Analizador Sintáctico

Flores, D. E.[1]
[1]Facultad de Ingeniería
Universidad Nacional de Cuyo

26 de abril de 2022

1. Introducción

En el presente informe se expone características del analizador sintáctico desarrollado. El mismo, desarrollado en Java, pretende emular la gramática en formato EBNF otorgada por la cátedra. La gramática, además, ha sido transformada a BNF y se aplicó los correspondientes algoritmos de *eliminación de recursión inmediata a izquierda* y *factorización a izquierda*

2. Diseño

Existe una única clase principal *SyntacticAnalyzer* que contiene dos categorías de métodos: aquellos que representan cada no terminal de la gramática BNF y los utilizados para *matchear* los terminales y utilidades relacionadas.

Por otro lado, el código se encuentra comentado principalmente en los métodos relacionados con el *match* ya que, a criterio del desarrollador, son los que nada familiarizado se puede encontrar el lector. Esto no debería suceder con los métodos que representan no terminales, cuyo comportamiento se describe en la gramatica BNF. El contenido se exportó con Doxygen y es accesible desde *./doc/index.html*.

2.1. Errores

Existe una única clase que hereda de *Exception* y se relaciona directamente con el analizador sintáctico: *SyntacticErrorException*. Esta se lanza cuando no es posible matchear el nombre del token con el terminal esperado por la gramática. Puede darse esta situación cuando los nombres difieren o cuando se acabó el archivo y no hubo match con éxito ($).

Los cuerpos de descripción de error siguen como el formato siguiente: *expected nombre_token_correcto but found token_actual_incorrecto*.

En algunos casos particulares puede haber una descripción más general, como: *Members definition not found*.

Adicionalmente, en ciertos métodos se agrega un mensaje inicial que podría mostrarse al usuario como: *In class Main: constructor: expected rbrace but found lbrace*

