MANUAL TÉCNICO

PROYECTO #1

USAC
TRICENTENARIA
Universidad de San Carlos de Guatemala

FIUSAC
Universidad de San Carlos de Guatemala

Organización de Lenguajes y de Compiladores 1

Javier Alejandro Gutierrez de León

202004765

Guatemala, 6 de marzo de 2022

Introducción

Se solicita crear un sistema que pueda realizar el metodo del arbol y el metodo de Thompson, con el fin de que los estudiantes puedan verificar sus resultados en hojas de trabajo y examenes cortos, siendo estos ingresados a travez de un archivo de extension ***".exp"*** utilizando un lenguaje en el cual se definan los conjuntos, las expresiones regulares en notacion polaca o prefija y con esto lograr analizar si son validas cadenas que se ingresen en la segunda seccion del archivo de entrada, generando reportes de cada paso realizado para la elaboracion de los metodos por el que se analizan las expresiones regulares, tambien generando un archivo de salida en formato JSON para mostrar si las cadenas son validas para la expresion regular a la que se asigno y ademas se genera un reporte de errores para notificar acerca de errores lexicos y sintacticos que puedan ser detectados a la hora de la ejecucion.

Objetivos

Objetivos Generales

- ✔ Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para construcción de una solución de software que permita generar análisis por medio del método del árbol.

Objetivos Específicos

- ✔ Reforzar el concepto del método de Árbol de expresiones regulares en Autómatas Finitos Deterministas (AFD).
- ✔ Reforzar el concepto del método de Thompson de expresiones regulares en Autómatas Finitos No Deterministas (AFND).
- ✔ Identificar y programar el proceso de reconocimiento de lexemas mediante el uso de Autómatas Finitos Determinista.

Especificación Técnica

Requisitos de Hardware

- RAM: 128 MB
- Procesador: Mínimo Pentium 2 a 266 MHz
- Espacio en disco: 124 MB para JRE; 2 MB para Java Update; 45MB para programa fuente

Requisitos de software

- **Sistema operativo**

  - Windows 10 (8u51 y superiores)
  - Windows 8.x (escritorio)
  - Windows 7 SP1
  - Windows Vista SP2
  - Windows Server 2008 R2 SP1 (64 bits)
  - Windows Server 2012 y 2012 R2 (64 bits)
  - Oracle Linux 5.5+1
  - Oracle Linux 6.x (32 bits), 6.x (64 bits)2
  - Oracle Linux 7.x (64 bits)2 (8u20 y superiores)
  - Red Hat Enterprise Linux 5.5+1 6.x (32 bits), 6.x (64 bits)2
  - Red Hat Enterprise Linux 7.x (64 bits)2 (8u20 y superiores)
  - Suse Linux Enterprise Server 10 SP2+, 11.x
  - Suse Linux Enterprise Server 12.x (64 bits)2 (8u31 y superiores)
  - Ubuntu Linux 12.04 LTS, 13.x
  - Ubuntu Linux 14.x (8u25 y superiores)
  - Ubuntu Linux 15.04 (8u45 y superiores)
  - Ubuntu Linux 15.10 (8u65 y superiores)
  - Mac con Intel que ejecuta Mac OS X 10.8.3+, 10.9+

- **Exploradores (Para visualizar reporte de errres):**
  - Google Chrome (en todas las plataformas)
  - Mozilla Firefox (en todas las plataformas)
  - Internet Explorer (Windows)
  - Microsoft Edge (Windows, Android, iOS, Windows 10 móvil)
  - Safari (Mac, iOS)
  - Opera (Mac, Windows)

- **Lenguaje de Programación e IDE**

  Para el desarrollo de este software se utilizó Netbeans 8.2

- **Tecnologías utilizadas**
  - **Java:** Es un lenguaje de programación de alto nivel , basado en clases y orientado a objetos que está diseñado para tener la menor cantidad posible de dependencias de implementación.
  - **HTML5:** lenguaje de marcado para la elaboración de páginas web, con el que se realizaron los reportes de errores lexicos y sintacticos.
  - **Boostrap:** Librería para el diseño del reporte HTML.
  - **Graphviz:** Es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT.

