

Diseño e Implementación de un Analizador
Léxico y Analizador Semántico para el
Lenguaje AVISMO

Aramis E. Matos

Lenier Gerena

Angel Berrios Pellot

Segundo Semestre, 2022-2023

Tabla de Contenido

1	Introducción	2
2	Analizador Léxico	3
2.1	Gramatica del Lenguaje AVISMO	3
2.2	Diseño del del Analizador Léxico	11
2.2.1	Autómatas Finitos Deterministas	11
2.2.2	Tabla de Símbolos	12
2.3	Implementación del Analizador Léxico	13
3	Implementación del Analizador Sintáctico	15
3.1	<i>grammar.py</i>	15
3.2	<i>tester.py</i>	17
4	Conclusiones y Recomendaciones	18
	Acrónimos	21

Capítulo 1

Introducción

La visualización molecular puede ser considerada como una de las áreas mas importante dentro de la bioinformática. Entre sus aplicaciones mas relevantes se destacan el diseño de nuevo fármacos... (?, ?). Este enunciado fue escrito hace mas de una década. Sin embargo, hoy día en un mundo pospandemia, reconocemos que tan sabio fue. El desarrollo de la vacuna contra el COVID-19 tan rápido fue gracias a herramientas de visualización como el Ambiente de visualización Molecular (AVISMO) (?, ?) El propósito de este proyecto es definir el automata de estado finito del lenguaje AVISMO, los patrones el cual caracterizan los lexemas del lenguaje, los atributos de los lexemas y la implementación del analizador léxico y sintáctico del lenguaje AVISO en Python. La implementación léxica y sintáctica fue desarrollada utilizando Python Lex Yacc (?, ?) con el lenguaje MAPL (?, ?) como base.

Capítulo 2

Analizador Léxico

2.1 Gramatica del Lenguaje AVISMO

- $\langle \text{SENTENCIAS} \rangle ::= \langle \text{FIN_DE_LINEA} \rangle \langle \text{SENTENCIAS} \rangle \mid \langle \text{SENTENCIA} \rangle \langle \text{FIN_DE_LINEA} \rangle$
- $\langle \text{FIN_DE_LINEA} \rangle ::= ":" \mid ";;"$
- $\langle \text{SENTENCIA} \rangle ::= \text{"defina"} \langle \text{ID} \rangle \text{"como"} \langle \text{TIPO} \rangle \mid \langle \text{ID} \rangle \text{"="} \langle \text{MODELO_MOLECULAR} \rangle \mid \langle \text{OPERACION} \rangle \text{"("} \langle \text{ID} \rangle \text{"}"}$
- $\langle \text{ID} \rangle ::= \text{"A"} \mid \text{"B"} \mid \text{"C"} \mid \text{"D"} \mid \text{"E"} \mid \text{"F"} \mid \text{"G"} \mid \text{"H"} \mid \text{"I"} \mid \text{"J"} \mid \text{"K"} \mid \text{"L"} \mid \text{"M"} \mid \text{"N"} \mid \text{"O"} \mid \text{"P"} \mid \text{"Q"} \mid \text{"R"} \mid \text{"S"} \mid \text{"T"} \mid \text{"U"} \mid \text{"V"} \mid \text{"W"} \mid \text{"X"} \mid \text{"Y"} \mid \text{"Z"} \mid \text{"a"} \mid \text{"b"} \mid \text{"c"} \mid \text{"d"} \mid \text{"e"} \mid \text{"f"} \mid \text{"g"} \mid \text{"h"} \mid \text{"i"} \mid \text{"j"} \mid \text{"k"} \mid \text{"l"} \mid \text{"m"} \mid \text{"n"} \mid \text{"o"} \mid \text{"p"} \mid \text{"q"} \mid \text{"r"} \mid \text{"s"} \mid \text{"t"} \mid \text{"u"} \mid \text{"v"} \mid \text{"w"} \mid \text{"x"} \mid \text{"y"} \mid \text{"z"} \mid \langle \text{LETRA} \rangle \langle \text{IDCONT} \rangle$

