Unidad II: Análisis semántico

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.
En compiladores de un solo paso, las llamadas a las rutinas semánticas se realizan directamente desde el analizador sintáctico y son dichas rutinas las que llaman al generador de código. El instrumento más utilizado para conseguirlo es la gramática de atributos.

2.1 Notaciones

Las notaciones son una forma especial en la que se pueden expresar una expresión matemática y puedan ser de 3 formas: infija, prefija y posfija. Los prefijos, Pre - Pos - In se refieren a la posición relativa del operador con respecto a los dos operandos.

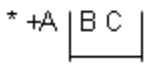
2.1.1 Prefija

La expresión o notación prefija nos indica que el operador va antes de los

Operandos sus características principales son:

- Los operadores conservan el mismo orden que la notación infija equivalente.
- No requiere de paréntesis para indicar el orden de precedencia de operadores ya que él es una operación.
- Se evalúa de izquierda a derecha hasta que encuentra el primer operador seguido inmediatamente de un par de operando.

- Se evalúa la expresión binaria y el resultado se cambia como un nuevo operando. Se repite hasta que nos quede un solo resultado.
- El orden es operador, primer operando, segundo operando.



(A+B)*C

2.1.2 Infija

La expresión o notación infija es la forma más común que utilizamos para escribir expresiones matemáticas, estas notaciones se refiere a que el operador esta entre los operadores. La notación infija puede estar completamente parentizada o puede basarse en un esquema de precedencia de operadores así como el uso de paréntesis para invalidar los arreglos al expresar el orden de evaluación de una expresión:

3*4 = 12
3*4+ = 14
3*(4+2) = 18

La notación infija tiene el problema de que en expresiones con más de un operador existe ambigüedad sobre cuál es el orden de evaluación. Por ejemplo, la expresión 8/4/2 se puede interpretar como (8/4)/2 o bien 8/(4/2). Las otras notaciones no sufren este problema.

La notación habitual. El orden es primer operando, operador, segundo operando.

2.1.3 Postfija

- Como su nombre lo indica se refiere a que el operador ocupa la posición después de los operandos sus características principales son:
- El orden de los operandos se conserva igual que la expresión infija equivalente no utiliza paréntesis ya que no es una operación ambigua.

- La operación posfija no es exactamente lo inverso a la operación prefija equivalente.
- El orden es primer operando, segundo operando, operando.

  (A+B)*C AB+C*

Ejemplo:

Si deseamos representar las expresiones (2+(3*4)) = xy ((2+3)*4) = x en las tres notaciones mencionadas, el resultado sería:

(2+(3*4)) = x

((2+3)*4) = x

Notación postfija

2 3 4 * + x =
2 3 + 4 * x =

2.2 Representaciones de código intermedio

2.2.1 Notación Polaca

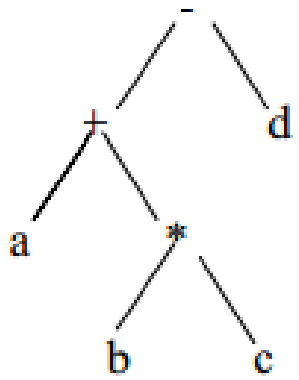
La notación polaca es la originada por un Autómata con pila, en la que los operadores siempre preceden a los operandos sobre los que actúan, y que tiene la ventaja de no necesitar paréntesis:

- Se utiliza principalmente para la representación de expresiones aritméticas.
- Expresión a notación polaca inversa.

Algoritmo

- ✓ Representa la expresión en forma de árbol sintáctico.
- ✓ Recorrer el árbol en postorden

Ejemplo: a + b * c-d



Código a b c * + d-

Ventajas y desventajas de la notación polaca

- *Generación de código:* simple, no utiliza registros.
- *Optimización:* es difícil de reordenar ya que hay que considerar el contenido de la pila.
- *Interpretación rápida:* es muy fácil de interpretar ya que solo necesita una pila.
- *Transportable:* si, ya que todos los procesadores implementan una pila.

2.2.2 Código P

El código P comenzó como un código ensamblador objetivo estándar producido por varios compiladores Pascal en la década de 1970 y principios de la de 1980. Fue diseñado para código real para una máquina de pila hipotética la idea era hacer que los compiladores de Pascal se transportaran fácilmente requiriendo solo que se volviera a escribir el intérprete de la maquina P para una plataforma, el código P también a probado ser útil como código intermedio y sean utilizado varias extensiones y modificaciones del mismo en diverso compiladores de código nativo, la mayor parte para lenguaje tipo Pascal.

Como el código P fue diseñado para ser directamente ejecutable, contiene una descripción implícita de un ambiente de ejecución particular que incluye tamaños de datos, además de mucha información específica para la maquina P, que debe conocer si se desea que un programa de código P se comprensible. La máquina P está compuesta por una memoria de código, una memoria de datos no específica para variables nombre das y una pila para datos temporales, junto como cualquiera registro que sea necesario para mantener la pila y apoyar la ejecución.

