Universidad Nacional Autónoma de México
Facultad de Ingeniería

Analizador léxico

Compiladores

Grupo: 1

Semestre 2021-1

Profesor: Adrian Ulises Mercado Martinez

18 de Noviembre del 2020.

Alumnos:

Aguilar Perez Paulina

Aviles Martinez Bryan

Meza Ortega Fernando

Índice

1. Objetivo

Crear un analizador léxico del subconjunto de Lenguaje GO, con el fin de poder aplicar los conocimientos teóricos vistos en clase.

2. Descripción del problema

Crear un analizador léxico que pueda identificar cada uno de los tokens que pertenecen al alfabeto del lenguaje GO.

3. Análisis de la solución

Para crear el analizador léxico necesitamos programar un AFD, existen muchas técnicas que nos sirven para representar el AFD y que nuestro analizador léxico funcione, pero en nuestro caso nos conviene que el AFD sea implementado con Jflex para que Jflex se encargue de las partes más complicadas y nosotros sólo tengamos que preocuparnos por las expresiones regulares.

4. Diseño de la solución

Las expresiones regulares que usamos son las siguientes

- Literales

$digito \rightarrow [0-9]$
$enteros \rightarrow digito + (\_digito+) *$
$letras \rightarrow [a-zA-Z]$
$ex \rightarrow [Ee]([+-])?enteros +$
$dec \rightarrow enteros * .enteros+ \mid enteros + .enteros *$
$decex \rightarrow ((dec)(ex)? \mid (enteros)(ex))$
$imaginario \rightarrow ((dec)(ex)?i \mid (enteros)(ex)?i)$
$espacios \rightarrow [\backslash n \setminus t] +$
$cadenas \rightarrow ((\backslash")([\wedge(\backslash") \setminus n] \mid \setminus \backslash(\backslash")) * (\backslash")\mid(\backslash')([\wedge(\backslash')]) * (\backslash'))$

- identificadores

$id \rightarrow (letras \mid \_)(digito \mid letras \mid \_) *$

- Operadores

$add_op1 \rightarrow +$
$add_op2 \rightarrow -$
$mul_op1 \rightarrow *$
$mul_op2 \rightarrow /$
$mul_op3 \rightarrow \%$
$unary_op1 \rightarrow \&$
$unary_op2 \rightarrow !$
$or \rightarrow ||$
$and \rightarrow \&\&$
$par_L \rightarrow ($

$par_R \rightarrow)$
$cor_L \rightarrow [$
$cor_R \rightarrow]$
$lla_L \rightarrow \{$
$lla_R \rightarrow \}$
$rel_op_1 \rightarrow ==$
$rel_op_2 \rightarrow ! =$
$rel_op_3 \rightarrow <$
$rel_op_4 \rightarrow <=$
$rel_op_5 \rightarrow >$
$rel_op_6 \rightarrow >=$
$inc \rightarrow ++$
$decre \rightarrow --$

- Palabras reservadas

$constbool \rightarrow (true|false)$
$pack \rightarrow package$
$var \rightarrow var$
$const \rightarrow const$
$uint8 \rightarrow uint8$
$uint16 \rightarrow uint16$
$int8 \rightarrow int8$
$int16 \rightarrow int16$
$int32 \rightarrow int32$
$float32 \rightarrow float32$
$float64 \rightarrow float64$
$complex64 \rightarrow complex64$
$byte \rightarrow byte$
$string \rightarrow string$
$bool \rightarrow bool$
$struct \rightarrow struct$
$func \rightarrow func$
$defer \rightarrow defer$
$if \rightarrow if$
$else \rightarrow else$
$switch \rightarrow switch$
$case \rightarrow case$
$default \rightarrow default$
$for \rightarrow for$
$return \rightarrow return$
$break \rightarrow break$
$continue \rightarrow continue$

Los tokens se clasifican de la siguiente manera

- Palabras reservadas
  package, var, const, uint8, unit16, int8, int16, int32, float32, float64, complex64, byte, string, bool, struct, func, defer, i, else , switch, case, default, for, return, break, continue
- Identificadores Para este lenguajes, un identificador es una cadena de una o más letras y dígitos, debe de empezar con una letra o un guión bajo.

