2023

# EXREGAN USAC

GRAMÁTICA

ALDO SAÚL VÁSQUEZ MOREIRA

**Para la solución del lenguaje solicitado se utilizó una serie de símbolos terminales y no terminales, los cuales serán detallados a continuación:**

- **No terminales**
    - **INICIAR**
    - **INSTRUCCIONES**
    - **CONJUNTOS**
    - **NOTACION**
    - **CONJUNTO_NUMEROS**
    - **CONJUNTO_LETRAS**
    - **CONJUNTO_CARACTERES_ESPECIALES**
    - **EXPRESIONES**
    - **EXPRESION**
    - **DEFINICION_EXPR**
    - **LEXEMAS**
    - **LEXEMA**
    - **IDENTIFICADOR_CONJUNTO**
    - **IDENTIFICADOR_EXPRESION**
    - **PUNTO_ER**
    - **BARRA_VERTICAL_ER**
    - **ASTERISCO_ER**
    - **SUMA_ER**
    - **INTERROGACION_ER**

- **Terminales**
    - **MENOR_QUE = <**
    - **MAYOR_QUE = >**
    - **ADMIRACION = !**
    - **DIAGONAL= /**
    - **LLAVE_IZQUIERDA = {**
    - **LLAVE_DERECHA = }**
    - **PORCENTAJE = %**
    - **PUNTO_COMA = ;**
    - **DOS_PUNTOS = :**
    - **MENOS = -**
    - **COMA = ,**
    - **INTERROGACIÓN = ?**
    - **SUMA = +**
    - **ASTERISCO = ***
    - **TILDE = ~**
    - **BARRA_VERTICAL = |**

- **RESERVADA_CONJUNTO = “Conj” o “CONJ”**
- **SALTO_LINEA = \n**
- **COMILLAS = “**
- **COMILLA_SIMPLE = \’**
- **COMILLA_DOBLE = \”**
- **NUMERO = Ejemplo (145)**
- **NUMERO_DECIMAL = Ejemplo (25.38)**
- **LETRA = Ejemplo (A)**
- **IDENTIFICADOR = Ejemplo (aldo_2023)**
- **CARÁCTER_ESPECIAL = Códigos ascii desde el 32 al 125**
- **CADENA = Ejemplo “esta es la gramática para compi1”**

**Dado que ya conocemos los símbolos terminales y no terminales utilizados, a continuación se detallará cada una de las producciones realizadas, así como su función dentro del sistema:**

```
INICIAR ::= LLAVE_IZQUIERDA INSTRUCCIONES LLAVE_DERECHA;

INSTRUCCIONES ::= CONJUNTOS EXPRESIONES PORCENTAJE PORCENTAJE PORCENTAJE PORCENTAJE LEXEMAS;
```

**La producción inicial o principal INICIAR define la estructura, separando el archivo en tres partes principales:**

**{**

**INSTRUCCIONES**

**}**

**Seguidamente observamos que la producción de INSTRUCCIONES separa el archivo en conjuntos, luego expresiones, luego porcentajes y lexemas. Obteniendo así, lo siguiente:**

**{**

**CONJUNTOS**

**EXPRESIONES**

**%%**

**%%**

**LEXEMAS**

**}**

**Ahora bien, veremos en qué consiste cada una de estas producciones para comprender mejor cómo funciona la gramática.**

```
CONJUNTOS ::= CONJUNTO CONJUNTOS
    | CONJUNTO;

CONJUNTO ::= RESERVADA_CONJUNTO DOS_PUNTOS IDENTIFICADOR_CONJUNTO MENOS MAYOR_QUE NOTACION PUNTO_COMA
```

**CONJUNTO consiste en una gramática recursiva por la derecha (LL), la cual consiste de la palabra reservada conjunto, dos puntos, un identificador, menos, mayor que, una notación y el punto y coma. Obteniendo lo siguiente:**

