Universidad Nacional Autonoma de Mexico

Materia: Compiladores

Profesor: MIGUEL ANGEL SANCHEZ HERNANDEZ

Proyecto Segundo Parcial: Diapositiva 27

Alumnos:

GARCIA VARGAS ANTONIO JAIR

MENDOZA ROSAS SEBASTIAN

ORTIZ UGALDE LEONARDO ADOLFO

ROMERO NAVARRO DANIEL ALEJANDRO

Grupo: 2609

📘 Implementación de un Autómata Determinista con Pila (PDA)

🎯 Propósito

Usar el archivo base en Java proporcionado para **implementar un autómata determinista con pila (PDA)**.
Se tomó la decisión de **empezar derivando de izquierda a derecha**, con el propósito de detectar primero una cadena que termine en punto y coma (`;`).
Una vez encontrado el punto y coma, **la derivación cambia de dirección (de derecha a izquierda)** para aplicar los axiomas y consumir la entrada con base en los tokens terminales.

📊 Tabla de transición con axiomas y tokens

Axiomas \ Tokens	a	b	c	d
S (inicial)	S := AB	S := AB	error	error
A	A := a	A := λ	error	error
B	error	B := bCd	error	error
C	error	error	C := c	C := λ

🧠 Ejemplo de derivación (Pila de símbolos)

Pila	Entrada	Regla o Acción
`;S`	`abcd;`	`S := AB`
`;BA`	`abcd;`	`A := a`
`;Ba`	`abcd;`	Se saca `a` de la pila y de la entrada
`;B`	`bcd;`	`B := bCd`
`;dCb`	`bcd;`	Se saca `b` de la pila y de la entrada
`;dC`	`cd;`	`C := c`
`;dc`	`cd;`	Se saca `c` de la pila y de la entrada
`;d`	`d;`	Se saca `d` de la pila y de la entrada
`;`	`;`	Aceptar

> 📄 El documento que se analiza como entrada es `archivo.txt`.

🛠 ¿Qué se hizo?