- <IDCONT> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | <LETRA> <IDCONT> | "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | <DIGITO> <IDCONT>
- <LETRA> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
- <DIGITO> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
- <TIPO> ::= "modelo"
- <OPERACION> ::= "graficar2d" | "graficar3d" | "pesomolecular"
- <MODELO_MOLECULAR> ::= "H" | "Li" | "Na" | "K" | "Rb" | "Cs" | "Fr" | "Be" | "Mg" | "Ca" | "Sr" | "Ba" | "Ra" | "Sc" | "Y" | "Ti" | "Zr" | "Hf" | "Db" | "V" | "Nb" | "Ta" | "Ji" | "Cr" | "Mo" | "W" | "Rf" | "Mn" | "Tc" | "Re" | "Bh" | "Fe" | "Ru" | "Os" | "Hn" | "Co" | "Rh" | "Ir" | "Mt" | "Ni" | "Pd" | "Pt" | "Cu" | "Ag" | "Au" | "Zn" | "Cd" | "Hg" | "B" | "Al" | "Ga" | "In" | "Ti" | "C" | "Si" | "Ge" | "Sn" | "Pb" | "N" | "P" | "As" | "Sb" | "Bi" | "O" | "S" | "Se" | "Te" | "Po" | "F" | "Cr" | "Br" | "I" | "At" | "He" | "Ne" | "Ar" | "Kr" | "Xe" | "Rn" | <ELEMENTO_QUIMICO> <VALENCIA> | <ELE-

MENTO> <GRUPO_FUNCIONAL> | <COMPUESTO> <ELEMENTO>
 <GRUPO_FUNCIONAL> | <COMPUESTO> <COMPUESTO>

- <COMPUESTO> ::= "H" | "Li" | "Na" | "K" | "Rb" | "Cs" | "Fr" | "Be" |
 "Mg" | "Ca" | "Sr" | "Ba" | "Ra" | "Sc" | "Y" | "Ti" | "Zr" | "Hf" | "Db" | "V"
 | "Nb" | "Ta" | "Ji" | "Cr" | "Mo" | "W" | "Rf" | "Mn" | "Tc" | "Re" | "Bh"
 | "Fe" | "Ru" | "Os" | "Hn" | "Co" | "Rh" | "Ir" | "Mt" | "Ni" | "Pd" | "Pt" |
 "Cu" | "Ag" | "Au" | "Zn" | "Cd" | "Hg" | "B" | "Al" | "Ga" | "In" | "Ti" |
 "C" | "Si" | "Ge" | "Sn" | "Pb" | "N" | "P" | "As" | "Sb" | "Bi" | "O" | "S" |
 "Se" | "Te" | "Po" | "F" | "Cr" | "Br" | "I" | "At" | "He" | "Ne" | "Ar" | "Kr" |
 "Xe" | "Rn" | <ELEMENTO_QUIMICO> <VALENCIA> | <ELEMENTO>
 <GRUPO_FUNCIONAL> | <ELEMENTO> <GRUPO_FUNCIONAL> <EN-
 LACE> | <ELEMENTO> <ENLACE>
- <COMPUESTOS> ::= <COMPUESTO> <COMPUESTO> | <COMPUESTOS>
- <ELEMENTO> ::= "H" | "Li" | "Na" | "K" | "Rb" | "Cs" | "Fr" | "Be" | "Mg"
 | "Ca" | "Sr" | "Ba" | "Ra" | "Sc" | "Y" | "Ti" | "Zr" | "Hf" | "Db" | "V" | "Nb"
 | "Ta" | "Ji" | "Cr" | "Mo" | "W" | "Rf" | "Mn" | "Tc" | "Re" | "Bh" | "Fe" |
 "Ru" | "Os" | "Hn" | "Co" | "Rh" | "Ir" | "Mt" | "Ni" | "Pd" | "Pt" | "Cu" |
 "Ag" | "Au" | "Zn" | "Cd" | "Hg" | "B" | "Al" | "Ga" | "In" | "Ti" | "C" | "Si"
 | "Ge" | "Sn" | "Pb" | "N" | "P" | "As" | "Sb" | "Bi" | "O" | "S" | "Se" | "Te" |
 "Po" | "F" | "Cr" | "Br" | "I" | "At" | "He" | "Ne" | "Ar" | "Kr" | "Xe" | "Rn" |
 <ELEMENTO_QUIMICO> <VALENCIA>
- <ELEMENTO_QUIMICO> ::= "H" | "Li" | "Na" | "K" | "Rb" | "Cs" | "Fr" |

