	UTN REGIONAL DELTA	TRABAIO PRACTICO	N°1
		SINTAXIS Y SEMÁNTICA DE LOS	AÑO 2022

UNIVERSIDAD TECNOLÓGICA NACIONAL

TRABAJO DE LABORATORIO N°1

SINTAXIS Y SEMANTICA DE LOS LENGUAJES

Docentes: Miranda Hernán y Santos Juan Miguel

Catino, Nicole Yuki – Ingeniería en Sistemas de Información.

Octavio, Pavón – Ingeniería en Sistemas de Información.

Bernardo, Basaldua – Ingeniería en Sistemas de Información.

Marinetto, Camila – Ingeniería en Sistemas de Información.

Índice

Enunciado

Se realizará la implementación de un analizador lexicográfico, mediante el cual, se pretende desarrollar un Autómata Finito(AF) por cada tipo de token y luego el analizador deberá ser implementado mediante un AFD que incluye a todos los AF construidos para cada token. El programa que resulte de la implementación deberá aceptar una cadena que representa código escrito en el lenguaje generado por la gramática provista. Este código, visto como una cadena de caracteres ASCII, deberá ser convertido a una cadena de tokens correspondiente a la gramática provista.

La gramática asignada fue la siguiente:

G=<VN, VT, P, S>

VT = {eq, id, num, *, +, op, clp, si, entonces, sino, mostrar, aceptar, mientras, esMenorQue, hacer, (,)}
VN = {Programa, Asignacion, Estructura, Expresion, Valor, ListaExpresiones, Termino, Factor}
S = Program
P = {

Program → Estructura Program
| λ

Estructura →**“mientras”** **“id”** **“esMenorQue”** Valor **“hacer”** **“op“** Program **“clp”**
| **“si”** Expresion **“entonces”** **“op“** Program **“clp”** **“sino”** **“op“**
Program **“clp”**
| **“mostrar”** Expresion
| **“aceptar”** **“id”**
| **“id”** **“eq”** Expresion

Valor → **“id”**
|**”num”**

Expresion → Termino Expresion2
Expresion2 → **“+”** Termino Expresion2
| λ

Termino → Factor Termino2
Termino2 → **“*”** Factor Termino2
| λ

Factor → “(“ Expresion “)”
| Valor

}

Explicación, observaciones y comentarios

En principio, ideamos el lexer definiendo todas las tuplas de tokens posibles con un orden de precedencia. A partir del paradigma de objetos, creamos una clase Lexer, en la cual hicimos un constructor donde inicializamos un nuevo lexer con los tokens posibles pasados por parámetro al instanciar la clase.

Creamos la función "_acomodar_listado_automatas(autómatas)" , la cual acomoda la lista de autómatas para que el que llega a estado final este en el principio de la lista.

Posteriormente, creamos una función " __alimentar_automatas(lexema)" ,en la cual, por cada tupla de "(token,autómata)" de "TOKENS_POSIBLES", nos fijamos si el lexema ingresado llega a un estado trampa.[1] Si esto ocurre, continuamos con la próxima tupla. En caso contrario, agregamos el token de la tupla actual a una lista de estados posibles en una tupla distinguiendo si llega a estado final o no final. Por último, si la lista de estados posibles es vacía la función nos devuelve la lista vacía. Sino, el primer elemento de la lista de posibles para así respetar el orden de precedencia de los "TOKENS_POSIBLES".

Luego, definimos la función" __recorriendo_lexema(source_code)", la cual se va a ejecutar mientras la variable start sea menor que la longitud del código fuente. Hacemos un while para ignorar cada espacio en blanco en el código fuente aumentando en una unidad la variable start. La variable start_lexema se iguala a start para empezar a construir un lexema. Por consiguiente, hacemos un ciclo infinito que se va a romper si la variable start supera la longitud del código fuente ya que no va haber más nada para analizar. Definimos la variable lexema que va a guardar una subcadena del código fuente que vamos a pasar por parámetro a la función " __alimentar_automatas(lexema)" , la cual nos va a devolver el primer autómata que aceptó el lexema o un array vacío. En caso de que todos los autómatas lleguen a un estado trampa, se rompe el ciclo. En caso contrario, se agrega el autómata a la lista de "tokens_posibles_extra_char".[2] Finalmente, avanzamos un carácter en el código fuente y aquí es donde termina el ciclo. Este ciclo lo utilizamos para encontrar las subcadenas correspondientes a los tokens.

Por último, si hay posibles tokens los agregamos a la lista de tokens, de manera contraria se agrega a la lista de tokens inválidos y retornamos ambas listas.

