PRÁCTICA 3

Abel Eduardo Robles Lázaro

20 DE MARZO DE 2023

UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Introducción:

Esta práctica requiere que convirtamos la gramática libre de contexto en una gramática libre de ambigüedades y libre de recursividad por la izquierda.

Gramática 2.1:

<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> → **+** | ***** | **-** | **/**

Desarrollo:

El primer paso es identificar qué reglas contienen ambigüedades y recursividad por la izquierda:

1 <programa> → **begin** <declaraciones> <órdenes> **end**
2 <declaraciones> → <declaración> **;** | <declaración>**;** <declaraciones>
3 <declaración> → <tipo> <lista_variables>
4 <tipo> → **entero** | **real**
5 <lista_variables> → <identificador**>** | <identificador**>** **,** <lista_variables>
6 <identificador> → <letra> | <letra> <resto_letras>
7 <letra> → **A | B | C | ... | Z | a | b | c | ... | z**
8 <resto_letras> → <letraN> | <letraN> <resto_letras>
9 <letraN> → **A | B | C | ... | Z | a | b | c | ... | z | 0 | 1 | ... | 9**
10 <órdenes> → <orden> **;** | <orden> **;** <órdenes>
11 <orden> → <condición> | <bucle_while> | <asignar>
12 <condición> → **if (** <comparación> **)** <órdenes> **end**
13 | **if (** <comparación> **)** <órdenes> **else** <órdenes> **end**
14 <comparación> → <operador> < condición_op> <operador>
15 <condición_op> → = | <= | >= | <> | < | >
16 <operador> → <identificador**>** | <números>
17 <números> → <número_entero> | <número_real>
18 <número_entero> → <número> | <número> <número_entero>
19 <número> → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
20 <número_real> → <número_entero> **.** <número_entero>
21 <bucle_while> → **while (** <comparación> **)** <órdenes> **endwhile**
22 <asignar> → <identificador> **:=** <expresión_arit>
23 <expresión_arit> → **(** <expresión_arit> <operador_arit> <expresión_arit> **)**
24 | <identificador>
25 | <números>
26 | <expresión_arit> <operador_arit> <expresión_arit>
27 <operador_arit> → + | * | - | /

Reglas adecuadas
Reglas ambiguas
Reglas recursivas por la izquierda

Empezando con la regla #2:

2 <declaraciones> → <declaración> ; | <declaración>; <declaraciones>

Ésta regla cuenta con ambigüedad porque puede tener más de una derivación por la izquierda, y, por lo tanto, más de un árbol de derivación (<declaración>), la solución propuesta elimina esta ambigüedad al agregar una nueva regla que puede ser una cadena vacía.

<declaraciones> → <declaración> ; <sig_declaraciones>
<sig_declaraciones> → <declaración> ; <sig_declaraciones> | ε

Regla #5:

5 <lista_variables> → <identificador> | <identificador> , <lista_variables>

Si prestamos atención, la regla cuenta con el mismo problema que la anterior, por lo que la solución es similar. La solución:

<lista_variables> → <identificador> <sig_lista_variables>
<sig_lista_variables> → , <lista_variables> | ε

Regla #10:

10 <órdenes> → <orden> ; | <orden> ; <órdenes>

Mismo problema que las dos anteriores, misma solución:

<órdenes> → <orden> ; <sig_órdenes>
<sig_órdenes> → <orden> ; <sig_órdenes> | ε

Regla #12:

```
12 <condición>      → if ( <comparación> ) <órdenes> end
13                  | if ( <comparación> ) <órdenes> else <órdenes> end
```

Al llegar a esta regla nos damos cuenta que el problema esta vez es diferente; la regla genera dos ramificaciones muy similares. Para solucionar este problema se agregó una regla nueva (similar a las soluciones anteriores) en la que el cierre de las condiciones se agrega por separado y la parte repetitiva de la regla se deja sola dentro de la regla original

```
<condición>       → if ( <comparación> ) <órdenes> <sig_condición>
<sig_condición>   → end | else <órdenes> end
```

Regla #23:

```
23 <expresión_arit> → ( <expresión_arit> <operador_arit> <expresión_arit> )
24                  | <identificador>
25                  | <números>
26                  | <expresión_arit> <operador_arit> <expresión_arit>
```

Esta regla es la primera y única de todo el conjunto que cuenta con recursividad por la izquierda, por lo que es importante eliminarla para poder generar un analizador sintáctico funcional.

Para lograrlo, se agregó una regla adicional con nombre similar a la original, se eliminaron las iteraciones en las que se comenzaba con “<expresión_arit>”, ya que estas generaban la recursividad. Se soluciona al agregar las iteraciones eliminadas sobre la nueva regla y agregando además una iteración de cadena vacía; también se eliminó ambigüedad al agregar esa nueva línea.

```
<expresión_arit>  → ( <expresión_arit> <operador_arit> <expresión_arit> )
                      <exp_arit>
                    | <identificador> <exp_arit>
                    | <números> <exp_arit>
<exp_arit>        → <operador_arit> <expresión_arit> <exp_arit> | ε
```

Conclusiones:

Es necesario generar una GLC adecuada antes de comenzar con el desarrollo del analizador sintáctico para evitar problemas a futuro.