Analizador léxico y sintáctico para el lenguaje MIO

Hernán J. Cervera Manzanilla

9 de diciembre de 2018

Índice de contenidos

Índice de figuras

1 Lenguaje MIO

El lenguaje está definido por la gramática de la figura 1, en la cual el símbolo inicial es <PROG>. Los símbolos con llaves cuadradas representan cadenas especiales que son terminales; se muestran en la figura 2.

<PROG> → PROGRAMA[id]<SENTS>FINPROG
<SENTS> → <SENT><SENTS>
<SENTS> → <SENT>
<SENT> → [id]=<ELEM>[op_ar]<ELEM>
<SENT> → [id]=<ELEM>
<SENT> → SI<COMPARA>ENTONCES<SENTS>SINO<SENTS>FINSI
<SENT> → SI<COMPARA>ENTONCES<SENTS>FINSI
<SENT> → REPITE<ELEM>VECES<SENTS>FINREP
<SENT> → IMPRIME<ELEM>
<SENT> → IMPRIME[txt]
<SENT> → LEE[id]
<SENT> → #[comentario]
<ELEM> → [id]
<ELEM> → [val]
<COMPARA> → [id][op_rel]<ELEM>

Figura 1: Gramática original para el lenguaje MIO

Símbolo	Descripción de cadenas representadas
[id]	Son cadenas de caracteres alfanuméricos que tienen una longitud de hasta 16 caracteres. Inician siempre con un carácter alfabético y son sensibles a mayúsculas y minúsculas.
[val]	Son cadenas de caracteres alfanuméricos. Inician con la subcadena `0x` y le sigue una secuencia cualquiera de los caracteres `0-9` y `A-F`. Representan números en hexadecimal.
[txt]	Son cadenas de caracteres alfanuméricos que pueden contener espacios vacíos.
[op_ar]	Uno de los siguientes símbolos: `+, -, *, /`.
[op_rel]	Uno de los siguientes símbolos: `<, >, <=, >=, ==, !=`.

Figura 2: Símbolos terminales especiales

2 Analizador léxico

2.1 Funcionamiento

El objetivo del analizador léxico es identificar los tokens que componen un archivo `MIO` de entrada. La salida consiste en dos archivos, uno de tipo `LEX`, que contiene un token por línea; otro, de tipo `SIM`, que contiene un listado enumerado de valores numéricos, de texto e identificadores que aparecen en el archivo `MIO`. El algoritmo empleado para obtener el contenido de estos dos archivos es el siguiente:

1. Extraer en un vector las líneas del archivo `MIO`.
2. Extraer en una matriz los lexemas de cada línea.

3. Evaluar cada lexema para determinar su tipo. Aquí se realizan dos acciones: se almacena el tipo identificado en un vector y, si el tipo es numérico, de texto o un identificador, se almacena en un mapa respectivo como llave de una entrada (el valor de la entrada es una etiqueta autogenerada).

Si el programa se ejecuta exitosamente, los archivos son generados; en caso contrario, se reporta la línea del primer error del archivo `MIO` y no se generan los archivos temporales. Si los archivos habían sido generados por una ejecución exitosa previa del programa, éstos son eliminados.

2.2 Ejecución

La aplicación es proporcionada como el JAR ejecutable `analex`. El programa requiere de un archivo `MIO`. Para proporcionarlo, debe simplemente colocarse el archivo en el mismo directorio donde se ubica `analex`. Para ejecutar el analizador léxico, se escribe lo siguiente en la línea de comandos:

`java -jar analex.jar` ⟨*nombre del archivo*⟩`.mio`.

2.3 Ejemplos de ejecución

2.3.1 Caso de éxito

Archivo MIO de entrada

```
1  # Programa que calcula el factorial de un número
2  PROGRAMA factorial
3  # VarX acumula los productos por iteración
4  VarX = 0x1
5  # VarY contiene el iterador del factor
6  VarY = 0x0
7  LEE Num
8  # Aplica Num! = 1 * 2 * 3 * ... * Num
9  REPITE Num VECES
10 VarY = VarY + 0x1
11 VarX = VarX * VarY
12 FINREP
13 IMPRIME "Factorial de "
14 IMPRIME Num
15 IMPRIME " es "
16 IMPRIME VarX
17 FINPROG
```

