Centro Universitario de Ciencias Exactas e Ingeniería

SEM. de Solución de Problemas de Traductores 2

Practica 2

LUIS FELIPE MUNOZ MENDOZA

Juan Pablo Hernández Orozco

219294285

Objetivo:

Realizar un analizador sintáctico que reconozca sentencias de asignación u operaciones simples, y valide su estructura con una gramática como la siguiente (ejemplo):
<program> -> <assignment> | <assignment> <program>
<assignment> -> <identifier> = <expression> ;
<identifier> -> [a-zA-Z][a-zA-Z0-9_]*
<expression> -> <term> | <term> + <expression> | <term>

- <expression>
  <term> -> <factor> | <factor> * <term> | <factor> / <term>
  <factor> -> <identifier> | <number> | (<expression>)
  <number> -> [0-9]+
  Pueden usar arboles sintácticos, tablas de derivación y/o autómatas. Se pueden apoyar con herramientas y librerias como yacc, o bison

El objetivo de esta práctica es *desarrollar un analizador sintáctico* que reconozca sentencias de asignación y operaciones aritméticas simples (+, -, *, /) para validar la estructura gramatical de una entrada. Se busca:

- Implementar un analizador léxico que identifique tokens como identificadores, números y operadores.
- Implementar un analizador sintáctico basado en la gramática propuesta.
- Probar interactivamente la correcta lectura de cadenas y la generación de un árbol sintáctico.

Introducción:

En la teoría de lenguajes formales y en el desarrollo de compiladores, el analizador sintáctico (o parser) es un componente esencial que se encarga de verificar que las secuencias de tokens generadas previamente por el analizador léxico se ajusten a las reglas gramaticales definidas para un lenguaje. Este proceso se fundamenta en la definición de una gramática libre de contexto (GLC), la cual describe de forma precisa la estructura y organización de las sentencias y expresiones válidas.

En esta práctica se utiliza la librería PLY (Python Lex-Yacc) para implementar tanto el analizador léxico (scanner) como el analizador sintáctico. El ejemplo se centra en el reconocimiento de sentencias de asignación y operaciones aritméticas simples, permitiendo analizar expresiones como:

x = 3 + 5;

y = (x - 2) * 4;

Desarrollo:

Definición del Analizador Léxico

Se establecieron los tokens básicos del lenguaje, incluyendo:

Números (NUMBER): Reconoce secuencias de dígitos y convierte la cadena a entero.

Identificadores (ID): Deben iniciar con una letra y pueden ir seguidos de letras, dígitos o guiones bajos.

Operadores Aritméticos (PLUS, MINUS, TIMES, DIVIDE): Permiten la suma, resta, multiplicación y división.

Símbolos: Se incluyen paréntesis (LPAREN, RPAREN), el operador de asignación (ASSIGN) y el punto y coma (SEMI).

Además, se implementó una función para gestionar errores léxicos, de manera que caracteres o secuencias inesperadas sean identificados y reportados.

Diseño de la Gramática

Se definió una gramática libre de contexto que describe la estructura del lenguaje. La gramática contempla:

Programa: Una secuencia de asignaciones.

Asignación: Consiste en un identificador, el operador =, una expresión y un punto y coma.

Expresiones Aritméticas: Se descomponen en términos y factores para asegurar la correcta precedencia de los operadores. Por ejemplo, se da mayor prioridad a las operaciones de multiplicación y división sobre la suma y la resta.

Esta descomposición en reglas permite generar un árbol sintáctico que refleja la estructura jerárquica de las operaciones.

Implementación del Analizador Sintáctico

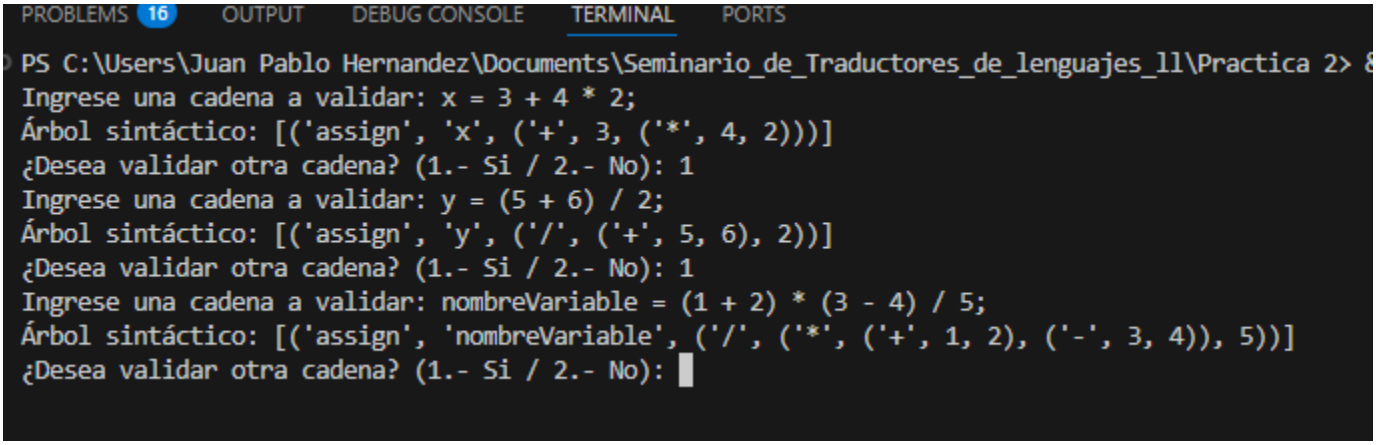
Utilizando PLY, se implementaron las reglas gramaticales mediante funciones específicas. Cada función correspondiente a una regla procesa los tokens recibidos y genera nodos del árbol sintáctico (generalmente representados como tuplas). Por ejemplo, una asignación se representa como:

('assign', 'x', ('+', 3, ('*', 4, 2)))

Adicionalmente, se incluyó una función de manejo de errores sintácticos para identificar y reportar de forma clara el token o elemento en el que se produce la falla.

Interfaz Interactiva para Pruebas

Se desarrolló una interfaz sencilla en la que el usuario puede ingresar cadenas de código. El programa analiza cada entrada, genera el árbol sintáctico correspondiente y lo muestra en pantalla. También permite repetir el proceso, lo que facilita la prueba de múltiples casos y el análisis de errores tanto léxicos como sintácticos.



Conclusión:

El analizador sintáctico desarrollado demuestra de forma práctica cómo, a partir de un conjunto de tokens y reglas gramaticales bien definidas, se puede validar la estructura de sentencias y expresiones aritméticas. La utilización de PLY simplifica la implementación y permite centrarse en el diseño de la gramática y la generación del árbol sintáctico.

El árbol obtenido no solo evidencia la correcta interpretación de la sintaxis, sino que también sienta las bases para fases posteriores en el proceso de compilación, como la generación de código y la optimización. Además, esta práctica se puede ampliar para incorporar nuevas funcionalidades o estructuras de lenguaje, profundizando en conceptos fundamentales de compiladores y lenguajes formales.

Repositorio

https://github.com/ELJuanP/Seminario_de_Traductores_de_lenguajes_ll/tree/c6a0e97650048032e6c5b17f3150fc868676bdc8/Practica%202