Unidad I: Análisis semántico

1.1 Arboles de expresiones

Los árboles de expresiones representan el código de nivel del lenguaje en forma de datos. Los datos se almacenan en una estructura con forma de árbol. Cada nodo del árbol de expresión representa una expresión, por ejemplo, una llamada al método o una operación binaria, como x < y.

Un árbol de expresión sirve para evaluar expresiones del tipo: **(a + b)*c/d**

Para que un árbol represente una expresión se deben tomar en cuenta 2 características muy importantes:

- Cualquier hoja está etiquetada sólo con un operando.
- Cualquier nodo interior *n* está etiquetado por un operador.

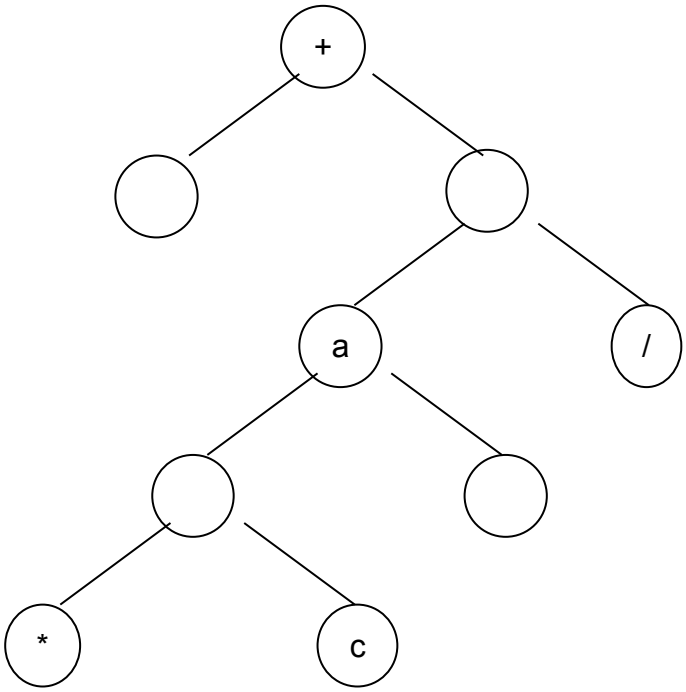

Imagen 1. Árbol de expresiones

Al introducir la expresión debemos de tomar en cuenta las siguientes características:

- La raíz siempre debe ser un operador
- Las hojas siempre deben ser operandos
- Los nodos deben estar etiquetados por operadores
- Si un operador tiene mayor prioridad que la raíz se coloca como hijo.
- Si un operador tiene igual o menor prioridad que un nodo se coloca como padre.
- Un nodo puede contener como hijo otro subárbol que contiene un pequeña expresión.

En los árboles de expresión, la sucesión del preorden de etiquetas nos da lo que se conoce como la forma prefijo de una expresión

Análogamente, la sucesión postorden de las etiquetas de un árbol expresión nos da lo que se conoce como la representación postfijo de una expresión

Finalmente, el inorden de una expresión en un árbol de expresión nos da la expresión infijo en sí misma, pero sin paréntesis

Construcción de un árbol de expresión

Algoritmo

- Mientras carácter diferente de nulo
- Leer carácter de la lista
- Si es paréntesis pasar al siguiente carácter
- Crear un nodo nuevo que contenga ese carácter

Operando

- Si el árbol está vacío hacer raíz a nuevo, si no recorrer el árbol por la derecha hasta llegar a un nodo con hojas, si la hoja izquierda, no está etiquetada colocar operando, si no colocarlo en la hoja derecha.

Operador

- Si la raíz es un operando, insertar nuevo en ese nodo, y convertir el operando en el hijo izquierdo, si no si hay un paréntesis abierto insertar nuevo en la última hoja derecha y colocar operando como hijo izquierdo.
- Si el carácter anterior es paréntesis izquierdo si el siguiente carácter es paréntesis derecho si solo hay un operador en el árbol nuevo se convierte en raíz, si no se inserta en el último nodo derecho, y el nodo se convierte en hijo izquierdo.
- Si no se cumple ninguna de las condiciones anteriores si la raíz es de igual prioridad o menor prioridad convertir la raíz en el hijo izq. de nuevo si no la prioridad del nodo raíz es mayor al de nuevo insertar nuevo como hijo derecho y colocar el nodo reemplazado como hijo izquierdo.

1.2 Acciones semánticas de un analizador sintáctico

Definición de un analizador sintáctico: es la fase del analizador que se encarga de chequear el texto de entrada en base a una gramática dada. Y en caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce.

En teoría, se supone que la salida del analizador sintáctico es alguna representación del árbol sintáctico que reconoce la secuencia de Token suministrada por el analizador léxico.

