Nombre: Dana Carolina Ramírez Velázquez

Código: 220286547

Materia: Traductores de Lenguaje 2

Actividad: Practica 3

Fecha: 19/10 /22

Introducción:

Para poder realizar un analizador que funcione de manera correcta, es necesario quitar la ambigüedad de la gramática y realizar las producciones con recursividad por la izquierda.

Sabemos que la ambigüedad es un fenómeno que se produce cuando un enunciado o una oración puede interpretarse en dos sentidos diferentes. En Ciencias de la computación, un a gramática ambigüa es una gramática libre del contexto para la que existe una cadena que puede tener más de una derivación a la izquierda. Mientras que las gramáticas sin ambigüedad y que son libres de contexto, valída una única derivación a la izquierda.

Desarrollo:

Para deshacernos de la ambigüedad, tenemos que identificar primero a las cadenas que tienen una llamada a si mismas en la gramática y agrupar todas sus producciones:

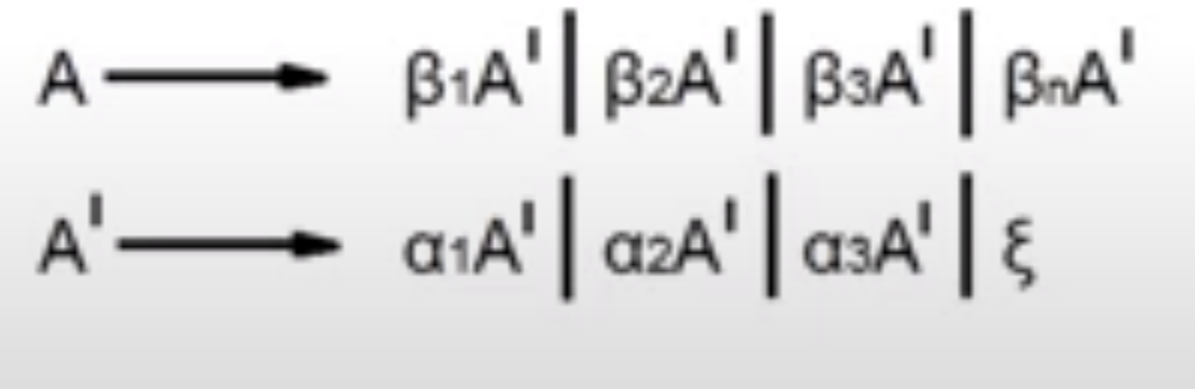
<programa> → **begin** <declaraciones> <órdenes> **end**
<declaraciones> → <declaración> **;** | <declaración>**;** <declaraciones>
<declaración> → <tipo> <lista_variables>
<tipo> → **entero** | **real**
<lista_variables> → <identificador> | <identificador> **,** <lista_variables>
<identificador> → <letra> | <letra> <resto_letras>
<letra> → **A | B | C | ... | Z | a | b | c | ... | z**
<resto_letras> → <letraN> | <letraN> <resto_letras>
<letraN> → **A | B | C | ... | Z | a | b | c | ... | z | 0 | 1 | … | 9**

<órdenes> → <orden> ; | <orden> ; <órdenes>
<orden> → <condición> | <bucle_while> | <asignar>
<condición> → **if (** <comparación> **)** <órdenes> **end**
| **if (** <comparación> **)** <órdenes> **else** <órdenes> **end**
<comparación> → <operador> < condición_op> <operador>
<condición_op> → = | <= | >= | <> | < | >
<operador> → <identificador> | <números>
<números> → <número_entero> | <número_real>
<número_entero> → <número> | <número> <número_entero>
<número> → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
<número_real> → <número_entero> **.** <número_entero>
<bucle_while> → **while (** <comparación> **)** <órdenes> **endwhile**
<asignar> → <identificador> **:=** <expresión_arit>
<expresión_arit> → **(** <expresión_arit> <operador_arit> <expresión_arit> **)**
| <identificador>
| <números>
| <expresión_arit> <operador_arit> <expresión_arit>
<operador_arit> → + | * | - | /

