Practica 3 Compiladores

Angel Sandoval Mendoza

Octubre 2024

1 Gramatica Original

```
programa → declaraciones sentencias
declaraciones → declaraciones declaracion | declaracion
declaracion → tipo lista_var ;
tipo → int | float
lista_var → lista_var , identificador | identificador
sentencias → sentencias sentencia | sentencia
sentencia → identificador = expresion ; | if ( expresion ) sentencias else sentencias |
            while ( expresion ) sentencias
expresion → expresion + expresion | expresion - expresion | expresion * expresion |
            expresion / expresion | identificador | numero
expresion → ( expresion )
```

- Símbolo inicial (S):

  {programa}

- No terminales (N):

  {programa, declaraciones, declaracion, tipo, lista_var, sentencias, sentencia, expresion}

- Terminales (Σ)

  {int, float, identificador, número, =, ;, if, else, while, +, -, *, /, (,),

2 Eliminación de ambiguedad

Lo que deberemos de hacer es quitar la ambiguedad de la dramatica, en este caso la gramatica original es ambigua en la parte de expresion debido a que no tenemos una jerarquía de operaciones, asi que la haremos nosotros

```
programa → declaraciones sentencias
declaraciones → declaraciones declaracion | declaracion
declaracion → tipo lista_var ;
```

```
tipo → int | float
lista_var → lista_var , identificador | identificador
sentencias → sentencias sentencia | sentencia
sentencia → identificador = expresion ; |
            if ( expresion ) sentencias else sentencias |
            while ( expresion ) sentencias
expresion → expresion + termino | expresion - termino | termino
termino → termino * factor | termino / factor | factor
factor → identificador | numero | ( expresion )
```

3 Eliminación de la recursividad izquierda

Aquí eliminaremos la recursión izquierda que tiene la gramatica, lo haremos de la siguiente forma:

```
declaraciones → declaracion declaraciones'
declaraciones' → declaracion declaraciones' | epsilon
```

Este mismo esquema lo repetimos para las producciones de lista_var, sentencias, espresion y termino, por lo que la gramatica quedaria de la siguiente manera:

```
programa → declaraciones sentencias
declaraciones → declaracion declaraciones'
declaraciones' → declaracion declaraciones' | epsilon
declaracion → tipo lista_var ;
tipo → int | float
lista_var → identificador lista_var'
lista_var' → , identificador lista_var' | epsilon
sentencias → sentencia sentencias'
sentencias' → sentencia sentencias' | epsilon
sentencia → identificador = expresion ; |
            if ( expresion ) sentencias else sentencias |
            while ( expresion ) sentencias
expresion → termino expresion'
expresion' → + termino expresion' | - termino expresion' | epsilon
termino → factor termino'
termino' → * factor termino' | / factor termino' | epsilon
factor → identificador | numero | ( expresion )
```

4 Factorización izquierda

Para esta gramatica no es necesario hacer una factorización izquierda, esto debido a que no se llegan a compartir prefijos comunes entre las producciones.

5 Nuevos conjuntos N y P

Ahora tendriamos como las producciones P las siguientes:

```
programa → declaraciones sentencias
declaraciones → declaracion declaraciones'
declaraciones' → declaracion declaraciones' | epsilon
declaracion → tipo lista_var ;
tipo → int | float
lista_var → identificador lista_var'
lista_var' → , identificador lista_var' | epsilon
sentencias → sentencia sentencias'
sentencias' → sentencia sentencias' | epsilon
sentencia → identificador = expresion ; |
            if ( expresion ) sentencias else sentencias |
            while ( expresion ) sentencias
expresion → termino expresion'
expresion' → + termino expresion' | - termino expresion'                | epsilon
termino → factor termino'
termino' → * factor termino' | / factor termino' | epsilon
factor → identificador | numero | ( expresion )
```

Y el conjunto de simbolos terminales estaría determinado por los siguientes elementos:

programa, declaraciones, declaraciones', declaracion, tipo