Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Segundo Semestre 2023
Catedráticos:
Ing. Mario Bautista
Laboratorista:
Ing. Kevin Lajpop

SubSetify

Tejiendo AFDs desde AFNs

Proyecto 1

Tabla de contenido

1. Objetivo General

Aplicar los conocimientos sobre las fases de análisis léxico de un compilador para la construcción de una solución de software.

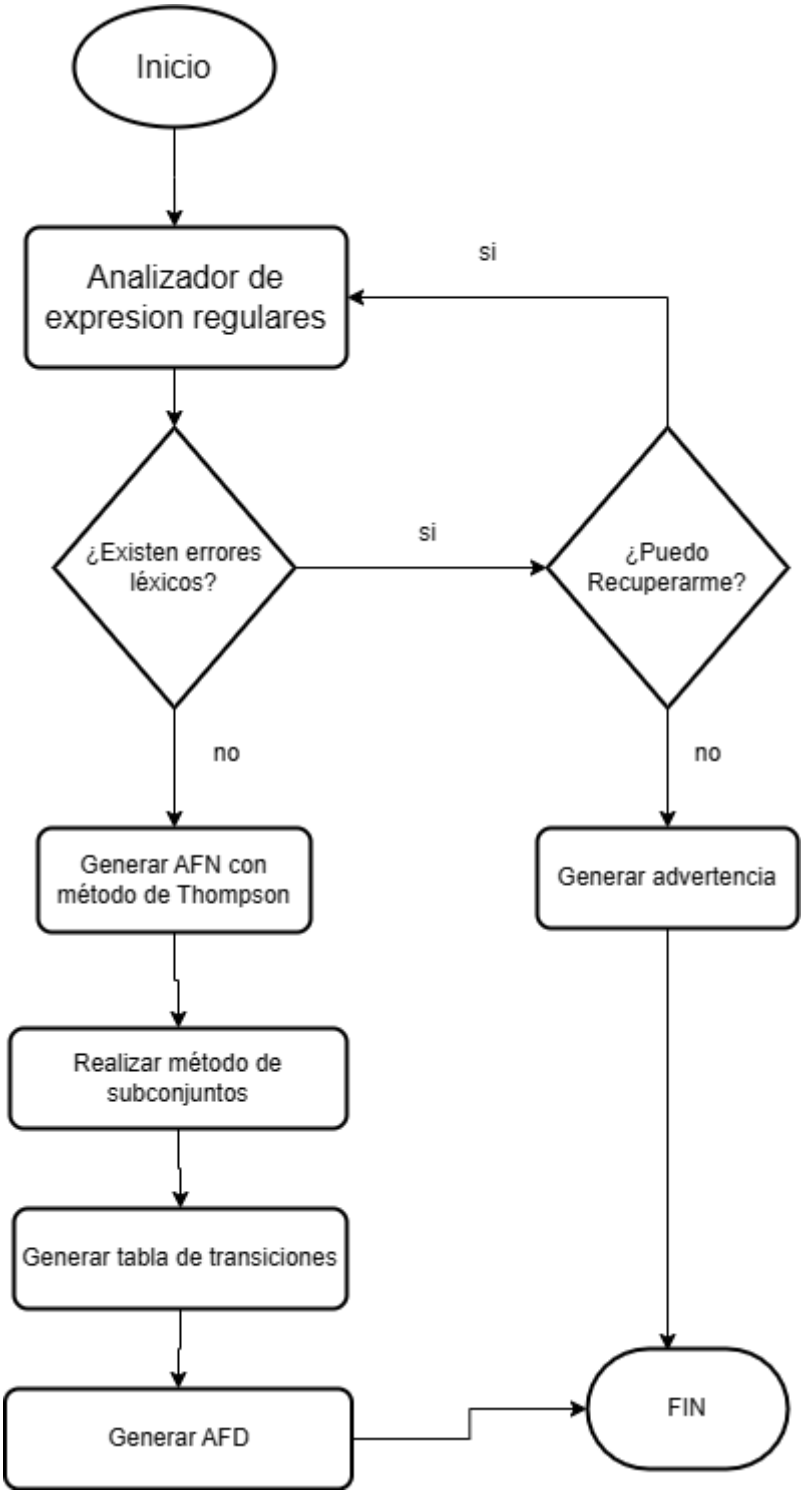
2. Objetivos Específicos

- Que el estudiante aprenda a generar analizadores léxicos y sintácticos utilizando las herramientas de JFLEX y CUP.
- Que el estudiante comprenda los conceptos de token, lexema, patrones y expresiones regulares.
- Que el estudiante pueda realizar correctamente el manejo de errores léxicos.
