https://github.com/trutadan/University-Work/tree/main/Semester%205/Formal%20Languages%20and%20Compiler%20Design/lab04

Transition

The Transition class represents a transition in a finite automata, where a specific input symbol triggers a change from one state to another. This class is used to encapsulate the details of a transition, such as the source state, input symbol, and destination state.

Attributes:

- sourceState (private final String): Represents the source state of the transition.
- inputSymbol (private final char): Represents the input symbol triggering the transition.
- destinationState (private final String): Represents the state resulting from the transition.

Methods:

- getSourceState(): Returns the source state of the transition.
- getInputSymbol(): Returns the input symbol of the transition.
- getDestinationState(): Returns the destination state of the transition.
- Static Method (fromLine(String line) throws IllegalArgumentException): Parses a string representation of a transition in the format "δ(sourceState, inputSymbol) = destinationState" and creates a Transition object. Throws an IllegalArgumentException if the input line is not in the expected format.
- toString(): Returns a string representation of the transition in the form "δ(sourceState, inputSymbol) = destinationState".

FiniteAutomata

The FiniteAutomata class represents the structure of a finite automata and provides methods for its initialization, manipulation, and interaction. It encapsulates the states, alphabet, transitions, initial state, and final states of a finite automaton.

Attributes:

- states (private Set<String>): Represents the set of states in the finite automaton.
- alphabet (private Set<Character>): Represents the alphabet of the finite automaton.
- transitions (private List<Transition>): Represents the list of transitions in the finite automaton.
- initialState (private String): Represents the initial state of the finite automaton.
- finalStates (private Set<String>): Represents the set of final states of the finite automaton.
- scanner (private final Scanner): Used for user input via the console.

Methods:

- resetFiniteAutomata(): Resets the finite automata by initializing empty sets for states, alphabet, and final states, an empty list for transitions, and an empty string for the initial state.
- Setter Methods (setStates(...), setAlphabet(...), setTransitions(...), setInitialState(...), setFinalStates(...)): Allows external modification of the states, alphabet, transitions, initial state, and final states of the finite automata.
- initializeElements(String inputFile): Reads an input file to initialize the states, alphabet, initial state, final states, and transitions of the finite automata. Handles input file format and updates the finite automaton accordingly. Prints error messages for invalid input.
- Display Methods (displayStates(), displayAlphabet(), displayTransitions(), displayInitialState(), displayFinalStates()): Display various components of the finite automaton, such as states, alphabet, transitions, initial state, and final states.
- User Interaction Methods (displayMenu(), checkIfSequenceIsAccepted(), runMenu()): Displays a menu-driven interface for users to interact with the finite automaton. Allows users to view different components, check if a sequence is accepted, and exit the menu.
- isDFA(): Checks if the finite automaton is deterministic (DFA) by examining the transitions. Verifies that each state has a unique transition for every symbol in the alphabet.
- isAccepted(String sequence): Checks if a given sequence is accepted by the finite automaton by traversing through transitions and checking the final state.
- toString(): Returns a string representation of the finite automaton, including states, alphabet, initial state, final states, and transitions.

<u>TokenScanner</u>

The TokenScanner class is designed to recognize and classify tokens using finite automata. It contains two finite automata instances: one for identifying identifiers and another for recognizing integer constants.

Attributes:

- identifierFiniteAutomata (private FiniteAutomata): Finite automaton for recognizing identifiers.
- integerConstantFiniteAutomata (private FiniteAutomata): Finite automaton for recognizing integer constants.
- Initialization Methods (initializeIdentifierFiniteAutomata(), initializeIntegerConstantFiniteAutomata()): Initializes the finite automata for identifiers and integer constants with specific states, alphabets, initial states, and final states. Defines transitions based on the structure of identifiers and integer constants.
- Utility Methods (generateAlphabetRange(...), generateLowercaseLetters(), generateUppercaseLetters(), generateDigits(), generateNonzeroDigits(), generateTransitionsList(...), concatenateSets(...), concatenateTransitionsLists(...)): Helper methods for generating character sets, transitions, and concatenating sets or transition lists.

