Lab5

https://github.com/915-Nichifor-Dragos/FLCD/tree/master/Lab4/lab-4

Finite Automaton (FA)

Overview

The Finite Automaton (FA) is a simple program designed to model and analyze a finite automaton. The FA can be initialized from a file, providing information about states, alphabet, output states, initial state, and transitions. It supports operations such as checking if a word is accepted and finding the accepted substring of a word.

Class Structure

1. FA Class

Properties:

_states: List of all possible states.

_alphabet: List of characters that can be used (alphabet).

_transitions: List of possible transitions between states.

_initialState: The initial state.

_outputStates: List of output states.

_filename: The name of the file from which the FA is initialized.

Constructor:

FA(string filename): Initializes the FA by reading information from a specified file.

Methods:

Initialize(): Initializes states, alphabet, output states, initial state, and transitions from the file.

PrintStates(): Prints the list of states.

PrintAlphabet(): Prints the list of characters in the alphabet.

PrintOutputStates(): Prints the list of output states.

PrintInitialState(): Prints the initial state.

PrintTransitions(): Prints the list of transitions.

CheckAccepted(string word): Checks if the inserted word is accepted by the FA.

GetNextAccepted(string word): Returns the substring of the word that matches the expression.

IsDFA(): checks if it is a DFA.

2. Transition Class

Properties

_from: From which state.

_to: To which state.

_label: Using what character (label).

Constructors

Transition(string from, string to, string label): Initializes a transition with the specified parameters.

Usage

Create an instance of the FA class by providing the filename of the file containing FA information.

Example:

```
fa = new FA("filename.txt");
```

Use the provided methods to interact with the FA.

```
fa.PrintStates();

fa.PrintAlphabet();

fa.PrintOutputStates();

fa.PrintInitialState();

fa.PrintTransitions();
```

Check if a word is accepted.

```
var result = fa.CheckAccepted("word");

Console.WriteLine(result);
```

Get the next accepted substring.

```
var substring = fa.GetNextAccepted("word");

Console.WriteLine(substring);
```

Example File Format (EBNF)

file = { states, alphabet, out_states, initial_state, transitions }

states: "states={" identifier_list "}"

alphabet: "alphabet={" character_list "}"

out_states: "out_states={" identifier_list "}"

initial_state: "initial_state=" identifier

transitions: "transitions={" transition_list "}"

identifier_list: identifier { "," identifier }

character_list: character { "," character }

transition_list: "(" identifier "," identifier "," character ")" { ";" "(" identifier "," identifier "," character ")" }

identifier = letter { letter | digit | "_" }

character = letter | digit

letter = "a" | "b" | ... | "z"

digit = "0" | "1" | ... | "9"

