Instituto Tecnológico Superior Zacatecas Norte

División de Informática
Carrera de Ingeniería en Sistemas Computacionales

Lenguajes y Autómatas II

Tema: Propuesta de Análisis semántico U2

Alumno(s): Eduardo García Delgado
Cinthia Griselda Almaraz Sierra

N.C. : 15010199
15010103

ÍNDICE

2.1 Notaciones.

Las notaciones sirven de base para expresar sentencias bien definidas. El uso más extendido de las notaciones sirve para expresar operaciones aritméticas.
Las expresiones aritméticas se pueden expresar de tres formas distintas: infija, prefija y postfija.
Los prefijos, Pre - Post - In se refieren a la posición relativa del operador con respecto a los dos operandos.

2.1.1. Prefija.

La notación prefija pone el operador primero que los dos operandos, por lo que la expresión anterior queda: +ab-5. Esto se representa con una estructura del tipo FIFO (First In First Out) o cola.

Las estructuras FIFO son ampliamente utilizadas pero tienen problemas con el anidamiento aritmético.

2.1.2. Infija.

La notación infija es la más utilizada por los humanos por que es la más comprensible ya que ponen el operador entre los dos operandos. Por ejemplo a+b-5.

No existe una estructura simple para representar este tipo de notación en la computadora por esta razón se utilizan otras notaciones.

2.1.3. Postfija.

La notación postfija pone el operador al final de los dos operandos, por lo que la expresión queda: ab+5-

La notación postfija utiliza una estructura del tipo LIFO (Last In First Out) pila, la cual es la más utilizada para la implementación.

2.2 Representaciones de código intermedio.

Existen maneras formales para representar código intermedio.
Estas notaciones simplifican la traducción de nuestro código fuente a nuestro código objeto ya que ahorran y acotan símbolos de la tabla de símbolos.

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

2.2.2. Código P.

El código P hace referencia a máquinas que utilizan o se auxilian de pilas para generar código objeto.

En muchos caso la P se asocia a código portable el cual garantiza que el código compilado en una máquina se pueda ejecutar en otras.
Para garantizar la portabilidad del código se necesita que el lenguaje está estandarizado por algún instituto y que dicho código no tenga extensiones particulares. También se recomienda la no utilización de características especiales exclusivas de alguna arquitectura de computadoras en particular.

2.2.3. Triplos.

La notación de tres direcciones es una forma abstracta de código intermedio. Esta notación se puede implementar como registros con campos para el operador y operadores.

Con una estructura de tres campos se pueden omitir los valores temporales, dicha estructura recibe el nombre de triples y tiene los siguientes campos: op, arg1 y arg2.

Generalmente el código que generan los triples recibe el nombre de código de dos direcciones, aunque en ocasiones puede variar. Cuando se utilizan triples se ocupan punteros a la misma estructura de los triples.
* b (0) //triple

Se debe tener en cuenta el proceso de asignación, de declaración, expresiones booleanas.
Las expresiones lógicas también pueden pasarse a código de tres direcciones, utilizando para ello expresiones en corto circuito.

La evaluación de expresiones en corto circuito implica que se evalúan condiciones revisando valores anteriores; por ejemplo, para el operador AND con una condición que se detecte como falsa toda la expresión es falsa, en el caso del operador OR si se encuentra una condición verdadera todo será verdadera.

2.2.4. Cuádruplos.

Los cuádruplos facilitan la aplicación de muchas optimizaciones, pero hay que tener un algoritmo para la reutilización de las variables temporales (reutilización de registros del procesador).
Los cuádruplos son una estructura tipo registro con cuatros campos que se llaman: op, arg1, arg2 y resultado. OP tiene un código intermedio.
Los operadores unarios como x:=-y no utilizan arg2. Generalmente arg1, arg2 y resultado son valores de tipo puntero y apuntan a una entrada en la tabla de símbolos. Ejemplo: (A+B)*(C+D)-E
<OP><arg1><arg2><resultado>
+, A, B, T1
+, C, D, T2
*, T1, T2, T3
-, T3, E, T4

2.3 Esquema de generación.

2.3.1. Variables y constantes.

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

Variables: Espacio de memoria en la computadora que permite almacenar temporalmente un dato durante la ejecución de un programa.
Ejemplo: areaTrapecio = [(B + b) *h] /2.

Constantes: Dato numérico o alfanumérico que no cambia durante la ejecución de un programa.
Ejemplo: pi = 3.1416.