"Be" | "Mg" | "Ca" | "Sr" | "Ba" | "Ra" | "Sc" | "Y" | "Ti" | "Zr" | "Hf" | "Db" |
 "V" | "Nb" | "Ta" | "Ji" | "Cr" | "Mo" | "W" | "Rf" | "Mn" | "Tc" | "Re" | "Bh" |
 "Fe" | "Ru" | "Os" | "Hn" | "Co" | "Rh" | "Ir" | "Mt" | "Ni" | "Pd" | "Pt" | "Cu"
 | "Ag" | "Au" | "Zn" | "Cd" | "Hg" | "B" | "Al" | "Ga" | "In" | "Ti" | "C" | "Si"
 | "Ge" | "Sn" | "Pb" | "N" | "P" | "As" | "Sb" | "Bi" | "O" | "S" | "Se" | "Te" |
 "Po" | "F" | "Cr" | "Br" | "I" | "At" | "He" | "Ne" | "Ar" | "Kr" | "Xe" | "Rn"

- <VALENCIA> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
- <GRUPO_FUNCIONAL> ::= <GRUPO_FUNCIONAL_INFERIOR>
 <GRUPO_FUNCIONAL_SUPERIOR> | <GRUPO_FUNCIONAL_SUPERIOR>
 <GRUPO_FUNCIONAL_INFERIOR> | "(" <MODELO_GRUPO_FUNCIONAL>
 ")" | "[" <MODELO_GRUPO_FUNCIONAL> "]"
- <GRUPO_FUNCIONAL_SUPERIOR> ::= "[" <MODELO_GRUPO_FUNCIONAL>
 "]"
- <GRUPO_FUNCIONAL_INFERIOR> ::= "(" <MODELO_GRUPO_FUNCIONAL>
 ")
- <MODELO_GRUPO_FUNCIONAL> ::= <ENLACE> <MODELO_MOLECULAR>
 | "H" | "Li" | "Na" | "K" | "Rb" | "Cs" | "Fr" | "Be" | "Mg" | "Ca" | "Sr" | "Ba"
 | "Ra" | "Sc" | "Y" | "Ti" | "Zr" | "Hf" | "Db" | "V" | "Nb" | "Ta" | "Ji" | "Cr"
 | "Mo" | "W" | "Rf" | "Mn" | "Tc" | "Re" | "Bh" | "Fe" | "Ru" | "Os" | "Hn" |
 "Co" | "Rh" | "Ir" | "Mt" | "Ni" | "Pd" | "Pt" | "Cu" | "Ag" | "Au" | "Zn" | "Cd" |
 "Hg" | "B" | "Al" | "Ga" | "In" | "Ti" | "C" | "Si" | "Ge" | "Sn" | "Pb" | "N" | "P"

| "As" | "Sb" | "Bi" | "O" | "S" | "Se" | "Te" | "Po" | "F" | "Cr" | "Br" | "I" | "At"
 | "He" | "Ne" | "Ar" | "Kr" | "Xe" | "Rn" | <ELEMENTO_QUIMICO> <VA-
 LENCIA> | <ELEMENTO> <GRUPO_FUNCIONAL> | <COMPUESTO>
 <ELEMENTO> | <COMPUESTO> <COMPUESTO> <COMPUESTOS>

En la tabla 2.1, en la columna de patrones, note que cuando dice $\{TOKEN\}$ donde *TOKEN* se refiere a el patrón asociado a *token*. Por ejemplo, si un patrón dice $\{ELEMENTO_QUIMICO\}$, esto significa que inserta el patrón asociado al *token ELEMENTO_QUIMICO*. Esto no significa que el analizador léxico espera un *token* de por si, sencillamente se hizo con el propósito de evitar redundancias.