En la práctica, el analizador sintáctico también hace:

- Acceder a la tabla de símbolos (para hacer parte del trabajo del analizador semántico).
- Chequeo de tipos (del analizador semántico).
- Generar código intermedio.
- Generar errores cuando se producen.
- En definitiva, realiza casi todas las operaciones de la compilación. Este método de trabajo da lugar a los métodos de compilación dirigidos por sintaxis.

Manejo de errores sintácticos

Los errores sintácticos son dados por una expresión aritmética o paréntesis no equilibrados.

El manejo de errores de sintaxis es el más complicado desde el punto de vista de la creación de compiladores. Nos interesa que cuando el compilador encuentre un error, se recupere y siga buscando errores. Por lo tanto el manejador de errores de un analizador sintáctico tiene como objetivos:

- Indicar los errores de forma clara y precisa. Aclarar el tipo de error y su localización.
- Recuperarse del error, para poder seguir examinando la entrada.
- No ralentizar significativamente la compilación

Tipo de gramática que acepta un analizador sintáctico

Nosotros nos centraremos en el análisis sintáctico para lenguajes basados en gramáticas formales, ya que de otra forma se hace muy difícil la comprensión del compilador, y se pueden corregir, quizás más fácilmente, errores de muy difícil localización, como es la ambigüedad en el reconocimiento de ciertas sentencias.

La gramática que acepta el analizador sintáctico es una gramática de contexto libre:

Gramática: G (N, T, P, S)

N = No terminales.
T = Terminales.
P = Reglas de Producción.
S = Axioma Inicial.

1.3 Comprobaciones de tipos en expresiones

La labor de comprobación de tipos consiste en conferir a las construcciones sintácticas del lenguaje la semántica de tipificación y en realizar todo tipo de comprobaciones de dicha índole. Por su naturaleza, sin embargo, ésta se encuentra repartida entre la fase de análisis semántico y la generación de código intermedio.

- *Comprobaciones estáticas*

Las comprobaciones estáticas recogen el compendio de todas aquellas tareas de carácter semántico que, por su naturaleza, pueden ser realizadas directamente durante la fase de compilación mediante el uso de los artefactos y mecanismos propios de dicha fase. Este tipo de comprobaciones son beneficiosas puesto que confieren seguridad a la ejecución del programa.

Características

-Diferente de la dinámica en runtime.
-Ejemplo: comprobación de tipos, flujo de control, unicidad.

- *Comprobaciones dinámicas*

Las comprobaciones dinámicas son aquellas que no se realizan durante la fase de compilación y se delegan al momento de la ejecución del programa. Ello requiere generar código ejecutable específicamente diseñado para realizar tales comprobaciones. Los lenguajes con una carga excesiva de comprobaciones dinámicas generan programas más largos, lentos e inseguros en ejecución.

Verificación de tipos

Comprueba la compatibilidad de tipos de todas las expresiones del código fuente recuperando la información durante la gestión de declaraciones. Además se asegura de que no existe en el programa ninguna referencia a ningún símbolo no declarado.

Inferencia de tipos

En lenguajes sin tipificación de variables o con sobrecarga se aplican tareas de inferencia de tipos en el nivel gramatical de las expresiones para resolver el tipo de datos de la expresión resultante en función del contexto de evaluación.

1.4 Pila semántica en un analizador sintáctico

Las pilas y colas son estructuras de datos que se utilizan generalmente para simplificar ciertas operaciones de programación. Estas estructuras pueden implementarse mediante arrays o listas enlazadas.

Pila: colección de datos a los cuales se les puede acceder mediante un extremo, que se conoce generalmente como tope. Las pilas tienen dos operaciones básicas:

- **Push** (para introducir un elemento)
- **Pop** (para extraer un elemento)

Sus características fundamentales es que al extraer se obtiene siempre el último elemento que acabe de insertarse. Por esta razón también se conoce como estructuras de datos LIFO, una posible implementación mediante listas enlazadas seria insertando y extrayendo siempre por el principio de la lista.

Las pilas se utilizan en muchas aplicaciones que utilizamos con frecuencia. Las pilas y colas son estructuras de datos que se utilizan generalmente para simplificar ciertas operaciones de programación. Estas estructuras pueden implementarse mediante arrays o listas enlazadas.

Un ***analizador sintáctico*** es un autómata de pila que reconoce la estructura de una cadena de componentes léxicos.

En general, el analizador sintáctico inicializa el compilador y para cada símbolo de entrada llama al analizador morfológico y proporciona el siguiente símbolo de entrada.

Al decir pila semántica no se refiere a que hay varios tipos de pila, hace referencia a que se debe programar única y exclusivamente en un solo lenguaje, es decir, no podemos mezclar código de C++ con Visual Basic.