2.3.2. Expresiones.

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Para generar expresiones estas deben representarse de manera más simple y más literal para que su conversión sea más rápida. Por ejemplo la traducción de operaciones aritméticas debe especificarse una por una, de tal forma que una expresión sea lo más mínimo posible.
Ejemplo: (n * 20) / 1.

2.3.3. Instrucciones de asignación.

Son aquellas que asignan un valor a una variable o a una expresión. Una instrucción de asignación consiste en asignar el resultado de la evaluación de una expresión a una variable.
Ejemplo: x = a + b /5

se simplifica como

y = b / 5

```
z = a + y
x = z
```

Las operaciones de asignación deben quedar expresadas por una expresión sencilla, si está es compleja se debe reducir hasta quedar un operador sencillo, comúnmente se utiliza el signo “=”.

2.3.4. Instrucciones de control.

Son aquellas que permiten modificar o variar el flujo de un programa. Existen 3 tipos de instrucciones de control las cuales son:

Instrucciones condicionales o alternativas: Las condiciones deben expresarse de manera lo más sencilla posible de tal forma que puedan evaluarse en cortocircuito.
Ejemplo: una instrucción if (a == b && f!=5 && f%3==0)

se evalúa primero

```
x = (a==b && f!=5) y = x && f%3==0
 if (y)
```

Instrucciones de salto: Las instrucciones de salto como switch se reducen a una versión complejas de if’s.
Ejemplo:

```
switch(opc) {
    case 1:
        statements;
        break;
    case 2:
        statements;
        break;
    default;
        statements;
        break;
}
```

Instrucciones repetitivas: Los ciclos se descomponen en un ciclo genérico, por lo que ciclos while, for y do- while tienen la misma representación interna. En el caso de C, todo queda en forma de while. Las condiciones lógicas también pueden ser evaluadas en cortocircuito y reducidas.
Ejemplo:

```
for (i = 0; i < 10; i++) {
        statements;
}

do {
        statements;
} while(i > 10);
```

2.3.5. Funciones.

Las funciones son un grupo de instrucciones con un objetivo en particular y que se ejecuta al ser llamada en otra función o procedimiento. Las funciones pueden reducir a en línea, lo que se hace es expandir el código original de la función. Las funciones se descomponen simplificando los parámetros de manera individual al igual que el valor de retorno.
Ejemplo:

```
public int Incrementar (int máximo) {
        statements;
}

int numero = Incrementar(7);
```

2.3.6. Estructuras.

Se utilizan para controlar la ejecución y flujo de código, se dividen en dos tipos:

Estructuras de Selección: Son aquellas que se utilizan para realizar operaciones basadas en el valor de una expresión.

Ejemplos: if, if else, switch.

Estructuras de Iteración: Son aquellas que nos permiten ejecutar un bloque de código repetidamente mientras una condición específica sea verdadera.
Ejemplos: for, while, do while.

Conclusión.

2.1 Notaciones.

2.1.1 Prefija.

Por los problemas que representa en el anidamiento aritmético, la notación prefija no se propone.

2.1.2 Infija.

Por su complejidad al momento de representarlo en lenguaje máquina, la notación infija no se propone.

2.1.3 Postfija.

La notación postfija maneja una estructura LIFO (last in first out), es de fácil entendimiento para el lenguaje máquina y es la que se propone a utilizar.

2.2 Representaciones de código intermedio.

2.2.1 Notación Polaca.

La adopción casi universal de la notación algebraica en los sistemas educativos hace que no haya razones de peso para que los alumnos aprendan la notación polaca, por lo cual no se recomienda su uso.

2.2.2 Código P.

El código P al ser muy estandarizado y no permitir extensiones particulares lo vuelve un poco limitado a la hora de ser compatible con arquitecturas específicas, aparte no permite la reutilización de espacios de memoria por lo que no se recomienda.

2.2.3 Triplos.

En el caso de los triplos al ser muy parecido a los cuádruplos, su uso puede ser conveniente, más sin embargo, el no permitir la reutilización de memoria nos hace tomar la representación en cuádruplos.

2.2.4 Cuádruplos.

La gran ventaja de la representación por cuádruplos es que nos permite la reutilización de variables lo cual nos ahorra el uso de registros en el procesador, al igual que la facilidad de entendimiento del código por los cuatros campos: op, arg1, arg2 y resultado.
Y como un plus, es que recordando la propuesta de Análisis Semántico U1 que hicimos, el esquema de traducción de Grafo de Dirección Acíclica (GDA) se adapta perfectamente a la representación por código intermedio por cuádruplos.