Después tenemos que identifiar los elementos que tienen recursividad y los elementos que no tienen recursividad, en la siguiente imagen muestro con azul los NO recursivos y con rosa los SI recursivos:

<programa> → **begin** <declaraciones> <órdenes> **end**
<declaraciones> → <declaración> **;** | <declaración>**;** <declaraciones>
<declaración> → <tipo> <lista_variables>
<tipo> → **entero** | **real**
<lista_variables> → <identificador> | <identificador> **,** <lista_variables>
<identificador> → <letra> | <letra> <resto_letras>
<letra> → **A | B | C | ... | Z | a | b | c | ... | z**
<resto_letras> → <letraN> | <letraN> <resto_letras>
<letraN> → **A | B | C | ... | Z | a | b | c | ... | z | 0 | 1 | ... | 9**
<órdenes> → <orden> **;** | <orden> **;** <órdenes>
<orden> → <condición> | <bucle_while> | <asignar>
<condición> → **if (** <comparación> **)** <órdenes> **end**
| **if (** <comparación> **)** <órdenes> **else** <órdenes> **end**
<comparación> → <operador> < condición_op> <operador>
<condición_op> → = | <= | >= | <> | < | >
<operador> → <identificador> | <números>
<números> → <número_entero> | <número_real>
<número_entero> → <número> | <número> <número_entero>
<número> → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
<número_real> → <número_entero> **.** <número_entero>
<bucle_while> → **while (** <comparación> **)** <órdenes> **endwhile**
<asignar> → <identificador> **:=** <expresión_arit>
<expresión_arit> → **(** <expresión_arit> <operador_arit> <expresión_arit> **)**
| <identificador>
| <números>
| <expresión_arit> <operador_arit> <expresión_arit>
<operador_arit> → + | * | - | /

Identificando los recursivos como alfas y los no recursivos como betas, lo siguiente sería colocarlos de la siguiente manera:



Donde A' sería lo que represento en la gramática como "r_", el cual produce las producciones que tenían recursividad en la gramática original, más su respectivo vacío, quedando de la siguiente manera:

<programa> → begin <declaraciones> <órdenes> end

<declaraciones> → <declaración>; <r_declaraciones>

<r_declaraciones> → <declaración>; <r_declaraciones> | ϵ

<declaración> → <tipo> <lista_variables>

<tipo> → entero | real

<lista_variables> → <identificador> <r_lista_variables>

<r_lista_variables> → , <lista_variables> | ϵ

<identificador> → <letra> | <letra> <resto_letras>

<letra> → A|B|C|...|Z|a|b|c|...|z|0|1|...|9

<órdenes> → <orden>; <r_órdenes>

<r_órdenes> → <orden>; <r_órdenes> | ϵ

<comparación> → <operador> <condición_op> <operador>

<condición_op> → = | <= | >= | <> | < | >

<operador> → <identificador> | <números>

<números> → <número_entero> | <número_real>

<número_entero> → <número> | <número> <número_entero>

<número> → 0|1|2|3|4|...|9

<numero_real> → <número_entero> | <número_real>

<bucle_while> → while (<comparación>) <órdenes> end while

<asignar> → <identificador> := <expresión_arit>

<expresión_arit> → (<expresion_arit> <operador_arit> <expresion_arit>) <exp_arit> | <identificador> <exp-arit> | <números> <exp_arit>

<exp-arit> → <operador_arit> <expresion_arit> <exp-arit> | ϵ

<operador_arit> → + | * | - | /

Conclusiones:

Lo más importante para realizar este ejercicio es poner atención en donde se produce la ambigüedad, para poder realizar los pasos necesarios para eliminarla de la manera correcta sin crear reglas innecesarias o que compliquen la gramática.