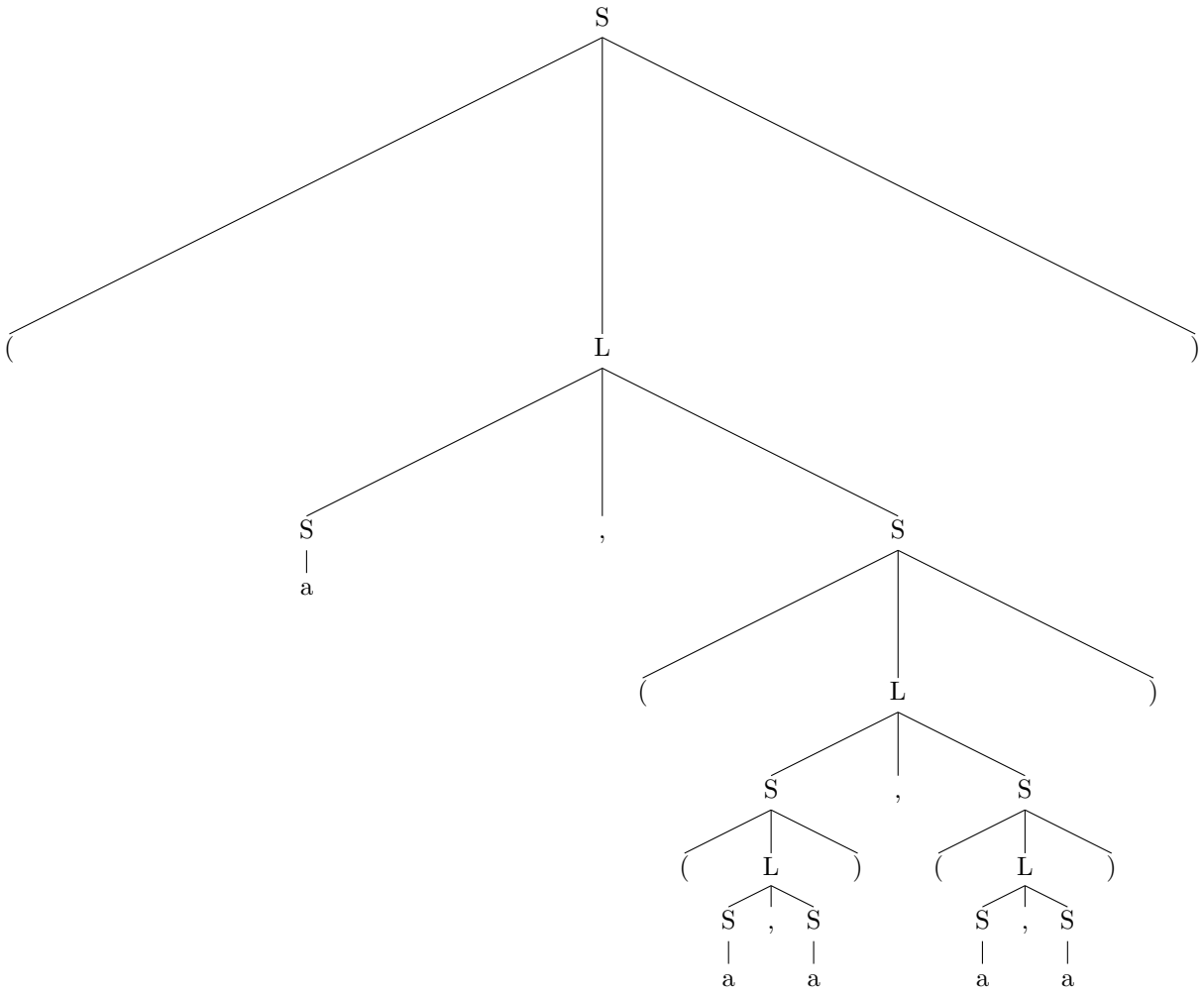
a) Árbol para (a, a)



b) Árbol para (a, (a, a))



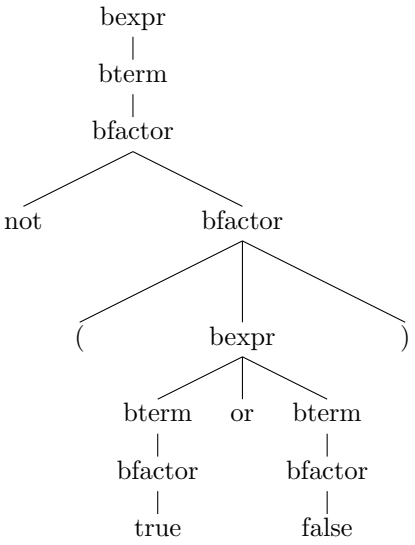
c) Árbol para (a, ((a, a), (a, a)))