- Que el estudiante pueda reforzar los conceptos sobre el método de Thompson y el método de subconjuntos para la transformación de un autómata tipo AFND a un AFD

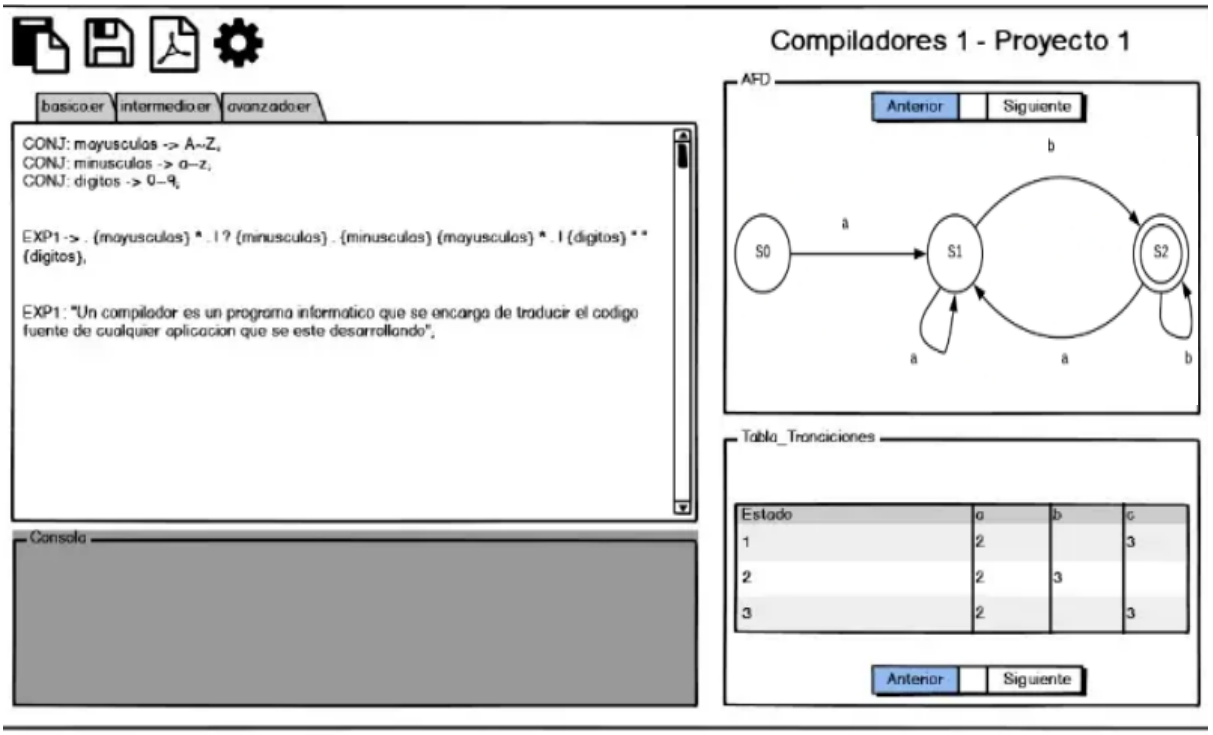
3.Descripción

El proyecto "SubSetify" es una solución de software que consiste en aplicar los conocimientos sobre las fases de análisis léxico aplicando el método de Thompson y el método de subconjuntos.Debido a lo tedioso que puede llegar a ser el realizar el método de subconjuntos para hacer la conversión de un AFN a un AFD, se le solicita a los estudiantes de sistemas que desarrollen un programa capaz de automatizar este proceso.

