TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
Compiladores e Interpretes(CE1108)

Tarea2: Implementando un Interprete

Investigación y trabajo de campo

Profesor:
Marco Hernandez Vasquez

Estudiantes:
Fabiola Meléndez Sequeira
Jeremy Serracin Oporta
Alisson Redondo Moya
Braulio Retana Murillo

II Semestre 2025

Link a la presentación:
https://www.canva.com/design/DAGzTP0DI88/ikNCxub4nk5ED54AsIWhHg/edit?utm$_content$ = *DAGzTP0DI88utm$_campaign$ = designshareutm$_medium$ = link2utm$_source$ = sharebutton*

TABLA DE CONTENIDOS

1. Investigación

1.1. Analisis Semantico

El proceso de compilación de un programa está compuesto por varias fases que permiten transformar el código fuente escrito en un lenguaje de alto nivel en un programa ejecutable en lenguaje máquina. La fase en la que nos centraremos es el analisis semantico.

El analisis semantico es una fase del compilador que toma como entrada el arbol sintactico(AST) generando por el analisis sintactico y realiza verificaciones de restricciones de tipo, comprobaciones semanticas y anotaciones nesesarias para preparar la generacion de codigo [1]

El analisis semantico tiene objetivos especificos para la funcion que realiza dentro del compilador de los cuales se resaltan los siguientes:

- Resolución de nombres y construcción de tablas de símbolos.
- Comprobación de tipos e inferencia.
- Chequeos semánticos adicionales (uso correcto de variables, control de flujo, conversiones, etc.)
- Preparación y anotación del AST con información para la generación de código.

Cabe destacar que las estruturas de datos utilizadas son las siguientes

- Árbol de Sintaxis Abstracta (AST).
- Tablas de símbolos por ámbito.
- Sistema de tipos y representación de tipos.
- Estructuras para unificación en inferencia de tipos.

En procesamiento de lenguaje natural (PLN),el análisis semántico incluye tareas como análisis léxico, análisis de entidades y análisis contextual [2]. Las cuales se detallan acontinuacion:

- **Análisis Léxico**:

  - Estudio de los significados individuales de las palabras.
  - Relaciones semánticas entre términos.
  - Sinónimos, antónimos y campos semánticos.

- **Análisis de Entidades**:
  - Identificación de personas, lugares y organizaciones.
  - Clasificación de entidades nombradas.
  - Resolución de referencias.

- **Análisis Contextual**:
  - Comprensión del significado según el contexto.
  - Resolución de ambigüedad léxica.
  - Interpretación de expresiones idiomáticas.

En conclusión, el análisis semántico representa un puente entre la forma y el sentido, garantizando tanto la corrección lógica en compiladores como la interpretación adecuada del lenguaje en aplicaciones de PLN.

1.2. Tabla de simbolos

La tabla de simbolos se utiliza casi que en todas las fases del compilador o intérprete ya que estas se apoyan en ella para escribir o para obtener información, las de análisis para insertar y actualizar y las de síntesis para obtener la información con la que generar el código [3]. En el caso del interprete, es la estructura central que soporta el análisis semántico. Sobre esta tabla se realizan operaciones comunes como: insertar, buscar, actualizar y eliminar aunque existen muchas otras operaciones que se pueden realizar sobre la misma.

En relacion con su integración con el proceso de compilación, Se utiliza inicialmente en el análisis sintáctico, a la hora de tomar las producciones relacionadas a las respectivas declaraciones y se van insertando los identificadores a la tabla. Posteriormente se utiliza cuando el arbol sintactico es accedido o creado, ya que cuando uno de los identificadores es utilizado en el arbol, el análisis semántico consulta la tabla para verificar su existencia y obtener su tipo. En terminos más generales, la tabla de simbolos se utiliza como una base de datos para realizar consultas, se pueden utilizar copias tambien pero una vez concluido su fase de montaje que como se mencionó anteriormente se realiza en la etapa del analisis sintatico [3].