Ejercicio 8: Árbol para "not (true or false)"

Gramática para expresiones booleanas

$$bexpr \rightarrow bexpr \text{ or } bterm \mid bterm$$
$$bterm \rightarrow bterm \text{ and } bfactor \mid bfactor$$
$$bfactor \rightarrow \text{ not } bfactor \mid (bexpr) \mid \text{ true } \mid \text{ false}$$



Ejercicio 9: Diseño de gramática para cadenas de 0 y 1

Requisito: Diseñar una gramática para el lenguaje del conjunto de todas las cadenas de símbolos 0 y 1 tales que todo 0 va inmediatamente seguido de al menos un 1.

Solución propuesta:

$$S \to A \mid \varepsilon$$
$$A \to 1A \mid 0B$$
$$B \to 1C$$
$$C \to 1C \mid A$$

Explicación

- S genera la cadena vacía o inicia la secuencia con A
- A puede generar secuencias que empiezan con 1 o con 0
- Si empieza con 0, obligatoriamente debe seguir al menos un 1 (a través de B y C)
- C permite generar múltiples 1's después de un 0

Ejercicio 10: Eliminación de recursividad por la izquierda

Gramática original

$$S \to (L) \mid a$$
$$L \to L, S \mid S$$

La recursividad por la izquierda está en la regla $L \to L, S$. Aplicamos el método estándar de eliminación:

$$L \rightarrow SL'$$
$$L' \rightarrow, SL' \mid \varepsilon$$

Gramática sin recursividad por la izquierda:

$$S \rightarrow (L) \mid a$$
$$L \rightarrow SL'$$
$$L' \rightarrow, SL' \mid \varepsilon$$

Ejercicio 11: Pseudocódigo para análisis sintáctico descendente recursivo

Gramática: $S \rightarrow (S) \mid x$

Algorithm 1 Análisis Sintáctico Descendente Recursivo

```
1:  procedure AnalizarS
2:      if el token actual es "(" then
3:          Consumir("(")
4:          AnalizarS()
5:          if el token actual es ")" then
6:              Consumir(")")
7:          else
8:              Error("Se esperaba )")
9:          end if
10:     else if el token actual es "x" then
11:         Consumir("x")
12:     else
13:         Error("Se esperaba ( o x")
14:     end if
15: end procedure
16: procedure Consumir(token esperado)
17:     if token actual = token esperado then
18:         Avanzar al siguiente token
19:     else
20:         Error("Token no esperado")
21:     end if
22: end procedure
```

Ejercicio 12: Movimientos de un analizador sintáctico predictivo

Entrada: (id+id)*id

rojo!15 **Pila**	**Entrada**
$E	(id+id)*id$
$T E'	(id+id)*id$
$F T' E'	(id+id)*id$
$(E) T' E'	(id+id)*id$
$E) T' E'	id+id)*id$
$T E') T' E'	id+id)*id$
$F T' E') T' E'	id+id)*id$
$id T' E') T' E'	id+id)*id$
$T' E') T' E'	+id)*id$
$E') T' E'	+id)*id$
$+T E') T' E'	+id)*id$
$T E') T' E'	id)*id$
$F T' E') T' E'	id)*id$
$id T' E') T' E'	id)*id$
$T' E') T' E'	)*id$
$E') T' E'	)*id$
$) T' E'	)*id$
$T' E'	*id$
$* F T' E'	*id$
$F T' E'	id$
$id T' E'	id$
$T' E'	$
$E'	$
$	$

Conclusión: La cadena es aceptada por el analizador.

Ejercicio 13: Modificación de gramática para operaciones adicionales

Gramática original

$$E \to E + T \mid E - T \mid T$$
$$T \to T * F \mid T/F \mid F$$
$$F \to (E) \mid \text{id}$$

Gramática modificada sin recursividad por la izquierda:

$$E \to TE'$$
$$E' \to +TE' \mid -TE' \mid \varepsilon$$
$$T \to FT'$$
$$T' \to *FT' \mid /FT' \mid \varepsilon$$
$$F \to (E) \mid \text{id} \mid \text{num}$$

Cambios realizados:

- Se eliminó la recursividad por la izquierda en todas las reglas

- Se agregaron las operaciones de resta y división
- Se añadió la posibilidad de usar literales numéricos (num)