- Operadores
  1. $Aritmeticos +, -, *, /, \%$
  2. $Deasignacion =$
  3. $Relacionales ==, != , <, <=, >, >=$
  4. $Logicos ||, \&\&, !$
- Literales numéricas
  Constantes numéricas, cualquiera de estos puede llevar un "_" entre 2 de sus dígitos.
  1. Enteros
  2. Decimales Incluye un punto decimal, y puede incluir la notación científica.
  3. Imaginarios Cualquier número seguido de una letra i.
- Signos especiales o de puntuación :, ;, , , &,
- Espacios en blanco

4.1. Construcción de AFD con 4 expresiones regulares

Mediante el uso del algoritmo goto, obtendremos un AFD haciendo uso de 4 expresiones regulares representativas
Las expresiones son las siguientes:

1. $decex \rightarrow ((dec)(ex)? \mid (enteros)(ex))$
2. $imaginario \rightarrow ((dec)(ex)?i \mid (enteros)(ex)?i)$
3. $enteros \rightarrow digito + (\_digito+)*$
4. $id \rightarrow (letras \mid \_)(\_digito \mid letras \mid \_)*$

$cerradura(decex \rightarrow \bullet((dec)(ex)? \mid (enteros)(ex)), imaginario \rightarrow \bullet((dec)(ex)?i \mid (enteros)(ex)?i, enteros \rightarrow \bullet digito + (\_digito+)*, id \rightarrow \bullet(letras \mid \_)(digito \mid letras \mid \_)*) = \{$

$decex \rightarrow ((\bullet dec)(ex)?|(enteros)(ex))$
$decex \rightarrow ((dec)(ex)?|(\bullet enteros)(ex))$
$imaginario \rightarrow ((\bullet dec)(ex)?(i)|(enteros)(ex)?i))$
$imaginario \rightarrow ((dec)(ex)?(i)|(\bullet enteros)(ex)?"i"))$
$id \rightarrow (\bullet letras|_{)}(digito|letras|_{)} *$
$id \rightarrow (letras| \bullet_{)} (digito|letras|_{)} *$
$enteros \rightarrow \bullet digito + ({}_{d}igito+) *$
$\} = q_0$

$goto(q_0, dec)$
$\{$
$decex \rightarrow ((dec)(\bullet ex)?|(enteros)(ex))$
$decex \rightarrow ((dec)(ex)?|(enteros)(ex)) \bullet$
$imaginario \rightarrow ((dec)(\bullet ex)?(i)|(enteros)(ex)?i))$
$imaginario \rightarrow ((dec)(ex)?(\bullet i)|(enteros)(ex)?i))$
$\} = q_1 *$

$goto(q_0, enteros)$
$\{$
$decex \rightarrow ((dec)(ex)?|(enteros)(\bullet ex))$
$imaginario \rightarrow ((dec)(ex)?(i)|(enteros)(\bullet ex)?i))$
$imaginario \rightarrow ((dec)(ex)?(i)|(enteros)(ex)?))$
$\} = q_2$

$goto(q_0, letras)$
$\{$
$id \rightarrow (letras|\_)(\bullet digito|letras|\_) *$
$id \rightarrow (letras|\_)(digito| \bullet letras|\_) *$
$id \rightarrow (letras|\_)(digito|letras| \bullet \_) *$
$id \rightarrow (letras|\_)(digito|letras|\_) * \bullet$
$\} = q_3 *$

$goto(q_0, \_) = q_3 *$

$goto(q_0, digito)$
$\{$
$enteros \rightarrow \bullet digito + (\_digito+) *$
$enteros \rightarrow digito + (\bullet \_digito+) *$
$enteros \rightarrow digito + (\_digito+) * \bullet$
$\} = q_4 *$

$goto(q_1, ex)$
$\{$
$decex \rightarrow (dec)(ex)?|(enteros)(ex)) \bullet$
$imaginario \rightarrow ((dec)(ex)?(\bullet i)|(enteros)(ex)?i))$
$\} = q_5 *$

$goto(q_1, i)$
$\{$
$imaginario \rightarrow ((dec)(ex)?(i)|(enteros)(ex)?i)) \bullet$
$\} = q_6 *$

$goto(q_2, ex)$
$\{$
$decex \rightarrow ((dec)(ex)?|(enteros)(ex)) \bullet imaginario \rightarrow ((dec)(ex)?(i)|(enteros)(ex)? \bullet i)$
$\} = q_7 *$

$goto(q_2, i) = q_6 *$
$goto(q_3, digito) = q_3 *$
$goto(q_3, letras) = q_3 *$
$goto(q_3, \_) = q_3 *$
$goto(q_4, digito) = q_4 *$

$goto(q_4, \_)$
$\{$
$enteros \rightarrow digito + (\bullet \_digito+) *$
$\} = q_8 *$