**CONJ: IDENTIFICADOR -> NOTACION;**

**La notación del conjunto consiste en:**

```
NOTACION ::= LETRA:a TILDE LETRA:b
    {:
        ManipuladorData.listaDeConjuntos.get(ubicacionConjunto(id_conjunto)).getElementos().add(a);
        char inicio = ManipuladorData.listaDeConjuntos.get(ubicacionConjunto(id_conjunto)).obtenerInicio();
        char fin = b.charAt(0);
        for (int i = (int) inicio + 1; i < (int) fin + 1; i++){
            ManipuladorData.listaDeConjuntos.get(ubicacionConjunto(id_conjunto)).getElementos().add(Character.toStr
        }
    :}
    |NUMERO:a TILDE NUMERO:b
    {:
        ManipuladorData.listaDeConjuntos.get(ubicacionConjunto(id_conjunto)).getElementos().add(a);
        int inicio = Integer.parseInt(ManipuladorData.listaDeConjuntos.get(ubicacionConjunto(id_conjunto)).obtenerI
        int fin = Integer.parseInt(b);
        for (int i = inicio + 1; i < fin + 1; i++){
            ManipuladorData.listaDeConjuntos.get(ubicacionConjunto(id_conjunto)).getElementos().add(Integer.toStrin
        }
    :}
    |CARACTER_ESPECIAL:a TILDE CARACTER_ESPECIAL:b
    {:
        ManipuladorData.listaDeConjuntos.get(ubicacionConjunto(id_conjunto)).getElementos().add(a);
        char inicio = ManipuladorData.listaDeConjuntos.get(ubicacionConjunto(id_conjunto)).obtenerInicio();
        char fin = b.charAt(0);
        for (int i = (int) inicio + 1; i < (int) fin + 1; i++){
            ManipuladorData.listaDeConjuntos.get(ubicacionConjunto(id_conjunto)).getElementos().add(Character.toStr
```

```
    |CONJUNTO_NUMEROS
    |CONJUNTO_LETRAS
    |CONJUNTO_CARACTERES_ESPECIALES
```

**La función es verificar la correcta declaración de los conjuntos así como determinar los elementos pertenecientes a dicho conjunto.**

**Ahora, continuaremos con la declaración de las expresiones regulares:**

```
EXPRESION ::= IDENTIFICADOR_EXPRESION MENOS MAYOR_QUE DEFINICION_EXPR PUNTO_COMA:a
```

**Consiste en un identificador, menos, mayor que, definición y punto y coma.**

**IDENTIFICADOR -> DEFINICION;**

**El identificador consiste en un ID cualquiera, lo interesante aquí es la parte de la definición de la expresión regular, ya que estas pueden variar mucho.**

**Por lo cual , para esto se utilizó la siguiente producción:**

```
DEFINICION_EXPR ::= PUNTO_ER DEFINICION_EXPR DEFINICION_EXPR
    | BARRA_VERTICAL_ER DEFINICION_EXPR DEFINICION_EXPR
    | ASTERISCO_ER DEFINICION_EXPR
    | SUMA_ER DEFINICION_EXPR
    | INTERROGACION_ER DEFINICION_EXPR
    | CARACTER:a
    {:
        String no_comillas = a.replace("\"","");
        ManipuladorData.listaDeExpresiones.get(ubicacionExpresionRegular(id_expresion)).nodoInsertar("valor", no_co
    :}
    | LLAVE_IZQUIERDA IDENTIFICADOR:a LLAVE_DERECHA
    {:
        ManipuladorData.listaDeExpresiones.get(ubicacionExpresionRegular(id_expresion)).nodoInsertar("valor", a);
    :}
    ;
```

**La cual reconoce cada una de las operaciones conocidas para una expresión regular, como:**

**Concatenación .ab**

**Or |ab**

**Kleene *a**

**Uno o más +a**

**Cero o uno ?a**

**Es decir, reconoce operaciones unarias o binarias por medio de producciones auxiliares, según sea el caso.**

**Finalmente, la parte de los lexemas se reconoce por medio de la siguiente producción:**

```
LEXEMAS ::= LEXEMA LEXEMAS
    |LEXEMA;

LEXEMA ::= IDENTIFICADOR DOS_PUNTOS CADENA:a PUNTO_COMA
    {:
        ManipuladorData.listaDeExpresiones.forEach(er -> {
            String palabra = a.replace("\"","");
            ManipuladorData.listaDeExpresiones.get(ubicacionExpresionRegular(er.getId())).cadenaInsertar(palabra);
        });
    :}
```

**Esta consiste de un identificador, dos puntos, una cadena y punto y coma.**

**IDENTIFICADOR: CADENA;**

**IDENTIFICADOR -> “cadena1”, “cadenaUno”, “cadena_1”, etc.**

**CADENA -> “esta es una cadena”, “15.35”, “cadena (expresión)”, etc.**

**Así, la función de esta parte es reconocer la declaración de una expresión para luego almacenarla en una lista para su posterior validación.**

**Ese es el funcionamiento básico de la gramática, tomando en cuenta que la sintaxis puede variar. En este caso se utilizó JAVA CUP como el generador del parser. Asimismo, se utilizó una gramática recursiva por la derecha.**