Compiladores e interprétes
Computación

Josué Rodríguez Alfaro
Insitituto Tecnológico de Costa Rica
Ingeniería en Computación
Alajuela, Costa Rica

Kevin Rodríguez Murillo
Insitituto Tecnológico de Costa Rica
Ingeniería en Computación
Alajuela, Costa Rica

Luis Salas Rojas
Insitituto Tecnológico de Costa Rica
Ingeniería en Computación
Alajuela, Costa Rica

***Resumen*—En el presente artículo se explica cómo es implementado un compilador para el lenguaje Mini Triángulo. Además, se explican las fases por las que se debe pasar para obtener un programa informático encargado de traducir un programa escrito en un lenguaje de programación a un lenguaje común. Por otra parte, se detalla la forma en que el equipo de trabajo se organizó, las estructuras de datos utilizadas y las reglas empleadas para contar con las etapas del proyecto. El propósito de la elaboración del Compilador es poner en práctica lo aprendido en el curso de Compiladores e Intérpretes del ITCR Sede Alajuela, Costa Rica.**

I. Introducción

Actualmente la computación ha tenido un gran avance, permitiéndonos contar con una gran variedad de herramientas que nos permiten facilitar las tareas diarias, ya sea a nivel personal,educativo o empresarial. Para la elaboración de estas herramientas se necesita de un lenguaje de programación el cual es definido como un lenguaje de programa especial utilizado para desarrollar programas de software, scripts u otros conjuntos de instrucciones para que las computadoras los ejecuten.

Cuando se trata del ambito informatico para que un programa desarrollado en "x" lenguaje se ejecute, necesita ser compilado, en este proceso se verifica si lo escrito por el programador se encuentra bien o no. En el momento que el compilador inicia se abarcan seis fases fundamentales como lo son:

- Analizador Léxico
- Analizador sintáctico
- Analizador semántico
- Generador de código intermedio
- Optimizador de código
- Generador de código

Si el proceso ha sido exitoso al final se puede obtener un código entendible por la máquina, ahora bien cada una de las etapas mencionadas anteriormente implican una serie de procesos para su correcta funcionalidad, se debe de diseñar de una buena forma como se resolverá el problema,debido a que una mala implementación puede ocasionar retrasos y problemas en el desarrollo de las etapas posteriores. Es importante manejar una buena estructura de datos en la que el acceso a los diversos elementos sea un proceso sencillo y óptimo.

A continuación se explica de una forma más detallada el proceso que se llevó a cabo para la creación de un compilador en el lenguaje Python versión 3.6.

II. Análisis Sintáctico

II-A. Scanner

El proceso de análisis léxico es la primera fase del compilador, en este punto se recibe un archivo de texto con el respectivo código fuente, como salida se debe de brindar una lista de objeto de tipo Token.

Para realizar el analisis lexico el equipo de trabajo toma como base el libro de Watt and Brown (2003), en el cual se explica la creación de un compilador. Se cuenta con una clase denominada Scanner en la que existe una lista de palabras reservadas y lista de operadores según las reglas de sintaxis del lenguaje mini triángulo, posteriormente se recorre el archivo linea por linea y dentro de cada línea se analiza caracter por caracter, de esta forma si existe algún símbolo que no pertenece a los operadores y no es ninguna letra o numero se podra detectar el error. Además de clasificarse como palabra reservada u operador , puede existir un identificador.

Una vez identificado el tipo de cada palabra se procede a crear un objeto de tipo Token el cual cuenta con el spelling, el tipo y el numero de linea en donde se encuentra. Luego de ello se agrega a la lista. Cabe destacar que en caso de algún caracter no permitido en el lenguaje mini triángulo, se hace la llamada a una clase la cual recibe una excepción. De esta forma se le hace saber al usuario si hubo algún inconveniente.

En general del analisis lexico emplea un ciclo en que el punto de parada será cuando se recorra cada una de las líneas del archivo de texto, permitiendo que el parser reciba la información necesaria y el proceso resulte más simple, al verificar la estructura según las reglas de producción.

