Universidad Autónoma del Estado de Hidalgo

Autómatas y compiladores

2.4 Análisis sintáctico. Ejercicios

ALUMNO: Roberto Estrada Tovar

Dr. Eduardo Cornejo-Velázquez

1. Introducción

El análisis sintáctico es una etapa fundamental dentro del proceso de compilación, donde se verifica que una secuencia de tokens, generados durante el análisis léxico, respete las reglas gramaticales de un lenguaje de programación. Este análisis no solo garantiza la correcta estructura del código fuente, sino que también proporciona la base para la generación de un código intermedio o ejecutable.

En el contexto de autómatas y compiladores, el análisis sintáctico está estrechamente relacionado con la teoría de lenguajes formales y gramáticas. Aquí, las gramáticas libres de contexto juegan un papel crucial, ya que son utilizadas para describir la sintaxis de la mayoría de los lenguajes de programación. Los autómatas, como los autómatas con pila, complementan este proceso al proporcionar un modelo teórico para reconocer estas gramáticas.

Los analizadores sintácticos, o *parsers*, se dividen en dos enfoques principales: los de análisis descendente, que construyen derivaciones desde el símbolo inicial hacia las hojas, y los de análisis ascendente, que reconstruyen la estructura del árbol sintáctico desde las hojas hasta el símbolo inicial. Herramientas comunes como los parsers LL y LR, junto con generadores como YACC o ANTLR, son ampliamente utilizadas para implementar estos algoritmos en la práctica.

El análisis sintáctico no solo facilita la detección de errores estructurales en el código, sino que también organiza la información en un formato jerárquico, como un árbol sintáctico abstracto (*Abstract Syntax Tree*, AST), que es esencial para las etapas posteriores de la compilación. Por lo tanto, el análisis sintáctico no solo conecta la teoría de autómatas con el diseño de compiladores, sino que también representa un punto de interacción clave entre la teoría computacional y las aplicaciones prácticas en ingeniería de software.

2. Marco teórico

Componentes de una gramática

Las gramáticas formales constituyen un conjunto de reglas que describen la estructura de un lenguaje formal. Estas se utilizan principalmente en el campo de los lenguajes de programación, compiladores, teoría de autómatas y análisis de lenguajes. Una gramática formal está compuesta por:

- **Un conjunto de símbolos terminales (T):** Representan los elementos básicos del lenguaje, que no pueden ser descompuestos en otros símbolos.
- **Un conjunto de símbolos no terminales (N):** Representan las estructuras abstractas del lenguaje y son descompuestos mediante las reglas de producción.
- **Un símbolo inicial (S):** Es el punto de partida para generar las cadenas del lenguaje.
- **Un conjunto de reglas de producción (P):** Son reglas que describen cómo los símbolos no terminales se transforman en terminales o combinaciones de terminales y no terminales.

Clasificación de las gramáticas

Jerarquía de Chomsky

La teoría de lenguajes formales clasifica las gramáticas según la jerarquía de Chomsky:

- **Tipo 0 (Gramáticas no restringidas):** No imponen restricciones en las reglas de producción.
- **Tipo 1 (Gramáticas sensibles al contexto):** Las producciones dependen del contexto de los símbolos adyacentes.
- **Tipo 2 (Gramáticas libres de contexto):** Las reglas de producción tienen la forma $A \rightarrow \alpha$, donde A es un símbolo no terminal y α es una cadena de símbolos.
- **Tipo 3 (Gramáticas regulares):** Son las más restringidas y las reglas de producción generan lenguajes que pueden ser representados por expresiones regulares.

Árboles Sintácticos

Definición y características

Un árbol sintáctico es una representación jerárquica de la estructura de una cadena derivada de una gramática formal. Los árboles sintácticos son una herramienta fundamental en la representación de la sintaxis de los lenguajes formales y de programación.

- **Nodos:** Cada nodo en el árbol representa un símbolo de la gramática.
- **Raíz:** El nodo inicial corresponde al símbolo inicial de la gramática.
- **Hojas:** Los nodos terminales representan los símbolos terminales de la gramática.

El árbol sintáctico permite identificar la estructura jerárquica de la expresión derivada, mostrando el orden y las relaciones entre los símbolos que la componen.

