Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Segundo Semestre 2024
Catedráticos:
Ing. Manuel Castillo
Ing. Kevin Lajpop
Ing. Mario Bautista
Tutores académicos:
Fabian Reyna
Carlos Acabal
Xhunik Miguel

ConjAnalyzer
Proyecto 1

Tabla de Contenido

1. Objetivos

1.1 Objetivos Generales

Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para la construcción de una solución de software.

1.2 Objetivos específicos

- Que el estudiante aprenda a generar analizadores léxicos y sintácticos utilizando las herramientas de JFLEX y CUP.
- Que el estudiante aprenda los conceptos de token, lexema, patrones y expresiones regulares.
- Que el estudiante pueda realizar correctamente el manejo de errores léxicos.
- Que el estudiante sea capaz de realizar acciones gramaticales utilizando el lenguaje de programación JAVA.

2. Descripción General

El curso de Organización de Lenguajes y Compiladores 1, perteneciente a la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, requiere de usted, como conocedor de la construcción de analizadores Léxico y Sintáctico, crear un sistema que sea capaz de realizar operaciones entre conjuntos para que los estudiantes del curso Matemática para Computación 1 puedan verificar las respuestas de sus tareas y exámenes del curso.

3. Entorno de Trabajo

3.1 Editor

El editor será parte del entorno de trabajo, cuya finalidad será proporcionar ciertas funcionalidades y herramientas que serán de utilidad para el usuario. La función principal del editor será el ingreso del código fuente que será analizado. Queda a discreción del estudiante el diseño. Queda a discreción del estudiante el diseño.

3.2 Funcionalidades

- **Nuevo Archivo**: se podrá crear un nuevo archivo con extensión [.ca].
- **Guardar**: nos permitirá guardar el archivo actual
- **Guardar como**: nos permitirá guardar el archivo con otro nombre.

3.3 Herramientas

- **Ejecutar**: se envía la entrada actual al intérprete con la finalidad de realizar el análisis léxico, sintáctico y la ejecución de instrucciones.

3.4 Reportes

- **Reporte de Tokens**: se mostrarán todos los tokens reconocidos en el analizador léxico.
- **Reporte de Errores**: se mostrarán todos los errores léxicos y sintácticos encontrados.

3.5 Consola

En la consola de salida se mostrarán los resultados, mensajes y todo lo que sea indiciado en el lenguaje. Tiene como restricción el no ser editable por el usuario y únicamente puede mostrar información.