4. Flujo recomendado



5. Área de Edición y Resultados



5.1 Funcionalidades:

- **Abrir Archivo:** El editor debe tener la capacidad de abrir archivos con las extensiones .ss cuyo contenido se mostrará en la entrada.
- **Guardar:** Se debe permitir guardar la entrada sobre el archivo en el que se está trabajando.
- **Guardar Como:** Se debe permitir guardar la entrada en un nuevo archivo.

5.2 Herramientas:

- **Editor de Entrada:** Se debe contar con un área de texto que permita ingresar y modificar el programa fuente.
- **Consola de Salida:** Mostrará los resultados de errores léxicos y mostrará el .
- **Ejecutar:** Ejecutar el analizador

6. Especificación del programa fuente

6.1 Case insensitive

El lenguaje no distinguirá entre mayúsculas o minúsculas.

```
CONJ:
conj:
Exp ->
EXP ->
```

6.2 Comentarios

6.2.1 Comentarios de una línea

```
// Este es un comentario de una línea
```

6.2.2 Comentarios multilínea

```
<!
        Este es un comentario
            multilínea
!>
```

6.3 Definición del lenguaje

El archivo deberá de contener la definición de las expresiones regulares se compone de una sección en la que cada sentencia define el token (identificador)

con que el analizador debe reconocer los lexemas ingresados, seguido de la definición de la expresión regular.

Cada sentencia se delimita utilizando punto y coma.

```
{
////// CONJUNTOS
CONJ: nombre_conjunto -> notacion;
CONJ: nombre_conjunto -> notacion
tld -> Expresión_regular_en_prefijo;
tid -> Expresión_regular_en_prefijo;
// Mas sentencias

// Mas sentencias
```

6.4 Expresiones regulares

Las expresiones regulares establecen el patrón que se debe cumplir para representar un token, estas se reconocerán en notación polaca o prefija. A continuación, se muestra la notación a utilizar.

Notación	Definición
. a b	Concatenación entre a y b
| a b	Disyunción entre a y b
* a	0 o más veces
+ a	1 o más veces
? a	0 o una vez

* **a** & **b** pueden ser caracteres, conjuntos designados con anterioridad o caracteres especiales.

6.5 Conjuntos

Para la definición de conjuntos se utiliza la palabra reservada “CONJ”. Un conjunto puede utilizarse dentro de una expresión regular, pero no en la definición de otro conjunto.

A continuación, la notación a utilizar para la definición de conjuntos

Notación	Definición
a~c	Conjunto {a, b, c}.
a~z	Conjunto de la a hasta la z en minúsculas.
A~Z	Conjunto de la A hasta la Z en mayúsculas.
0~7	Conjunto del 0 al 7.
0,2,4,6,8	Conjunto {0, 2, 4, 6, 8}
A,b,C,d	Conjunto {A, b, C, d}
!~&	Conjunto de signos entre ! (33 en código ascii) y & (38 en código ascii). Nota: el rango valido será desde el ascii 32 hasta 125 omitiendo los ascii de las letras y dígitos.

Los conjuntos vistos anteriormente son ejemplos de las diferentes variantes que éstos pueden tomar (Ej: también puede existir a~d o 0~9).