Example File

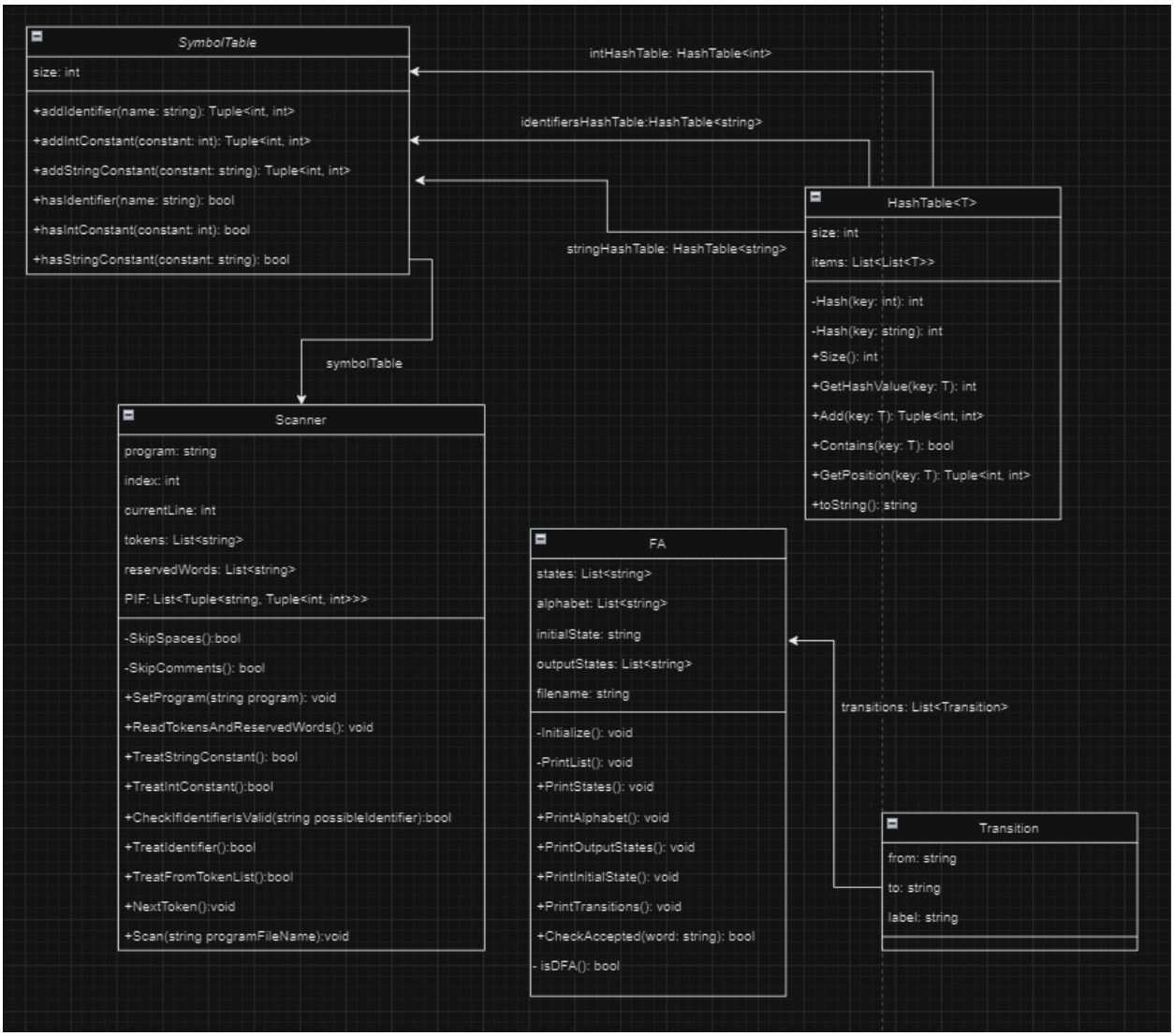
states={p, q, r}

alphabet={0, 1}

out_states={q, r}

initial_state=p

transitions={(p, q, 1); (q, q, 0); (q, r, 1); (r, r, 0)}


SymbolTable
size: int
+addIdentifier(name: string): Tuple<int, int>
+addIntConstant(constant: int): Tuple<int, int>
+addStringConstant(constant: string): Tuple<int, int>
+hasIdentifier(name: string): bool
+hasIntConstant(constant: int): bool
+hasStringConstant(constant: string): bool
intHashTable: HashTable<int>
identifiersHashTable:HashTable<string>
stringHashTable: HashTable<string>
HashTable<T>
size: int
items: List<List<T>>
-Hash(key: int): int
-Hash(key: string): int
+Size(): int
+GetHashValue(key: T): int
+Add(key: T): Tuple<int, int>
+Contains(key: T): bool
+GetPosition(key: T): Tuple<int, int>
+toString(): string
symbolTable
Scanner
program: string
index: int
currentLine: int
tokens: List<string>
reservedWords: List<string>
PIF: List<Tuple<string, Tuple<int, int>>>
-SkipSpaces():bool
-SkipComments(): bool
+SetProgram(string program): void
+ReadTokensAndReservedWords(): void
+TreatStringConstant(): bool
+TreatIntConstant():bool
+CheckIfIdentifierIsValid(string possibleIdentifier):bool
+TreatIdentifier():bool
+TreatFromTokenList():bool
+NextToken():void
+Scan(string programFileName):void
FA
states: List<string>
alphabet: List<string>
initialState: string
outputStates: List<string>
filename: string
-Initialize(): void
-PrintList(): void
+PrintStates(): void
+PrintAlphabet(): void
+PrintOutputStates(): void
+PrintInitialState(): void
+PrintTransitions(): void
+CheckAccepted(word: string): bool
- isDFA(): bool
transitions: List<Transition>
Transition
from: string
to: string
label: string