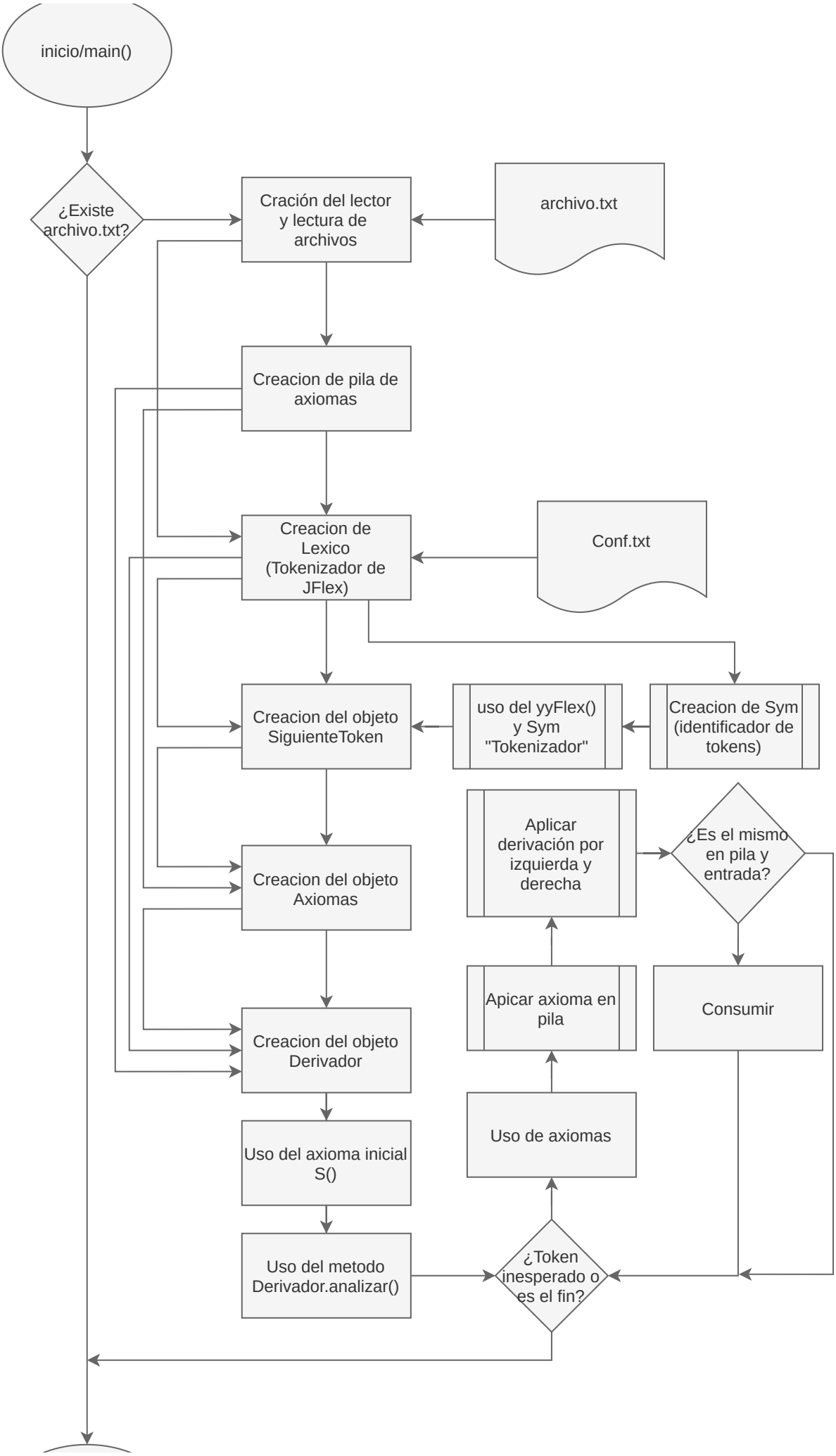
- Se tomó como base un archivo compartido en la clase de Compiladores con el profesor **Miguel Ángel Sánchez Sánchez**.
- Se adaptó para cumplir con los requisitos de un **autómata determinista con pila (PDA)**.
- Antes, el flujo era para identificar A+B: leer tokens y analizarlos directamente.
- Ahora, el análisis se divide en dos fases:
  - **Derivación izquierda → derecha**: construye la pila hasta el punto y coma.
  - **Derivación derecha ← izquierda**: analiza la pila y los tokens aplicando axiomas.

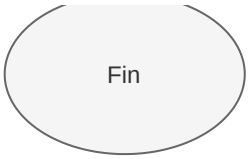
⚔ Comparación del flujo anterior vs. actual

🔙 Antes

> Por cuestiones prácticas está simplificado

🔁 Ahora


inicio/main()
¿Existe archivo.txt?
Cración del lector y lectura de archivos
archivo.txt
Creacion de pila de axiomas
Creacion de Lexico (Tokenizador de JFlex)
Conf.txt
Creacion del objeto SiguienteToken
uso del yyFlex() y Sym "Tokenizador"
Creacion de Sym (identificador de tokens)
Creacion del objeto Axiomas
Aplicar derivación por izquierda y derecha
¿Es el mismo en pila y entrada?
Creacion del objeto Derivador
Apicar axioma en pila
Consumir
Uso del axioma inicial S()
Uso de axiomas
Uso del metodo Derivador.analizar()
¿Token inesperado o es el fin?



Explicación del nuevo flujo

Explicación de objetos del diagrama

Se optó por la división de tareas para que el código sea más mantenible.

