UNIVERSIDAD DEL VALLE DE GUATEMALA

Construcción de Compiladores

Laboratorio 2

Diseño de Código Intermedio / Funcionalidad en Tabla de Símbolos

JESSICA PAMELA ORTIZ IXCOT 20192

ESTEBAN ALDANA GUERRA 20591

GUATEMALA, 20 de Septiembre del 2023

1. **Definir la arquitectura de la estructura(s) de datos que soportarán la implementación del Código Intermedio dentro de la Fase de Compilación**

   Para la representación intermedia de un código, vamos a utilizar la representación de Tres Direcciones usando cuádruplos.

   Cada cuádruplo, será representado como una instrucción, que consiste en una operación y tres operandos. Se almacenan en una lista o arreglo, y cada instrucción es un objeto que contiene:

   Operador (suma, resta, multiplicación, etc.)
   Dos argumentos.(Arg1, Arg2)
   Un operando destino (Resultado).

   ```
   class Cuadruplo:
      def __init__(self, operador, arg1, arg2, destino):
         self.operador = operador
         self.op1 = op1
         self.op2 = op2
         self.destino = destino
   ```

2. **Identificar las oraciones gramaticales del lenguaje YAPL que serán traducidas a distintas secuencias de instrucciones de tres direcciones.**

1. *Sentencias Condicionales (if - then - else - fi)*

   Traducido a instrucciones de salto condicional representadas por cuádruplas como if_goto y goto, además de etiquetas label para controlar el flujo.

2. *Sentencias de Bucle (while - loop - pool)*

   Traducido a instrucciones de salto condicional, también representadas por cuádruplas, y etiquetas label de manera similar a las sentencias condicionales.

3. *Bloques de Código ({ - })*

   No se traducen directamente al Código de Tres Direcciones (TAC), pero sirven como delimitadores de bloques de instrucciones en el TAC.

4. *Asignaciones (expressionStatement)*

   Traducidas a cuádruplas que representan asignaciones utilizando el operador =, y potencialmente operaciones aritméticas como +, -, *, y /.

5. *Sentencias de Retorno (returnStatement)*

Traducido directamente a una instrucción de retorno en el TAC.

6. *Expresiones (expression)*

   Operaciones aritméticas se traducen directamente a sus respectivas cuádruplas: +, -, *, /.
   Comparaciones se traducen a cuádruplas con operadores <, <=, ==, !=.
   Operaciones booleanas como not se traducen directamente en cuádruplas.
   Para negaciones específicas, se usará el "menos" como en el ejemplo - c.

3. Definir el conjunto de instrucciones de tres direcciones que constará el lenguaje intermedio a generar.

- Operaciones aritméticas: +, -, *, /.
- Operaciones de asignación: =.
- Operaciones de control: goto, if_goto, label.
- Operaciones lógicas: <, <=, ==, !=, not.
- Su usará “menos” para distinguir al operador de resta. ej (- c)

Ejemplos

1. Sentencias Condicionales (if - then - else - fi)

   YAPL:

```
if a <= b then
   c <- 1;
else
   c <- 0;
fi;
```

   Código Intermedio:

```
t1 = a <= b
if_false t1 goto L1
c = 1
goto L2
L1: c = 0
L2:
```

2. Sentencias de Bucle (while - loop - pool)

   YAPL:

```
while a < b loop
   a <- a + 1;
pool;
```

Codigo Intermedio:

```
L1:
t1 = a < b
if_false t1 goto L2
a = a + 1
goto L1
L2:
```

3. Bloques de Código ({ - })

   No se traducen directamente a TAC pero se utilizan para delimitar un bloque de instrucciones TAC.

4. Asignaciones (expressionStatement)

   YAPL:

   ```
   a <- b + c;
   ```

   Codigo Intermedio:

   ```
   t1 = b + c
   a = t1
   ```

5. Sentencias de Retorno (returnStatement)

   YAPL:

   ```
   return a;
   ```

   Codigo intermedio:

   ```
   return a;
   ```

6. Expresiones (expression)

   YAPL:

   ```
   d <- a * b;
   ```

   Codigo Intermedio:

   ```
   t1 = a * b
   d = t1
   ```