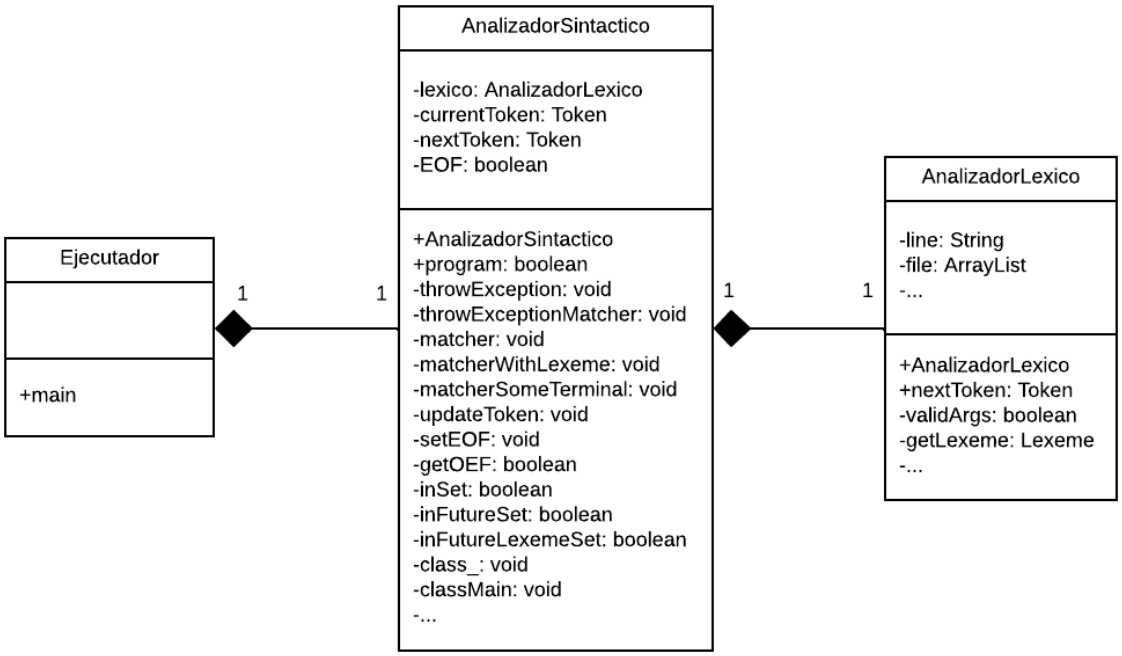

Figura 1. Diagrama de clases para el proyecto Analizador Léxico

3. Transformación de EBNF a BNF

Para la transformación de la gramática EBNF a BNF se utilizó las reglas proporcionadas por la cátedra. Se tuvo especial cuidado en ciertos no terminales como *<Argumentos-Formales>* y *<Lista-Argumentos-Formales>* con el fin de aplicar correctamente la regla borradora.

```
Program ::= <Clase_> <ClaseMain> <Clase_>

Clase_ ::= <Clase> <Clase_> |
           empty

ClaseMain ::= 'class' 'Main' <Herencia> '{' <Miembro> 'static' 'func'
'void' 'main' '(' ')' <Bloque-Metodo> <Miembro> '}'

Clase ::= 'class' 'idclass' <Herencia> '{' <Miembro_> '}'

Herencia ::= ':' <Tipo> |
             empty

Miembro_ ::= <Miembro> <Miembro_> |
             empty
```

```
Miembro ::= <Atributo> |
            <Constructor> |
            <Metodo>

Constructor ::= 'init' <Argumentos-Formales> <Bloque-Metodo>

Atributo ::= <Visibilidad> 'var' <Tipo> <Lista-Declaracion-Variables> ';'

Metodo ::= <Forma-Metodo> 'func' <Tipo-Metodo> 'id' <Argumentos-Formales>
<Bloque-Metodo>

Visibilidad ::= 'private' |
                empty

Forma-Metodo ::= 'static' |
                 empty

Bloque-Metodo ::= '{' <Decl-Var-Locales_> <Sentencia_> '}'

Decl-Var-Locales_ ::= <Decl-Var-Locales> Decl-Var-Locales_> |
                      empty

Decl-Var-Locales ::= 'var' <Tipo> <Lista-Declaracion-Variables> ';'

Lista-Declaracion-Variables ::= 'id' |
                                'id' ',' <Lista-Declaracion-Variables>

Argumentos-Formales ::= '(' <Lista-Argumentos-Formales> ')' |
                        '(' ')'

Lista-Argumentos-Formales ::= <Argumento-Formal> ','
                                <Lista-Argumentos-Formales> |
                              <Argumento-Formal>

Argumento-Formal ::= <Tipo> 'id'

Tipo-Metodo ::= <Tipo> |
                'void'
```

```
Tipo ::= <Tipo-Primitivo> |
         <Tipo-Referencia> |
         <Tipo-Arreglo>

Tipo-Primitivo ::= 'Bool' |
                   'Int' |
                   'String' |
                   'Char'

Tipo-Referencia ::= 'idclass'

Tipo-Array ::= 'Array' <Tipo-Primitivo>

Sentencia_ ::= <Sentencia> <Sentencia_> |
                 empty

Sentencia ::= ';' |
              <Asignacion> ';' |
              <Sentencia-Simple> ';' |
              'if' '(' <Expresion> ')' <Sentencia> |
              'if' '(' <Expresion> ')' <Sentencia> 'else' <Sentencia> |
              'while' '(' <Expresion> ')' <Sentencia> |
              <Bloque> |
              'return' <Return> ';'