2. Implementación y Diseño

Para este trabajo se realizaron ciertas mejoras al interprete, siendo estas las siguientes; logic_ope, este es un atributo el cual al igual que el resto de atributos modifica el orden de ejecucion de las operaciones, la implementacion de logic_ope se puede encontrar en la siguiente imagen

```
logic_ope returns [ASTNode node]:
    t1=expression {$node = $t1.node;}
        (EQ e2=expression {$node = new Equal($node, $e2.node);})*;
```

Figure 2.1: Implementacion logic_ope

en la figura 2.1 se puede obersvar el identificador EQ, esto corresponde a ==, esta es otra de las mejoras realizadas al interprete, la implemtacion de esta mejora se puede ver en la siguiente imagen

```
public class Equal implements ASTNode {
    private ASTNode operand1;
    private ASTNode operand2;

    //constructor to initialize atributes
    public Equal(ASTNode operand1, ASTNode operand2) {
        super();
        this.operand1 = operand1;
        this.operand2 = operand2;
    }



    @Override
    public Object execute(Map<String, Object> symbolTable) {
        if (operand1.execute(symbolTable) instanceof Number && operand2.execute(symbolTable) instanceof Number) {
            return ((Number)operand1.execute(symbolTable)).doubleValue() == ((Number)operand2.execute(symbolTable)).doubleValue();
        }
        return  operand1.execute(symbolTable) == operand2.execute(symbolTable);
    }


}
```

Figure 2.2: Implementacion Equal

Equal se encarga de comparar dos valores y retorna rue en caso de que sean iguales y false si no son iguales.

Otra mejora que realizo fue la adicion del token RAND_FLIP (?), la funcion ligada a este token se puede observar en la siguiente imagen

```java
public Object execute(Map<String, Object> symbolTable) {

    if (percent instanceof Number) {
        double p = ((Number)percent).doubleValue();
        if (p < 0.0 || p > 100.0) {
            throw new RuntimeException("Probability must be between 0 and 100");
        }
        return Math.random() * 100 < p;
    }

    throw new RuntimeException("Invalid type for probability in RandFlip");

}
```

Figure 2.3: Implementacion Rand_Flip

RandFlip se encarga de ”lanzar una moneda” con un porcentaje de acierto asignado, es decir ?50, ”lanzara una moneda” con un porcentaje de acierto del 50%.

La ultima mejora agregada fue la adicion de la operacion de exponenciacion, esta se puede ver reflejada en la siguiente imagen

```java
public Object execute(Map<String, Object> symbolTable) {
    Object base = operand1.execute(symbolTable);
    Object exponent = operand2.execute(symbolTable);

    if (base instanceof Number && exponent instanceof Number) {
        return Math.pow(((Number)base).doubleValue(), ((Number)exponent).doubleValue());
    }

    throw new RuntimeException("Invalid types for power operation");
}
```

Figure 2.4: Implementacion Power

Power se encarga de elevar un numero dado a un segundo numero dado. Para finalizar la siguiente imagen muestra un codigo utilizado para probar el interprete con estas nuevas implementaciones

