Documentación de programa:Regex2DFA

Omar Alexis Palencia-1152270

Andres Felipe Beltran-1152262

Flujo del programa

Interpretación de la Expresión Regular

- **Entrada**: La expresión regular en notación de Thompson.
- **ProcesadorRegexThompson**: Analiza la expresión regular y la convierte a notación posfija, preparando el camino para la construcción del NFA.

Construcción del NFA

- **ProcesadorRegexThompson**: Utiliza el árbol de expresión posfija para construir un NFA. Este paso implica la creación de estados y transiciones basadas en las operaciones de Thompson (concatenación, alternativa, cierre de Kleene).
- **Resultado**: Se obtiene un objeto `NFA` que representa el autómata obtenido a partir de la expresión regular.

Eliminación de Transiciones Epsilon

- **EpsilonNFA_DFA**: Implementa el algoritmo de construcción por subconjuntos para convertir el NFA a DFA, eliminando transiciones ε. Este proceso es fundamental para simplificar el autómata y hacerlo determinista.
- **FuncionDeTransicion**: Durante este paso, se crean y gestionan funciones de transición que representan el comportamiento del NFA en términos de DFA. Cada función de transición encapsula un conjunto de estados y las transiciones entre ellos, facilitando la representación del DFA.

Representación del DFA

- **EpsilonNFA_DFA**: Una vez eliminadas las transiciones ε, se imprime el DFA resultante. Esto incluye la impresión de las funciones de transición

generadas y un mapa final que resume el DFA, mostrando claramente el estado inicial, los estados finales, el alfabeto, y las transiciones entre estados.

Clase: `MainAutomata`

Propósito

La clase `MainAutomata` actúa como el controlador principal de la aplicación, facilitando la interacción con el usuario y desencadenando el proceso de generación de un DFA a partir de una expresión regular (regex). Continuamente solicita al usuario entradas de regex hasta que el usuario decide salir del programa.

Atributos

- No tiene atributos específicos mencionados.

Métodos

- `main(String[] args)`: El método principal de la aplicación, que contiene la lógica para la interacción del usuario y el inicio del procesamiento de patrones regex.

Descripción Detallada del Método

- `public static void main(String[] args)`
  - **Descripción**: El método principal de la aplicación, que se ejecuta indefinidamente hasta que el usuario decida salir.
  - **Flujo de Proceso**:
    1. Inicializa un objeto `Scanner` (`myObj`) para leer la entrada del usuario desde la consola.
    2. Solicita al usuario que introduzca una expresión regular siguiendo la notación de Thompson. La solicitud también incluye instrucciones para salir del programa escribiendo "Salir".
    3. Lee la entrada del usuario (`regex`) y verifica si es igual a "Salir" (ignorando las diferencias de mayúsculas y minúsculas). Si es así, el método retorna, efectivamente terminando el programa.

4. Muestra la regex ingresada por el usuario en la consola para confirmación.
5. Crea una nueva instancia de la clase `NFA`, pasando la regex ingresada como argumento. Esto inicializa el NFA basado en la regex.
6. Llama al método `eliminarTransicionesVacias` de la clase `EpsilonNFA_DFA`, pasando el objeto `NFA` inicializado. Este paso elimina cualquier transición vacía del NFA, preparándolo para procesos adicionales.

Interacción con el Usuario

- El programa continúa solicitando expresiones regulares al usuario, procesando cada una creando un NFA y eliminando sus transiciones vacías. Este bucle continúa hasta que el usuario ingresa "Salir".

Condición de Salida

- El programa sale cuando el usuario ingresa "Salir", tal como lo determina la comparación de la entrada del usuario contra esta cadena, ignorando las diferencias de mayúsculas y minúsculas.

Clase `NFA`

Propósito

La clase `NFA` representa un Autómata Finito No Determinista (NFA) construido a partir de una expresión regular utilizando operaciones de Thompson. Esta clase gestiona los estados, el estado inicial, el estado final y las transiciones del NFA.

Atributos

- `estados`: Una lista de objetos `Estado` que representan los estados del NFA.
- `estadoInicial`: Un objeto `Estado` que representa el estado inicial del NFA.
- `estadoFinal`: Un objeto `Estado` que representa el estado final del NFA.
- `alfabeto`: Una lista de cadenas que representa el alfabeto del NFA.