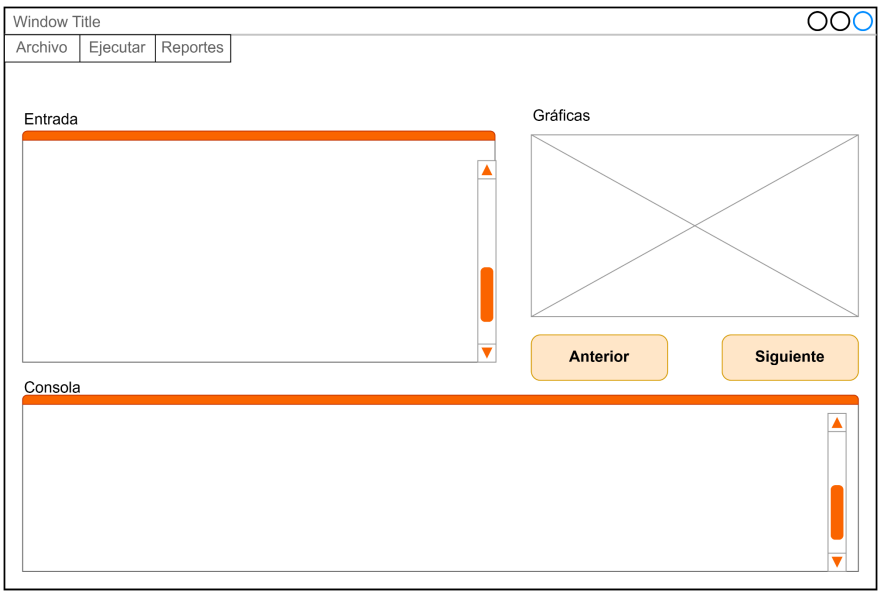


Figura 1. Propuesta de interfaz

4. Descripción del Lenguaje

4.1 Case Sensitive

El lenguaje es case sensitive por lo que reconoce entre mayúsculas y minúsculas.

```
CONJ : conjuntoA -> a~z;
CONJ : conjuntoB -> A~Z;
# Nota: Son conjuntos diferentes
```

4.2 Encapsulamiento

Todas las sentencias del lenguaje deben venir entre llaves.

```
{
    <CÓDIGO>
}
```

4.3 Comentarios

El lenguaje permite el manejo de comentarios que son ignorados por los analizadores con el objetivo de dejar mensajes a otros desarrolladores en el código fuente.

4.3.1 Comentarios de una línea

Este comentario comenzará con # y deberá terminar con un salto de línea.

4.3.2 Comentarios multilínea

Este comentario comenzará con <! y terminará con !>.

```
# Esto es un comentario de una sola línea
<!
Esto es un
comentario multilínea
!>
```

4.4 Conjunto Universo

El universo está conformado por los símbolos entre ! (33 en código ascii) hasta el ~ (126 en código ascii).

4.5 Definición de Conjuntos

Para la definición de conjuntos se utiliza la palabra reservada “CONJ”. Un conjunto puede utilizarse dentro de las operaciones pero no dentro de otro conjunto.

CONJ : nombre_conjunto **->** notacion**;**

A continuación se describe la notación para definir su valor:

Notación	Definición
a~c	Conjunto {a, b ,c}
0~4	Conjunto {0, 1, 2, 3, 4}
i, j, k. l	Conjunto {i, j, k. l}

 # Ejemplos
CONJ : conjuntoA -> a~z;
CONJ : conjuntoB -> m, j, d, 1;

4.6 Operaciones

Las operaciones se reconocen en notación polaca o prefija, siguiendo la siguiente forma:

OPERA : nombre_operacion **->** operacion**;**

A continuación se detalla la notación a utilizar:

Notación	Definición
U {conjA} {conjB}	Unión
& {conjA} {conjB}	Intercesión
^ {conjA}	Complemento
- {conjA} {conjB}	Diferencia

#Ejemplos
OPERA : operacion1 -> U U {conjA} {conjB} {conjC}; # A U B U C
OPERA : operacion2 -> & U {conjC} {conjA} {conjB}; # (C U A) ∩ C

4.7 Evaluar elementos

Luego de haber realizado todas las operaciones entre conjuntos se puede validar que cierto conjunto de datos pertenezca al conjunto resultante.

EVALUAR(<CONJUNTO>, nombre_operacion **);**

El conjunto de datos a evaluar únicamente vendrá delimitado por llaves y separando cada elemento por comas.

#Ejemplo
EVALUAR ({1,2,3}, operacion1);

El resultado de la evaluación de los elementos deberá mostrarse en la consola, indicando si fue exitoso o si falló en algún elemento, especificando cuál fue el elemento que falló.

4.8 Ejemplo

```
{
    #Definición de conjuntos
    CONJ : conjuntoA -> 1,2,3,a,b ;
    CONJ : conjuntoB -> a~z;
    CONJ : conjuntoC -> 0~9;

    #Definición de operaciones
    OPERA : operacion1 -> & {conjuntoA} {conjuntoB};
    OPERA : operacion2 -> & U {conjuntoB} {conjuntoC} {conjuntoA};

    #Evaluamos conjuntos de datos
     EVALUAR ( {a, b, c} , operacion1 );
     EVALUAR ( {1, b} , operacion1 );
}
# Salida en consola
<!
==============
Evaluar: operacion1
==============
a -> exitoso
b -> exitoso
c -> fallo
```

```
==============
Evaluar: operacion1
==============
1 -> exitoso
b -> exitoso
!>
```