Dicha implementación facilita el acceso a los datos debido a que se cuenta con objetos, por ejemplo si se quisiera obtener

la palabra, el proceso es el siguiente:
-Token.spelling

La organización que presentó el grupo en este punto fue la siguiente:

Se reúnen los miembros del equipo de trabajo con el objetivo de comprender las reglas establecidas con respecto a la sintaxis del lenguaje, luego de entender las reglas, se definen los puntos a realizar. Cómo será el diseño de las clases? Cuáles son las estructuras de datos utilizadas? Cuál será el medio del control de versiones? Cómo será la comunicación entre el equipo?

El trabajo se reparte entre los miembros del equipo, por lo tanto un integrante es el encargado de llevar a cabo el análisis léxico. Los demás miembros le ayudarán en caso de que se presente algún problema, también se cuenta con la guia del libro y la ayuda del profesor. El cual nos puede brindar recomendaciones para una mejor implementación.

Cada una de las versiones del análisis léxico son guardadas en la plataforma gitlab, en la cual se pueden realizar issues para informar sobre las tareas que aún faltan o inconvenientes presentados.

II-B. Parser

El analizador sintáctico (parser) analiza una cadena de símbolos según las reglas de producción establecidas en el lenguaje mini triangulo, las cadenas de símbolos analizadas podran ser:

- AssignCommand
- CallCommand
- SequentialCommand
- IfCommand
- WhileCommand
- LetCommand
- IntegerExpression
- VnameExpression
- UnaryExpression
- BinaryExpression
- SimpleVname
- ConstDecaration
- VarDeclaration
- SequentialDeclaration
- SimpleTypeDenoter

Para el caso del parser se procede a implementar la estructura de datos árbol ast, en el cual se utiliza herencia, realizar este proceso trae consigo una serie de beneficios en etapas que se deban desarrollar luego, debido a que permite una implementación más simple y sencilla.

La entrada del parser corresponde a una lista de objeto Token que cuenta con tres atributos: palabra, tipo y línea en que se encuentra en el archivo de texto.

En este proceso se debe detectar de una forma clara cada uno de los errores con respecto a la estructura de los objetos.

La elaboración del parser implicó el trabajo de los tres integrantes del equipo debido a que requiere un análisis más profundo y detallado, además de que se debe crear un árbol AST, esto implica que se debe retornar una representación abstracta de la sintaxis del código fuente, en donde cada uno de los nodos hace referencia a una construcción del código fuente. Al final de las hojas se encuentran los terminales de una gramática, ya sea operadores, variables, etc.

El proceso llevado a cabo utiliza la recursión, en donde primero se parsea el nodo program, luego de ello el command y posteriormente el single command. En el se abarca cada uno de los command correspondientes. Cada método retorna el nodo respectivo, es importante mencionar que dentro de ese proceso se irá realizando el árbol AST. Existen algunos métodos como lo es el identificar Token que retorna el tipo de acuerdo a la clase Token implementada en el Scanner, lo cual nos ayuda a verificar si la estructura de algunas reglas es la adecuada.

El diseño en general de la clase Parser es basada en el libro de Watt and Brown (2003), en donde se puede observar una implementación para cada una de las reglas de producción las cuales devuelven un nodo, en donde su nombre es definido por las reglas de produccion abstractas.

Como resultado se obtiene un AST, con cada uno de los nodos que abarca el segmento de código establecido en el archivo de texto.

III. Reglas de producción

Las reglas de producción permiten verificar que la estructura con la que está conformada la declaración, expresión o comando sea válida, se deben de aceptar ciertas palabras reservadas o símbolos para la correcta sintaxis. Es importante mencionar que las reglas de producción abstracta son similares a las de sintaxis concreta. Cada una de ellas cuenta con una etiqueta adecuada.

Por ejemplo:
- while b do begin n := 0; b := false end

En este ejemplo el nodo raíz de este AST está etiquetado como WhileCommand, lo que significa que se trata de un comando while. El segundo hijo del nodo raíz se denomina Comando Secuencial, lo que significa que el cuerpo del comando while es un comando secuencial. Finalmente ,los dos hijos del nodo de comando Sequencial se etiquetan como Assigncommand.

Las reglas de producción permiten elaborar el árbol y verificar que la estructura sea correcta.