<i>Token</i>	Patrón	Lexema	Atributos
<FIN_DE_LINEA>	; :	:	Símbolo reservado
<PALABRA _RESERVADA>	defina como	defina	Palabra reservada
<ID>	[A-Za-z][A-Za-z0-9]*	var1	Modelo molecular asociado
<IDCONT>	[A-Za-z0-9]+	1ar	ID asociado
<LETRA>	[A-Za-z]	a	ID asociado
<DIGITO>	[0-9]	7	Valor numérico, lexema asociado
<TIPO>	modelo	modelo	ID asociado
<OPERACION>	graficar2d graficar3d pesomolecular	pesomolecular	ID asociado
<MODELO _MOLECULAR>	({ELEMENTO _QUIMICO} {ELEMENTO _QUIMICO} {VALENCIA} {ELEMENTO} {GRUPO _FUNCIONAL} {ELEMENTO} {GRUPO _FUNCIONAL} {ENLACE} {ELEMENTO} {ENLACE})	CH3(CH3)CHH	ID asociado

<COMPUESTO>	COMPUESTO ({ELEMENTO _QUIMICO} {ELEMENTO _QUIMICO} {VALENCIA}) {ELEMENTO} {GRUPO_FUNCIONAL} {ELEMENTO} {GRUPO_FUNCIONAL} {ENLACE} {ELEMENTO} {ENLACE})	CH3::	Modelo molecular asociado, enlaces, valencias
<COMPUESTOS>	{COMPUESTO}+	CH3:::(OH)3	Modelo molecular asociado, enlaces, valencias
<ELEMENTO>	{ELEMENTO _QUIMICO} {VALENCIA}?	Ag3	Elemento, valencia
<ELEMENTO _QUIMICO>	("H" "Li" "Na" "K" "Rb" "Cs" "Fr" "Be" "Mg" "Ca" "Sr" "Ba" "Ra" "Sc" "Y" "Ti" "Zr" "Hf" "Db" "V" "Nb" "Ta" "Ji" "Cr" "Mo" "W" "Rf" "Mn" "Tc" "Re" "Bh" "Fe" "Ru" "Os" "Hn" "Co" "Rh" "Ir" "Mt" "Ni" "Pd" "Pt" "Cu" "Ag" "Au" "Zn" "Cd" "Hg" "B" "Al" "Ga" "In" "Tl" "C" "Si" "Ge" "Sn" "Pb" "N" "P" "As" "Sb" "Bi" "O" "S" "Se" "Te" "Po" "F" "Cr" "Br" "I" "At" "He" "Ne" "Ar" "Kr" "Xe" "Rn")	I	Elemento
<VALENCIA>	[1-9]	2	Valor

<GRUPO _FUNCIONAL>	({GRUPO _FUNCIONAL _INFERIOR} {GRUPO _FUNCIONAL _SUPERIOR} {GRUPO _FUNCIONAL _SUPERIOR} {GRUPO _FUNCIONAL _INFERIOR} "(" {MODELO _GRUPO _FUNCIONAL} ")" "[" MODELO _GRUPO _FUNCIONAL "]")	(CH3){Ag2}	Grupos funcionales, grupo funcional inferior, grupo funcional superior
<GRUPO _FUNCIONAL _INFERIOR>	"[" {MODELO _GRUPO _FUNCIONAL} "]"	[CVHe3]	Elementos, valencias
<GRUPO _FUNCIONAL _SUPERIOR>	"(" {MODELO _GRUPO _FUNCIONAL} ")"	(CVHe3)	Elementos, valencias
<MODELO _GRUPO _FUNCIONAL>	((ELEMENTO _QUIMICO)+ {VALENCIA}?) + ({ELEMENTO}+ {ENLACE} {ELEMENTO}+)+	FeH=C3Si4	Elementos, enlaces, valencias
<ENLACE>	("-" "=" ":" "::")	-	Valencia