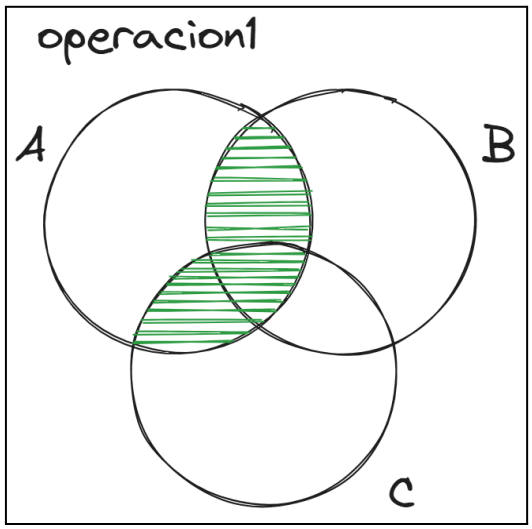
5. Reportes

Los reportes son parte fundamental del sistema, ya que muestra de forma visual las herramientas utilizadas para realizar la ejecución del código. Cada reporte debe ser generado luego de cada ejecución por lo que no deberá mostrarse reportes de análisis anteriores.

A continuación, se muestran ejemplos de estos reportes. Queda a discreción del estudiante el diseño de estos, solo se pide que sean totalmente legibles.

5.1 Gráfica de conjuntos resultantes

Se debe de generar el diagrama de Venn con los conjuntos analizados en cada operación. Este diagrama debe permitir visualizar de forma práctica el área que abarca la operación resultante. Además, debe ser visible en la interfaz gráfica y permitir la navegación a través de los diversos gráficos generados en cada ejecución.



5.2 Tabla de tokens

El reporte de tokens debe tener la información suficiente para poder identificar los tokens reconocidos en el análisis léxico. Este debe ser en formato HTML.

#	Lexema	Tipo	Línea	Columna
1	arr	id	5	3
2	3	Double	6	3
3	“Hola”	String	8	3

5.3 Tabla de errores

El reporte de errores debe contener la información suficiente para detectar y corregir errores en el código fuente. Este debe ser en formato HTML.

#	Tipo	Descripción	Línea	Columna
1	Léxico	El carácter “$” no pertenece al lenguaje	5	3
2	Sintáctico	Se encontró Identificador y se esperaba Expresión.	6	3

5.4 JSON de salida

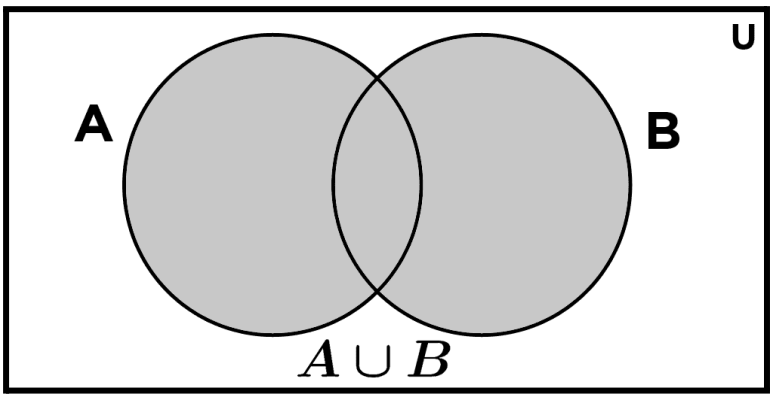
Al finalizar de evaluar las cadenas se debe de usar las propiedades de la teoría de conjuntos indicada en la sección 7 para simplificar la operación si es posible. De cada ley aplicada se debe colocar en un archivo .json en el siguiente formato:

```
{
    "operacion1": {
        "leyes": [
            "ley de doble complemento",
            "leyes de DeMorgan"
             ],
        "conjunto simplificado": "U & {A} {B} {C}"
     },
    "operacion2": "No se puede simplificar la operación"
}
```