$goto(q_5, i) = q_6 *$

$goto(q_7, i) = q_6 *$

$goto(q_8, digito)$
$\{$
$enteros \rightarrow digito + (\bullet\_digito+) *$
$enteros \rightarrow digito + (\_ \bullet digito+) *$
$enteros \rightarrow digito + (\_digito+) * \bullet$
$\} = q_9 *$

$goto(q_9, \_) = q_8$
$goto(q_9, digito) = q_9 *$

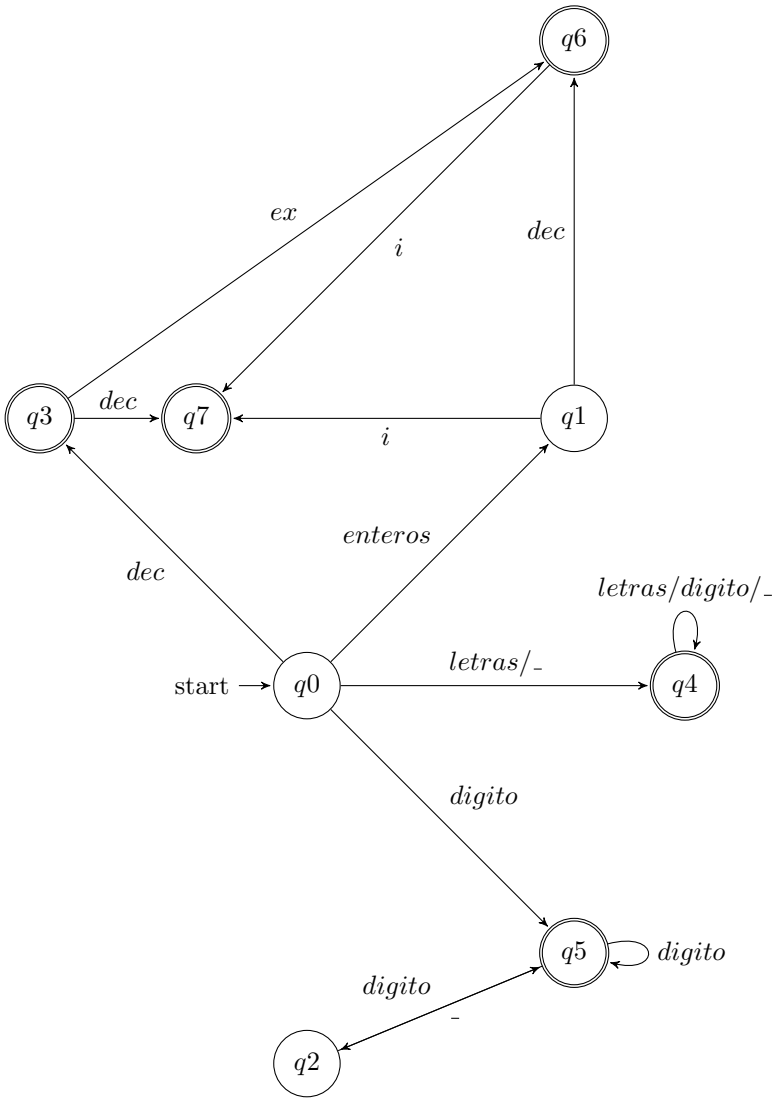
4.2. Tabla de transiciones

Estados	dec	enteros	letras	_	digito	ex	i
q0	q1*	q2	q3*	q3*	q4*	ϕ	ϕ
q1*	q6*	ϕ	ϕ	ϕ	ϕ	q5*	ϕ
q2	q7*	ϕ	ϕ	ϕ	ϕ	ϕ	q6*
q3*	ϕ	ϕ	q3*	q3*	q3*	ϕ	ϕ
q4*	ϕ	ϕ	ϕ	q8	q4*	ϕ	ϕ
q5*	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	q6*
q6*	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
q7*	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	q6*
q8	ϕ	ϕ	ϕ	ϕ	q9*	ϕ	ϕ
q9*	ϕ	ϕ	ϕ	q8*	q9*	ϕ	ϕ

