Instituto Tecnológico de Costa Rica

Compiladores e Intérpretes

Segundo Proyecto:

# Analizador Semántico y Código Tres Direcciones

Profesor:

Allan Rodríguez Dávila

Estudiantes:

Dylan Rodríguez Chavarría

Bryan Londoño Marchena

Segundo Semestre 2025

## Manual de Usuario y Pruebas de Funcionalidad

Para este apartado se realizó un vídeo donde se muestran y explican las características del proyecto. Link

## Descripción del Problema

Se presenta la asignación que da continuidad a lo realizado en el primer proyecto donde se debe de llevar a cabo la construcción de un analizador semántico y el código tres direcciones.

Para el analizador semántico se deben de tomar en cuenta todas las consideraciones necesarias referentes a la validación de tipos, esto puede ser, por ejemplo: la existencia de un identificador cuando se hace referencia a este, validación de los tipos de los operandos en las expresiones aritméticas, relacionales y lógicas, verificación de las llamadas a las funciones y sus parámetros entre otras más.

Para el código tres direcciones este debe de tomar en cuenta que en cada línea solamente se pueden ubicar tres instrucciones, además de otros aspectos como el manejo de la memoria y ciertas señales que se deben de dejar en el código generado para por ejemplo indicar si una operación es entre operandos enteros o flotantes.

## Diseño del Programa

Para este se continuó con lo que ya se contaba desarrollado previamente del primer proyecto. Esto quiere decir que se va a trabajar bajo la utilización de liberías como cup y jflex para la lectura de los archivos de entrada y verificar que estos respeten las reglas gramaticales establecidas. Bajo esta base mientras se están llevando a cabo estos procesos de análisis sintáctico y léxico de forma paralela se da el análisis semántico y la construcción del código tres direcciones ya que en el código generado por la librería cup resultaba mucho más sencillo realizar estos procesos. Se aprovecho la característica de que en analizador sintáctico lleva a cabo un

análisis desde las hojas hacia los no terminales para poder capturar datos vitales (especialmente los literales operandos, sus tipos, identificadores) para esto aprovecharlo en el análisis semántico y la generación del código tres direcciones. Esto se ve evidenciado con el uso del RESULT para enviar datos en el análisis bottom up y el código dentro de las producciones para su uso.

Además de esto, se utilizan objetos creados especialmente para el proyecto. Estos se utilizan para la creación y manejo de la tabla de símbolos, además de ciertas partes de la creación del código tres direcciones.

Cabe destacar que para la tabla de símbolos se utilizó una estructura donde se almacena dentro de la clase objetos de tipo símbolo y con estos se llevan a cabo todas las operaciones, esto a diferencia del enfoque mostrado en clase donde se utilizaba el hash.

## Librerías Utilizadas

java_cup.runtime.* comunicación entre parser y leer.

java.io.* esta es para entrada y salida.

java.util.ArrayList para utilizar listas.

## Análisis de Resultados

| **Requerimiento** | **¿Implementado?** |
|---|---|
| Reconoce la gramática del proyecto 1 | Si |
| Realiza la comprobación semántica de los tipos para asignaciones y reasignaciones. | Si |
| Realiza la comprobación semántica para las operaciones aritméticas, relacionales y | Sí |

| | |
|---|---|
| lógicas (tipado fuerte, ambos operandos del mismo tipo) | |
| Crear la tabla de símbolos | Sí |
| Utilizar la tabla de símbolos para comprobar que una función o identificador se encuentre creado al momento de referenciarlo | Sí |
| Validaciones utilizando la tabla de símbolos (tipos, referencias y que el identificador exista) | Sí |
| Validar la cantidad y tipo de parámetros al invocar una función | Sí |
| Validar que el retorno de una función coincida con su tipo | Sí |
| Validar que las condiciones en estructuras de control sean expresiones de tipo bool | Sí |
| Validar que el tipo de los elementos de un array coincidan con el tipo del arreglo | Sí |
| Informar de los errores semánticos indicando su línea y el detalle del error | Sí |
| Informar si se puede llevar a cabo la creación del código tres direcciones de acuerdo a si no existen errores léxicos, sintácticos o semánticos | Sí |
| Identificar el tipo de dato que se utiliza para mostrarlo en el código de tres direcciones. | Sí |

| | |
|---|---|
| Generar el código de tres direcciones siguiendo los lineamientos del sentido semántico. | Sí |
| Generar un archivo fuente a partir del código intermedio. | Sí |