Tabla 2.1: Tabla de Componentes Léxicos de AVISMO

2.2 Diseño del del Analizador Léxico

2.2.1 Autómatas Finitos Deterministas

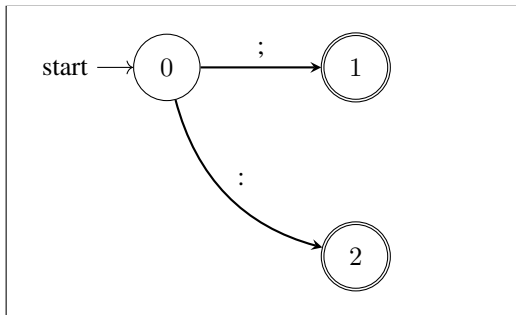


Figura 2.1: Automata del patrón para el token <FIN_DE_LINEA>

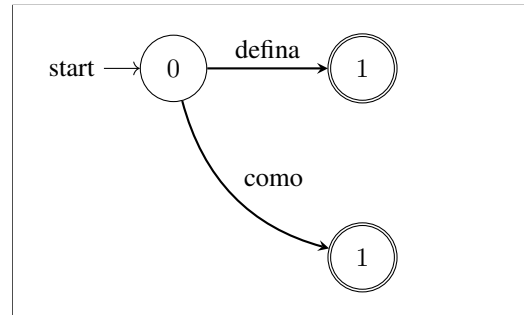


Figura 2.2: Automata del patrón para el token <PALABRAS_RESERVADA>

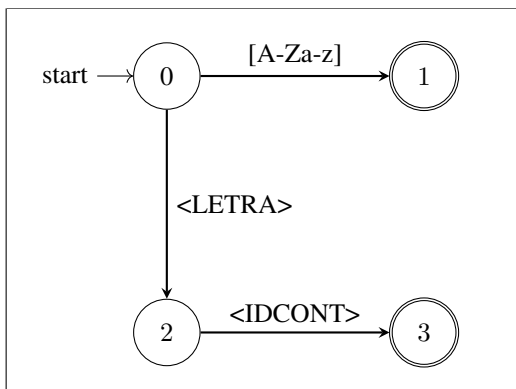


Figura 2.3: Automata del patrón para el token <ID>

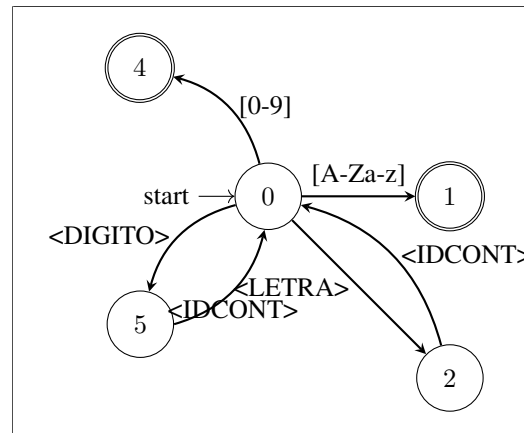


Figura 2.4: Automata del patrón para el token <IDCONT>

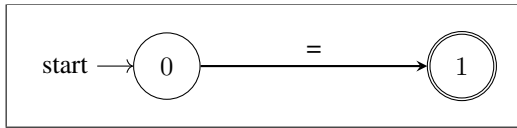


Figura 2.5: Automata del patrón para el token <ASIGNACION>

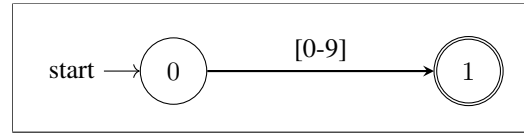


Figura 2.6: Automata del patrón para el token <LETRA>

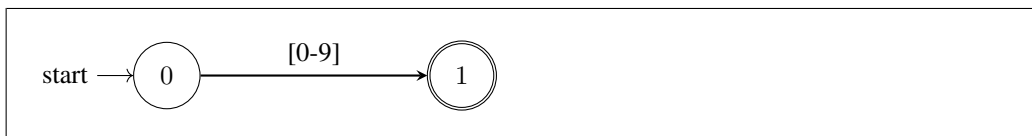


Figura 2.7: Automata del patrón para el token <DIGITO>

2.2.2 Tabla de Símbolos

Identificador

Atributo

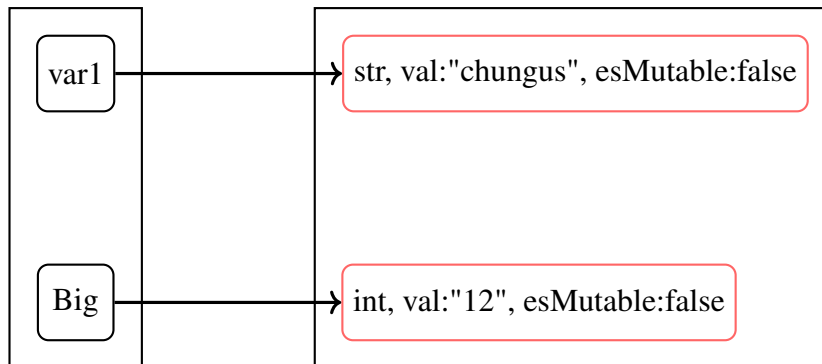


Figura 2.8: Tabla de símbolos implementada como un diccionario

2.3 Implementación del Analizador Léxico

Como se ha mencionado anteriormente, la implementación lexica del proyecto fue inspirada por la MAPL (?, ?) y adaptada para la gramática de AVISMO.

