Link for github repo:

<https://github.com/AndreiFoidas/LFTC>

<https://github.com/AndreiFoidas/LFTC/tree/main/src/main/java>

# Finite Automaton

Is a class containing 5 fields: states (represented as a List), the alphabet (also a List), transitions (a Map which has Pair of strings as key and a List of Strings as value), initial state (a String) and the final states (a List), where each field is equivalent to the theoretical definition.

## Operations

* isDeterministic(): verifies if the finite automaton is deterministic by going through all the dictionary keys and checking if there is any List which has a length greater than 1, if none is found, the finite automaton is deterministic
* isValid(): verifies that the finite automaton is valid by checking to see if the initial state, final states and states in the transitions are in the set of states and checking that all the values of the transitions are in the alphabet
* acceptsSequence(sequence): checking if sequence is accepted by the finite automaton by going through each symbol of the sequence and checking that it can be reached by following the finite automaton’s transitions

## Integration in Scanner

Instead of regex matching the number constants and identifiers, the finite automaton checks that a given sequence is accepted by the finite automaton.

The respective finite automatons are kept in the files: identifierFA.in and integerFA.in and are read as they are needed. They follow the following EBNF definitions:

**Integer:**

integer ::= "0" | non\_zero\_digit {digit}

digit ::= "0" | non\_zero\_digit

non\_zero\_digit ::= "1" |...| "9"

**Identifier:**

identifier ::= (letter | \_) {(letter | digit | \_)}

letter ::= "A" | "B" | . ..| "Z" | "a" | "b" | ... | "z"

## Tests

states = a b c

alphabet = 0 1

initial state = a

final states = a b c

transitions: (a, 1) -> a

(a, 0) -> b

(b, 1) -> c

(c, 0) -> c