6.6 Caracteres especiales

Notación	Definición
\n	Salto de línea
\’	Comilla Simple
\”	Comilla Doble

6.7 Ejemplo

Notas:

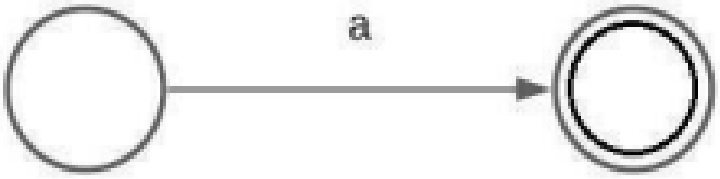
- La definición de conjuntos CONJ puede existir en cualquier parte del archivo.
- El uso de conjuntos se verá delimitado por **{** llaves **}**

```
{
// CONJUNTOS
CONJ: letra -> a~z;
CONJ: digito -> 0-9;
////// EXPRESIONES REGULARES
ExpReg1 -> . (letra) * | * * | (letra) (digito);
```

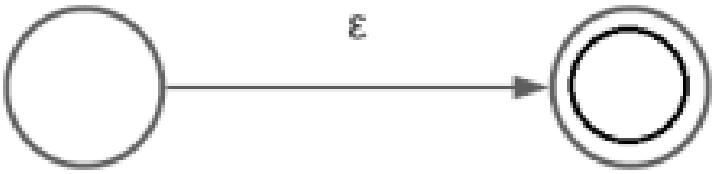
```
ExpresionReg2 -> . (digito). *." + (digito);
RegEx3 -> . (digito) * I *_* | (letra) (digito);

}
```

