Instituto Tecnológico de Costa Rica
Sede Regional San Carlos

Escuela de Computación
Compiladores e intérpretes

Proyecto 2

Responsables:

Crisly González Sánchez
Kevin Zamora Arias

16 de Octubre del 2016
Santa Clara, San Carlos

Introducción

Un compilador es un programa que tiene la funcionalidad de traducir el código fuente escrito en un lenguaje de alto nivel de un sistema en ejecución a lenguaje máquina, para que pueda ser procesado por el ordenador.

Es relevante iniciar por retomar conceptos relevantes para entender ¿Qué es la semántica de un lenguaje de programación? *La semántica es el campo que tiene que ver con el estudio riguroso desde un punto de vista matemático del significado de los lenguajes de programación. Esto se hace evaluando el significado de cadenas sintácticamente legales definidas por un lenguaje de programación específico, mostrando el proceso computacional involucrado*. (Wikipedia, 2016). [2]

El siguiente proyecto comprende el Análisis Contextual(AC), donde se deberán verificar la concordancia semántica de cada símbolo o variable que se utiliza en el código y hacer un chequeo de tipos para que se cumplan, además de hacer un reporte de errores para que el usuario entienda porque el lenguaje no está aceptando la semántica escrita y pueda corregirla.

Haremos uso de la implementación de las reglas de gramática Mini Java, teniendo el Analizador léxico y Scanner podremos hacer el chequeo de semántica y tipos de las expresiones, para luego proceder a la etapa de generación de código.

Análisis del problema

Se deberá realizar un Análisis Contextual de las reglas previamente definidas con la gramática Mini Java y con esto indirectamente se debe trabajar con el chequeo de tipos de datos para que las reglas que rigen el programa se cumplan y se ejecuten correctamente, en el contexto adecuado.

Se deberá valorar estrategias por parte del equipo de desarrollo para crear las estructuras necesarias para almacenar los métodos, variables y clases declaradas en el lenguaje, quedará a discreción del equipo su valoración ya que esto influye en cómo será la generación de código. Es recomendable utilizar tablas para las variables declaradas, además de ingresarles el nombre, tipo , nivel y dirección de memoria si pertenecen a un tipo de clase, esto facilitaría para ubicar su tipo en caso de que sea requerido en una suma, resta u operación que requiera chequear el tipo del identifier. Además el equipo de desarrollo deberá conocer ampliamente las reglas producidas en el primer entregable y la validación de tipos que se deberán realizar.

Los desarrolladores deberán tener una estrategia para conocer los tipos de datos existentes, los cuales son Integer , Float , Array y Clase. En Caso de el tipo de dato Clase deberá verificarse que existe una vez que se desea agregar a la tabla de símbolos.

Deberá realizarse un método Open Scope y Close Scope que permitan conocer las variables dentro de un método únicamente o de la clase, de manera que no puedan ser utilizadas fuera del método en el cual fueron declaradas y deberán borrarse de la tabla una vez que haya terminado la definición del método. Las variables globales deberán permanecer todo el tiempo en la tabla.

Es importante mencionar que las reglas que contengan [“expression”] deberán devolver en su visita un número entero, al igual cuando se le asigne a un **identifier [“expression”]** deberá verificar que ese identifier haya sido declarado como tipo arreglo ejemplo reglas como:

ArrayAssignmentStatement ::= Identifier “[" Expression "]” “=” Expression “;”

Cabe destacar que las variables que se utilizan para hacer procedimientos deberán de tener un puntero a su declaración con el fin de obtener el tipo de dato del identifier. Por lo tanto cuando sea impreso el árbol deberá salir la dirección de memoria de la variable en declaración.

Se deberá tomar decisión sobre los tipos de variables que serán impresos en el print ya que es una gramática parecida al lenguaje Java sin embargo no será igual el manejo de cadenas y números en un print por lo tanto el equipo tendrá que valorarlo de acuerdo a sus criterios para etapas posteriores.

Cuando se extiende una clase, se debe verificar que la clase a extender existe y además sus variables tendrán que ser agregadas a las variables globales de la clase hija, es importante tener claro que la gramática no permite el acceso a métodos de la clase del padre de la herencia, sin embargo si podemos acceder métodos de otras clases ya creadas a través de la regla MessageSend (creación de instancias).

Solución del problema

El equipo de desarrollo hizo uso de la herramienta ANTLR versión 4 para realizar la etapa anterior de Análisis Sintáctico que sirvió para la implementación del Análisis Contextual, cabe destacar los elementos previos con los cuales se debió contar para el desarrollo, los cuales son un archivo de Analizador Lexer donde se incluyen todos Tokens a utilizar en las reglas sintácticas y un archivo Parser que incluye todas las reglas de la gramática MiniJava ambos archivos deben tener relación para la construcción de gramática y su interpretación.

El archivo Scanner(Paser MiniJava) se le han agregado identificadores con un símbolo # en el cual se referencian a las clases que estarán presentes en el árbol y de las cuales haremos el Análisis Contextual(AC), tema principal de dicha etapa.

Se requirió de la creación de tres tablas para almacenar las estructuras manejadas en el desarrollo del AC, de las cuales son las siguientes:

Tabla de símbolos: Fue creada con el objetivo de guardar todos las variables presentes en el archivo de prueba, consideramos que era necesario guardar el nombre, tipo de la variable, nombre de la clase a la cual pertenece y la declaración de la variable.