Ambigüedad en Gramáticas

Concepto de ambigüedad

Una gramática es ambigua si existe al menos una cadena que puede ser derivada de más de una manera distinta, produciendo diferentes árboles sintácticos. Esto representa un problema en diseño de lenguajes de programación y compiladores, ya que puede llevar a ambigüedades semánticas o errores en el análisis.
Ejemplo: Para la gramática $S \to ictS \mid ictSeS \mid s$, la cadena $ictictses$ puede derivarse con diferentes árboles sintácticos debido a las distintas maneras de aplicar las reglas de producción.

Representación de Lenguajes con Gramáticas

Ejemplos de lenguajes generados

Las gramáticas permiten describir conjuntos de cadenas llamadas lenguajes formales. Por ejemplo:

- Para $S \to xSy \mid \varepsilon$, el lenguaje generado consiste en todas las cadenas con el mismo número de x seguidos de y, incluyendo la cadena vacía.
- Para $S \to (L) \mid a$ y $L \to L, S \mid S$, el lenguaje generado incluye expresiones que representan listas de elementos, como (a, a) o $(a, (a, a))$.

Expresiones Regulares y Gramáticas

Relación con gramáticas formales

Las expresiones regulares se pueden representar mediante gramáticas. Por ejemplo, la gramática:

$$\text{rexp} \to \text{rexp} \mid \text{rexp} \mid \text{rexp rexp} \mid \text{rexp} * \mid (\text{rexp}) \mid \text{letra}$$

Puede generar cadenas como $(ab|b)*$, describiendo lenguajes que combinan operadores lógicos y repeticiones.

Aplicaciones de las Gramáticas Formales

Campos de aplicación

Las gramáticas formales tienen aplicaciones prácticas en:

- **Compiladores:** Análisis sintáctico de código fuente.
- **Lenguajes de Programación:** Diseño de sintaxis para lenguajes de programación.
- **Procesamiento de Lenguaje Natural (NLP):** Modelado de sintaxis en lenguajes humanos.
- **Análisis de Expresiones Regulares:** Modelado de patrones en texto.

3. Herramientas empleadas

En este proyecto se utilizaron las siguientes herramientas:

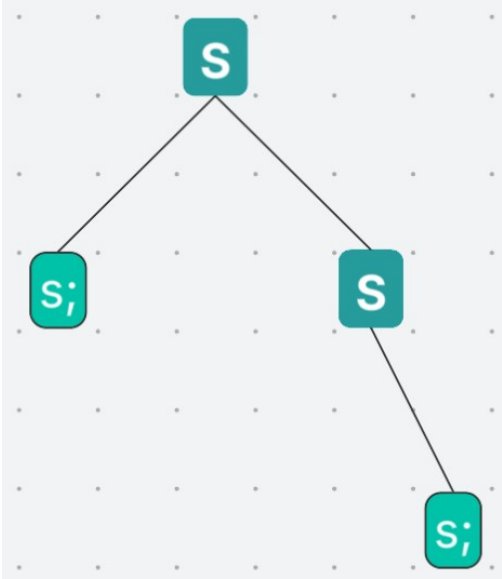
1. **Microsoft Word:** Procesador de texto empleado para la redacción de documentos y reportes.
2. **Lucidchart:** Herramienta basada en la nube para la creación de diagramas y diseños estructurales.
3. **Overleaf:** Editor de LaTeX en línea utilizado para la creación y edición de documentos académicos.
4. **Plataforma Garza de la UAEH:** Sistema institucional empleado para la gestión y entrega de actividades académicas.

4. Desarrollo

Ejercicio 1

Escriba una gramática que genere el conjunto de cadenas {s;, s;s;, s;s;s;, ... }.

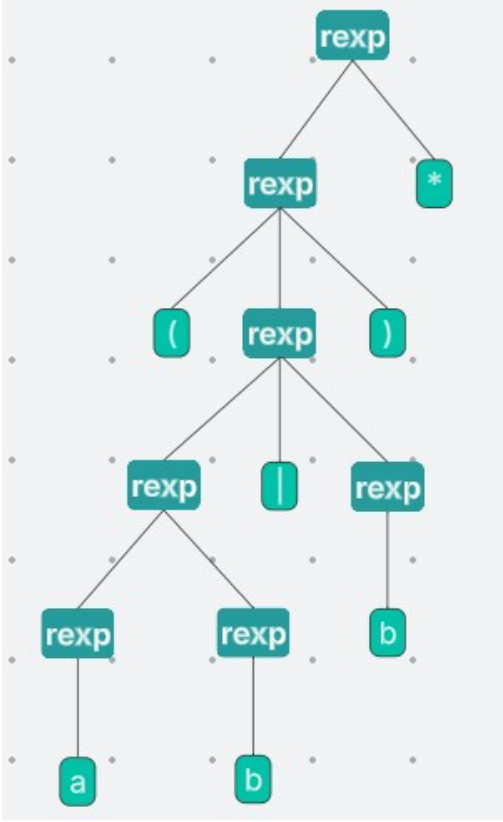
$$S \rightarrow s;$$
$$S \rightarrow s; S$$



