Universidad

Nacional Autónoma de México

Facultad de Ciencias

Preguntas práctica 3

Integrantes:
Yonathan Berith Jaramillo Ramírez. 419004640

Profesor: Adrián Ulises Mercado Martínez
Ayudantes: Yessica Janeth Pablo Martínez
Carlos Gerardo Acosta Hernández

5 Oct, 2024

Compiladores

Gramática Original

La gramática original se describe mediante las siguientes producciones:

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)

Pregunta 1

Determina los conjuntos N, Σ, y el símbolo inicial S.

Solución:

Con base en esta gramática:

- El conjunto de **No terminales** N es:

$$N = \left\{ \begin{array}{l} \text{programa, declaraciones, declaracion,} \\ \text{tipo, lista\_var, sentencias,} \\ \text{sentencia, expresion} \end{array} \right\}$$

- El conjunto de **Terminales** Σ es:

$$\Sigma = \left\{ \begin{array}{l} \text{"int", "float", "identificador",} \\ \text{"numero", "+", "-", "*", "/",} \\ \text{"=", "(", ")", "if",} \\ \text{"else", "while", ";", ","} \end{array} \right\}$$

- El **símbolo inicial** S es:

$$S = \text{programa}$$

Pregunta 2

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

Solución:

La gramática original es ambigua, especialmente porque el no terminal `expresion` se declara en múltiples reglas de producción con diferentes operadores sin indicar precedencia o asociatividad. Además, la declaración de `expresion` aparece dos veces, lo que genera ambigüedad sobre qué producción utilizar al derivar cadenas.

Consideremos la siguiente producción, donde se ve claramente la duplicidad:

$$\begin{aligned} \text{expresion} \rightarrow \text{expresion} + \text{expresion} \,|\, \text{expresion} - \text{expresion} \\ |\, \text{expresion} * \text{expresion} \,|\, \text{expresion}/\text{expresion} \end{aligned}$$

$$\text{expresion} \rightarrow (\text{expresion})$$

Sin reglas adicionales, esta producción es ambigua, ya que no está claro cómo evaluar expresiones como:

$$a + b * c$$

Esto genera más de un árbol de derivación posible. Para eliminar la ambigüedad, introducimos diferentes niveles de precedencia en la gramática, diferenciando entre suma/resta y multiplicación/división, y estableciendo que los operadores de suma y resta son asociativos por la izquierda.

La gramática sin ambigüedad queda así:

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

Pregunta 3

Introducción a la Recursividad Izquierda Inmediata

Solución:

La **recursividad izquierda inmediata** ocurre cuando una producción tiene la forma:

$$A \rightarrow A\alpha \mid \beta$$

lo cual puede causar problemas en parsers de descenso recursivo al derivarse el no terminal A a sí mismo sin consumir tokens. Para eliminar esta recursividad, seguimos este procedimiento:

1. Introducimos un nuevo no terminal A'. 2. Reescribimos la regla de la siguiente manera:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \epsilon$$

Aplicación del Procedimiento a la Gramática

La gramática original contiene recursividad izquierda en varias producciones que necesitan ser transformadas para que sea compatible con un parser LL(1). Aplicamos el procedimiento a cada producción recursiva.

1. **Para** `declaraciones`

   - **Producción Original:**

$$\text{declaraciones} \rightarrow \text{declaraciones declaracion} \mid \text{declaracion}$$

   - **Eliminación de la Recursividad:**

$$\text{declaraciones} \rightarrow \text{declaracion declaraciones'}$$

$$\text{declaraciones'} \rightarrow \text{declaracion declaraciones'} \mid \epsilon$$

2. **Para** `sentencias`

   - **Producción Original:**

$$\text{sentencias} \rightarrow \text{sentencias sentencia} \mid \text{sentencia}$$

   - **Eliminación de la Recursividad:**

$$\text{sentencias} \rightarrow \text{sentencia sentencias'}$$

$$\text{sentencias'} \rightarrow \text{sentencia sentencias'} \mid \epsilon$$

3. **Para** `expresion_suma`

   - **Producción Original:**

$$\begin{aligned}\text{expresion\_suma} \rightarrow\ & \text{expresion\_suma} + \text{expresion\_producto} \\ \mid\ & \text{expresion\_suma} - \text{expresion\_producto} \\ \mid\ & \text{expresion\_producto}\end{aligned}$$

