Set de Ejercicios 5

Javier Falcón (2016-5265)

Ejercicio 6.14

Muestre que, dada la gramática con atributos (mostrada en el ejercicio) si el atributo *type.dtype* se mantiene en la pila de valores durante un análisis sintáctico LR, entonces este valor no se puede encontrar en una posición fija en la pila cuando ocurren reducciones a una *var-list*.

Efectivamente no es posible poder identificar la posición del atributo *type.dtype* debido a que, en vista de que la regla coloca a *var-list* para que sea recursiva por la derecha, no habría manera de predecir cúantos *IDs* están en la pila, lo cual hace imposible conocer la posición de *dtype* dentro de ella, a menos que exista una forma de que el programador pueda acceder directamente a la pila.

Ejercicio 6.20

Considere la siguiente gramática (ambigua) de expresiones:

$$exp \rightarrow exp + exp | exp - exp | exp * exp | exp / exp | (exp) | num | num.num$$

Suponga que se siguen las reglas de C al calcular el valor de cualquier expresión así: si dos subexpresiones son de tipo mixto, entonces la subexpresión en modo entero se convierte a modo de punto flotante, y se aplica el operador de punto flotante. Escriba una gramática con atributos que convertirá a tales expresiones en expresiones legales en Modula-2: las conversiones de números enteros a números de punto flotante se expresan aplicando la función FLOAT, y el operador de división / se considera como div si sus operitndos son enteros.

```
exp: exp + exp
{
     if($1.type != float && $3.type != float) {
           FLOAT(\$1);
           FLOAT(\$3);
    }
        $$.type = $1.type;
        $$ = 1+$3;
}

exp: exp - exp
{
        if($1.type != float && $3.type != float) {
                       FLOAT($1);
                       FLOAT($3);
        }
         $$.type = $1.type;
         $$ = 1-$3;
}

exp: exp * exp
{
        if($1.type != float & $3.type != float){
                       FLOAT($1);
                       FLOAT($3);
        }
        $$.type = $1.type;
        $$ = $1*$3;
}

exp: exp / exp
{
        if($1.type != float && $3.type != float){
                FLOAT($1);
                FLOAT($3);
                $$ = 1/$3;
        }
        else{
                $$ = div($1,$3);
        }
        $$.type = $1.type;
}

exp: (exp)
{
        $$.type = $2.type;
}
```

```
exp:  num
{
        $$.type = $1.type;
}

exp: num.num
{
        FLOAT($1);
        FLOAT($3);
        $$.type = $1.type;
}
```

Ejercicio 6.1

- Diseñe una estructura adecuada de árbol para las nuevas estructuras de tipo de función, y escriba una función typeEqual para dos tipos de función.

Ejercicio 6.22

a. Describa cómo puede utilizar el analizador sintáctico la tabla de símbolos para eliminar la ambigüedad de estas dos interpretaciones.

   En vista de que la tabla de símbolos maneja las propiedades de cada identificador, considero que la eliminación de la ambigüedad vendría dada en el momento en el que se analiza **A**. Por ejemplo, si se busca **A** en la tabla de símbolos y se encuentra registrada en ella, quiere decir que **A** es un ID de tipo entero y, al terminar de analizar la línea, efectuará la resta entre **A** y **x**. Ahora bien, si **A** no está en la tabla y fue definido como un tipo, se precederá a hacer el casting de *int* a *double*.

b. Describa cómo puede emplear el analizador léxico la tabla de símbolos para eliminar la ambigüedad de estas dos interpretaciones.

   Según lo que hemos visto en clase y en experiencias trabajando con *Flex*, considero que este problema no podría ser solucionado en el análisis léxico debido a que éste no sabe manejar cuáles son las operaciones ni cuáles son los tipos, ya que su trabajo consiste solucionar la decisión de si los lexemas del código pertenecen o no al lenguaje, por lo que considero que es un problema que solo puede ser resuelto a nivel de *parsing.*