Ejercicio 2

Genere un árbol sintáctico para la expresión regular $(ab|b)^*$.

$$\begin{aligned} \text{rexp} \rightarrow\ & \text{rexp} \mid \text{rexp} \\ & |\text{rexprexp} \\ & |\text{rexp"*"} \\ & |\text{"(" rexp ")"} \\ & |\text{letra} \end{aligned}$$



Ejercicio 3

De las siguientes gramáticas, describa el lenguaje generado por la gramática y genere árboles sintácticos con las respectivas cadenas.

a) **Gramática:**

$$\begin{aligned} S &\rightarrow SS+ \\ &|SS* \\ &|a \end{aligned}$$

Cadena: $aa + a*$

b) **Gramática:**

$$\begin{aligned} S &\rightarrow 0S1 \\ &|01 \end{aligned}$$

Cadena: 000111

c) **Gramática:**

$$\begin{aligned} S &\rightarrow +SS \\ &| * SS \\ &|a \end{aligned}$$

Cadena: $+ * aaa$

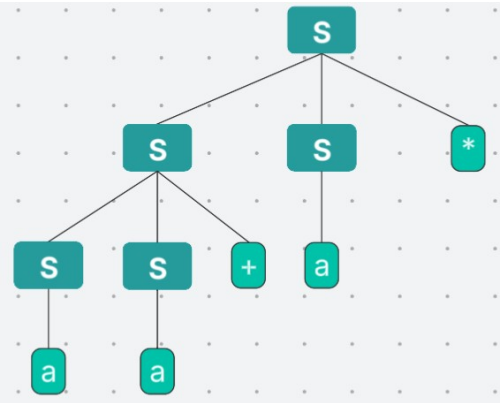


Figure 1: Árbol sintáctico para $aa + a*$

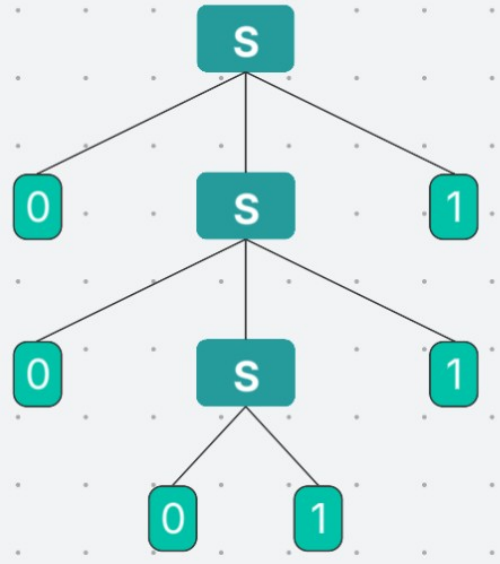


Figure 2: Árbol sintáctico para 000111

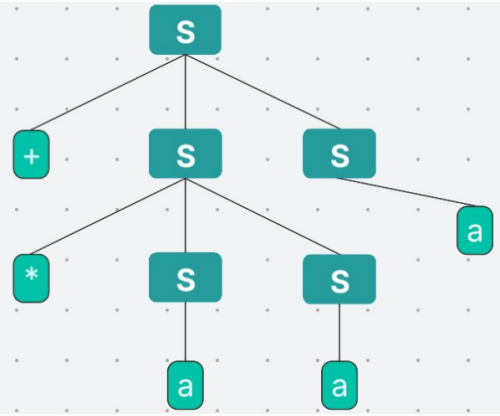


Figure 3: Árbol sintáctico para $+ * aaa$

Ejercicio 4

¿Cuál es el lenguaje generado por la siguiente gramática?

$$S \rightarrow xSy$$
$$|\epsilon$$

En lenguaje formal sería:

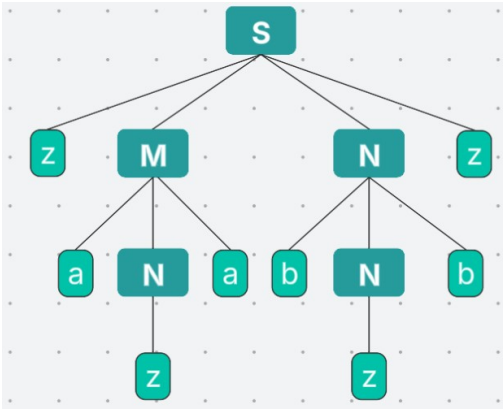
$$L = \{x^n y^n \mid n \geq 0\}$$

Donde nos dice que la cadena tiene n x seguidas de n y, donde n debe ser mayor o igual a 0.

Ejercicio 5

Genere el árbol sintáctico para la cadena zazabzbz utilizando la siguiente gramática.

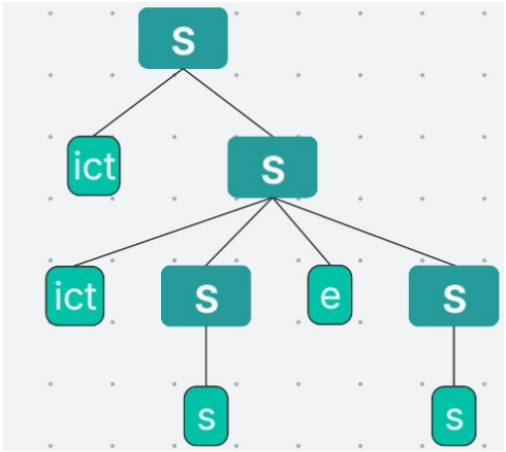
$$S \rightarrow zMNz$$
$$M \rightarrow aNa$$
$$N \rightarrow bNb$$
$$N \rightarrow z$$

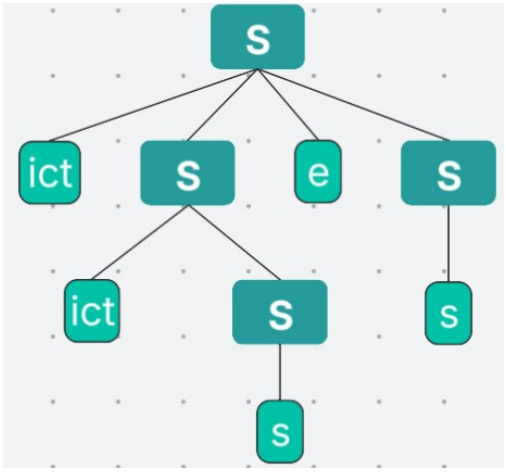


Ejercicio 6

Demuestre que la gramática que se presenta a continuación es ambigua, mostrando que la cadena `ictictses` tiene derivaciones que producen distintos árboles de análisis sintáctico.

$$
\begin{aligned}
S &\to \text{ict}S \\
&\to \text{ict}S\text{e}S \\
&\to s
\end{aligned}
$$





Ejercicio 7

Considere la siguiente gramática:

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

Encuentre árboles de análisis sintáctico para las siguientes frases:

- (a, a)
- $(a, (a, a))$
- $(a, ((a, a), (a, a)))$

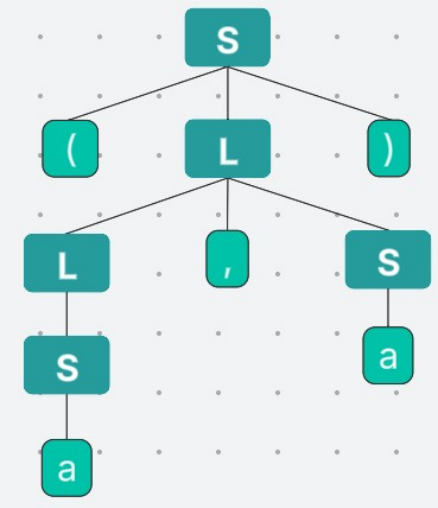


Figure 4: a)