Ejercicio 14: Implementación del analizador sintáctico

Algorithm 2 Método Descendente Recursivo para la Gramática Modificada

```
1: procedure AnalizarE
2:     AnalizarT()
3:     AnalizarEPrima()
4: end procedure
5: procedure AnalizarEPrima
6:     if el token actual es "+" then
7:         Consumir("+")
8:         AnalizarT()
9:         AnalizarEPrima()
10:    else if el token actual es "-" then
11:        Consumir("-")
12:        AnalizarT()
13:        AnalizarEPrima()
14:    else
15:        return                                   ▷ Producción vacía ()
16:    end if
17: end procedure
18: procedure AnalizarT
19:    AnalizarF()
20:    AnalizarTPrima()
21: end procedure
22: procedure AnalizarTPrima
23:    if el token actual es "*" then
24:        Consumir("*")
25:        AnalizarF()
26:        AnalizarTPrima()
27:    else if el token actual es "/" then
28:        Consumir("/")
29:        AnalizarF()
30:        AnalizarTPrima()
31:    else
32:        return                                   ▷ Producción vacía ()
33:    end if
34: end procedure
35: procedure AnalizarF
36:    if el token actual es "(" then
37:        Consumir("(")
38:        AnalizarE()
39:        Consumir(")")
40:    else if el token actual es "id" then
41:        Consumir("id")
42:    else if el token actual es "num" then
43:        Consumir("num")
44:    else
45:        Error("Se esperaba (, id o num")
46:    end if
47: end procedure
```

Implementación en Java

```java
public class AnalizadorSintactico {
    private Token tokenActual;
    private AnalizadorLexico lexico;

    public AnalizadorSintactico(AnalizadorLexico lexico) {
        this.lexico = lexico;
        this.tokenActual = lexico.siguienteToken();
    }

    public void analizar() {
        analizarE();
        if (tokenActual.tipo != TipoToken.FIN) {
            error("Se esperaba fin de entrada");
        }
    }

    private void analizarE() {
        analizarT();
        analizarEPrima();
    }

    private void analizarEPrima() {
        if (tokenActual.tipo == TipoToken.SUMA) {
            consumir(TipoToken.SUMA);
            analizarT();
            analizarEPrima();
        } else if (tokenActual.tipo == TipoToken.RESTA) {
            consumir(TipoToken.RESTA);
            analizarT();
            analizarEPrima();
        }
        // Si no es ninguno, se aplica la produccion vacia
    }
```

```
  private void analizarT() {
        analizarF();
        analizarTPrima();
    }

    private void analizarTPrima() {
        if (tokenActual.tipo == TipoToken.MULTIPLICACION) {
            consumir(TipoToken.MULTIPLICACION);
            analizarF();
            analizarTPrima();
        } else if (tokenActual.tipo == TipoToken.DIVISION) {
            consumir(TipoToken.DIVISION);
            analizarF();
            analizarTPrima();
        }
        // Si no es ninguno, se aplica la produccion vacia
    }

    private void analizarF() {
        if (tokenActual.tipo == TipoToken.PARENTESIS_IZQUIERDO) {
            consumir(TipoToken.PARENTESIS_IZQUIERDO);
            analizarE();
            consumir(TipoToken.PARENTESIS_DERECHO);
        } else if (tokenActual.tipo == TipoToken.IDENTIFICADOR) {
            consumir(TipoToken.IDENTIFICADOR);
        } else if (tokenActual.tipo == TipoToken.NUMERO) {
            consumir(TipoToken.NUMERO);
        } else {
            error("Se esperaba (, identificador o n mero");
        }
    }

    private void consumir(TipoToken tipoEsperado) {
        if (tokenActual.tipo == tipoEsperado) {
            tokenActual = lexico.siguienteToken();
        } else {
            error("Se esperaba " + tipoEsperado);
        }
    }

    private void error(String mensaje) {
        throw new RuntimeException("Error sint ctico: " + mensaje);
    }
}
```

Conclusión

El estudio y análisis del documento proporcionado han permitido comprender en profundidad los conceptos fundamentales abordados en este trabajo. A través del uso de LaTeX, se logró estructurar el contenido de manera clara y organizada, facilitando la presentación de la información con un formato profesional y académico.

Asimismo, la revisión del material en formato PDF permitió extraer los elementos clave necesarios para fundamentar el marco teórico y desarrollar una perspectiva crítica sobre el tema tratado. Este proceso contribuyó a fortalecer el conocimiento en la materia, fomentando un enfoque analítico y estructurado en la investigación.

En conclusión, la combinación de herramientas adecuadas y un análisis detallado del contenido han resultado esenciales para la elaboración de este documento, proporcionando una base sólida para futuras investigaciones y aplicaciones prácticas relacionadas con el tema.