Analizador sintáctico para un lenguaje de consultas tipo Google

Mieres Julian Edgardo; Sanchez Octavio; Freixes Hutchings Matias

Universidad Nacional del Nordeste,
julianmierez@gmail.com
Universidad Nacional del Nordeste,
octaviosanchez070103@gmail.com
Universidad Nacional del Nordeste,
matufreixes@gmail.com

Resumen

Este artículo presenta el diseño e implementación de un analizador sintáctico para un mini lenguaje de consultas inspirado en los buscadores web tipo Google, que permite la combinación de términos mediante operadores lógicos como AND, OR y NOT, así como el uso de paréntesis para agrupar expresiones. La herramienta fue desarrollada como parte de un trabajo integrador de la materia Teoría de la Computación, aplicando los conceptos de gramáticas libres de contexto y análisis sintáctico descendente. El sistema permite verificar si una consulta está estructuralmente bien formada, generar su árbol sintáctico y detectar errores de forma clara, este proyecto se implementa con una interfaz gráfica en WPF con C#.

1. Introducción

El análisis sintáctico permite validar la estructura de una expresión en función de una gramática formal que define un lenguaje que simula consultas del estilo Google, es decir, cadenas de texto que pueden combinar palabras o frases entre comillas con conectores booleanos y paréntesis. Esta herramienta tiene como objetivo permitir a un sistema interpretar y validar la estructura lógica de consultas compuestas, facilitando así el aprendizaje de conceptos clave como gramáticas libres de contexto, derivaciones, árboles sintácticos y parseo descendente.

2. Análisis sintético para un lenguaje de consultas tipo Google

Los lenguajes que permiten construir expresiones de búsqueda, como los utilizados en motores de consulta de filtros lógicos, pueden ser definidos sintácticamente mediante gramáticas libres de contexto, o el tipo 2 según la clasificación de Chomsky.

A. Gramática libre de contexto

Una gramática de contexto libre se puede entender como:

G = {VT, VN, P, S}

Donde G representa la gramática del lenguaje, VT los símbolos terminales, VN los símbolos no terminales, P las reglas de producción y S el Símbolo inicial del lenguaje. En los lenguajes que implementan estas gramáticas se definen dos conjuntos finitos disjuntos; el primer conjunto está formado por los símbolos terminales VT, equivalentes a símbolos atómicos (es decir que no pueden desagregarse en otros) y el segundo conjunto está formado por los símbolos no terminales VN, equivalentes a los roles sintácticos. Este conjunto incluye una cantidad finita de reglas de producción P, correspondientes a las reglas gramaticales, y un símbolo S.

B. Derivaciones y árboles sintácticos

Analizar sintácticamente una cadena de tokens es encontrar para ella un árbol sintáctico o derivación, que tenga como raíz el símbolo inicial de la gramática libre de contexto y mediante la aplicación sucesiva de sus reglas de derivación se pueda alcanzar dicha cadena como hojas del árbol sintáctico. En caso de éxito la sentencia pertenece al lenguaje generado por la gramática. En caso contrario, es decir cuando no se encuentra el árbol que genera dicha gramática, se dice entonces que la sentencia no pertenece al lenguaje.

C. Ejemplo: gramática de expresiones de consulta booleanas

Supongamos que se desea construir una gramática:

G = {VN, VT, S, P} que describe un lenguaje basado en la utilización de expresiones de consultas lógicas, como las empleadas en motores de búsqueda o

filtros avanzados. Este lenguaje permite el uso de operadores booleanos (AND, OR, NOT), paréntesis para agrupar expresiones, términos simples y frases entre comillas dobles. Su objetivo es modelar expresiones que combinen palabras clave o frases completas mediante conectores lógicos.

La gramática está compuesta por un vocabulario terminal VT formado por los operadores lógicos, paréntesis y los tokens palabra y frase. El vocabulario no terminal VN contiene los símbolos no terminales <CONSULTA>, <EXP> y <TERMINO>. El símbolo inicial S será <CONSULTA>.

VT = {AND, OR, NOT, (,), palabra, frase}

VN = {<CONSULTA>, <EXP>, <TERMINO>}

S = <CONSULTA>

Las reglas de producción P que representan al lenguaje se numeran a continuación del (1) al (6). Esta numeración se utilizará para referenciar las reglas aplicadas en las derivaciones:

1. <CONSULTA> ::= <EXP>

2.<EXP> ::= <EXP> AND <EXP>

3. <EXP> ::= <EXP> OR <EXP>

4. <EXP> ::= NOT <EXP>

5. <EXP> ::= (<EXP>)

6.<EXP>::=<TERMINO>

7.<TERMINO>::= palabra

8.<TERMINO> ::= Frase

Ejemplo: determinar si la expresión NOT (ropa OR "sin stock") pertenece al lenguaje generado por G.