III-A. Análisis Semánticco

El análisis semántico emplea como entrada el árbol AST generado por el parser el cual está construido por una serie de nodos, en este análisis se comprueban restricciones de tipo o declaración de variables poniendo en práctica el nivel en el que se encuentren según hayan sido declaradas. Para realizar el análisis semántico lo primero que se debe hacer es especificar cuál será la semántica de cada frase en el lenguaje Mini triángulo, lo cual incluye comandos, expresiones y declaraciones.

Con respecto a este punto en un inicio se presentaron problemas con la estructura de nuestro árbol, debido a que consistia en una lista de listas, por ende recorrerlo resultaría difícil, luego de consultar con el profesor y leer el capítulo del libro de Watt and Brown (2003) asignado al análisis semántico, como equipo de trabajo se decide basarse en la implementación explicada en el libro, en donde se detalla cual debe ser el proceso a seguir para poder identificar errores semánticos.

Se cuenta con un método visit en cada uno de los nodos, lo cual permite realizar el recorrido del árbol AST e ir verificando si la estructura de la frase es correcta o no, para ello se debe de evaluar los atributos de cada uno de los nodos, esto incluye verificar que el tipo sea correcto y que en caso de utilizar variables se encuentren inicializadas y declaradas.

Para una mayor claridad a continuación se puede analizar el siguiente ejemplo:
- while n $>$ 0 do C1

La función que cumple el analizador semántico en el proceso anterior es verificar que n se encuentre declarada y que la expresión n $>$ 0.

Para ello se verifica el tipo de n y el tipo de 0, suponiendo que ambos pertenecen al tipo Integer lo que continua es evaluar el operador, para poder concluir si n ¿0 retorna bool o no.Si el ciclo while da como resultado un booleano podrá ser ejecutado correctamente.

La tabla de símbolos es un recurso fundamental requerida para distintas funcionalidades como lo son:

1) Almacenar los nombres de las variables o constantes declaradas al llevar a cabo un let.

2) Verificar la existencia de las variables, cuando se emplea a lo largo del código.

3) Comprobar el tipo con el que se está operando.

4) Determinar el nivel de cada una de las variables o constantes, esto se realiza debido a que existen estructuras anidadas, esto permite que en el código el nombre de una variable se pueda repetir en diversos niveles. Además el scope, restringe el uso de las variables es decir una variable declarada en un nivel 2 , no se puede utilizar en el nivel 0.

En la estructura del AST cada uno de los nodos cuenta con un atributo tipo, el cual es actualizado en el proceso del análisis semántico, debido a que en el visitor los métodos que hereden de Expresión deben ser decorados, así que dependiendo de la expresión que sea así será el tipo.

Como grupo de trabajo se analiza la estructura a implementar, se determina que para poder aprovechar el potencial del lenguaje python , utilizar como tabla de símbolos un diccionario es adecuado debido a que la llave corresponde al nombre del identificador y su alcance, y como valor se obtienen los atributos de acuerdo al nodo.

Es fundamental mencionar que antes de iniciar la primera fase es necesario crear un buen diseño, ya que para el análisis semántico se necesita evaluar cada uno de los atributos independientemente de lo que es y si se tiene una estructura poco legible puede que la implementación de esta fase sea complicada y tediosa.

IV. VISITOR

El proceso seguido para decorar el árbol sintáctico y por ende para hacer el análisis semántico se basó específicamente en el patrón de diseño llamado Visitor. Este patrón fue fundamental para recorrer el AST e ir validando las respectivas reglas de semántica acorde al lenguaje de Mini Triangulo. Al ir recorriendo cada uno de los distintos nodos del árbol, se pudo ir decorando el árbol con los tipos de cada variable, constante o expresión, además de ir insertando los datos necesarios a la tabla de símbolos o diccionario, como por ejemplo la variable, su valor, su tipo y su nivel de scope. Este patrón brinda una gran ayuda en este último aspecto, ya que permite, por medio de la herencia implementada en el AST, ir ordenadamente recorriendo los hijos e ir insertando esta información a la tabla, que más adelante sería útil para analizar semánticamente los distintos componentes. Durante este mismo trayecto se realizó de manera paralela el análisis semántico, donde lo que se hace es una búsqueda en la tabla de símbolos, primero que todo para la validar la existencia del componente o token, en otras palabras, verificar si ha sido declarada / inicializada y luego obtener así el tipo de la constante, variable o expresión que se estaba buscando. Luego de ello se procede a validar si los tipos corresponden a la estructura determinada, también si su nivel de scope corresponde al scope actual en el que se está trabajando. Si al hacer cada una de estas validaciones existe algún componente que no coincida con las reglas de semántica de Mini Triangulo, se procede a imprimir el error correspondiente.