Creacion del analisis Léxico

Ejemplo de cadenas que se deben reconocer:

```
{
//CONJUNTOS
CONJ:letra->a~z;
CONJ:digito->0~9;
CONJ:especial->!~&;
CONJ: vocales->a,e,i,o,u;

//EXPRESIONES REGULARES
ExpReg1 -> .{letra}+|"_"|{letra}{digito};
ExpresionReg2 -> .{letra}*|"_"|{letra}{digito};
decimal->..+{digito}"."+{digito};
Raro->+..{vocales}?{especial}"\n";

%%
%%

ExpReg1: "Lexemaprimero";
ExpresionReg2 : "33.3";
}
```

Tabla de tokens

Token	Descripcion	Expresion regular
Reservadas	Secuencia de una o más letras que forman la palabra CONJ	"CONJ"
Se ignora	Comentarios de una línea	//.*
Se ignora	Comentarios multilínea	<! (.*|\n)*"!>"
FLECHA	Secuencia de un guion y una flecha	"->"
TILD	Único símbolo de virgulilla o tilde	"~"
Línea	Salto de línea	"\n"
PTCOMA	Único símbolo de punto y coma	";"
LLAVEI	Único símbolo Llave inicial	"{"
LLAVEF	Único símbolo Llave final	"}"
PARENTESISI	Único símbolo Paréntesis inicial	"("
PARENTESISF	Único símbolo Paréntesis final	")"
CORCHETEI	Único símbolo Corchete inicial	"["
CORCHETEF	Único símbolo Corchete final	"]"
DPUNTOS	Único símbolo Dos puntos	":"
PUNTO	Único símbolo Punto	"."
COMA	Único símbolo Coma	","
ASTERISCO	Único símbolo Asterisco	"*"
BARRA	Único símbolo Barra	"|"
MAS	Único símbolo Mas	"+"
INTERROG	Único símbolo Interrogación	"?"
SIMBOLO	Único símbolo entre el ASCCII 32 y 125 omitiendo letras y dígitos	[|"!"|"\""|"#"|"$"|"%"|"&"|"\'"|"("|")"|"*"|"+"|","|"-"|"."|"/"|":"|";"|"<"|"="|">"|"?"|"@"|"["|"\\"|"]"|"^"|"_"|"`"|"{"|"|"|"}"]
Ignorar	Espacios en blanco o tabulación	[\t\r]+
LETRAU	Única letra, ya sea en mayúscula o minúscula	[a-zA-Z]
DIGITU	Único digito	[0-9]
Identificador	Secuencia de letras, numero o guion bajo, siempre debe empezar por letra	{[a-zA-Z_]+}({[a-zA-Z_]+}|{[0-9]+})*

Gramática Libre de Contexto

G=(V,T,P,S)

V Conjunto de variables

decConjuntos, tConjuntos,vConj, vConjL, vConjD, Probador, ExpReg, ER, CONTENIDO, CABECERA, PIE, CARACT

T Conjunto de terminales

PTCOMA, Reservadas, SEPARADOR, STRING, LLAVEI, LLAVEF, DPUNTOS, FLECHA, Identificador, TILD, ERROR, LETRAU, DIGITU, SIMBOLO,PUNTO, ASTERISCO, BARRA, PARENTESISI, PARENTESISF, CORCHETEI, CORCHETEF, COMA, INTERROG, MAS, COMDOB

P Conjunto finito de producciones

<INICIO>::= LLAVEI <CONTENIDO> LLAVEF

<CONTENIDO>::= <CABECERA> <SEPARADOR> <PIE> | error LLAVEF

<CABECERA>::= <decConjuntos> <CABECERA> | <ExpReg> <CABECERA> | <decConjuntos> | <ExpReg> | error PTCOMA

<PIE>::= Probador <PIE> | <Probador> | error PTCOMA

<decConjuntos>::= Reservadas DPUNTOS Identificador FLECHA <tConjuntos>

<tConjuntos>::= LETRAU TILD LETRAU | DIGITU TILD DIGITU | <vConj> | CARACT TILD CARACT

<vConj>::= LETRAU| LETRAU <vConjL> | DIGITU | DIGITU <vConjD>

<vConjL>::= COMA LETRAU <vConjL> | COMA LETRAU

<vConjD>::= COMA DIGITU <vConjD> | COMA DIGITU

<CARACT>::= PTCOMA | LLAVEI | LLAVEF | PARENTESISI | PARENTESISF | CORCHETEI | CORCHETEF | DPUNTOS | PUNTO | COMA | ASTERISCO | BARRA | MAS | INTERROG | COMDOB | SIMBOLO