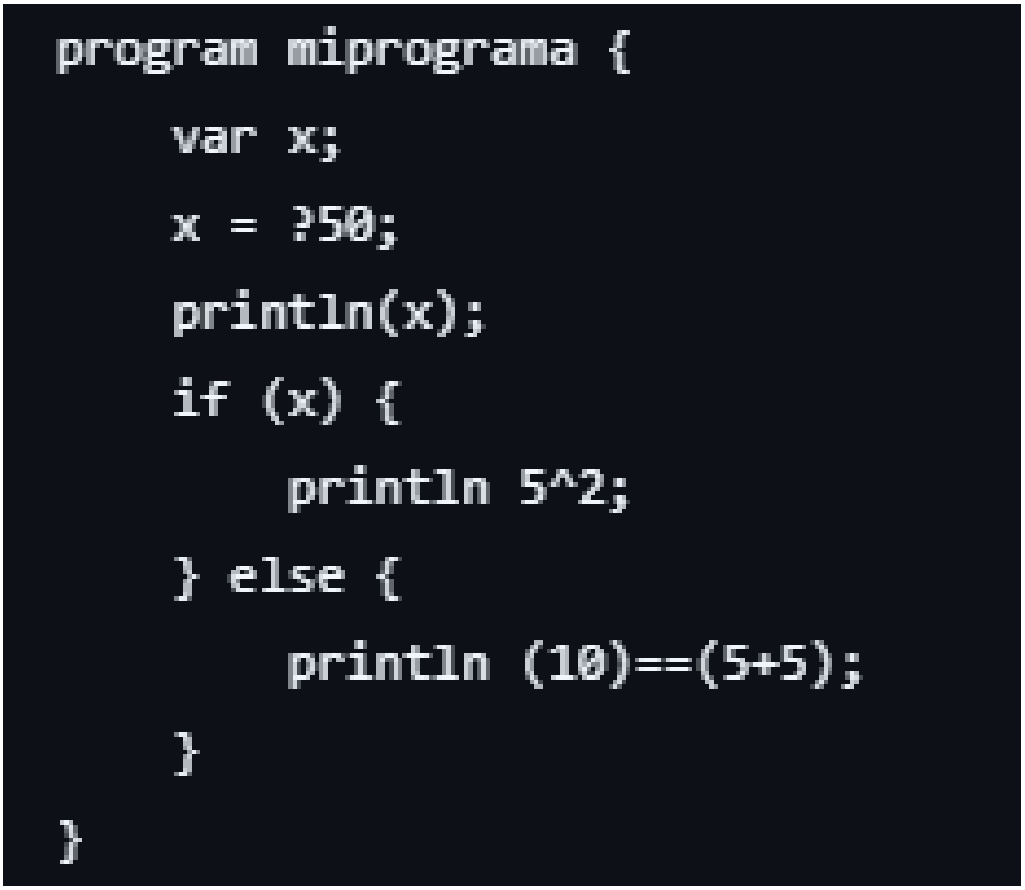

Figure 2.5: Caso de prueba

Este caso de prueba primero declara una variable x mediante **var x** luego asigna a **x** el resultado de realizar **x = ?50**, esto dara true o false dependiendo del caso, luego se imprime el resultado valor de **x** mediane **println(50)** el bloque **if** imprime el resultado de realizar $\mathbf{5}^2$ en caso de que **x** sea verdadero, caso contrario imprime **true**, lo cual corresponde a la ejecucion de **(10) == (5+5)**.

3. Arbol de Parseo

A continuacion, se muestra el arbol de parseo obtenido para el interprete desarrollado en este trabajo,

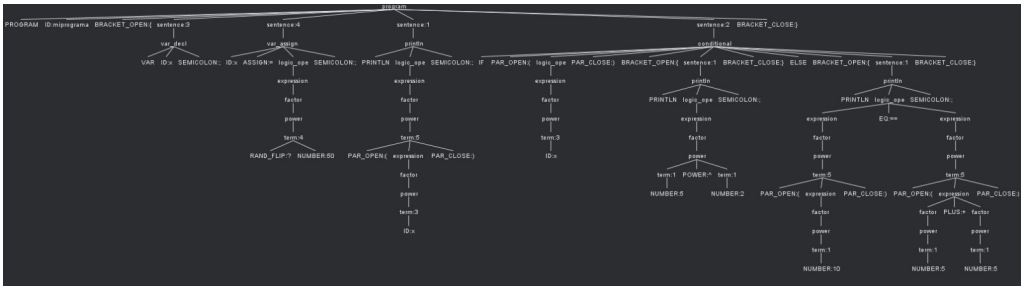

Figure 3.1: Arbol de parseo

References

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd ed. Boston: Pearson, 2007.

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Draft, 2023.

[3] Universidad Europea de Madrid, *Análisis semántico / La tabla de símbolos*, Apuntes en línea. Disponible en: `https://www.cartagena99.com/recursos/alumnos/apuntes/ININF2_M4_U5_T2.pdf`.