6. Operaciones de conjuntos

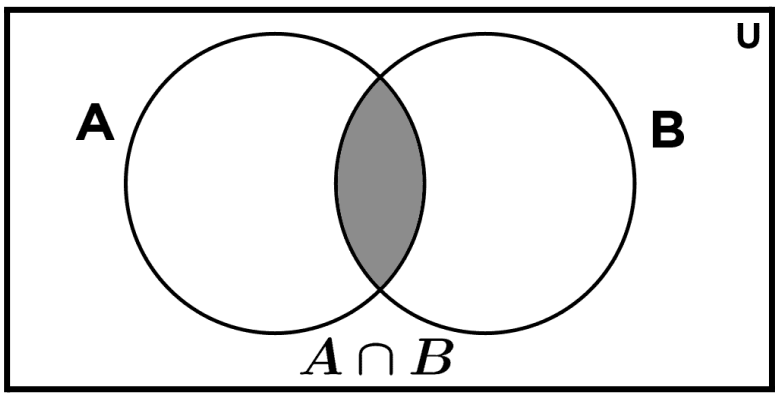
6.1 Unión

Es la operación que nos permite unir dos o más conjuntos para formar otro conjunto que contendrá a todos los elementos que queremos unir pero sin que se repitan. Es decir dado un conjunto A y un conjunto B, la unión de los conjuntos A y B será otro conjunto formado por todos los elementos de A, con todos los elementos de B sin repetir ningún elemento.



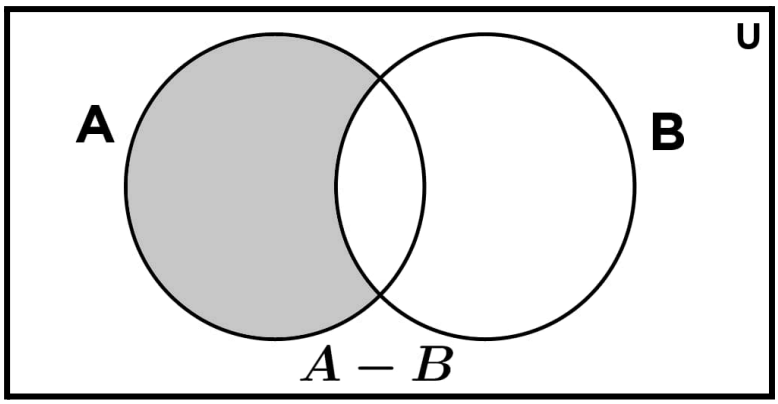
6.2 Intersección

Es la operación que nos permite formar un conjunto, sólo con los elementos comunes involucrados en la operación. Es decir, dados dos conjuntos A y B, la intersección de los conjuntos A y B, estará formado por los elementos de A y los elementos de B que sean comunes, los elementos no comunes A y B, serán excluidos.