Return ::= <Expresion> |
           empty

Bloque ::= '{' <Sentencia_> '}'

Asignacion ::= <AccesoVar-Simple> '=' <Expresion> |
               <AccesoSelf-Simple> '=' <Expresion>

AccesoVar-Simple ::= 'id' <Encadenado-Simple_> |
                     'id' '[' <Expresion> ']'

AccesoSelf-Simple ::= 'self' <Encadenado-Simple_>

Encadenado-Simple_ ::= <Encadenado-Simple> <Encadenado-Simple_> |
```

```
                        empty

Encadenado-Simple ::= '.' 'id'

Sentencia-Simple ::= '(' <Expresion> ')'

Expresion ::= <ExpOr>

ExpOr ::= <ExpOr> '||' <ExpAnd> |
          <ExpAnd>

ExpAnd ::= <ExpAnd> '&&' <ExpIgual> |
           <ExpIgual>

ExpIgual ::= <ExpIgual> <OpIgual> <ExpCompuesta> |
              <ExpCompuesta>

ExpCompuesta ::= <ExpAd> <OpCompuesto> <ExpAd> |
                  <ExpAd>

ExpAd ::= <ExpAd> <OpAd> <ExpMul> |
          <ExpMul>

ExpMul ::= <ExpMul> <OpMul> <ExpUn> |
           <ExpUn>

ExpUn ::= <OpUnario> <ExpUn> |
          <Operando>

OpIgual ::= '==' |
            '!='

OpCompuesto ::= '<' |
                '>' |
                '<=' |
                '>='

OpAd ::= '+' |
         '-'
```

```
OpUnario ::= '+' |
             '-' |
             '!'

OpMul ::= '*' |
          '/' |
          '%'

Operando ::= <Literal> |
             <Primario> <Encadenado>

Literal ::= 'nil' |
            'true' |
            'false' |
            'intLiteral' |
            'stringLiteral' |
            'charLiteral'

Primario ::= <ExpresionParentizada> |
             <AccesoSelf> |
             <AccesoVar> |
             <Llamada-Metodo> |
             <Llamada-Metodo-Estatico> |
             <Llamada-Constructor>

ExpresionParentizada ::= '(' Expresion ')' <Encadenado>

AccesoSelf ::= 'self' <Encadenado>

AccesoVar ::= 'id' <Encadenado>

Llamada-Método ::= 'id' <Argumentos-Actuales> <Encadenado>

Llamada-Método-Estático ::= 'idclass' '.' <Llamada-Metodo> <Encadenado>

Llamada-Constructor ::= 'new' 'idclass' <Argumentos-Actuales> <Encadenado> |
                        'new' <Tipo-Primitivo> '[' <Expresion> ']'
```

```
Argumentos-Actuales ::= '(' <Lista-Expresiones> ')' |
                        '(' ')'

Lista-Expresiones ::= <Expresion> |
                      <Expresion> ',' <Lista-Expresiones>

Encadenado ::= '.' <Llamada-Metodo-Encadenado> |
               '.' <Acceso-Variable-Encadenado> |
               empty

Llamada-Método-Encadenado ::= 'id' <Argumentos-Actuales> <Encadenado>

Acceso-Variable-Encadenado ::= 'id' <Encadenado> |
                               'id' '[' <Expresion> ']'
```