Para ello, se debe alcanzar dicha cadena a partir de

derivaciones desde el símbolo inicial <consulta>, y luego construir el árbol sintáctico correspondiente. A continuación, se muestra la secuencia de derivaciones, indicando el número de regla aplicada en cada paso

1. <consulta> -> <EXP>

4. -> NOT<EXP>

5. -> NOT(<EXP>)

3. -> NOT (<EXP> OR <EXP>)

6. ->NOT (<TERMINO> OR <EXP>)

7. ->NOT (palabra OR <EXP>)

6. ->NOT (palabra OR <termino>)

8. -> NOT (palabra OR frase)

Reemplao con los tokens reales que reconoce el analizador léxico:

<consulta>, -> NOT<EXP>, -> NOT(<EXP>), -> NOT (<EXP> OR <EXP>), ->NOT (ropa OR <EXP>), -> ->NOT (ropa OR “sin stock”).

El analizador léxico es responsable de reconocer los símbolos terminales, asignando la categoría palabra al token ropa, y frase al token entre comillas "sin stock".

D. Caso de Estudio

En esta sección se muestre un ejemplo práctico que ofrece solución aplicable a un caso de estudio propuesto por la catedra de Teoría de la Computación que consiste en la implementación de este lenguaje de mini consultas con interfaz gráfica, en el cual se puede escribir, analizar, importar o exportar el análisis de una expresión.

A continuación, se hará demostración de los casos de prueba en la ya mencionada aplicación

En este caso de prueba se deriva la expresión en símbolos terminales como el AND y derivar los no terminales de forma sucesiva hasta obtener los terminales, en este caso para la consulta: perro AND gato

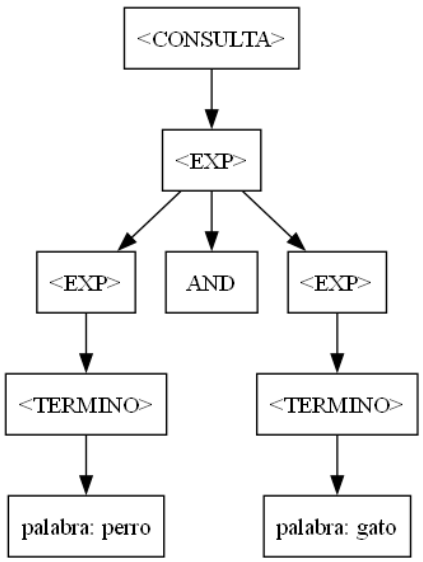

Fig. 1 Imagen generada por el analizador de un caso de prueba

El siguiente caso de prueba será: “frase de prueba” OR NOT palabraX and palabraY, que genera el siguiente árbol de derivación

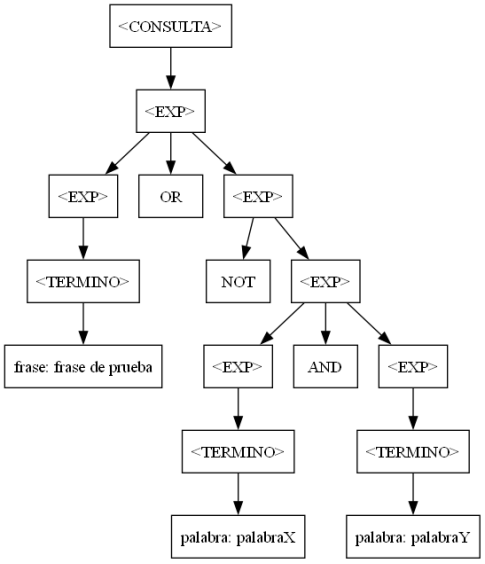

Fig. 2 Imagen generado por el analizador léxico para el caso de prueba mencionado anteriormente

El siguiente caso de prueba será: “camisa azul” OR “camiseta blanca”, que genera el siguiente árbol de derivación

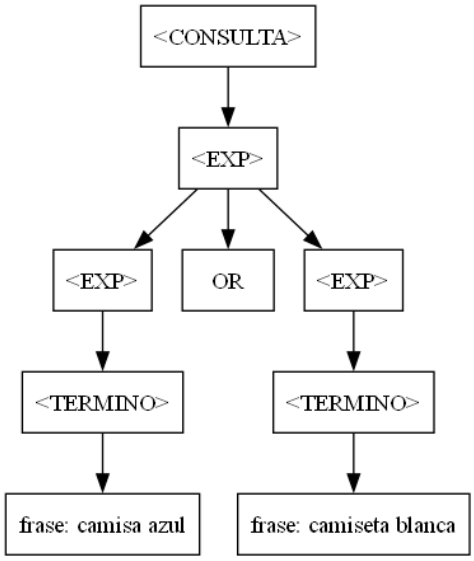

Fig. 3 Imagen que generada por el analizador léxico para el caso de prueba “camisa azul” OR “camisa blanca”