<Probador>::= Identificador DPUNTOS STRING PTCOMA

<ExpReg>::= Identificador FLECHA <ER> PTCOMA

<ER>::= LLAVEI Identificador LLAVEF | ASTERISCO | MAS | INTERROG | BARRA | PUNTO | STRING | CORCHETEI <tConjuntos> CORCHETEF | LLAVEI Identificador LLAVEF <ER>| ASTERISCO <ER> | MAS <ER> | INTERROG <ER> | BARRA <ER> | PUNTO <ER> | STRING <ER> | CORCHETEI tConjuntos CORCHETEF <ER>

S Simbolo Inicial

INICIO

Lógica del programa (Métodos principales)

Nombre	Descripción	Parámetros
sintactico.parse()	Metodo con el cual se ejecuta el analisis Lexico y Sintactico	• **Ninguno**
Metodos.ordenarExpresionRegular(ER)	Convierte la expresion regular de la notacion prefija a la normal, comienza a armar el arbol en el proceso.	• **ER: Expresion regular**
Metodos.parametrosArbolito(ac)	Verifica los nodos anulables del arbol y enumera los nodos	• **Ac: nodo inicial**
Metodos.listaPrimeros(ac)	Coloca los primeros de cada nodo	• **Ac: nodo inicial**
Metodos.listaUltimos(ac)	Coloca los ultimos de cada nodo	• **Ac: nodo inicial**
Metodos.grafica(ac, er)	Grafica el arbol en dot y convertirla a imagen.	• **ac: nodo inicial** • **er: string con el identidficador y la expresion regular que representa el arbol**
Metodos.tablaSiguientes(ac,nombre,ER)	Genera la tabla de siguientes en base a los ultimos y primeros de cada nodo del arbol, ademas de generar la tabla en dot y convertirla a imagen.	• **Ac: nodo inicial** • **Nombre: identificador de la expresion regular que representa.** • **ER: Expresion regular que representa**
transicions.tablaEstados(ts, ac, nombre,ER)	Crea la tabla de estados en base a lo obtenido en la tabla de siguientes ademas de generar la tabla en dot y convertirla a imagen.	• **Ac: nodo inicial** • **Nombre: identificador de la expresion regular que representa.** • **ER: Expresion regular que representa** • **Ts: LinkedList con los datos de la tabla de siguientes.**
transicions.Automata(ac,nombre,ER)	Genera el automata en dot y lo converte a imagen.	• **Ac: nodo inicial** • **Nombre: identificador de la expresion regular que representa.** • **ER: Expresion regular que representa**

AnalizarCadena.analiza(CONJ, PRUEBA, automatas);	Analiza la cadena, en el AFD generado para el automata con el que se declara.	• **CONJ: LinkedList con los conjutos declarados** • **PRUEBA: LinkedList con las cadenas que se desean probar.** • **Automatas: LinkedList con los AFD generados.**
Thompson.grafica1(c1,simbolo,c2);	Genera con el metodo de thompson el AFND para el carácter 1 y carácter 2.	• **c1: primer carácter del fragmento de la expresion regular.** • **c2: segundo carácter del fragmento de la expresion regular.** • **Simbolo: Simbolo con el cual estan unidos los simbolos.**
Thompson.grafica2(c1,simbolo);	Genera con el metodo de thompson el AFND para el carácter 1.	• **c1: primer carácter del fragmento de la expresion regular.** • **Simbolo: Simbolo para el carácter 1.**
Thompson.grafica(grafo, cadena,nombre);	Grafica el grafo que se ha generado durante la ejecucion.	• **Grafo: AFND que se ha generado en la ejecucion.** • **Cadena: ER que representa.** • **Nombre: Nombre de la ER que representa.**

Conclusiones

La utilizacion de las herramientas JFlex y CUP ayudaron bastante en el desarrollo de la aplicación ya que simplifican bastante el trabajo a la hora de crear el analizador lexico y sintactico ya que la cantidad de codigo creado se reduce en gran medida en comparacion al que se tenia que hacer paso a paso en el curso anterior (Lenguajes Formales y de Programacion), por lo cual se podia dedicar mas tiempo a realizar las funciones requeridas.