Para ejecutar el analizador léxico, primero se tiene que instalar *Python 3*. Luego, se ejecuta por la línea de comando. Finalmente, se ejecuta

Al ejecutarse el comando anterior, el programa procede a leer **cada caracter** del programa e identificar si una serie de caracteres sigue un patrón que forma parte del lenguaje AVISMO. Al encontrar un patrón reconocido, tales como un identificador o modelo molecular, lo clasifica con un *token* correspondiente, lo emprime en el archivo de *output.txt* y lo devuelve al analizador sintáctico. Note, que el patrón de identificador reconoce palabra reservadas también. Esto crea ambigüedad semántica debido a que la gramática no tiene un mecanismo para diferenciar entre una palabra reservada y un identificador. Por esta razón, si una serie de caracteres se identifica como un lexema de categoría identificador, se compara con los valores ya existentes del diccionario *variables*. Al inicializar el programa, este se encarga de abrir el archivo de palabras reservadas (*keywords.txt*) y añadir las palabras reservadas antes que cualquier variable se pueda inicializar. Mas aún, a las palabras reservadas se les asigna el valor de la cadena vacía. Esto se hace con el propósito de poder diferenciar entre palabras reservadas e identificadores, ya que al nivel sintáctico, no es posible asignarle a un identificador una cadena vacía, como se puede apreciar a continuación:

Figura 2.9: Patrón que ceeza identificadores de palabras reservadas

Figura 2.11: Gestión de errores

Con el propósito de visualizar los lexemas generados por , colocado en el archivo *lexer.py*, se utiliza la siguiente función:

Figura 2.10: Código para imprimir los *tokens* encontrados

Esto imprime el *token* resultante en el archivo *output.txt*. En el caso de un error léxico, se ejecuta el siguiente código: Cuando se encuentra un error léxico, no se retorna al analizador sintáctico. Los patrones que se utilizan están en (líneas 58-103) son una adaptación de la gramática en la tabla 2.1. Note la tupla *tokens*. Esta contiene todos los *tokens* del lenguaje AVISMO. Sin embargo, no todos están definidos a el nivel léxico. La mayoría de los *tokens* se construyen en la etapa sintáctica, particularmente los que tienen que ver con compuestos químicos y modelos moleculares. Mas aún, note el objeto *tok* en la línea 5 de el listado 2.10. Este contiene los atributos *type*, *value*, *lineno* y *lexpos*. Estos devuelven, respectivamente, el tipo, lexema, en que línea del archivo se encuentra y la posición del primer caracter de el *token* llamado *tok*. Otro detalle importante de la implementación léxica es el orden de aplicación de los patrones. Los patrones en forma de variables, como en el listado 2.12, tienen que estar escrito antes que los patrones escritos en forma de funciones, como en el listado 2.9 y 2.13

Figura 2.12: Patrón en forma de variable

Figura 2.13: Patrón en forma de función

Capítulo 3

Implementación del Analizador Sintáctico

3.1 *grammar.py*

Un código fuente pasa por al menos dos fases:

1. Análisis léxico
2. Análisis sintáctico

Como se ha mencionado antes, en esta primera fase se evalúa el código fuente carácter a carácter. Este se tokeniza, es decir se le otorga una categoría sintáctica, y se devuelve al analizador sintáctico. Ahora, la cuestión es, cuál es el propósito del analizador sintáctico? El analizador sintáctico recibe una lista de *tokens* del analizador léxico y las convierte, a través de reglas de producción, en sentencias

gramaticales del lenguaje en cuestión. Relas de producción tienen la siguiente forma: (**Regla** : *Definición*) donde **Regla** es un no terminal y *Definición* es una serie de 0 o mas terminales o no terminales. Un terminal se define como un *token* y un no terminal es un una regla gramatical en si.