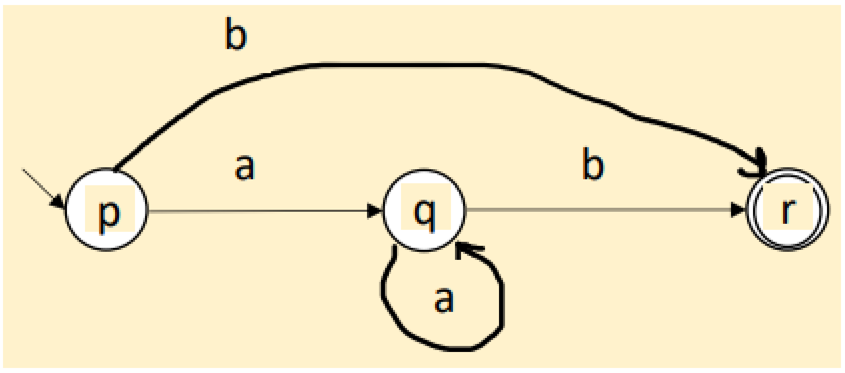
- Token Recognition Methods (isIdentifier(String token), isIntegerConstant(String token)): Uses the finite automata to determine whether a given token is an identifier or an integer constant. Calls the isAccepted method of the respective finite automaton.

BNF of the form in which the Finite Automata file should be written for initialization:

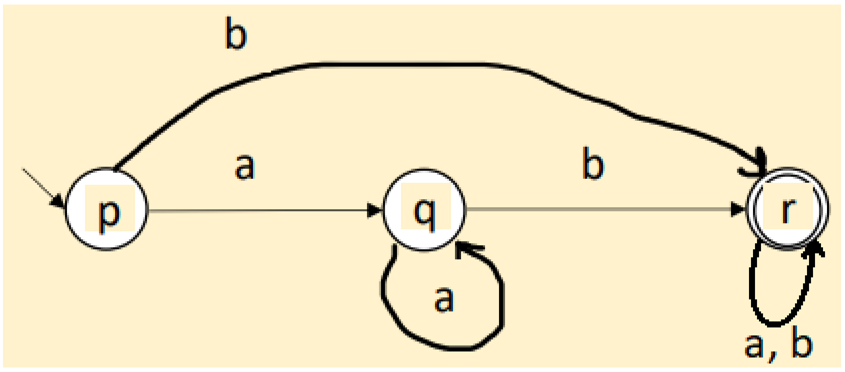
```
<FA_Config> ::= Q = {<State_List>}
                Σ = {<Symbol_List>}
                <State>
                F = {<State_List>}
                <Transition_List>

<State_List> ::= <State>|<State>, <State_List>
<Symbol_List> ::= <Symbol>|<Symbol>, <Symbol_List>
<Transition_List> ::= <Transition>|
                      <Transition>
                      <Transition_List>

<letter> ::= A|B|...|Z|a|b|...|z
<digit> ::= 0|1|...|9
<character> ::= <letter>|<digit>
<string_prefix> ::= <letter>
<string_suffix> ::= <character>|<string_suffix><character>
<string> ::= <string_prefix>|<string_prefix><string_suffix>

<State> ::= <string>
<Symbol> ::= <character>
<Transition> ::= δ(<State>, <Symbol>) = <State>
```

Example of a non-deterministic finite automata that can be used to check the isDFA() method(image extracted from the course)

