Gramática de la práctica 3

Kevin Isaac Alcántara Estrada

October 2024

1 Identificar elementos de la gramática

Primero debes tomar en cuenta las expresiones regulares que deben ser identificadas por el autómata.

1.1 Producciones

P = {
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) }

1.2 Símbolos no terminales

El conjunto de símbolos no terminales N es el siguiente:

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

1.3 Símbolos terminales

El conjunto de símbolos terminales Σ es el siguiente:

Σ = { "identificador", "numero", "int" ,"float", "if", "else", "while", "(", ")", ",", ";", "+", "-", "*", "/", "=" }

1.4 Símbolos inicial

Nuestro símbolo inicial $S \in N$ para la gramática sería $S = programa$

2 Eliminación de la ambigüedad en la gramática

Podemos ver que la gramática es ambigüa porque tiene más de una producción del estilo $S \rightarrow S\alpha S$, en particular, las producciones que generan ambigüedad en la gramática son:

expresion → expresion + expresion | expresion - expresion | expresion * expresion | expresion / expresion

Es por ello que debemos eliminar la ambigüedad en la gramática, ello de acuerdo al algoritmo visto en clase.

De acuerdo a las producciones que debemos modificar, la precedencia de los operadores que nos interesan es la siguiente (de menor a mayor precedencia de arriba hacia abajo según se vio en clase):

+ -
* /
( )

Así pues, tomando a consideración dicha precendencia, las producciones resultantes de las modificaciones serán las siguientes:

expresion → expresion + expresion_1 | expresion - expresion_1 | expresion_1

expresion_1 → expresion_1 * expresion_2 | expresion_1 / expresion_2 | expresion_2

expresion_2 → (expresion) | identificador | numero

Hata ahora, tenemos que:

P = {
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_1 | expresion - expresion_1 | expresion_1

expresion_1 → expresion_1 * expresion_2 | expresion_1 / expresion_2 | expresion_2

expresion_2 → (expresion) | identificador | numero
}

3 Eliminar recursividad izquierda de la gramática

Hay diversas producciones que poseen recursividad por la izquierda, para eliminar dicha recrusión, modificaremos el conjunto de producciones tal y como se observó en clase.

Para "declaraciones → declaraciones declaracion | declaracion" tendremos:

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

Para "lista_var → lista_var , identificador | identificador" tendremos:

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

Para "sentencias → sentencias sentencia | sentencia" tendremos:

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

Para "expresion → expresion + expresion_1 | expresion - expresion_1 | expresion_1" tendremos:

expresion → expresion_1 expresion'
expresion' → + expresion_1 expresion'| - expresion_1 expresion' | ϵ

Para "expresion_1 → expresion_1 * expresion_2 | expresion_1 / expresion_2 | expresion_2" tendremos:

expresion_1 → expresion_2 expresion"
expresion" → * expresion_2 expresion"| / expresion_2 expresion" | ϵ

Ya hemos eliminado la recursividad por la izquierda para la gramática. Hasta ahora tenemos lo siguiente:

P = {
programa → declaraciones sentencias

declaraciones → declaracion declaracion'

declaracion' → declaracion declaracion' | 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 → expresion_1 expresion'

expresion' → + expresion_1 expresion'| - expresion_1 expresion' | ϵ

expresion1 → expresion_2 expresion"

expresion" → * expresion_2 expresion"| / expresion_2 expresion" | ϵ

expresion_2 → (expresion) | identificador | numero
}

4 Factorización izquierda de la gramática

En este caso, no es necesario realizar una factorización izquierda de la gramática debido a que no existe más de una producción tal que para algún $A \in N$ ocurra que $A \rightarrow \alpha\beta_1|\alpha\beta_2|....|\alpha\beta_i|\gamma_1|\gamma_2|...|\gamma_k$

5 Nuevos conjuntos N y P

El conjunto de producciones ahora es
P = {
programa → declaraciones sentencias

declaraciones → declaracion declaracion'

declaracion' → declaracion declaracion' | 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 → expresion_1 expresion'

expresion' → + expresion_1 expresion'| - expresion_1 expresion' | ϵ

expresion1 → expresion_2 expresion"

expresion" → * expresion_2 expresion"| / expresion_2 expresion" | ϵ

expresion_2 → (expresion) | identificador | numero
}

El conjunto de símbolos no terminales ahora es:

N = {programa, declaraciones, declaracions' sentencias, sentencias', declaracion, sentencia, tipo, lista_var, lista_var', expresion, expresion_1, expresion_2, expresion', expresion" }