2.2.3 Triplos

• <Operador>, <operando1>, <operando2>

• El resultado se asocia al número de tripleta

Ejemplo: W * X + (Y + Z)

1. *, W, X

2. +, Y, Z

3. +, (1), (2)

Control de flujo:

IF X>Y THEN Z=X ELSE Z=Y+1

1. >, X, Y

2. Saltar si falso, (1), 5

3. =, Z, X

4. Saltar,, 7

5. +, Y, 1

6. =, Z, (5)

Problema

La optimización supone mover tripletas y hay que recalcular las referencias.

2.2.4 Cuádruplos

• <Operación>, <operando1>, <operando2>, <resultado>

Ejemplo:

(A+B)*(C+D)-E

+, A, B, T1

+, C, D, T2

*, T1, T2, T3

-, T3, E, T4

Las cuádruplas facilitan la aplicación de muchas optimizaciones, pero hay que tener un algoritmo para la reutilización de las variables temporales (reutilización de registros del procesador).

2.3 Esquema de generación

Los esquemas de generación son las estrategias o acciones que se deberán realizarse y tomarse en cuenta en el momento de generar código intermedio.

Los esquemas de generación dependen de cada lenguaje. Tomaremos algunos esquemas de generación del lenguaje C.

2.3.1 Variables y constantes

Las variables y constantes deben separarse de tal manera que queden las expresiones una por una de manera simple.

Por ejemplo int a,b,c; se descompone a int a; int b; intc; respectivamente.

2.3.2 Expresiones

En esta función recibe una cadena que representa una línea de código intermedio y toma las medidas oportunas para que ese código se utilice. Estas medidas pueden ser escribir la línea en un fichero adecuado, almacenar la instrucción en una lista que después se pasará a otros módulos, o cualquier otra que necesitemos en nuestro compilador.

Expresiones aritméticas

Son aquella donde los operadores que intervienen en ella son numéricos, el resultado es un número y los operadores son aritméticos. Los operadores aritméticos más comúnmente utilizados son: +, - , * , / y %.

Comenzamos el estudio por las expresiones aritméticas. Lo que tendremos que hacer es crear por cada tipo de nodo un método que genere el código para calcular la expresión y lo emita. Ese código dejará el resultado en un registro, cuyo nombre devolverá el método como resultado.

Para reservar estos registros temporales, utilizaremos una función, reserva. En principio bastar ‘a con que esta función devuelva un registro distinto cada vez que se la llame.

Cada nodo generará el código de la siguiente manera:

- Por cada uno de sus operandos, llamara al método correspondiente para que se evalúe la sub expresión. Si es necesario, reservara un registro para guardar su resultado.
- Emitirá las instrucciones necesarias para realizar el cálculo a partir de los operandos.

2.3.3 Instrucciones de asignación

La sintaxis general de la instrucción de asignación es:

nombre_de_la_variable = valor

El valor a la derecha del signo igual puede ser una constante, otra variable o una expresión que combine constantes y variables, pero siempre la variable y su valor deben ser del mismo tipo de dato.

Ejemplos:

edad% = 5

area! = 12.3

nombre$ = “Pedro”

- *Instrucciones de asignación compuesta*

Las *instrucciones de asignación compuesta* realizan primero una operación en una expresión antes de asignarla a un elemento de programación. En el siguiente ejemplo se muestra uno de estos operadores, **+=**, que incrementa

el valor de la variable del lado izquierdo del operador con el valor de la expresión de la derecha.

Una instrucción de asignación asigna el valor de una expresión a una variable. En general, si la variable que se va a asignar es una propiedad, la propiedad debe ser de lectura y escritura o de sólo escritura; en caso contrario, se produce un error de compilación. Si la variable es una variable de sólo lectura, la asignación debe producirse en un constructor Shared o un constructor de instancia apropiado para el tipo de la variable; en caso contrario, se producirá un error de compilación.

2.3.4 Instrucciones de control

Esta forma de programación sólo permite resolver problemas sencillos. Para resolver problemas más complejos, nos puede interesar que dependiendo de los valores de los datos, se ejecuten unas instrucciones u otras.

Las instrucciones condicionales nos van a permitir representar éste tipo de comportamiento. Sentencias IF y SWITCH. En otros casos, nos encontraremos con la necesidad de repetir una instrucción o instrucciones un número determinado de veces. En éstos casos utilizaremos instrucciones de control iterativas o repetitivas (ciclos). Sentencias WHILE, DO-WHILE y FOR.

2.3.5 Funciones

Las funciones pueden reducir a en línea, lo que se hace que expandir el código original de la función.

Las funciones se descomponen simplificando los parámetros de manera individual al igual que el valor de retorno.