



- Dada la gramática  $G = (N, \Sigma, P, S)$

Donde  $P$  es:

```
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 )
```

1. Se determinan los conjuntos  $N, \Sigma$  y el símbolo inicial  $S$ . (0.5 pts.)

- $N$  (No terminales):

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

- $\Sigma$  (Terminales):

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

- $S$  (Símbolo inicial):

$$S = \{\text{programa}\}$$

## 2. Se realiza el proceso de eliminación de ambigüedad. (1 pts.)

### i. Se identifica la ambigüedad

$$\begin{aligned} \text{expresion} \rightarrow & \text{expresion} + \text{expresion} \\ & | \text{expresion} - \text{expresion} \\ & | \text{expresion} * \text{expresion} \\ & | \text{expresion} / \text{expresion} \\ & | \text{identificador} \\ & | \text{numero} \\ & | (\text{expresion}) \end{aligned}$$

Esta gramática es ambigua porque no especifica la precedencia ni la asociatividad de los operadores, permitiendo derivaciones múltiples para una misma expresión. La multiplicación y división tienen mayor precedencia que la suma y resta. Se necesita separar la gramática para reflejar esta diferencia.

### ii. Se define la recursividad según la asociatividad

Los operadores +, -, \* y / son asociativos a la izquierda. Por lo tanto, las producciones deben ser recursivas a la izquierda para reflejar esta propiedad.

### iii. Se reescribe la gramática respetando precedencia y asociatividad

#### o Nivel más bajo:

$$\text{factor} \rightarrow (\text{expresion}) \mid \text{identificador} \mid \text{numero}$$

#### o Nivel intermedio:

$$\text{termino} \rightarrow \text{termino} * \text{factor} \mid \text{termino} / \text{factor} \mid \text{factor}$$

#### o Nivel superior:

$$\text{expresion} \rightarrow \text{expresion} + \text{termino} \mid \text{expresion} - \text{termino} \mid \text{termino}$$

### Gramática Actualizada:

$$\begin{aligned} \text{programa} &\rightarrow \text{declaraciones} \text{ sentencias} \\ \text{declaraciones} &\rightarrow \text{declaraciones} \text{ declaracion} \mid \text{declaracion} \\ \text{declaracion} &\rightarrow \text{tipo} \text{ lista\_var} ; \\ \text{tipo} &\rightarrow \text{int} \mid \text{float} \\ \text{lista\_var} &\rightarrow \text{lista\_var}, \text{identificador} \mid \text{identificador} \\ \text{sentencias} &\rightarrow \text{sentencias} \text{ sentencia} \mid \text{sentencia} \\ \text{sentencia} &\rightarrow \text{identificador} = \text{expresion} ; \\ &\mid \text{if} ( \text{expresion} ) \text{ sentencias} \text{ else} \text{ sentencias} \\ &\mid \text{while} ( \text{expresion} ) \text{ sentencias} \\ \text{expresion} &\rightarrow \text{expresion} + \text{termino} \mid \text{expresion} - \text{termino} \mid \text{termino} \\ \text{termino} &\rightarrow \text{termino} * \text{factor} \mid \text{termino} / \text{factor} \mid \text{factor} \\ \text{factor} &\rightarrow (\text{expresion}) \mid \text{identificador} \mid \text{numero} \end{aligned}$$

### 3. Se muestra el proceso de eliminación de la recursividad izquierda. (1 pts.)

#### i. Se identifica la recursividad izquierda

- $\text{declaraciones} \rightarrow \text{declaraciones declaracion} \mid \text{declaracion}$
- $\text{lista\_var} \rightarrow \text{lista\_var}, \text{identificador} \mid \text{identificador}$
- $\text{sentencias} \rightarrow \text{sentencias sentencia} \mid \text{sentencia}$
- $\text{expresion} \rightarrow \text{expresion} + \text{termino} \mid \text{expresion} - \text{termino} \mid \text{termino}$
- $\text{termino} \rightarrow \text{termino} * \text{factor} \mid \text{termino} / \text{factor} \mid \text{factor}$

#### ii. Se elimina la recursividad izquierda.

- **declaraciones:**

$$\begin{aligned} \text{declaraciones} &\rightarrow \text{declaracion declaraciones}' \\ \text{declaraciones}' &\rightarrow \text{declaracion declaraciones}' \mid \epsilon \end{aligned}$$
- **lista\_var:**

$$\begin{aligned} \text{lista\_var} &\rightarrow \text{identificador lista\_var}' \\ \text{lista\_var}' &\rightarrow , \text{identificador lista\_var}' \mid \epsilon \end{aligned}$$
- **sentencias:**

$$\begin{aligned} \text{sentencias} &\rightarrow \text{sentencia sentencias}' \\ \text{sentencias}' &\rightarrow \text{sentencia sentencias}' \mid \epsilon \end{aligned}$$
- **expresion:**

$$\begin{aligned} \text{expresion} &\rightarrow \text{termino expresion}' \\ \text{expresion}' &\rightarrow + \text{termino expresion}' \\ &\quad \mid - \text{termino expresion}' \mid \epsilon \end{aligned}$$
- **termino:**

$$\begin{aligned} \text{termino} &\rightarrow \text{factor termino}' \\ \text{termino}' &\rightarrow * \text{factor termino}' \\ &\quad \mid / \text{factor termino}' \mid \epsilon \end{aligned}$$

#### 4. Se muestra por qué no es necesario aplicar factorización izquierda. (1 pts.)

La gramática no requiere factorización izquierda debido a que no se presentan producciones con prefijos comunes. Esto significa que las producciones de la gramática comienzan de manera diferente, lo que permite que un parser determine cuál regla aplicar sin ambigüedades.

#### 5. Se muestran los nuevos conjuntos $N$ y $P$ . (0.5 pts.)

- $N$  (No terminales):

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

- $P$  (Producciones):

```

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

```