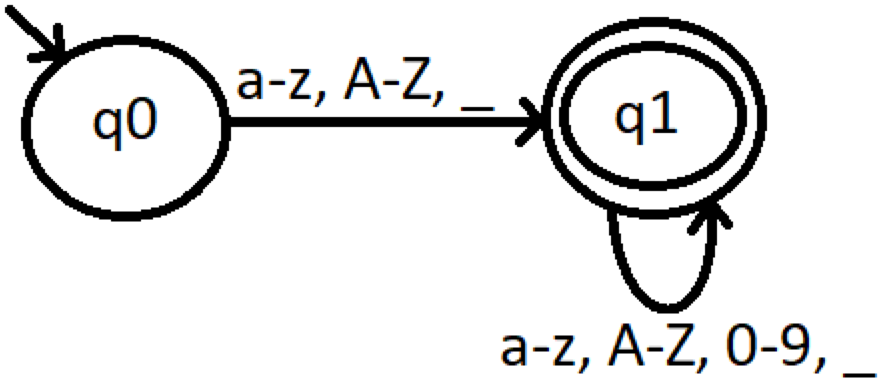
nfa.txt



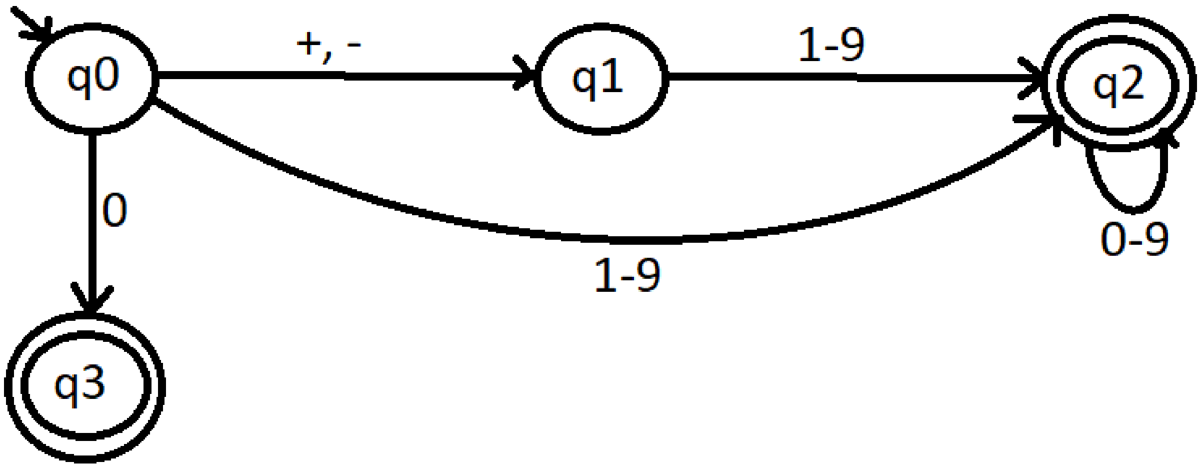
Example of a deterministic finite automata that can be used to check the isDFA() method(image extracted from the course and slightly modified)
dfa.txt:



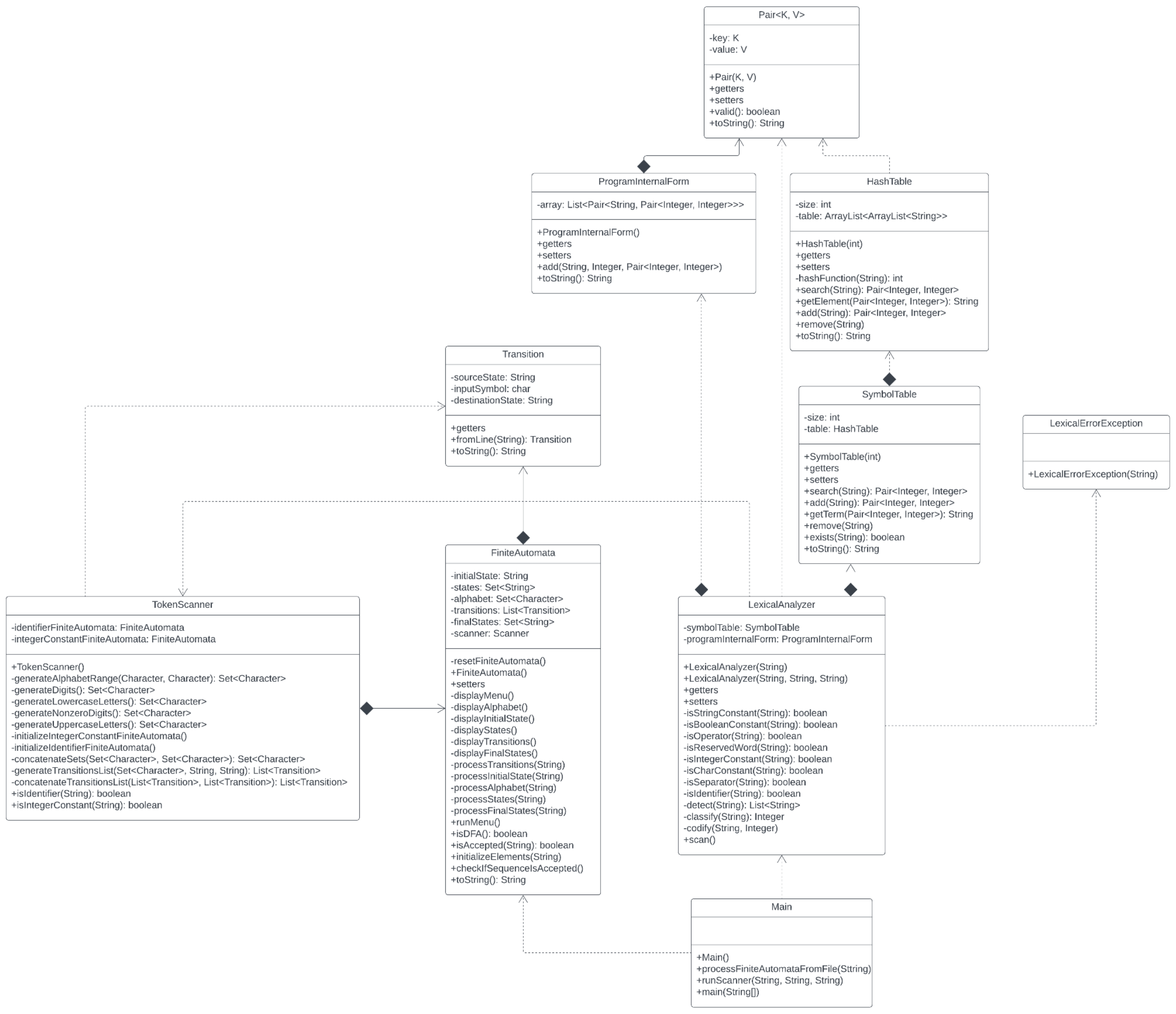
The representation of the finite automata designed and used for identifiers:



The representation of the finite automata designed and used for integer constants:



The class diagram of the whole project(created using LucidChart):


Pair<K, V>
-key: K
-value: V
+Pair(K, V)
+getters
+setters
+valid(): boolean
+toString(): String
ProgramInternalForm
-array: List<Pair<String, Pair<Integer, Integer>>>
+ProgramInternalForm()
+getters
+setters
+add(String, Integer, Pair<Integer, Integer>)
+toString(): String
HashTable
-size: int
-table: ArrayList<ArrayList<String>>
+HashTable(int)
+getters
+setters
-hashFunction(String): int
+search(String): Pair<Integer, Integer>
+getElement(Pair<Integer, Integer>): String
+add(String): Pair<Integer, Integer>
+remove(String)
+toString(): String
Transition
-sourceState: String
-inputSymbol: char
-destinationState: String
+getters
+fromLine(String): Transition
+toString(): String
SymbolTable
-size: int
-table: HashTable
+SymbolTable(int)
+getters
+setters
+search(String): Pair<Integer, Integer>
+add(String): Pair<Integer, Integer>
+getTerm(Pair<Integer, Integer>): String
+remove(String)
+exists(String): boolean
+toString(): String
LexicalErrorException
+LexicalErrorException(String)
FiniteAutomata
-initialState: String
-states: Set<String>
-alphabet: Set<Character>
-transitions: List<Transition>
-finalStates: Set<String>
-scanner: Scanner
-resetFiniteAutomata()
+FiniteAutomata()
+setters
-displayMenu()
-displayAlphabet()
-displayInitialState()
-displayStates()
-displayTransitions()
-displayFinalStates()
-processTransitions(String)
-processInitialState(String)
-processAlphabet(String)
-processStates(String)
-processFinalStates(String)
+runMenu()
+isDFA(): boolean
+isAccepted(String): boolean
+initializeElements(String)
+checkIfSequenceIsAccepted()
+toString(): String
TokenScanner
-identifierFiniteAutomata: FiniteAutomata
-integerConstantFiniteAutomata: FiniteAutomata
+TokenScanner()
-generateAlphabetRange(Character, Character): Set<Character>
-generateDigits(): Set<Character>
-generateLowercaseLetters(): Set<Character>
-generateNonzeroDigits(): Set<Character>
-generateUppercaseLetters(): Set<Character>
-initializeIntegerConstantFiniteAutomata()
-initializeIdentifierFiniteAutomata()
-concatenateSets(Set<Character>, Set<Character>): Set<Character>
-generateTransitionsList(Set<Character>, String, String): List<Transition>
-concatenateTransitionsList(List<Transition>, List<Transition>): List<Transition>
+isIdentifier(String): boolean
+isIntegerConstant(String): boolean
LexicalAnalyzer
-symbolTable: SymbolTable
-programInternalForm: ProgramInternalForm
+LexicalAnalyzer(String)
+LexicalAnalyzer(String, String, String)
+getters
+setters
-isStringConstant(String): boolean
-isBooleanConstant(String): boolean
-isOperator(String): boolean
-isReservedWord(String): boolean
-isIntegerConstant(String): boolean
-isCharConstant(String): boolean
-isSeparator(String): boolean
-isIdentifier(String): boolean
-detect(String): List<String>
-classify(String): Integer
-codify(String, Integer)
+scan()
Main
+Main()
+processFiniteAutomataFromFile(Sttring)
+runScanner(String, String, String)
+main(String[])