Métodos

- `getEstadoInicial()`: Devuelve el estado inicial del NFA.

- `getEstadoFinal()`: Devuelve el estado final del NFA.
- `toString()`: Retorna una representación en cadena del NFA.
- `imprimirTabla()`: Imprime una tabla que muestra los estados y las transiciones del NFA.

Constructor

- `NFA(String thompsonRegex)`: Construye un NFA a partir de una expresión regular en notación de Thompson.
- `NFA()`: Constructor vacío.

Clase `ProcesadorRegexThompson`

Propósito

La clase `ProcesadorRegexThompson` procesa una expresión regular en notación de Thompson y genera un NFA correspondiente. Utiliza operaciones de Thompson para construir el NFA.

Atributos

- `regex`: La expresión regular en notación de Thompson.
- `alfabeto`: Una lista de caracteres que representa el alfabeto de la expresión regular.
- `op`: Una lista de operadores permitidos en la expresión regular.
- `generado`: El NFA resultante de procesar la expresión regular.

Métodos

- `ProcesadorRegexThompson(String regex)`: Constructor que inicializa el procesador con una expresión regular.
- `generarAutomata(LinkedList<TreeNode> colaDeOperaciones)`: Genera un NFA a partir de una lista de nodos de árbol que representan la expresión regular en notación posfija.
- `encontrarAlfabeto(char[] regex, int n)`: Encuentra el alfabeto de la expresión regular eliminando caracteres de operación y caracteres repetidos.

Clase `Transicion`

Propósito

La clase `Transicion` representa una transición entre estados en un autómata, especificando el símbolo que activa la transición y el estado destino a donde lleva dicha transición.

Atributos

- `simbolo`: Una cadena que representa el símbolo o carácter que activa la transición.
- `estadoDestino`: Un objeto `Estado` que representa el estado al cual se llega mediante la transición.

Constructores

- `Transicion(String simbolo, Estado estadoFinal)`: Constructor que inicializa una transición con un símbolo y un estado destino.
- `Transicion(String simbolo)`: Constructor que inicializa una transición con un símbolo y mantiene el estado destino actual.

Métodos

- `equals(Object obj)`: Sobreescribe el método `equals` para comparar dos instancias de `Transicion`.

Clase `Estado`

Propósito

La clase `Estado` representa un estado individual dentro de un autómata, incluyendo su nombre, si es un estado de aceptación, y las transiciones asociadas a dicho estado.

Atributos

- `nombre`: Una cadena que identifica al estado.
- `transiciones`: Un conjunto de objetos `Transicion` que representan las transiciones salientes desde este estado.

- `esAceptacion`: Un booleano que indica si el estado es un estado de aceptación.

Métodos

- `getNombre()`: Retorna el nombre del estado.
- `setEsAceptacion(boolean esAceptacion)`: Establece si el estado es un estado de aceptación.
- `esAceptacion()`: Indica si el estado es un estado de aceptación.
- `equals(Object obj)`: Sobreescribe el método `equals` para comparar dos instancias de `Estado`.
- `getEstadosAlcanzablesPorSimbolo(String simbolo)`: Retorna un conjunto de estados alcanzables desde este estado mediante transiciones con el símbolo dado.
- `añadirTransicion(String simbolo, Estado estadoFinal)`: Agrega una nueva transición al estado.
- `getAllFollowingEstados()`: Retorna todos los estados siguientes alcanzables desde este estado.
- `toString()`: Retorna una representación en cadena del estado, incluyendo sus transiciones.

Constructores

- `Estado()`: Constructor vacío.
- `Estado(String nombre)`: Constructor que inicializa un estado con un nombre específico.

Clase `EpsilonNFA_DFA`

Propósito

La clase `EpsilonNFA_DFA` contiene métodos estáticos para convertir un Autómata Finito No Determinista (NFA) que utiliza transiciones epsilon (ε-transiciones) en un Autómata Finito Determinista (DFA), eliminando así las transiciones ε. Este proceso es crucial para simplificar el análisis de patrones en expresiones regulares.

Método `eliminarTransicionesVacias`

Este método realiza la conversión de NFA a DFA, eliminando las transiciones ε y generando funciones de transición equivalentes que representan el comportamiento del NFA en términos de DFA.

Parámetros

- `nfa`: Un objeto `NFA` que representa el autómata a ser convertido.