Ventajas

- Los problemas de integración entre los subsistemas son sumamente costosos y muchos de ellos no se solucionan hasta que la programación alcanza la fecha límite para la integración total del sistema.
- Se necesita una memoria auxiliar que nos permita guardar los datos para poder hacer la comparación.

Objetivo teórico

Es construir un árbol de análisis sintáctico, este raramente se construye como tal, sino que las rutinas semánticas integradas van generando el árbol de Sintaxis abstracta. Se especifica mediante una gramática libre de contexto.

El análisis semántico detecta la validez semántica de las sentencias aceptadas por el analizador sintáctico. El analizador semántico suele trabajar simultáneamente al analizador sintáctico y en estrecha cooperación. Se entiende por semántica como el conjunto de reglas que especifican el significado de cualquier sentencia sintácticamente correcta y escrita en un determinado lenguaje.

Las rutinas semánticas deben realizar la evaluación de los atributos de las gramáticas siguiendo las reglas semánticas asociadas a cada producción de la gramática.

El análisis sintáctico es la fase en la que se trata de determinar el tipo de los resultados intermedios, comprobar que los argumentos que tiene un operador pertenecen al conjunto de los operadores posibles, y si son compatibles entre sí, etc.

En definitiva, comprobará que el significado de la que se va leyendo es válido. La salida teórica de la fase de análisis semántico sería un árbol semántico. Consiste en un árbol sintáctico en el que cada una de sus ramas ha adquirido el significado que debe tener.

Se compone de un conjunto de rutinas independientes, llamadas por los analizadores morfológico y sintáctico. El análisis semántico utiliza como entrada el árbol sintáctico detectado por el análisis sintáctico para comprobar restricciones de tipo y otras limitaciones semánticas y preparar la generación de código.

Las rutinas semánticas suelen hacer uso de una pila que contiene la información semántica asociada a los operadores en forma de registros semánticos.

Reglas semánticas

Son el conjunto de normas y especificaciones que definen al lenguaje de programación y están dadas por la sintaxis del lenguaje, las reglas semánticas asignan un significado lógico a ciertas expresiones definidas en la sintaxis del lenguaje.

La evaluación de las reglas semánticas define los valores de los atributos en los nodos del árbol de análisis sintáctico para la cadena de entrada. Una regla semántica también puede tener efectos colaterales, por ejemplo, imprimir un valor o actualizar una variable global.

Compatibilidad de tipos

Durante la fase de análisis semántico, el compilador debe verificar que los tipos y valores asociados a los objetos de un programa se utilizan de acuerdo con la especificación del lenguaje.

Además debe detectar conversiones implícitas de tipos para efectuarlas o insertar el código apropiado para efectuarlas así como almacenar información relativa a los tipos de los objetos y aplicar las reglas de verificación de tipos.

Analizadores descendentes:

Parten del axioma inicial de la gramática, se va descendiendo utilizando las derivaciones izquierdas, hasta llegar a construir la cadena analizada.

Se va construyendo el árbol desde sus nodos terminales. Es decir, se construye desde los símbolos de cadena hasta llegar al axioma de la gramática.

Bottom up

Es un principio de muchos años del estilo de programación que los elementos funcionales de un programa no deben ser demasiado grandes. Si un cierto componente de un programa crece más allá de la etapa donde está fácilmente comprensible, se convierte en una masa de la complejidad que encubre errores tan fácilmente como una ciudad grande encubre a fugitivos.

Top-down

Este método consiste en dividir los problemas en subproblemas más sencillos para conseguir una solución más rápida. El diseño descendente es un método para resolver el problema que posteriormente se traducirá a un lenguaje compresible por la computadora.

Un **parser** ascendente utiliza durante el análisis una pila. En esta va guardando datos que le permiten ir haciendo las operaciones de reducción que necesita.

Para incorporar acciones semánticas como lo es construir el árbol sintáctico, es necesario incorporar a la pila del parser otra columna que guarde los atributos de los símbolos que se van analizando. Estos atributos estarían ligados a la correspondiente producción en la tabla de parsing.

La ***pila*** juega un papel fundamental en el desarrollo de cualquier analizador semántico. Dentro de cada elemento de la pila se guardan los valores que pueden tener una expresión.

1.5 Esquema de traducción

Un esquema de traducción es una gramática independiente de contexto en la que se asocian atributos con los símbolos gramaticales y se insertan acciones semánticas encerradas entre llaves { } dentro de los lados derechos de las producciones. Los esquemas de traducción pueden tener tantos atributos sintetizados como heredados.

Cuando se diseña un esquema de traducción, se deben respetar algunas limitaciones para asegurarse de que el valor de un atributo esté disponible cuando una acción se refiera a él. Estas limitaciones, motivadas por las definiciones con atributos por la izquierda, garantizan que las acciones no hagan referencia a un atributo que aún no haya sido calculado. El ejemplo más sencillo ocurre cuando sólo se necesitan atributos sintetizados, en este caso, se puede construir el esquema de traducción creando una acción que conste de una asignación para cada regla semántica y colocando esta acción al final del lado derecho de la producción asociada.