4. Aplicación del primer algoritmo

A la nueva gramática, se le aplicó el algoritmo de eliminación de recursividad inmediata a izquierda. Los resultados pueden verse a continuación:

```
Program ::= <Clase_> <ClaseMain> <Clase_>

Clase_ ::= <Clase> <Clase_> |
           empty

ClaseMain ::= 'class' 'Main' <Herencia> '{' <Miembro> 'static' 'func'
'void' 'main' '(' ')' <Bloque-Metodo> <Miembro> '}'

Clase ::= 'class' 'idclass' <Herencia> '{' <Miembro_> '}'

Herencia ::= ':' <Tipo> |
             empty

Miembro_ ::= <Miembro> <Miembro_> |
             empty

Miembro ::= <Atributo> |
            <Constructor> |
            <Metodo>
```

```
Constructor ::= 'init' <Argumentos-Formales> <Bloque-Metodo>

Atributo ::= <Visibilidad> 'var' <Tipo> <Lista-Declaracion-Variables> ';'

Metodo ::= <Forma-Metodo> 'func' <Tipo-Metodo> 'id' <Argumentos-Formales>
<Bloque-Metodo>

Visibilidad ::= 'private' |
                empty

Forma-Metodo ::= 'static' |
                 empty

Bloque-Metodo ::= '{' <Decl-Var-Locales_> <Sentencia_> '}'

Decl-Var-Locales_ ::= <Decl-Var-Locales> Decl-Var-Locales_> |
                      empty

Decl-Var-Locales ::= 'var' <Tipo> <Lista-Declaracion-Variables> ';'

Lista-Declaracion-Variables ::= 'id' |
                                'id' ',' <Lista-Declaracion-Variables>

Argumentos-Formales ::= '(' <Lista-Argumentos-Formales> ')' |
                        '(' ')'

Lista-Argumentos-Formales ::= <Argumento-Formal> ','
                              <Lista-Argumentos-Formales> |
                             <Argumento-Formal>

Argumento-Formal ::= <Tipo> 'id'

Tipo-Metodo ::= <Tipo> |
                'void'

Tipo ::= <Tipo-Primitivo> |
         <Tipo-Referencia> |
         <Tipo-Arreglo>
```

```
Tipo-Primitivo ::= 'Bool' |
                   'Int' |
                   'String' |
                   'Char'

Tipo-Referencia ::= 'idclass'

Tipo-Array ::= 'Array' <Tipo-Primitivo>

Sentencia_ ::= <Sentencia> <Sentencia_> |
                 empty

Sentencia ::= ';' |
              <Asignacion> ';' |
              <Sentencia-Simple> ';' |
              'if' '(' <Expresion> ')' <Sentencia> |
              'if' '(' <Expresion> ')' <Sentencia> 'else' <Sentencia> |
              'while' '(' <Expresion> ')' <Sentencia> |
              <Bloque> |
              'return' <Return> ';'

Return ::= <Expresion> |
           empty

Bloque ::= '{' <Sentencia_> '}'

Asignacion ::= <AccesoVar-Simple> '=' <Expresion> |
               <AccesoSelf-Simple> '=' <Expresion>

AccesoVar-Simple ::= 'id' <Encadenado-Simple_> |
                     'id' '[' <Expresion> ']'

AccesoSelf-Simple ::= 'self' <Encadenado-Simple_>

Encadenado-Simple_ ::= <Encadenado-Simple> <Encadenado-Simple_> |
                       empty

Encadenado-Simple ::= '.' 'id'
```

```
Sentencia-Simple ::= '(' <Expresion> ')'

Expresion ::= <ExpOr>

ExpOr ::= <ExpAnd> <ExpOr_>

ExpOr_ ::= '||' <ExpAnd> <ExpOr_> |
           empty

ExpAnd ::= <ExpIgual> <ExpAnd_>

ExpAnd_ ::= '&&' <ExpIgual> <ExpAnd_> |
            empty

ExpIgual ::= <Expcompuesta> <ExpIgual_>

ExpIgual_ ::= <OpIgual> <ExpCompuesta> <ExpIgual_> |
              empty

ExpCompuesta ::= <ExpAd> <OpCompuesto> <ExpAd> |
                    <ExpAd>

ExpAd ::= <ExpMul> <ExpAd_>

ExpAd_ ::= <OpAd> <ExpMul> <ExpAd_> |
           empty

ExpMul ::= <ExpUn> <ExpMul_>

ExpMul_ ::= <OpMul> <ExpUn> <ExpMul_> |
            empty

ExpUn ::= <OpUnario> <ExpUn> |
          <Operando>

OpIgual ::= '==' |
            '!='
```

```
OpCompuesto ::= '<' |
                '>' |
                '<=' |
                '>='

OpAd ::= '+' |
         '-'

OpUnario ::= '+' |
             '-' |
             '!'

OpMul ::= '*' |
          '/' |
          '%'

Operando ::= <Literal> |
             <Primario> <Encadenado>

Literal ::= 'nil' |
            'true' |
            'false' |
            'intLiteral' |
            'stringLiteral' |
            'charLiteral'

Primario ::= <ExpresionParentizada> |
             <AccesoSelf> |
             <AccesoVar> |
             <Llamada-Metodo> |
             <Llamada-Metodo-Estatico> |
             <Llamada-Constructor>

ExpresionParentizada ::= '(' Expresion ')' <Encadenado>

AccesoSelf ::= 'self' <Encadenado>

AccesoVar ::= 'id' <Encadenado>
```

```
Llamada-Método ::= 'id' <Argumentos-Actuales> <Encadenado>

Llamada-Método-Estático ::= 'idclass' '.' <Llamada-Metodo> <Encadenado>

Llamada-Constructor ::= 'new' 'idclass' <Argumentos-Actuales> <Encadenado> |
                        'new' <Tipo-Primitivo> '[' <Expresion> ']'

Argumentos-Actuales ::= '(' <Lista-Expresiones> ')' |
                        '(' ')'

Lista-Expresiones ::= <Expresion> |
                      <Expresion> ',' <Lista-Expresiones>

Encadenado ::= '.' <Llamada-Metodo-Encadenado> |
               '.' <Acceso-Variable-Encadenado> |
               empty

Llamada-Método-Encadenado ::= 'id' <Argumentos-Actuales> <Encadenado>

Acceso-Variable-Encadenado ::= 'id' <Encadenado> |
                               'id' '[' <Expresion> ']'
```