7. Reglas del método de Thompson

7.1 Transición a:

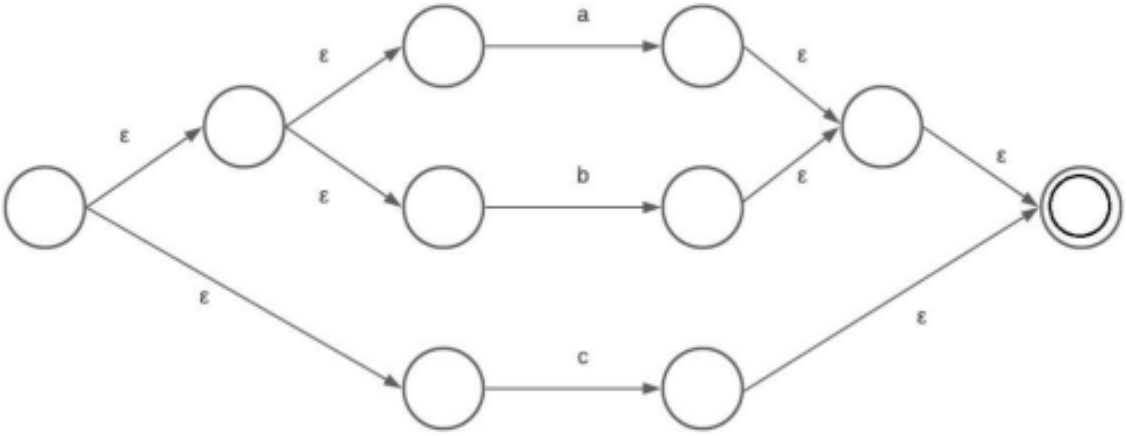


7.2 Transición “ε” :

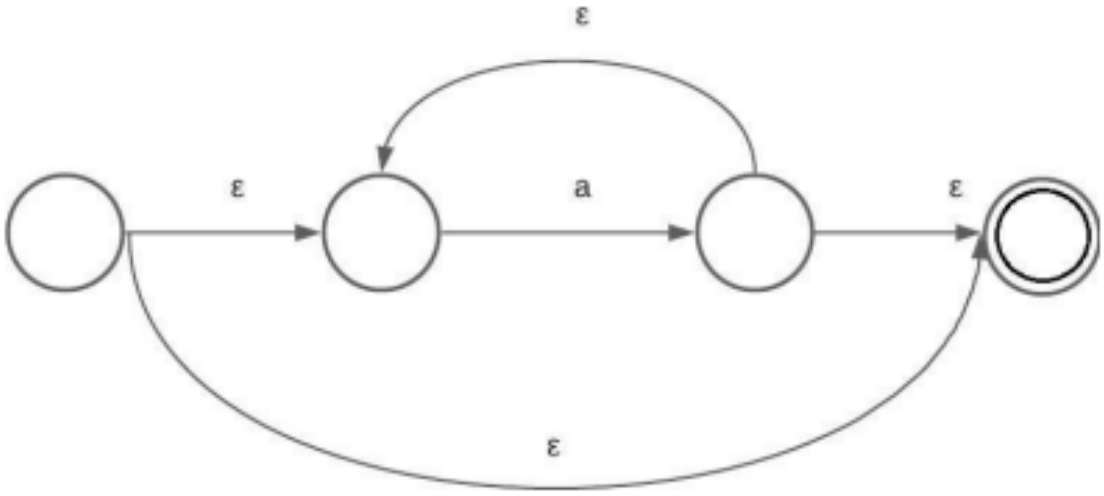


7.3 Transición “ab” :

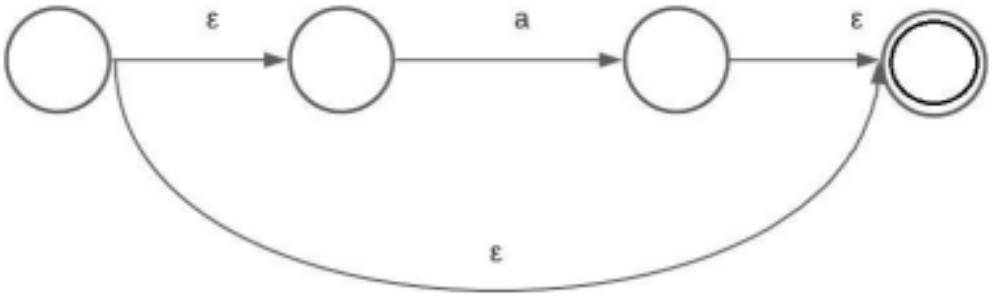
7.4 Transición “a|b|c” :



7.5 Transición “a*” :



7.7 Transición “a?” :



8. Conversión AFN a AFD por método de subconjuntos

La idea general de la construcción de subconjuntos es que cada estado del AFD construido corresponde a un conjunto de estados del AFN. Después de leer la entrada a1a2 ⋯ an, el AFD se encuentra en el estado que corresponde al conjunto de estados que el AFN puede alcanzar, desde su estado inicial, siguiendo los caminos etiquetados como a1a2 ⋯ an.
Es posible que el número de estados del AFD sea exponencial en el número de estados del AFN, lo cual podría provocar dificultades al tratar de implementar este AFD. No obstante, parte del poder del método basado en autómatas para el análisis léxico es que para los lenguajes reales, el AFN y el AFD tienen aproximadamente el mismo número de estados, y no se ve el comportamiento exponencial.

Su algoritmo construye una tabla de transición Dtran para D. Cada estado de D es un conjunto de estados del AFN, y construimos Dtran para que D pueda simular “en paralelo” todos los posibles movimientos que N puede realizar sobre una cadena de entrada dada. Nuestro primer problema es manejar las transiciones de N en forma apropiada. En la figura 3.31 vemos las definiciones de varias funciones que describen cálculos básicos en los estados de N que son necesarios en el algoritmo. Observe que s es un estado individual de N,mientras que T es un conjunto de estados de N.

OPERACIÓN	DESCRIPCIÓN
ϵ-*cerradura*(s)	Conjunto de estados del AFN a los que se puede llegar desde el estado s del AFN, sólo en las transiciones ϵ.
ϵ-*cerradura*(T)	Conjunto de estados del AFN a los que se puede llegar desde cierto estado s del AFN en el conjunto T, sólo en las transiciones ϵ; $= \cup_{s \text{ en } T}$ ϵ-*cerradura*(s).
mover(T, a)	Conjunto de estados del AFN para los cuales hay una transición sobre el símbolo de entrada a, a partir de cierto estado s en T.