2.3 Esquema de generación.

2.3.1 Variables y constantes.

Se propone declarar las variables y constantes tal como es el en el caso de swift donde se utiliza la palabra reservada “let” para una constante y la palabra reservada “var” para una variable. Se propone utilizarlo para definir el tipo de dato junto con su nombre.
Ejemplo:

```
var Mayor = 10;
const velocidadLuz = 300000000;
```

2.3.2 Expresiones

Para las expresiones se propone utilizar la misma sintaxis que se ha utilizado en la mayoría de los lenguajes, se leerá de la misma manera respetando jerarquía de operadores. Quedaría de la siguiente manera (caso particular):

(b * a) / 2

2.3.3 Instrucciones de asignación.

En el caso de la asignación y siguiendo lo propuesto en el punto 2.3.2 de la conclusión, para mayor entendimiento se propone a utilizar el símbolo “=” para la asignación de un elemento ó resultado de una expresión a otro elemento. Se propone usar de la siguiente manera (caso particular):

Resultado = (b * a) / 2

2.3.4 Instrucciones de control.

Instrucciones condicionales o alternativas: Se propone usar la palabra reservada “if” enseguida de una condición que se encuentra en paréntesis “()” la cual si se cumple deberá ejecutar todo el bloque de código que se encuentre delimitado por un par de llaves “{}”, de lo contrario, si la condición es false se deberá de ejecutar otro bloque de código encontrado en un par de llaves, pero este bloque de código deberá de encontrarse después de la palabra reservada “else”.

Ejemplo:

```
if (condición) {
        statements_true;
} else {
        statements_false;
}
```

Instrucciones de salto: Para las instrucciones de salto se propone la sintaxis utilizada por el lenguaje swift, conservando la palabra reservada “case” y eliminando así el uso de la palabra reservada “break” junto con los paréntesis que delimitan la variable que servirá de parámetro en la instrucción, a continuación un ejemplo de cómo se manejaría:

```
switch opcion {
        case 1:
                statements;
```

```
        case 2:
            statements;
        case 3:
            statements;
}
```

Instrucciones repetitivas: Para las instrucciones repetitivas se propone la sintaxis utilizada por el lenguaje swift, en el caso del for se vuelve más funcional la instrucción eliminando el contador y añadiendo la palabra reservada “in”. Mientras que con el while se sustituye la palabra reservada “true” por “repeat”, aquí un pequeño ejemplo:

```
for ct in 1...10 {
}
repeat {
} while (condición);
```

2.3.5 Funciones.

Para las funciones, se propone hacer uso de una declaración que haga uso del caracter reservado “->” al igual de una palabra reservada “func”, seguida por el nombre de la función, unos paréntesis que contengan los parámetros que recibe la función (si van solos quiere decir que no recibe nada), después el caracter reservado “->” seguido de unos paréntesis que contengan los valores que manda la función (si van solos quiere decir que no manda nada). Las funciones se puedan declarar de la siguiente manera.

Función void (no retorna valores):

```
func Saludo() -> () { }
```

Función void que retorna un valor:

```
func Saludo() -> (String Nombre) { }
```

Función void que recibe parámetros:

```
func Saludo(String Nombre) -> () { }
```

Función que retorna valor y recibe parámetros:
func Saludo(String Nombre) -> (String saludoCompleto) { }

2.3.6 Estructuras.

Las estructuras se definen con lo propuesto en todos los puntos anteriores a partir del 2.3.1.

Referencias.

2.1
http://webcache.googleusercontent.com/search?q=cache:http://dsc.itmorelia.edu.mx/~jcolivares/courses/ps207a/ps2_u6.pdf
https://prezi.com/htmcq3v71qdu/unidad-ii-generacion-de-codigo-intermedio/

2.2
http://itpn.mx/recursosisc/7semestre/leguajesyautomatas2/Unidad%20II.pdf
http://webcache.googleusercontent.com/search?q=cache:http://dsc.itmorelia.edu.mx/~jcolivares/courses/ps207a/ps2_u6.pdf
http://www.academia.edu/28108829/Undiad_II_Analisis_semantico

2.3
https://www.goconqr.com/p/7083495-2-3-esquemas-de-generaci-n--mind_maps
https://prezi.com/3cybbmmxjwcw/23-esquema-de-generacion/
http://webcache.googleusercontent.com/search?q=cache:http://dsc.itmorelia.edu.mx/~jcolivares/courses/ps207a/ps2_u6.pdf