Universidad Nacional Autónoma de México
Facultad de Ciencias
Compiladores
Práctica 3

Osorio Escandón Huriel — 317204652 — Huriel117@ciencias.unam.mx

1. Determinar los conjuntos N, Σ y el símbolo inicial S.

   Conjuntos de la Gramática G

   Conjunto de No Terminales (N)

   El conjunto de no terminales es:

   $$N = \{\text{programa, declaraciones, declaracion, tipo, lista\_var, sentencias, sentencia, expresion}\}$$

   Conjunto de Terminales (Σ)

   El conjunto de terminales es:

   $$\Sigma = \{\text{int, float, ,, =, ;, (,), +, -, *, /, if, else, while, identificador, numero}\}$$

   Símbolo Inicial (S)

   El símbolo inicial de la gramática es:

   $$S = \text{programa}$$

2. Mostrar en el archivo el proceso de eliminación de ambigüedad o justificar, en caso de no ser necesario.

   Eliminación de Ambigüedad

   Expresiones Aritméticas

   La gramática original para las expresiones aritméticas presenta una potencial ambigüedad, ya que no especifica la precedencia ni la asociatividad de los operadores `+`, `-`, `*` y `/`. Para resolver esta ambigüedad, se van a introducir reglas que establecen una precedencia adecuada y una asociatividad por la izquierda.

   La producción original:

   ```
   expresion → expresion + expresion
             | expresion - expresion
             | expresion * expresion
             | expresion / expresion
             | identificador
             | numero
             | ( expresion )
   ```

   Las modificaciones son las siguientes:

```
expresion → expresion_suma
expresion_suma → expresion_suma + expresion_mult | expresion_suma - expresion_mult |
expresion_mult
expresion_mult → expresion_mult * expresion_unaria | expresion_mult / expresion_unaria |
expresion_unaria
expresion_unaria → identificador | numero | ( expresion )
```

De esta manera, garantizamos que `*` y `/` tienen mayor precedencia que `+` y `-`, y todas las operaciones son asociativas por la izquierda.

Problema del if-else colgante

La gramática original presenta el problema del *if-else colgante*, que puede llevar a ambigüedades en la interpretación de las sentencias. Para resolverlo, se harán las siguientes modificaciones:

```
sentencia → sentencia_if | sentencia_otra
sentencia_if → if ( expresion ) sentencia_if else sentencia_if
             | if ( expresion ) sentencia_otra
sentencia_otra → identificador = expresion ; | while ( expresion ) sentencias | { sentencias }
```

De esta manera, garantizamos que un `else` siempre se empareja con el `if` más cercano que lo requiere, eliminando así la ambigüedad.

3. Mostrar en el archivo el proceso de eliminación de la recursividad izquierda o justificar, en caso de no ser necesario.

Eliminación de la Recursividad Izquierda

En la gramática original, se presentan varias producciones con recursividad izquierda, lo que puede causar problemas en un analizador sintáctico de descenso recursivo.

Producción de `declaraciones`

La producción original es recursiva a la izquierda:

```
declaraciones → declaraciones declaracion | declaracion
```

La nueva producción, sin recursividad izquierda, es:

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

Producción de `lista_var`

La producción original:

```
lista_var → lista_var , identificador | identificador
```

La nueva producción sin recursividad izquierda:

```
lista_var → identificador lista_var'
lista_var' → , identificador lista_var' |
```

Producción de `sentencias`

La producción original:

```
sentencias → sentencias sentencia | sentencia
```

La nueva producción sin recursividad izquierda:

```
sentencias → sentencia sentencias'
sentencias' → sentencia sentencias' |
```

4. Mostrar en el archivo el proceso de factorización izquierda o justificar, en caso de no ser necesario.

   Factorización Izquierda

   Después de analizar las producciones de la gramática, notemos que no es necesario realizar factorización izquierda, ya que no existen prefijos comunes entre las alternativas de las producciones.

   Producción de `expresion`

   La producción original para `expresion` no contiene prefijos comunes:

```
expresion → expresion + expresion
          | expresion - expresion
          | expresion * expresion
          | expresion / expresion
          | identificador
          | numero
          | ( expresion )
```

   Producción de `sentencia`

   La producción de `sentencia` tampoco presenta prefijos comunes:

```
sentencia → identificador = expresion ;
        | if ( expresion ) sentencias else sentencias
        | while ( expresion ) sentencias
```

   Producción de `declaraciones`

   Las producciones para `declaraciones` tampoco requieren factorización izquierda, ya que no hay prefijos comunes:

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

   Tomando en cuenta lo anterior, se tiene que, no es necesario realizar la factorización izquierda en la gramática.

5. Mostrar en el archivo los nuevos conjuntos N y P.

   Nuevos Conjuntos N y P

   Después de realizar las modificaciones necesarias, los conjuntos de no terminales y producciones son los siguientes:

   Conjunto de No Terminales (N)

   El conjunto de no terminales es:

   N = {programa, declaraciones, declaraciones', declaracion, tipo, lista_var, lista_var', sentencias, sentencias', sentencia, expresion, expresion_suma, expresion_mult, expresion_unaria }

   Conjunto de Producciones (P)

   Las producciones de la gramática son las siguientes:

```
programa → declaraciones sentencias

declaraciones → declaracion declaraciones'
declaraciones' → declaracion declaraciones' |

declaracion → tipo lista_var ;

tipo → int | float

lista_var → identificador lista_var'
lista_var' → , identificador lista_var' |

sentencias → sentencia sentencias'
sentencias' → sentencia sentencias' |

sentencia → identificador = expresion ;
          | if ( expresion ) sentencias else sentencias
          | while ( expresion ) sentencias

expresion → expresion_suma
expresion_suma → expresion_suma + expresion_mult | expresion_suma - expresion_mult |
expresion_mult
expresion_mult → expresion_mult * expresion_unaria | expresion_mult / expresion_unaria |
expresion_unaria
expresion_unaria → identificador | numero | ( expresion )
```