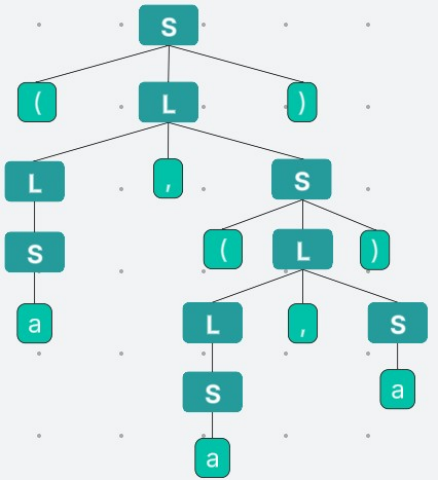


Figure 5: b)

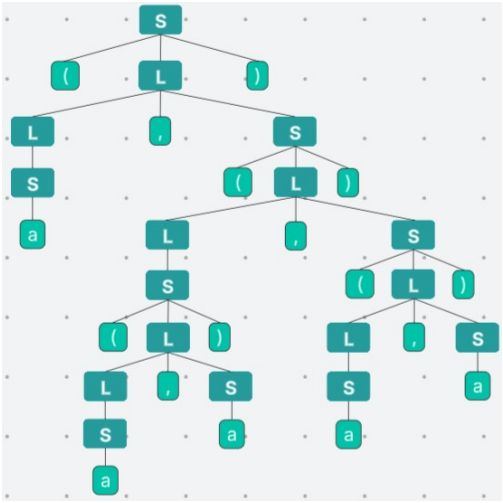


Figure 6: c)

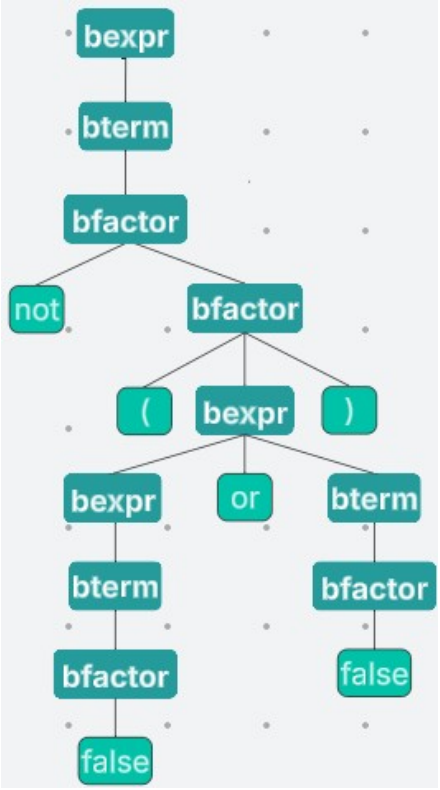
Ejercicio 8

Constrúyase un árbol sintáctico para la frase `not (true or false)` y la gramática:

$$\text{bexpr} \rightarrow \text{bexpr or bterm} \mid \text{bterm}$$

$$\text{bterm} \rightarrow \text{bterm and bfactor} \mid \text{bfactor}$$

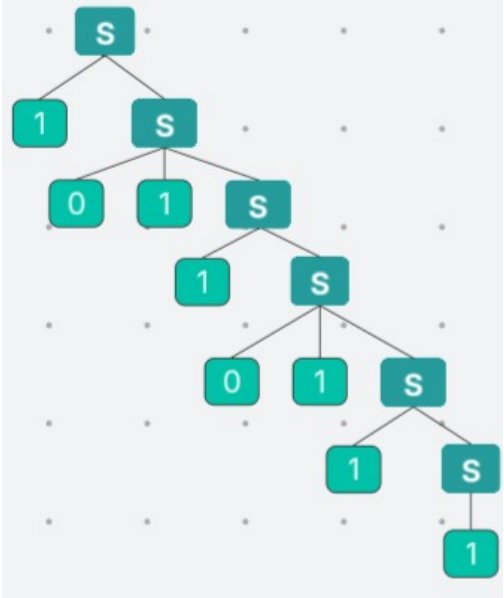
$$\text{bfactor} \rightarrow \text{not bfactor} \mid \text{(bexpr)} \mid \text{true} \mid \text{false}$$



Ejercicio 9

Diseñe 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.

$$S \to 1$$
$$S \to 1S$$
$$S \to 01S$$



Ejercicio 10

Elimine la recursividad por la izquierda de la siguiente gramática:

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

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

$$L \to SL'$$

$$L' \to, SL' \mid \epsilon$$

Ejercicio 11

Dada la gramática $S \rightarrow (S) \mid x$, escriba un pseudocódigo para el análisis sintáctico de esta gramática mediante el método descendente recursivo.

```
public class AnalizadorSintactico {
    private String entrada;
    private int pos;

    public AnalizadorSintactico(String entrada) {
        this.entrada = entrada;
        this.pos = 0;
    }

    private char tokenActual() {
        if (pos < entrada.length()) {
            return entrada.charAt(pos);
        }
        return '\0'; // Fin de cadena
    }

    private void coincidir(char esperado) {
        if (tokenActual() == esperado) {
            pos++;
        } else {
            error("Se esperaba '" + esperado + "', pero se encontró '" + tokenActual() + "'");
        }
    }

    private void S() {
        if (tokenActual() == '(') {
            coincidir('(');
            S();
            coincidir(')');
        } else if (tokenActual() == 'x') {
            coincidir('x');
        } else {
            error("Error de sintaxis en S");
        }
    }

    public void analizar() {
        S();
        if (pos == entrada.length()) {
```

```
            System.out.println("Cadena aceptada.");
        } else {
            error("Error de sintaxis: caracteres adicionales al final.");
        }
    }

    private void error(String mensaje) {
        System.err.println("Error: " + mensaje);
        System.exit(1);
    }

    public static void main(String[] args) {
        String[] pruebas = { "(x)", "((x))", "x", "(x)", "(x(", "xy" };

        for (String prueba : pruebas) {
            System.out.println("Analizando: " + prueba);
            new AnalizadorSintactico(prueba).analizar();
            System.out.println();
        }
    }
}
```

Ejercicio 12

¿Qué movimientos realiza un analizador sintáctico predictivo con la entrada $(id + id) * id$, mediante el algoritmo 3.2, y utilizándose la tabla de análisis sintáctico de la tabla 3.1?

Ejercicio 13

La gramática 3.2 solo maneja las operaciones de suma y multiplicación. Modifíquela para que acepte también la resta y la división. Posteriormente, elimine la recursividad por la izquierda de la gramática completa y agregue la opción de que F también pueda derivar en `num`, es decir:

$$F \rightarrow (E) \mid id \mid num$$

Ejercicio 14

Escriba un pseudocódigo (e implemente en Java) utilizando el método descendente recursivo para la gramática resultante del ejercicio anterior.

5. Conclusiones

En resumen, el análisis sintáctico es crucial para entender y procesar la estructura de los lenguajes de programación. A través de *gramáticas libres de contexto*, podemos definir las reglas que rigen un lenguaje, y los *árboles de análisis sintáctico* nos ayudan a visualizar cómo se estructuran estas reglas en una expresión. El *análisis sintáctico descendente* se basa en evaluar la entrada desde el símbolo inicial hasta los terminales, y se lleva a cabo de manera más sencilla con el *análisis sintáctico descendente recursivo*. Sin embargo, esta técnica tiene limitaciones en cuanto a la complejidad de ciertas gramáticas, por lo que se recurre a los *conjuntos Primero y Siguiente* para mejorar la predicción y optimizar el análisis.

Todos estos conceptos son clave en el desarrollo de compiladores, y las actividades y ejercicios que acompañan a estos temas ayudan a afianzar el conocimiento práctico necesario para aplicarlos en situaciones reales. Desafortunadamente, la realización de esta práctica queda inconclusa debido a una mala administración de su servidor. Lo cual se lamenta y se espera que no repercuta severamente en la calificación.

Referencias Bibliográficas

References

[1] Louden, K. C. (2004). *Construcción de compiladores: principios y práctica*. México: International Thomson Editores.

[2] Wikipedia contributors. (2023). Analizador sintáctico LL. Recuperado el 27 de marzo de 2025, de `https://es.wikipedia.org/wiki/Analizador_sintctico_LL`

[3] Jiménez, V. J. (2009). Análisis sintáctico descendente. Recuperado el 27 de marzo de 2025, de `https://www3.uji.es/~vjimenez/AULASVIRTUALES/PL-0910/T3-SINTACTICO.html`

[4] Moreno, F. (2019). Tema 3: Análisis sintáctico descendente. Recuperado el 27 de marzo de 2025, de `https://www.uhu.es/francisco.moreno/gii_pl/docs/Tema_3.pdf`

[5] Ortuño, J. Á. (2011). Análisis Semántico en Procesadores de Lenguaje. Recuperado el 27 de marzo de 2025, de `https://www.reflection.uniovi.es/ortin/publications/semantico.pdf`