Proceso
1. Se marca el estado final del NFA como un estado de aceptación.
2. Se elimina el símbolo ε del alfabeto del NFA.
3. Se inicializa una cola de funciones de transición (`FuncionDeTransicion`) y un conjunto de contenedores para almacenar las funciones de transición generadas.
4. Se ejecuta un bucle mientras la cola de funciones de transición no esté vacía:
    - Se extrae una función de transición de la cola.
    - Se llama al método `generarFuncionesTransicion` para expandir la función de transición actual basándose en cada símbolo del alfabeto.
5. Al finalizar el proceso, se imprimen las funciones de transición generadas y se muestra un mapa final que resume el DFA resultante.

Método `generarFuncionesTransicion`

Este método auxiliar se encarga de expandir una función de transición dada, generando nuevas funciones de transición para cada símbolo del alfabeto que no haya sido manejado aún. Esto permite representar el comportamiento del NFA en términos de DFA.

Parámetros

- `contenedor`: Un conjunto de funciones de transición existentes.
- `currentFt`: La función de transición actual que se está expandiendo.
- `cola`: Una cola de funciones de transición para manejar las nuevas funciones generadas.
- `alfabeto`: El alfabeto del NFA.

Proceso

1. Para cada símbolo en el alfabeto:
    - Se crea una nueva función de transición `ft` basada en la función de transición actual y el símbolo.
    - Se verifica si una función de transición equivalente ya existe en el contenedor. Si es así, se agrega la transición correspondiente a la función de transición actual sin crear una nueva función.
    - Si no hay una función equivalente, se agrega la nueva función de transición al contenedor y se actualiza la cola de funciones de transición.

Resultado

El resultado de aplicar estos métodos es un DFA equivalente al NFA original, pero sin utilizar transiciones ε, lo que facilita el análisis y la implementación de patrones de búsqueda y coincidencia de texto.

Clase `FuncionDeTransicion`

Propósito

La clase `FuncionDeTransicion` representa una función de transición en un autómata, específicamente diseñada para ser utilizada en la conversión de un Autómata Finito No Determinista (NFA) a un Autómata Finito Determinista (DFA). Esta clase encapsula un conjunto de estados y las transiciones entre ellos, permitiendo modelar el comportamiento de un DFA de manera eficiente.

Atributos

- `estadosFuncion`: Un conjunto de estados que forman parte de esta función de transición.
- `eClosure`: Un conjunto de estados alcanzables por transiciones epsilon (ε) desde cualquier estado en `estadosFuncion`. Esencial para determinar si la función de transición representa un estado de aceptación.
- `transiciones`: Un mapa que mapea símbolos a funciones de transición, representando las transiciones entre conjuntos de estados.
- `nombre`: Una cadena que identifica a la función de transición.
- `FuncionDeAceptacion`: Un booleano que indica si la función de transición representa un estado de aceptación en el DFA.

Métodos

- `isFuncionDeAceptacion()`: Verifica si la función de transición representa un estado de aceptación en el DFA. Lo hace comprobando si alguno de los estados en `eClosure` es un estado de aceptación.
- `FuncionDeTransicion(String nombre, HashSet<Estado> estadosFuncion)`: Constructor que inicializa una función de transición con un nombre y un conjunto de estados.
- `agregarTransicion(String simbolo, FuncionDeTransicion destino)`: Agrega una transición a la función de transición, mapeando un símbolo a otra función de transición.
- `SigmaDeSimbolo(String simbolo)`: Retorna un conjunto de estados alcanzables desde cualquier estado en `eClosure` mediante transiciones con el símbolo dado, representando el efecto de esa transición en el DFA.
- `toString()`: Retorna una representación en cadena de la función de transición, útil para depuración y visualización.
- `imprimirFuncion()`: Retorna una descripción detallada de la función de transición, incluyendo su nombre, estados, e-closure, y transiciones.
- `hashCode()` y `equals(Object obj)`: Sobreescribe estos métodos para permitir la comparación de instancias de `FuncionDeTransicion` basadas en sus estados y e-closure.

Funcionalidad

La clase `FuncionDeTransicion` juega un papel crucial en la conversión de NFA a DFA, permitiendo representar el comportamiento del NFA de manera que sea fácilmente traducible a un DFA. Las funciones de transición capturan las relaciones entre conjuntos de estados y símbolos, facilitando la construcción del DFA equivalente.