Universidad Nacional del Centro de la Provincia de Buenos Aires

Trabajo Práctico

Diseño de Compiladores

Optimización por Reducción Simple

Grupo 8

INTEGRANTES:	CORREO:
Chiozza, Juan Ignacio	chiozzajuani@gmail.com
González, Cristian	cris_mdq22@hotmail.com
Rodríguez, Facundo Hernán	farodriguez@alumnos.exa.unicen.edu.ar

Introducción:

Como parte final del trabajo se realizará una optimización de reducción simple durante la construcción de los tercetos, la cual consiste en evaluar expresiones que son conocidas en tiempo de compilación (entre constantes), y se sustituyen por su valor.

Reducción Simple:

Para llevar a cabo dicha reducción simple se modificó la gramática, ya que se pedía que se efectuará en la construcción de los tercetos. Debido a este motivo las reglas que se vieron afectadas fueron las de Expresión y Término, la cuales se encargaban de las operaciones suma y resta (expresión), multiplicación y división (término).

A ambas reglas se le agregó una parte de código, en el cual se evalúa si los dos operandos involucrados en la operación se tratan de constantes, ya sean de tipo int o double. En el caso de que sean constantes, se utiliza una variable auxiliar para almacenar el resultado de dicha operación, la cual luego es utilizada para armar el Token, siendo este último agregado en la tabla de símbolos. Adicionalmente, en caso de ser una operación entre Doubles, al susodicho Token se le agrega el atributo AuxDouble para indicar al convertidor de assembler que debe declarar una variable de tipo QWORD con valor igual a la variable auxiliar. Por último se retorna el token como valor de la regla.

Se procede a mostrar dicha explicación, tomando como ejemplo la expresión de la suma:

```
if (es_cte_entera(t1) && es_cte_entera(t2)){
    int calculo_int = Integer.parseInt(t1.getValor()) + Integer.parseInt(t2.getValor());
    Token t = new Token(AnalizadorLexico.tipoInt, AnalizadorLexico.CTEENTERA, calculo_int);
    tablaSimbolo.add(String.valueOf(calculo_int),t);
    $$=new ParserVal(t);
        }
else if (es_cte_double(t1) && es_cte_double(t2)){
        double calculo_double = Double.parseDouble(t1.getValor()) + Double.parseDouble(t2.getValor());
        Token t = new Token(AnalizadorLexico.tipoDouble, AnalizadorLexico.CTEDOUBLE, calculo_double);
        t.AgregarAtributo("AuxDouble",analizadorL.getVAuxDouble());
        tablaSimbolo.add(String.valueOf(calculo_double),t);
        $$=new ParserVal(t);
            }
```

Conclusión:

En esta etapa del trabajo, se logró ver la optimización que se genera en la construcción de los tercetos, ya que si en el programa se cuenta con muchas operaciones que están compuestas solo de constantes del mismo tipo, estas mismas no van a formar un terceto nuevo, por lo que se va a ver reducida la cantidad de tercetos involucrados en el programa completo, y por lo tanto el tiempo de ejecución del programa final.