5. Aplicación del segundo algoritmo

Finalmente, se aplicó el algoritmo de factorización cuyo resultado fue:

```
Program ::= <Clase_> <ClaseMain> <Clase_>

Clase_ ::= <Clase> <Clase_> |
           empty

ClaseMain ::= 'class' 'Main' <Herencia> '{' <Miembro> 'static' 'func'
'void' 'main' '(' ')' <Bloque-Metodo> <Miembro> '}'

Clase ::= 'class' 'idclass' <Herencia> '{' <Miembro_> '}'

Herencia ::= ':' <Tipo> |
             empty
```

```
Miembro_ ::= <Miembro> <Miembro_> |
             empty

Miembro ::= <Atributo> |
            <Constructor> |
            <Metodo>

Constructor ::= 'init' <Argumentos-Formales> <Bloque-Metodo>

Atributo ::= <Visibilidad> 'var' <Tipo> <Lista-Declaracion-Variables> ';'

Metodo ::= <Forma-Metodo> 'func' <Tipo-Metodo> 'id' <Argumentos-Formales>
<Bloque-Metodo>

Visibilidad ::= 'private' |
                empty

Forma-Metodo ::= 'static' |
                 empty

Bloque-Metodo ::= '{' <Decl-Var-Locales_> <Sentencia_> '}'

Decl-Var-Locales_ ::= <Decl-Var-Locales> Decl-Var-Locales_> |
                      empty

Decl-Var-Locales ::= 'var' <Tipo> <Lista-Declaracion-Variables> ';'

Lista-Declaracion-Variables ::= 'id' <Lista-Declaracion-Variables-F>

Lista-Declaracion-Variables-F ::= empty |
                                  ',' <Lista-Declaracion-Variables>

Argumentos-Formales ::= '(' <Argumentos-Formales-F>

Argumentos-Formales-F ::= <Lista-Argumentos-Formales> ')' |
                          ')'

Lista-Argumentos-Formales ::= <Argumento-Formal>
                              <Lista-Argumentos-Formales-F>
```

```
Lista-Argumentos-Formales-F ::= ',' <Lista-Argumentos-Formales> |
                                empty

Argumento-Formal ::= <Tipo> 'id'

Tipo-Metodo ::= <Tipo> |
                'void'

Tipo ::= <Tipo-Primitivo> |
         <Tipo-Referencia> |
         <Tipo-Arreglo>

Tipo-Primitivo ::= 'Bool' |
                   'Int' |
                   'String' |
                   'Char'

Tipo-Referencia ::= 'idclass'

Tipo-Array ::= 'Array' <Tipo-Primitivo>

Sentencia_ ::= <Sentencia> <Sentencia_> |
                empty

Sentencia ::= ';' |
              <Asignacion> ';' |
              <Sentencia-Simple> ';' |
              'if' '(' <Expresion> ')' <Sentencia> <Sentencia-F> |
              'while' '(' <Expresion> ')' <Sentencia> |
              <Bloque> |
              'return' <Return> ';'

Sentencia-F ::= empty |
                'else' <Sentencia>

Return ::= <Expresion> |
           empty
```

```
Bloque ::= '{' <Sentencia_> '}'

Asignacion ::= <AccesoVar-Simple> '=' <Expresion> |
               <AccesoSelf-Simple> '=' <Expresion>

AccesoVar-Simple ::= 'id' <AccesoVar-Simple-F>

AccesoVar-Simple-F ::= <Encadenado-Simple_> |
                       '[' <Expresion> ']'

AccesoSelf-Simple ::= 'self' <Encadenado-Simple_>

Encadenado-Simple_ ::= <Encadenado-Simple> <Encadenado-Simple_> |
                       empty

Encadenado-Simple ::= '.' 'id'

Sentencia-Simple ::= '(' <Expresion> ')'

Expresion ::= <ExpOr>

ExpOr ::= <ExpAnd> <ExpOr_>