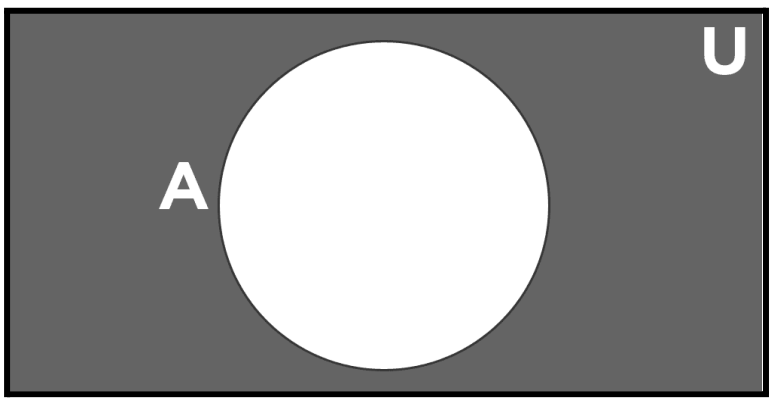
6.3 Diferencia

Es la operación que nos permite formar un conjunto, en donde de dos conjuntos el conjunto resultante es el que tendrá todos los elementos que pertenecen al primero pero no al segundo. Es decir dados dos conjuntos A y B, la diferencia de los conjuntos entra A y B, estará formado por todos los elementos de A que no pertenezcan a B.



6.4 Complemento

Es la operación que nos permite formar un conjunto con todos los elementos del conjunto de referencia o universal, que no están en el conjunto. Es decir dado un conjunto A que está incluido en el conjunto universal U, entonces el conjunto complemento de A es el conjunto formado por todos los elementos del conjunto universal pero sin considerar a los elementos que pertenezcan al conjunto A.



7. Propiedades de la teoría de conjuntos

Ley	Nombre
^ ^A = A	Ley del doble complemento
^(A U B) = ^A ∩ ^B ^(A ∩ B) = ^A U ^B	Leyes de DeMorgan
A U B = B U A A ∩ B = B ∩ A	Propiedades conmutativas
A U (B U C) = (A U B) U C A ∩ (B ∩ C) = (A ∩ B) ∩ C	Propiedades asociativas
A U (B ∩ C) = (A U B) ∩ (A U C) A ∩ (B U C) = (A ∩ B) U (A ∩ C)	Propiedades distributivas
A U A = A A ∩ A = A	Propiedades idempotentes
A U (A ∩ B) = A A ∩ (A U B) = A	Propiedades de absorción

8. Requerimientos Mínimos

Para que el estudiante tenga derecho a calificación, deberá cumplir con lo siguiente:

- Interfaz gráfica funcional
- Carga de archivos
- Analizador Léxico y Sintáctico
- Salidas en consola
- Reportes
- Documentación completa

9. Entregables

- **Código fuente del proyecto**

- **Archivo Ejecutable (JAR)**

- **Manual de Usuario en un archivo pdf**
    - Capturas de pantalla detallando cómo funciona su entorno de trabajo y los reportes que se generan.

- **Manual Técnico en un archivo pdf**
    - Información importante del proyecto para que se pueda realizar el mantenimiento en el futuro. Especificar el lenguaje, herramientas utilizadas, métodos y funciones más importantes.

- **Archivo de Gramática en un archivo txt**
    - El archivo debe contener su gramática y debe de ser limpio, entendible y no debe ser una copia del archivo de Cup.
    - La gramática debe estar escrita en formato **BNF(Backus-Naur form).**

10. Restricciones

- La entrega debe ser realizada mediante UEDI enviando el enlace del **repositorio privado** de Gitlab en donde se encuentra su proyecto y el archivo ejecutable **JAR**.
- El nombre del repositorio de Gitlab debe ser **OLC1_Proyecto1_#Carnet**
- Se debe agregar al auxiliar encargado como colaborador al repositorio de Gitlab.
- Lenguaje de Programación a utilizar: **Java**
- Herramientas para el análisis léxico y sintáctico: **JFlex/Cup**
- **El proyecto debe ser realizado de forma 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.
- La calificación tendrá una duración de 30 minutos, acorde al programa del laboratorio.
- **Solo se calificará sobre el archivo ejecutable JAR**

11. Fecha de Entrega

Domingo, 01 de septiembre de 2024 a las 23:59. La entrega será por medio de la plataforma UEDI. Entregas fuera de la fecha indicada, no se calificarán.

SE LE CALIFICARA DEL ÚLTIMO COMMIT REALIZADO ANTERIOR A ESTA FECHA.