Autómata Finito Determinista (AFD), Expresión Regular Asociada y Método del Árbol

Autómata Finito Determinista (AFD)

Descripción

El AFD diseñado para el análisis léxico reconoce el lenguaje utilizado en archivos .nlex. Este autómata identifica tokens clave, incluyendo palabras reservadas, operadores, identificadores, números y cadenas.

Diagrama del AFD

Estados: {S0, S1, S2, S3, S4, S5, S6, S7, S8}

Alfabeto: {letras, dígitos, "=", ":", ",", "{", "}", "[", "]", "/", "\"", otros}

Transiciones:

S0 -> S1 [letras]

S0 -> S2 [dígitos]

S0 -> S3 [=, :, ,, {, }, [,]]

S0 -> S4 [/]

S0 -> S5 [\"]

S1 -> S1 [letras, dígitos]

S2 -> S2 [dígitos]

S2 -> S3 [.]

S4 -> S6 [/]

S4 -> S7 [*]

S6 -> S6 [no salto de línea]

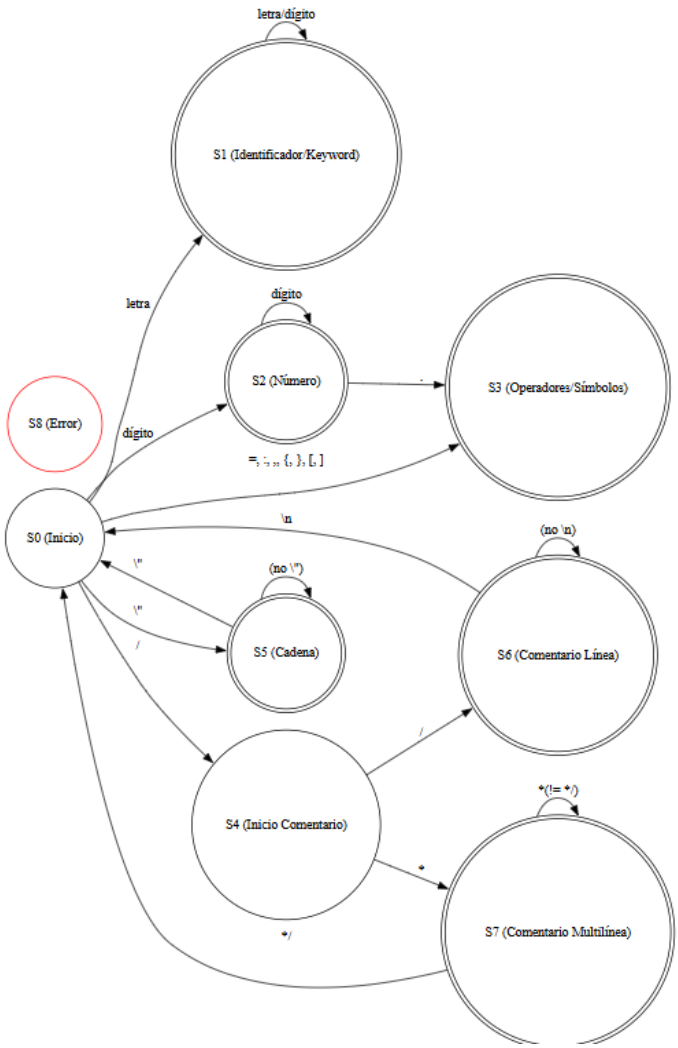
S6 -> S0 [salto de línea]

S7 -> S7 [* != */]

S7 -> S0 [*/]

S5 -> S5 [no \"]

S5 -> S0 [\"]



S3 -> S0 [otros]

Estados Finales

- **S1**: Identificador o palabra clave.
- **S2**: Número entero o decimal.
- **S3**: Operadores o símbolos especiales.
- **S6**: Comentario de una línea.
- **S7**: Comentario multilínea cerrado.
- **S5**: Cadena cerrada.

Expresión Regular Asociada

La expresión regular que describe el lenguaje es:

(?:[a-zA-Z_][a-zA-Z0-9_]*|\d+(?:\.\d+)?|\"(?:[^\"])*\"|[=:\{\}\[\],]|\/\/.*|\/*(?:.|\n)*?*\/)

Desglose:

1. **Palabras clave e identificadores:** [a-zA-Z_][a-zA-Z0-9_]*
2. **Números:** \d+(?:\.\d+)?
3. **Cadenas:** \"(?:[^\"])*\"
4. **Operadores y símbolos:** [=:\{\}\[\],]
5. **Comentarios de una línea:** \/\/.*
6. **Comentarios multilínea:** \/*(?:.|\n)*?*\/

Método del Árbol

Proceso Paso a Paso

1. Construcción del Árbol de Derivación

El método del Árbol de Derivación utiliza una gramática libre de contexto para validar y organizar los componentes del código en un formato jerárquico. Por ejemplo:

Gramática (Simplificada):

Programa → ConfiguracionesLex ConfiguracionesParser Operaciones

ConfiguracionesLex → "ConfiguracionesLex" = [Configuracion]

ConfiguracionesParser → "ConfiguracionesParser" = [Configuracion]

Configuracion → clave : valor (, Configuracion)?

Operaciones → "Operaciones" = [Operacion (, Operacion)*]

Operacion → { clave : valor (, clave : valor)* }

clave → IDENTIFICADOR

valor → NUMERO | CADENA | Operacion

2. Derivación del Ejemplo

Entrada:

Operaciones = [

{ nombre: "op1", operacion: "suma", valor1: 5, valor2: 10 },

{ operacion: "potencia", valor1: 2, valor2: 3 }

]

Árbol de Derivación:

Programa

├— ConfiguracionesLex

├— ConfiguracionesParser

└— Operaciones

  ├— Operación 1

  │ ├— Clave: nombre

  │ ├— Valor: "op1"

  │ ├— Clave: operacion

  │ ├— Valor: suma

  │ ├— Clave: valor1

  │ ├— Valor: 5

│ ├── Clave: valor2

│ └── Valor: 10

└── Operación 2

├── Clave: operacion

├── Valor: potencia

├── Clave: valor1

├── Valor: 2

├── Clave: valor2

└── Valor: 3

3. Validación con el AFD

Cada token derivado del árbol debe ser validado contra el AFD para garantizar que cumpla con las reglas léxicas.

4. Resultados y Errores

- Si se encuentran tokens inválidos durante la derivación, se registran como errores.
- La construcción del árbol permite identificar estructuras incompletas o malformadas.

Paso	Producción Aplicada	Derivación Parcial
1	`Programa → ConfiguracionesLex ConfiguracionesParser Operaciones`	`ConfigLex ConfigParser Operaciones`
2	`Operaciones → "Operaciones" = [ Operacion (, Operacion)* ]`	`"Operaciones" = [ Operacion (, Operacion)* ]`
3	`Operacion → { clave : valor (, clave : valor)* }`	`"Operaciones" = [ { clave: valor (, clave: valor)* } (, Operacion)* ]`
4	Descomposición de la primera operación	`"Operaciones" = [ { nombre: "op1", operacion: "suma", valor1: 5, valor2: 10 } (, Operacion)* ]`
5	Descomposición de la segunda operación	`"Operaciones" = [ { nombre: "op1", operacion: "suma", valor1: 5, valor2: 10 }, { operacion: "potencia", valor1: 2, valor2: 3 } ]`
6	Terminal alcanzado	Derivación completa: `"Operaciones" = [ { nombre: "op1", operacion: "suma", valor1: 5, valor2: 10 }, { operacion: "potencia", valor1: 2, valor2: 3 } ]`