Y el último caso de éxito que se analizará será para la expresión palabra123, que genera el siguiente árbol

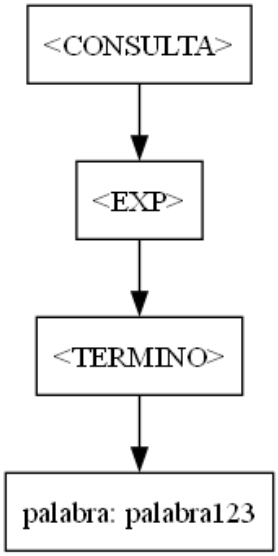

Fig. 4 Imagen generada por el analizador lexico de la expresión palabra123

A continuación, se van a mostrar como hay casos de prueba que generan error debido a errores en la congruencia de la expresión

En un primer caso se usara la expresión oferta OR OR descuento

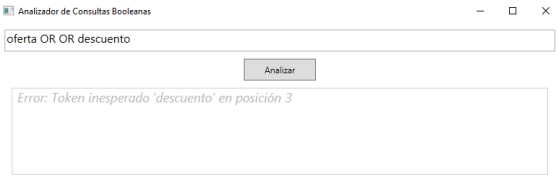

Fig. 5 Captura de error de sintaxis, generada por la expresión: oferta OR OR descuento

Como se ve, esta expresión genera error, ya que el OR al ser un operador de carácter binario no puede tener concatenado otro OR

En el siguiente caso se usará la expresión: AND zapatilla, que genera la siguiente captura

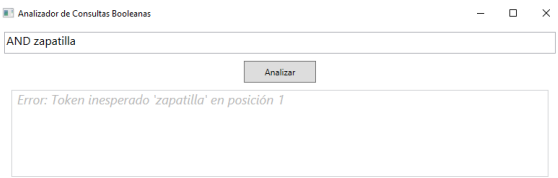

Fig. 6 Captura de error de sintaxis, generada por la expresión: AND zapatilla

En este caso, lo que sucede es que el AND necesita 2 expresiones a ambos lados para estar correctamente escrito, y este caso no se cumple

3. Conclusiones

Para concluir la finalización de este trabajo, se puede decir que para realizar una correcta implementación de este analizador y que genere las salidas correctas, se debe hacer lo posible porque el analizador tenga el control de la situación, y que puede responder a todas las situaciones posibles, y que las únicas veces que falle sean por error humano, y que cuando esto pase, se detecte y se señale de forma clara y coherente el error cometido, evitando hacer muchos mensajes para el mismo error.
Cabe aclarar que la información que existe con respecto a este tema es muy extensa y existen muchísimos recursos a los que se pueden recurrir para la realización de cualquier proyecto relacionado al tema debido que es uno de los conceptos más utilizados dentro del mundo de la computación

Para implementar un analizador sintáctico bien definido se deben considerar múltiples factores. Para algunos autores como Wirth [1] quien especifica que las características de un buen analizador sintáctico son: que ninguna sentencia debe dar lugar a que el analizador sintáctico pierda el control; que todos los errores sintácticos deben de ser detectados y señalados; y que los errores muy frecuentes e imputables a verdaderos fallos de compresión o descuido del programador, habrán de ser diagnosticados correctamente, siendo esta tercera característica la más difícil de lograr, ya que es habitual ver muchos analizadores que emiten dos o más mensajes para un determinado error. Existen también diferentes métodos y enfoque para el diseño de analizadores sintácticos, así como también una infinidad de material relacionado con este tema y herramientas para su implementación. En esta propuesta se aplicó todo lo relacionado con los conceptos básicos descriptos en este artículo que han servido de soporte para una introducción a los primeros pasos entre teoría e implementación.

Bibliografía

(Las referencias van al final y enumeradas de acuerdo a la secuencia de lectura):

[1] Universidad Nacional del Nordeste - Facultad de Ciencias Exactas y Naturales y Agrimensura, “Teoría de la Computación - Unidad 1: Lenguajes y Gramáticas Formales”, TC01.pdf, 2020. [En línea]. Disponible en el aula virtual. [Último acceso: 06 2025]

[2] Universidad Nacional del Nordeste - Facultad de Ciencias Exactas y Naturales y Agrimensura, “Teoría de la Computación - Unidad 3: Autómatas Push-Down y Análisis Sintáctico”, TC03.pdf, 2020. [En línea]. Disponible en el aula virtual. [Último acceso: 06 2025].