Visitor es un patrón que se acopla muy bien a este tipo de casos no solamente para llevar a cabo el análisis semántico de manera ordenada, sino también para lo que mas adelante seria la generación de código.

V. Generador de código

El proceso de generar código consiste en tomar el código establecido en archivo de texto plano y convertirlo en código con extensión asm el cual pueda ejecutarse de forma correcta en intel. Al contar con el patrón Visit implementado el proceso no resulta difícil, pero si bien es cierto se debe contar con un buen conocimiento en ensamblador para realizar cada una de las plantillas adecuadamente.

Como equipo de trabajo se intentó llevar a cabo el generador de código en su totalidad, el proceso a seguir fue primeramente investigar sobre las instrucciones de asm, luego desarrollar plantillas para cada uno de los distintas expresiones, comandos o declaraciones. En el método visit de cada uno de los nodos luego de validar que la expresión fuese correcta se podría agregar la plantilla con los valores adecuados, tomando en cuenta las diferentes secciones como lo son:

- section bss
- section data
- section text

El método de salir sería implementado en cada uno de los casos, debido a que en ensamblador para evitar la violación de segmento se debe de actualizar el valor del registro eax y luego una interrupción al sistema.

Conclusiones / Resultados

Los resultados y conclusiones obtenidas luego de trabajar en la creación del compilador son:

El Scanner realiza la función adecuadamente, debido a que lee el contenido en el archivo de texto plano y verifica si existe algún símbolo incorrecto, además permite retornar una lista de objeto tipo Token, el cual cuenta con tres atributos: palabra, tipo y línea en que se encuentra, esto por razón de que es recomendable para el usuario mostrar los errores de una forma que no cueste encontrarlos.

El Parser recibe como entrada la lista de objetos de tipo Token y se encarga de verificar si la estructura en que está compuesta la declaración, expresión o comando es correcta de acuerdo a las reglas de producción establecidas para el lenguaje mini triángulo.

El analizador semántico se encarga de verificar los tipos de cada una de las constantes o declaraciones utilizadas, también se valida si se está operando con tipos distintos, si la variable no se encuentra declarada o si el nivel de scope no es correcto.Para realizar algunas de las validaciones mencionadas anteriormente el árbol AST es decorado en caso de ser expresión mediante el patrón Visit en cada uno de los nodos.

Se agregan estructuras extra como lo es el print (d) y el for, las estructura son:

- print(identificador)
- for i in lim 5 en donde i corresponde a un integer

El generador de código no fue completado en su totalidad, las estructuras realizadas corresponden al for, let y print, cabe destacar que el código que se genera puede ser compilado de forma adecuada. Este código es almacenado en un archivo que se genera luego de ejecutar el compilador el nombre es generador.asm

VI. Recomendaciones

1) Sacar el tiempo necesario para realizar un buen diseño de como va ser desarrollado el compilador, cuales van a ser las clases a implementar y la función de cada una de ellas.

2) Determinar las estructuras de datos a utilizar.

3) En caso de alguna duda, consultar al profesor o a estudiantes con un mayor conocimiento en este ámbito para evitar atrasos de tiempo o funciones incompletas.

4) Investigar lo suficiente sobre el lenguaje en el que se esté llevando a cabo el desarrollo del compilador, debido a que pueden existir funcionalidades que nos faciliten el proceso.

5) Utilizar lo aprendido en cursos anteriores de la carrera en computación.

Referencias

[1] Watt, D. and Brown, D. (2003). Programming language processors in Java. Delhi, India: Pearson Education (Singapore) Pte. Ltd.