## Justificación

En el primer Proyecto la parte de la tabla de símbolos que fue creada se realizó de una forma totalmente distinta al enfoque utilizado en este proyecto. En primera instancia se creaba posterior al análisis léxico y sintáctico. Fue necesario cambiar esto debido a que en las clases se nos enseñó que el análisis semántico se llevaba a cabo en conjunto con el léxico y sintáctico, es decir, mientras el “parser” realizaba su trabajo. Por lo que la tabla de símbolos también debía de construirse mientras se llevaba a cabo este proceso. Se utilizaron dos objetos propios para esto donde se recrea una estructura de tabla en la cual se almacenan los símbolos y se llevan a cabo diversas operaciones como por ejemplo la búsqueda de un identificador.

Tal y como ya se mencionó en diseño del programa se aprovechó la parte del análisis bottom up donde se parte desde los terminales hacia los no terminales en las producciones y de esta manera se logró obtener datos importantes tales como el tipo de un literal, su valor, fila, columna, alcance (scope) entre otros datos. Esto se utilizó para llevar a cabo el proceso que consideramos más importante en el análisis semántico: el análisis de las expresiones lógicas, relacionales y aritméticas.

En la gramática utilizada estas tres expresiones van ligadas entre sí pues derivando desde las expresiones lógicas se podían llegar a los literales enteros, flotantes, booleanos, acceso a arreglos y en general cualquier elemento que pudiera ser un operador en estas operaciones. La producción logical_expression_and es la que nos permitía en la gramática tener

expresiones tales como, por ejemplo, let int a = 5$, let int b = arr[5+2] * identificador * miFunc() entre otras posibilidades de combinación. Esta producción junto a los literales char y string eran los valores para la asignación y reasignación. Además, por la mencionada combinación también se utilizaba para las condiciones en estructuras de control.

El enfoque utilizado donde por medio del RESULT se pasaban valores durante el análisis bottom up fue lo que permitió resolver los tipos de estas expresiones. Una vez resuelto esto el resto de las validaciones de este estilo fue más fácil de realizar y de forma general siempre se utilizó el mismo enfoque para resolver tipos.

Además, también se utilizaron diversas funciones externas creadas en el mismo archivo del cup con el objetivo de por ejemplo llevar a cabo validaciones para saber si los operandos de una suma eran del mismo tipo y de lo contrario reportar el error.

El enfoque del código intermedio se basó en lo visto en la demo que realizó el profesor en clase, esto se tomó como base para poder desarrollar el código intermedio por medio de strings los cuales se iban guardando y enviando de manera constante generando así que el CUP realice gran parte del trabajo en las expresiones de forma que el trabajo se realiza practicamente automática, claramente hay que realizar ciertas acciones en los no terminales para que funciones como los append al string builder o enviarlo al RESULT para que así la próxima producción comprenda lo que uno está intentando construir.

Además, creamos otros no terminales para que las producciones se mostraran de forma más bonita y hasta cierto punto correcta, ya que muchas veces los títulos de las variables, funciones o estructuras de control quedaban por debajo de su producción generando así una mayor dificultad en la comprensión del código intermedio, por esto lo que se hacía es una producción como ayudante la cual lo que tenía era una pequeña parte del código distinguible y que no sea repetida en distintas producciones y solo se referenciaba a ella.

Para lo que son validaciones de tipos o datos generalmente utilizaba lo que ya estaba creado en el semántico y sino creaba contadores o por medio de REGEX comprobaba si los temporales o variables eran lo que yo buscaba para validar, de esta forma es que logré hacer la diferenciación entre los enteros y flotantes.

## Bitácora

Link: https://github.com/DylanRodriguez22/Proyecto2_Compiladores