   - **Eliminación de la Recursividad:**

$$\begin{aligned}\text{expresion\_suma} &\rightarrow \text{expresion\_producto expresion\_suma'} \\ \text{expresion\_suma'} &\rightarrow + \text{expresion\_producto expresion\_suma'} \\ &\mid - \text{expresion\_producto expresion\_suma'} \\ &\mid \epsilon\end{aligned}$$

4. **Para** `expresion_producto`

- **Producción Original:**

$$\begin{aligned} \text{expresion\_producto} \rightarrow\ & \text{expresion\_producto} * \text{expresion\_termino} \\ & \mid \text{expresion\_producto/expresion\_termino} \\ & \mid \text{expresion\_termino} \end{aligned}$$

- **Eliminación de la Recursividad:**

$$\begin{aligned} \text{expresion\_producto} &\rightarrow \text{expresion\_termino expresion\_producto'} \\ \text{expresion\_producto'} &\rightarrow *\text{expresion\_termino expresion\_producto'} \\ &\mid /\text{expresion\_termino expresion\_producto'} \\ &\mid \epsilon \end{aligned}$$

Nueva Gramática Sin Recursividad Izquierda

La gramática, una vez eliminada la recursividad izquierda, queda de la siguiente manera:

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

Pregunta 4

Justificación

La **factorización izquierda** es una técnica que permite eliminar prefijos comunes en una gramática, lo cual es útil para parsers de tipo LL(1), ya que estos requieren que cada producción tenga prefijos únicos para decidir qué regla aplicar. Si una producción tiene prefijos comunes, el parser puede enfrentar problemas de ambigüedad.

En nuestra gramática, la producción que necesita factorización es `lista_var`, ya que presenta un prefijo común que puede generar conflictos.

Producción a Factorizar

1. **Para** `lista_var`:

- **Producción Original:**

$$\text{lista\_var} \rightarrow \text{lista\_var , identificador} \mid \text{identificador}$$

- **Factorización:** Para eliminar el prefijo común `lista_var`, introducimos un nuevo no terminal `lista_var'`:

$$\text{lista\_var} \rightarrow \text{identificador lista\_var'}$$

$$\text{lista\_var'} \rightarrow \text{, identificador lista\_var'} \mid \epsilon$$

Con esta modificación, hemos eliminado el prefijo común y la gramática ahora es compatible con un parser LL(1) para esta producción.

Producción Final Factorizada

La gramática con la producción `lista_var` factorizada queda de la siguiente manera:

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

Pregunta 5

Mostrar en el archivo los nuevos conjuntos N y P.

Solución:

Conjunto de No Terminales N

El conjunto de no terminales N contiene todos los símbolos que pueden expandirse en otras producciones en la gramática. Después de los procesos de eliminación de

ambigüedad, recursividad izquierda y factorización, los no terminales de la gramática se organizan de la siguiente manera:

$$N = \left\{ \begin{array}{l} \text{programa, declaraciones, declaraciones', declaracion, tipo, lista\_var, lista\_var',} \\ \text{sentencias, sentencias', sentencia, expresion, expresion\_suma, expresion\_suma',} \\ \text{expresion\_producto, expresion\_producto', expresion\_termino} \end{array} \right\}$$

Cada uno de estos no terminales es esencial para representar la estructura de la gramática en su versión final:

- **programa**: No terminal principal que define el inicio de la gramática.
- **declaraciones** y **declaraciones'**: Se utilizan para manejar la lista de declaraciones con recursividad eliminada.
- **declaracion** y **tipo**: Representan las declaraciones y tipos de variables.
- **lista_var** y **lista_var'**: Manejadas con factorización para eliminar prefijos comunes.
- **sentencias** y **sentencias'**: Organizan las sentencias con recursividad eliminada.
- **sentencia**: Describe los tipos de sentencias (asignación, condicional, bucle).
- **expresion**, **expresion_suma**, **expresion_suma'**, **expresion_producto**, **expresion_producto'**, **expresion_termino**: Controlan las operaciones aritméticas, respetando la precedencia y asociatividad.

Conjunto de Producciones P

El conjunto de producciones P se construye siguiendo la gramática final derivada de los ejercicios anteriores. Esta versión está libre de ambigüedad, sin recursividad izquierda y sin prefijos comunes.

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