Se utiliza PLY (?, ?) para el análisis sintáctico. En particular, su implementación de yacc (?, ?). En el archivo *grammar.py* se puede apreciar que la todas de las reglas del lenguaje AVISMO, con la exección de relas que fueron utilizadas en analizador léxico, fueron adaptadas. En PLY, una regla gramatical se define como una función en Python cuyo nombre es **p_** seguido del nombre de la regla de producción, por ejemplo:

Figura 3.1: Ejemplo de una regla de producción en PLY

Note que el argumento *p* es una lista que contiene objetos *LexToken*. Los terminales tienen una variable de valor asignada mientras que los no terminales no. Cada objeto *LexToken* tiene una posición léxica (como una variable miembro llamada *lexpos*) y la línea dentro del código fuente (como una variable miembro llamada *lineno*). La manera de definir la regla de producción se puede ver en la figura 3.1, línea ???. Esta sigue el formato previamente establecido pero con un detalle importante. Los no terminales están escritos en letras minúsculas y los terminales en mayúsculas. Esto se hizo con el motivo de clarificar en que categoría, si terminal o no terminal, es clasificada cada ítem en la regla de producción. Más aún, la documentación de *PLY* sugiere esta convención.

Toda gramática parte desde un axioma y *PLY* sigue este principio. Por de-

facto, *PLY* asume que la primera regla que se define en el archivo de *grammar.py* es el axioma de la gramática. Sin embargo, es preferible que se defina un axioma explícito. En *PLY*, si se le asigna a la variable *start* el nombre de la regla de producción como una cadena de caracteres, como se hace a continuación, *PLY* explícitamente comienza la derivación desde esa regla. Declarar el axioma explícitamente tiene dos ventajas:

- Claridad en el código
- Eliminación de errores por tokens no utilizados

Debido a que no se está implementando la funcionalidad del lenguaje AVISMO, las reglas de producción no tienen código relevante. Sin embargo, todas las reglas de producción en *grammar.py* ejecutan una función llamada *format_expr* que guarda información acerca de la regla gramatical que se utilizó en la derivación del código fuente. Esto se hace con intenciones pedagógicas. El código de *format_expr* se presenta a continuación:

Figura 3.2: Código para guardar información acerca de las derivaciones

3.2 *tester.py*

Para poder correr (*grammar.py*) en un archivo escrito en AVISMO, es necesario invocar de correr el programa *tester.py* de la siguiente manera:

Capítulo 4

Conclusiones y Recomendaciones

Lista de Figuras

2.1	Automata del patrón para el token <FIN_DE_LINEA>	11
2.2	Automata del patrón para el token <PALABRAS_RESERVADA>	11
2.3	Automata del patrón para el token <ID>	11
2.4	Automata del patrón para el token <IDCONT>	11
2.5	Automata del patrón para el token <ASIGNACION>	12
2.6	Automata del patrón para el token <LETRA>	12
2.7	Automata del patrón para el token <DIGITO>	12
2.8	Tabla de símbolos implementada como un diccionario	12
2.9	Patrón que ceeza identificadores de palabras reservadas	13
2.11	Gestión de errores	14
2.10	Código para imprimir los <i>tokens</i> encontrados	14
2.12	Patrón en forma de variable	14
2.13	Patrón en forma de función	14
3.1	Ejemplo de una regla de producción en PLY	16
3.2	Código para guardar información acerca de las derivaciones	17

Lista de Tablas

2.1	Tabla de Componentes Léxicos de AVISMO	10
-----	--	----

Referencias Bibliográficas

Man yacc (1): An LALR(1) parser generator. (n.d.). Retrieved 2023-04-15, from

`https://manpages.org/yacc`

Narciso Farias, F., Rios, A., Hidrobo, F., & Vicuña, O. (n.d.). Una gramática

libre de contexto para el lenguaje del ambiente de visualización molecular -

AVISMO..

PL-Project-LGM-YVV-AMN/MAPL. (n.d.). Retrieved 2023-03-29, from

`https://github.com/PL-Project-LGM-YVV-AMN/MAPL`

PLY (Python Lex-Yacc) — ply 4.0 documentation. (n.d.). Retrieved

2023-03-29, from `https://ply.readthedocs.io/en/latest/`

`index.html`