1. Para obtener si el lexema ingresado llega a un estado trampa, hacemos "autómata(lexema)", en donde va a variar el autómata y así utilizamos la función correspondiente a cada autómata de la tupla (estas funciones se encuentran en el archivo de los autómatas).
2. Se utiliza la lista "tokens_posibles" para guardar los autómatas que aceptan el lexema, y "tokens_posibles_extra_char" para guardarlos luego de avanzar un carácter, para que cuando se rompa el ciclo por tener una cadena que incluye un espacio en blanco, queden guardados los autómatas aceptados para el lexema con un carácter menos.

Ejemplos de cadenas prueba

Cadenas prueba	Resultados
lexer("si 20 &&&&&&& 5 && 4 & 2")	- AUTOMATA: si - LEXEMA: si - AUTOMATA: num - LEXEMA: 20 - AUTOMATA: num - LEXEMA: 5 - AUTOMATA: num - LEXEMA: 4 - AUTOMATA: num - LEXEMA: 2
lexer("cont*")	- AUTOMATA: id - LEXEMA: cont* - AUTOMATA: * - LEXEMA: *
lexer("333vaux")	- AUTOMATA: num - LEXEMA: 333v - AUTOMATA: id - LEXEMA: vaux
lexer("si 10 esMenorQue 20 entonces mostrar 21")	- AUTOMATA: si - LEXEMA: si - AUTOMATA: num - LEXEMA: 10 - AUTOMATA: esMenorQue - LEXEMA: esMenorQue - AUTOMATA: num - LEXEMA: 20 - AUTOMATA: entonces - LEXEMA: entonces - AUTOMATA: mostrar - LEXEMA: mostrar - AUTOMATA: num - LEXEMA: 21
lexer("mostrar 80 sino 21 21 + 55")	- AUTOMATA: mostrar - LEXEMA: mostrar - AUTOMATA: num - LEXEMA: 80 - AUTOMATA: sino - LEXEMA: sino - AUTOMATA: num - LEXEMA: 21 - AUTOMATA: num - LEXEMA: 21 - AUTOMATA: + - LEXEMA: + - AUTOMATA: num - LEXEMA: 55
lexer("mientras 0 esMenorQue 1 hacer mostrar 55 sino hacer mostrar -1")	AUTOMATA: mientras LEXEMA: mientras AUTOMATA: num LEXEMA: 0 AUTOMATA: esMenorQue LEXEMA: esMenorQue AUTOMATA: num LEXEMA: 1 AUTOMATA: hacer LEXEMA: hacer AUTOMATA: mostrar LEXEMA: mostrar AUTOMATA: num LEXEMA: 55

	AUTOMATA: sino LEXEMA: sino AUTOMATA: hacer LEXEMA: hacer AUTOMATA: mostrar LEXEMA: mostrar AUTOMATA: num LEXEMA: 1
lexer("si 40 entonces (contador eq 21) entonces 10 mientras 0 eq 20")	- AUTOMATA: si - LEXEMA: si - AUTOMATA: num - LEXEMA: 40 - AUTOMATA: entonces - LEXEMA: entonces - AUTOMATA: (- LEXEMA: (c - AUTOMATA: id - LEXEMA: contador - AUTOMATA: eq - LEXEMA: eq - AUTOMATA: num - LEXEMA: 21) - AUTOMATA:) - LEXEMA:) - AUTOMATA: entonces - LEXEMA: entonces - AUTOMATA: num - LEXEMA: 10 - AUTOMATA: mientras - LEXEMA: mientras - AUTOMATA: num - LEXEMA: 0 - AUTOMATA: eq - LEXEMA: eq - AUTOMATA: num - LEXEMA: 20
lexer("&&/$$$···### 2 + 2 ··$")	- AUTOMATA: num - LEXEMA: 2 - AUTOMATA: + - LEXEMA: + - AUTOMATA: num - LEXEMA: 2
lexer("(((((((((( ))))))))))")	- AUTOMATA: (- LEXEMA: ((((((((((- AUTOMATA:) - LEXEMA:))))))))))
lexer("contador eq 20 si (2 * 10) eq contador mostrar esNumeroPar")	- AUTOMATA: id - LEXEMA: contador - AUTOMATA: eq - LEXEMA: eq - AUTOMATA: num - LEXEMA: 20 - AUTOMATA: si - LEXEMA: si - AUTOMATA: (- LEXEMA: (- AUTOMATA: num - LEXEMA: 2 - AUTOMATA: * - LEXEMA: * - AUTOMATA: num - LEXEMA: 10 - AUTOMATA:) - LEXEMA:) - AUTOMATA: eq - LEXEMA: eq - AUTOMATA: id - LEXEMA: contador - AUTOMATA: mostrar - LEXEMA: mostrar

	`- AUTOMATA:  id  - LEXEMA: esNumeroPar`