ExpOr_ ::= '||' <ExpAnd> <ExpOr_> |
           empty

ExpAnd ::= <ExpIgual> <ExpAnd_>

ExpAnd_ ::= '&&' <ExpIgual> <ExpAnd_> |
            empty

ExpIgual ::= <Expcompuesta> <ExpIgual_>

ExpIgual_ ::= <OpIgual> <ExpCompuesta> <ExpIgual_> |
              empty

ExpCompuesta ::= <ExpAd> <ExpCompuesta-F>

ExpCompuesta-F ::= <OpCompuesto> <ExpAd> |
```

```
                    empty

ExpAd ::= <ExpMul> <ExpAd_>

ExpAd_ ::= <OpAd> <ExpMul> <ExpAd_> |
           empty

ExpMul ::= <ExpUn> <ExpMul_>

ExpMul_ ::= <OpMul> <ExpUn> <ExpMul_> |
            empty

ExpUn ::= <OpUnario> <ExpUn> |
          <Operando>

OpIgual ::= '==' |
            '!='

OpCompuesto ::= '<' |
                '>' |
                '<=' |
                '>='

OpAd ::= '+' |
         '-'

OpUnario ::= '+' |
             '-' |
             '!'

OpMul ::= '*' |
          '/' |
          '%'

Operando ::= <Literal> |
             <Primario> <Encadenado>

Literal ::= 'nil' |
            'true' |
```

```
            'false' |
            'intLiteral' |
            'stringLiteral' |
            'charLiteral'

Primario ::= <ExpresionParentizada> |
             <AccesoSelf> |
             'id' <Primario-ID> |
             <Llamada-Metodo-Estatico> |
             <Llamada-Constructor>

ExpresionParentizada ::= '(' Expresion ')' <Encadenado>

AccesoSelf ::= 'self' <Encadenado>

Primario-ID ::= <AccesoVar> |
                <Llamada-Metodo>

AccesoVar ::= <Encadenado>

Llamada-Método ::= <Argumentos-Actuales> <Encadenado>

Llamada-Método-Estático ::= 'idclass' '.' 'id' <Llamada-Metodo>
<Encadenado>

Llamada-Constructor ::= 'new' <Llamada-Constructor-F>

Llamada-Constructor-F ::= 'idclass' <Argumentos-Actuales> <Encadenado> |
                          <Tipo-Primitivo> '[' <Expresion> ']'

Argumentos-Actuales ::= '(' <Argumentos-Actuales-F>

Argumentos-Actuales-F ::= <Lista-Expresiones> ')' |
                          ')'

Lista-Expresiones ::= <Expresion> <Lista-Expresiones-F>

Lista-Expresiones-F ::= empty |
                        ',' <Lista-Expresiones>
```

```
Encadenado ::= '.' 'id' <Encadenado-F> |
               empty

Encadenado-F ::= <Llamada-Metodo-Encadenado> |
                 <Acceso-Variable-Encadenado>

Llamada-Método-Encadenado ::= <Argumentos-Actuales> <Encadenado>

Acceso-Variable-Encadenado ::= <Encadenado> |
                               '[' <Expresion> ']'
```

6. Pruebas **.swift*

En el directorio *./test* existen diversos archivos de prueba exitosos (*test_xy.swift*) y no exitosos (*err_xy.swift*).

7. Conclusión

Se trató de seguir fielmente las guías proporcionadas por la cátedra, aunque modificaciones menores escapan de lo esperado por el método de no terminal: es el caso del método *"metodo()"* que matchea términos *"static", "func"* en su propio cuerpo en caso de haber sido llamado por la clase Main.

Adicionalmente, faltó aplicar el algoritmo de eliminación de recursividad indirecta. Se desconoce si habría modificado en gran medida el código resultante, pero se espera que lo desarrollado sea igualmente exitoso.