4.3. Después de minimizar

	dec	enteros	letras	_	digito	ex	i
q0	q3*	q1	q4*	q4*	q5*	ϕ	ϕ
q1	q6*	ϕ	ϕ	ϕ	ϕ	ϕ	q7*
q2	ϕ	ϕ	ϕ	ϕ	q5*	ϕ	ϕ
q3*	q7*	ϕ	ϕ	ϕ	ϕ	q6*	ϕ
q4*	ϕ	ϕ	q4*	q4*	q4*	ϕ	ϕ
q5*	ϕ	ϕ	ϕ	q2	q5*	ϕ	ϕ
q6*		ϕ	ϕ	ϕ	ϕ	ϕ	q7*
q7*		ϕ	ϕ	ϕ	ϕ	ϕ	

4.4. Autómata con 4 expresiones regulares



4.5. Muestra representativa del ADF completo

La siguiente imagen muestra una pequeña parte del Autómata completo (minimizado), este fue generado con la ayuda de JFlex. En la carpeta donde se incluye este documento se anexara el archivo que muestra el autómata completo.

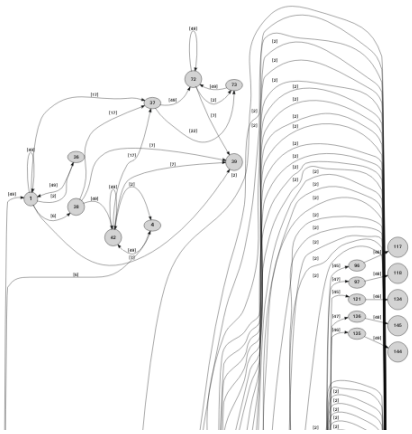


Figura 1: Muestra representativa automata completo

5. Implementación

Ya habiendo analizado y diseñado una solución por medio de la función cerradura y goto con las expresiones regulares, metimos las expresiones a un archivo para implementar nuestra solución en Jflex. Esta herramienta hace el autómata minimizado a partir de las expresiones proporcionadas y la acción a realizar en cada caso. Después de agregar las expresiones regulares, modificamos el programa para que retornara un número entero distinto por cada expresión, operador y palabra reservada encontrada. Por esto fue que tuvimos que escribir una expresión diferente para cada una de estas palabras y operadores. Además agregamos una función main en una clase diferente para poder probar que nuestro analizador léxico funcione correctamente, esta nueva clase es un molde muy simple para que en el análisis semántico nosotros ya seamos capaces de llamar a Yylex de manera externa

6. Conclusiones

Aguilar Pérez Paulina

Con la realización de este trabajo pudimos aplicar y reforzar mucha de la teoría vista en clase acerca del análisis léxico en un compilador. Empezando con el desarrollo de las expresiones regulares a partir de la gramática y unas pequeñas descripciones de los literales en este lenguaje. También aplicamos nuestro conocimiento al obtener un AFD a partir de tales expresiones por medio de las funciones goto. Hicimos esto con 4 expresiones regulares para darnos una idea de cómo armar el AFD. Para obtener el AFD completo utilizamos Jflex. Introdujimos las expresiones regulares en un archivo, donde también lo programamos para que retornara un entero diferente por cada expresión que reconociera. A mi parecer no fue difícil solucionar el problema planteado en esta primera parte del proyecto, me ayudó mucho a recordar la teoría ya vista y también pude aprender un poco mejor a escribir las expresiones en Jflex.

Aviles Martinez Bryan

Esta parte del proyecto que corresponde al analizador léxico, nos ayudó tanto a mi, como al equipo a entender completamente la teoría que se ha visto en clase, desde diferenciar entre un lexema y token, hasta generar el AFD minimizado. Durante el desarrollo del analizador se usaron todos los recursos teóricos: elemento punteado, formación de expresiones regulares, construcción de un AFD, todo esto con la ayuda de la herramienta JFlex , la cual nos facilitó mucho el trabajo en cuanto el reconocimiento de expresiones regulares, el retorno de lexemas, entre otras funciones que nos brinda. Por último, comparto que la forma de trabajo de hacer el compilador por fases (no todo en una sola

entrega) es de las maneras más prácticas de comprender lo que se está realizando.

Meza Ortega Fernando

con este proyecto comprendí más la fase de análisis léxico que realiza el compilador, en especial la parte que considero más provechosa es el uso de las expresiones regulares ya que con la práctica que obtuvimos en la clase esta parte fue relativamente sencilla y además no utilizamos estados en Jflex porque sentimos que con definir bien nuestras expresiones eran más que suficiente. El uso de los elementos punteados es una parte fundamental del análisis léxico desde un punto de vista teórico y considero que también lo pudimos reforzar al obtener el AFD. Por último aprendimos a cómo organizarnos un poco mejor considerando la carga de trabajo que todo el equipo tiene.