- El archivo `archivo.txt` es el que será sometido a una evaluación por nuestro analizador léxico.
- El archivo `Conf.txt` define los tokens y sirve como guía para la generación de nuestro objeto `Lexico` con JFlex.
- El objeto `pila` almacena las reglas de producción y será revisado en cada iteración del derivador.
- El objeto `Lexico` se encarga de identificar y proporcionarnos los métodos para tokenizar el contenido de `archivo.txt`.
- El objeto `SiguienteToken` es un método extraído del código del profesor **Miguel Ángel Sánchez Sánchez**, convertido en clase reutilizable.
- El objeto `Axiomas` define las reglas de producción. El programa está pensado para que se puedan agregar reglas nuevas sin modificar otras clases.
- El objeto `Derivador` se encarga de derivar de izquierda a derecha y luego de derecha a izquierda, aplicando las reglas y consumiendo tokens según corresponda.

Explicación con pseudocódigo del funcionamiento

```
Inicio del programa:
    Crear objeto Inicio
    Abrir archivo archivo.txt como buffer
    Crear objeto Lexico con el buffer
    Inicializar reglas de producción (Axiomas)
    Llamar a S() para comenzar el análisis

Reglas de producción():
Función S():
    Apilar ";" y luego "S"
    Llamar a siguienteToken() para obtener el primer token
    Llamar a Derivador()

    Registrar la regla A:
        si token == 'a' → regresar "a"
        si token == 'b' → regresar "" (lambda)
        otro → marcar error
    Registrar la regla B:
        si token == 'b' → regresar "bCd"
        otro → marcar error
    Registrar la regla C:
        si token == 'c' → regresar "c"
```

```
        si token == 'd' → regresar "" (lambda)
        otro → marcar error
    Registrar la regla S:
        siempre regresar "AB"

Derivador():
    Mientras pila no esté vacía y no haya error:
        Obtener cima de la pila

        Si cima es un no terminal registrado:
            pop de la pila
            aplicar producción con el token actual
            si hay resultado:
                apilar resultado en orden inverso

        Si cima es ";":
            si token es punto y coma:
                pop de la pila
                siguienteToken()
            otro:
                marcar error

        Si cima es terminal:
            si cima coincide con el token actual:
                pop de la pila
                siguienteToken()
            otro:
                marcar error

SiguienteToken():
    Llamar a analizador.yylex()
    Si null → lanzar excepción de fin de archivo
    Mostrar token leído
```

Datos tecnicos a tomar en cuenta

Es importante contar con el con las siguientes depedecias para que el proyecto se ejecute de forma conrrecta:

- Contar con `OpenJDK 64-Bit Server VM (build 17.0.14+7, mixed mode, sharing)` o equivalente
- El proyecto ya cuneta con el `jflex-full-1.7.0.jar` pero en caso de que se corrompa decargar uno equivalente
- Configurar UTF-8 para que no exista problema en la comilacion, ya que JFLex en ocaciones no es capas de detectar estos caracteres por si mimismo
- Se recomienda solo modificar el contenido de `archivo.txt`, si se decea usar otro `txt` hacer las modificaciones en `inicio.class`

📁 Estructura del proyecto

```
/
```

```
RutaProyecto
├── archivo.txt
├── Conf.txt
├── Documentacion
│   └── README.md
├── jar
│   └── jflex-full-1.7.0.jar
├── out
│   └── production
│       └── sintacticov2
│           └── fes
│               └── aragon
│                   ├── codigo
│                   │   ├── Axiomas.class
│                   │   ├── Derivador.class
│                   │   ├── ErrorSintactico.class
│                   │   ├── Lexico.class
│                   │   ├── SiguienteToken.class
│                   │   ├── Sym.class
│                   │   └── Tokens.class
│                   └── inicio
│                       └── Inicio.class
├── sintacticov2.iml
└── src
    └── fes
        └── aragon
            ├── codigo
            │   ├── Axiomas.java
            │   ├── Derivador.java
            │   ├── ErrorSintactico.java
            │   ├── Lexico.java
            │   ├── SiguienteToken.java
            │   ├── Sym.java
            │   └── Tokens.java
            └── inicio
                └── Inicio.java
```

Conclución:

Estuvo entretenido, me diveru por una tarde, lo que mas me costo es rehacer Lexico, no queria leer la doc de JFlex, pero cundo la leí me dí cuneta de por que no me salia.