Además la tabla de simbolos cuenta con un método Open Scope y Close Scope que es utilizado cada vez que debemos abrir(“{“) o cerrar (“ }”) llaves, permitiendo almacenar variables en niveles diferentes, de esta manera se sabrá cuáles variables pueden ser utilizadas en algún momento particular del código fuente.

Además se implementaron métodos de búsqueda para verificación de existencia de la variable antes de ser agregado a la tabla, esto permitiría comprobar que no había sido agregado antes y así evitamos la repetición de elementos.

Y Finalmente un método de imprimir las variables, el cual es manejado de acuerdo a la Constante que se obtiene del símbolo, cabe recalcar que las constantes facilitan la identificación de tipos de datos existentes.

Tabla de Métodos: Tiene una lista de métodos ingresados, los cuales nunca se borran, es decir permanecen en memoria una vez ejecutado el Analizador Contextual, esta tabla de métodos posee Token de métodos, que es un objeto con información sobre: Nombre del método, tipo de dato , Clase a la cual pertenece y declaración.

Se implementó un insertar método que verifica que no haya sido ingresado el método anteriormente y luego lo agrega a la tabla. Consideramos un buscar método en la lista para obtener el Token Método cuando fuera necesario y conocer su contenido, ya que algunas reglas requieren más comprobación sobre tipos de datos de métodos.

Y finalmente un imprimir método que obtiene cada método existente en la lista y los imprime de acuerdo a su tipo y sus especificaciones.

Tabla de clases: Consideramos crear una tabla de clases ya que creemos que es fundamental la modularización e independencia de los datos, en esta Tabla Clases tenemos una lista total de clases agregadas al sistema, con su nombre, declaración.

Para el manejo de tipos de datos utilizamos una Clase Constantes donde cada tipo existente en la gramática está identificado como un número, identificamos las variables: tipo indefinido = 0 , tipo int = 1, tipo String = 2, tipo Boolean es = 3, Tipo Char = 4 , Tipo Class = 5, Arreglo de int = 6, arreglo de String = 7, arreglo de Boolean =8 y Arreglo de Char es =9.

Como antes hemos mencionado existen 3 estructuras de objetos: Objeto Método, Objeto Token y Objeto Clase, utilizamos Orientación a Objetos porque consideramos que son estructuras sencillas y fáciles de utilizar, además que tenemos estructuras de listas propias de la implementación.

Se realizaron chequeos de tipos de datos en todas las reglas de la gramática Mini Java, en las cuales algunas requirieron comprobar si el identificador estaba almacenado en la tabla de símbolos, métodos o clases para comprobar si era viable permitir el funcionamiento de la regla.

Las reglas de arreglos requirieron una comprobación exhaustiva para verificar primeramente que las variables fueran declaradas como tipo de arreglo int, float o boolean y además que los expression [“expression”] fueran valores enteros únicamente.

El método print deberá imprimir únicamente valores enteros, para lo cual la concatenación de dos tipos de datos no está implementado para la etapa actual, ya que no fue considerado necesario.

Finalmente se logró la impresión del árbol correctamente en consola e interfaz donde su única modificación de la etapa anterior era agregar la dirección de memoria de declaración.

Análisis de resultados

Tarea	Estado	Observaciones
Creación de tablas de estructura para Tokens, Clases y Métodos	100%	
Creación de clase Tabla simbolos, métodos, Clases	100%	
Chequeo de tipos de datos en reglas de sintaxis MiniJava	100%	
Creación e impresión de árboles con punteros a memoria en consola e interfaz	100%	
Creación de métodos insertar símbolos, métodos y clases openScope y closeScope	100%	
Creación de métodos buscar elementos en la lista, comprobación antes de una inserción.	100%	
Implementación de Clase y métodos previamente insertados	100%	

Conclusión

El desarrollo del proyecto asignado tuvo como base la comprensión e investigación de la gramática de MiniJava, para realización de comprobación de tipos de datos entre las expresiones aritméticas y asignaciones que lo requirieron.

Se logró comprender el proceso que siguen los lenguajes de programación al realizar un Análisis Contextual con sus reglas gramaticales, además logramos conocer cómo se realiza exhaustivamente el chequeo de tipos en las distintas reglas de la gramaticales.

La etapa de Análisis contextual requirió el desarrollo de lógica sobre el planteamiento de las estructuras a utilizar y además el análisis para justificar que realmente las queremos y así optimizar recursos.

Recomendaciones

- Es recomendable realizar la impresión del árbol en un apartado del editor de texto para una mejor visualización de la estructura de las clases. Además de que dicha impresión sea atractiva y comprensiva a la lectura del usuario.

- El sistema de errores podrá mejorarse a una mayor cobertura de problemas que la clase actual no cubre, de la cual se pueden desarrollar casos de prueba que verifiquen los escenarios posibles de fallo y sean corregidos antes de la creación del árbol.

- Es recomendable comprender la gramática de MiniJava antes de realizar un chequeo de tipos, ya que es necesario saber ejemplificar las reglas para asociarlas a la realidad de una gramática de lenguaje.

Bibliografía

[1] Luis Martinez. (2012). Sintaxis de lenguaje de programación. 15 de Octubre del 2016, de SR Sitio web: http://es.slideshare.net/luismart05/sintaxis-de-lenguaje-de-programacion

[2] Semántica de lenguajes de programación. (s.f.). En *Wikipedia*. Recuperado el 14 de octubre de 2016 de https://es.wikipedia.org/wiki/Sem%C3%A1ntica_de_lenguajes_de_programaci%C3%B3n