Figura 3: factorial.mio

```
1  PROGRAMA
2  [id]
3  [id]
4  =
5  [val]
6  [id]
7  =
8  [val]
9  LEE
10 [id]
11 REPITE
12 [id]
13 VECES
14 [id]
15 =
16 [id]
17 [op_ar]
18 [val]
19 [id]
20 =
21 [id]
22 [op_ar]
23 [id]
24 FINREP
25 IMPRIME
26 [txt]
27 IMPRIME
28 [id]
29 IMPRIME
30 [txt]
31 IMPRIME
32 [id]
33 FINPROG
```

Figura 4: factorial.lex

```
1  IDS
2  factorial,ID01
3  VarX,ID02
4  VarY,ID03
5  Num,ID04
6
7  TXT
8  "Factorial de ",TX01
9  " es ",TX02
10
11 VAL
12 0x1,1
13 0x0,0
```

Figura 5: factorial.lex

2.3.2 Caso de fallo

Archivo MIO de entrada

```
1 PROGRAMA contar
2 VarX = 0x00
3 SI varx < 0x0SA ENTONCES
4 VarX = VarX + 0x01
5 IMPRIME VarX
6 FINSI
7 FINPROG
```

Figura 6: contar.mio

Salida en la consola

```
Invalid token at line: 3
```

Figura 7: Fallo analex (contar.mio)

3 Analizador sintáctico

3.1 Funcionamiento

El analizador sintáctico es predictivo y tiene una implementación recursiva. Para lograr esto, la gramática tuvo que ser convertida a una equivalente de tipo LL(1); la siguiente herramienta fue usada para la conversión: `http://smlweb.cpsc.ucalgary.ca/`. El símbolo inicial se mantiene como <PROG> y los símbolos de llaves cuadradas también conservan su definición (figura 2 en la página 1).

<PROG> → PROGRAMA[id]<SENTS>FINPROG
<SENTS> → <SENT><FSENT>
<FSENT> → <SENTS>
<FSENT> → ε
<SENT> → [id]<FID>
<SENT> → SI<FSI>
<SENT> → REPITE<ELEM>VECES<SENTS>FINREP
<SENT> → IMPRIME<FIMPRIME>
<SENT> → LEE[id]
<SENT> → #[comentario]
<FID> → =<FE>
<FE> → <ELEM><FELEM>
<FELEM> → [op_ar]<ELEM>
<FELEM> → ε
<FSI> → <COMPARA><FCOMPARA>
<FCOMPARA> → ENTONCES<FENTONCES>
<FENTONCES> → <SENTS><FSENTS>
<FSENTS> → SINO<SENTS>FINSI
<FSENTS> → ε
<FIMPRIME> → <ELEM>
<FIMPRIME> → [txt]
<ELEM> → [id]
<ELEM> → [val]
<COMPARA> → [id][op_rel]<ELEM>

Figura 8: Gramática LL(1) para el lenguaje MIO.

Si el programa se ejecuta sin errores, se reporta, con el mensaje `The word is accepted`, que se aceptó la cadena extraída del archivo `LEX`. En caso contrario, se reporta, con el mensaje `ERROR: No match`, que hubo un error, pero no se reporta qué tipo de error ocurrió.

3.2 Ejecución

La aplicación es proporcionada como el JAR ejecutable `anasin`. El programa requiere de un archivo `LEX`, generado por la ejecución del `analex` sobre un archivo `MIO`. Para proporcionarlo, debe simplemente colocarse el archivo en el mismo directorio donde se ubica `anasin`. Para ejecutar el analizador léxico, se escribe lo siguiente en la línea de comandos:

`java -jar anasin.jar` ⟨*nombre del archivo*⟩`.lex`.

3.3 Ejemplos de ejecución

3.3.1 Caso de éxito

Archivo LEX de entrada

```
1  PROGRAMA
2  [id]
3  LEE
4  [id]
5  LEE
6  [id]
7  SI
8  [id]
9  [op_rel]
10 [id]
11 ENTONCES
12 IMPRIME
13 [id]
14 SINO
15 IMPRIME
16 [id]
17 FINSI
18 FINPROG
```

Figura 9: max.lex

Salida en la consola

```
The word is accepted
```

Figura 10: Éxito anasin (max.lex)

3.3.2 Caso de fallo

Archivo LEX de entrada

```
1  PROGRAMA
2  [id]
3  LEE
4  [id]
5  LEE
6  [id]
7  SI
8  [id]
9  [op_rel]
10 [id]
11 ENTONCES
12 IMPRIME
13 [id]
14 SINO
15 IMPRIME
16 [id]
17 FINSI
18 [id]
19 FINPROG
```

Figura 11: min.lex

Salida en la consola

```
ERROR: No match
```

Figura 12: Fallo anasin (min.lex)