Traducción descendente

Se trabaja con esquema de traducción en lugar de hacerlo con definiciones dirigidas por sintaxis, así que se puede ser explícito en cuanto al orden en que tienen que lugar las acciones y las evaluaciones de los atributos.

Eliminacion de la recursividad izquierda de un esquema de traducción

Como la mayoría de los operadores aritméticos son asociativos por la izquierda, es natural utilizar gramáticas recursivas por la izquierda para las expresiones. La transformación se aplica a esquemas de traducción con atributos sintetizados.

Para el análisis sintáctico descendente, se supone que una acción se ejecuta en el mismo momento en que se expandiría un símbolo en la misma posición. Un atributo heredado de un símbolo debe ser calculado por una acción que aparezca antes que el símbolo, y un atributo sintetizado del no terminal de la izquierda se

debe calcular después de que hayan sido calculados todos los atributos de los que depende.

Un atributo heredado de un símbolo debe ser calculado por una acción que aparezca antes que el símbolo, y un atributo sintetizado del no terminal de la izquierda se debe calcular después de que hayan sido calculados todos los atributos de los que depende.

Los fragmentos de código así insertados se denominan acciones semánticas. Dichos fragmentos actúan, calculan y modifican los atributos asociados con los nodos del árbol sintáctico. El orden en que se evalúan los fragmentos es el de un recorrido primero-profundo del árbol de análisis sintáctico.

Obsérvese que, en general, para poder aplicar un esquema de traducción hay que construir el árbol sintáctico y después aplicar las acciones empotradas en las reglas en el orden de recorrido primero-profundo. Por supuesto, si la gramática es ambigua una frase podría tener dos árboles y la ejecución de las acciones para ellos podría dar lugar a diferentes resultados. Si se quiere evitar la multiplicidad de resultados (interpretaciones semánticas) es necesario precisar de qué árbol sintáctico concreto se está hablando.

1.6 Generación de la tabla de símbolo y de direcciones

Las tablas de símbolos (también llamadas tablas de identificadores y tablas de nombres), realizan dos importantes funciones en el proceso de traducción: verificar que la semántica sea correcta y ayudar en la generación apropiada de código. Ambas funciones se realizan insertando o recuperando desde la tabla de símbolos los atributos de las variables usadas en el programa fuente. Estos atributos, tales como: el nombre, tipo, dirección de almacenamiento y dimensión de una variable, usualmente se encuentran explícitamente en las declaraciones o más implícitamente a través del contexto en que aparecen los nombres de variables en el programa.

Una de las estructuras de datos que se encuentran relacionadas con las fases del proceso de compilación es la tabla de símbolos, la cual tiene como propósito registrar información que se comparte entre varias etapas y que permite administrar los recursos asociados a las entidades que manipulará el programa. La tabla de símbolos tiene típicamente la siguiente estructura:

Una tabla de símbolos puede conceptualizarse como una serie de renglones, cada uno de los cuales contiene una lista de valores de atributos que son asociados con una variable en particular. Las clases de los atributos que aparecen en una tabla de símbolos dependen en algún grado de la naturaleza del lenguaje de programación para el cual se escribe el compilador.

Por ejemplo, un lenguaje puede ser sin tipos, y por lo tanto el atributo tipo no necesita aparecer en la tabla. Similarmente, la organización de la tabla de símbolos variará dependiendo de las limitaciones de memoria y tiempo de acceso.

1.7 Manejo de errores semánticos

Es una de las misiones más importantes de un compilador, aunque, al mismo tiempo, es lo que más dificulta su realización. A veces unos errores ocultan otros.

A veces un error provoca una avalancha de muchos errores que se solucionan con el primero.

Es conveniente un buen manejo de errores, y que el compilador detecte todos los errores que tiene el programa y no se pare en el primero que encuentre. Hay, pues, dos criterios a seguir a la hora de manejar errores:

- Pararse al detectar el primer error.
- Detectar todos los errores de una pasada.

El análisis semántico es posterior al sintáctico y mucho más difícil de formalizar que éste. Se trata de determinar el tipo de los resultados intermedios, comprobar que los argumentos que tiene un operador pertenecen al conjunto de los operadores posibles, y si son compatibles entre sí, etc. En definitiva, comprobará que el significado de lo que se va leyendo es válido.

La salida "teórica" de la fase de análisis semántico sería un árbol semántico. Consiste en un árbol sintáctico en el que cada una de sus ramas ha adquirido el significado que debe tener. En el caso de los operadores polimórficos (un único símbolo con varios significados), el análisis semántico determina cuál es el aplicable.