Figura 3.31: Operaciones sobre los estados del AFN

Procedimiento
1. Se aplica la operación sobre el estado inicial del AFN:

Cerradura(0)=A

El conjunto de estados resultantes debe nombrarse, en este ejemplo lo llamaremos el conjunto "A".
.
2. Debe crearse un nuevo conjunto de estados para cada terminal "a" de la forma:

Cerradura(Mover(A,a))=B

Este nuevo conjunto deberá de nombrarse por ejemplo "B". Si ya existe un conjunto de estados idénticos al resultado de la operación se le coloca el mismo nombre.
3. Se repite el paso 2 para cada conjunto de estados que se genere como resultado del mismo. Este paso acaba cuando ya no hay más conjuntos nuevos a los que aplicar el paso 2.
4. Se toman los conjuntos de estados como los nuevos estados del AFD y para cada operación Mover aplicada se utiliza como una transacción en la Tabla de Transiciones. Cada conjunto de estados que posea al último estado del método de Thompson se convierte en un estado de aceptación.
El módulo deberá generar tanto el AFN como el AFD para poder mostrarlos por cada expresión analizado. En las gráficas del AFN y AFD se debe mostrar los estados del autómata, el estado inicial, los estados de aceptación y las transiciones del mismo.

9. Reportes

9.1 Reporte de tokens

Se deberá generar un reporte en HTML con todos los tokens y lexemas reconocidos durante la fase de análisis léxico. Se debe considerar el siguiente formato:

Lexema	Token	LÍnea	Columna
Conj	Res_conj	5	1
expr1	identificador	12	5
expr2	identificador	25	10

9.2 Reporte de errores léxicos

Se deberá generar un reporte en HTML con todos los errores y lexemas reconocidos durante los análisis correspondientes. Se debe considerar el siguiente formato

Lexema	Descripción	LÍnea	Columna
°	error léxico	5	1
¬	error léxico	7	10

10. Entregables

10.1 Manual de Usuario

Deberá entregar un manual de usuario en donde se detalle cómo se hace uso de la aplicación, es de vital importancia que se adicional a incluir capturas de pantalla sobre el funcionamiento de la aplicación el estudiante incluya cómo funciona el lenguaje, esto con el objetivo de que en caso de que cualquier usuario haga uso de la aplicación, sepa cómo funciona tanto el lenguaje de generación de reportes como el editor de texto creado para la aplicación.

10.2 Manual técnico

En este manual el estudiante deberá incluir toda la información que considere importante para el caso en el que otro desarrollador desee darle mantenimiento o realizar mejoras en el código fuente, se recomienda incluir versiones de las herramientas utilizadas, especificaciones del entorno en donde se desarrolló la solución, diagramas de clases, etc.

10.3 Archivos de gramática

El estudiante deberá incluir la gramática que utilizo tanto para el lenguaje StatPy como también la gramática que se realizó para analizar los archivos JSON, es de vital importancia que no se realice una copia literal del archivo utilizado para la herramienta CUP, en este archivo deberán escribir las gramáticas con el formato **BNF(Backus-Naur form)**

10.4 Requerimientos mínimos

Para tener derecho a calificación, el estudiante deberá cumplir con los siguientes requerimientos mínimos:

- Carga de los archivos
- analizador para los archivos descritos para el proyecto
- Manual de Usuario.
- Manual Técnico.
- Archivo de gramáticas.

10.5 Restricciones

- Se debe hacer uso de un repositorio en Github para realizar la entrega de su proyecto, Nombre del repositorio: **[OLC1]Proyecto1_#Carnet** no olvidar invitar al laboratorista.
- Lenguajes de programación a usar: Java
- Herramientas de análisis léxico y sintáctico: JFlex/CUP
- Herramienta para graficar: se recomienda el uso de graphviz para la graficación del AFN y AFD
- **El proyecto es individual**
- Copias completas/parciales de: código, gramática, etc. serán merecedoras de una nota de 0 puntos, los responsables serán reportados al catedrático de la sección y a la Escuela de Ciencias y Sistemas

10.6 Fecha de Entrega

Sábado 16 de diciembre de 2023 la entrega se realizará por medio de UEDI, en caso exista algún problema, se estará habilitando un medio alterno.