Finite Automaton

Student: Alexandrescu Andrei-Robert, 931/1

Current lab: Repository Link

Integration with previous lab: Repository Link

1. File Structure

a) Mathematic (Natural Language) description

s_1 s_2 ... s_n (states)
a_1 a_2 ... a_m (alphabet)
noTran (number of transitions)
p_1 b_1 q_1 (delta(p_1, b_1) = q_1)
...
p_{noTran} b_{noTran} q_{noTran}
q_0 (initial state)
f_1 f_2 ... f_o (final states)

b) EBNF:

identifier ::= letter | letter {letter | digit}

letter ::= "A" | "B" | . ..| "Z"

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

non_zero_digit ::= “1” | ... | “9”

constno ::= [(“+” | “-“)] non_zero_number | zero

zero ::= 0

non_zero_number::= non_zero_digit{digit}

Previously defined

states ::= identifier | identifier {identifier}

alphabet ::= { constno }

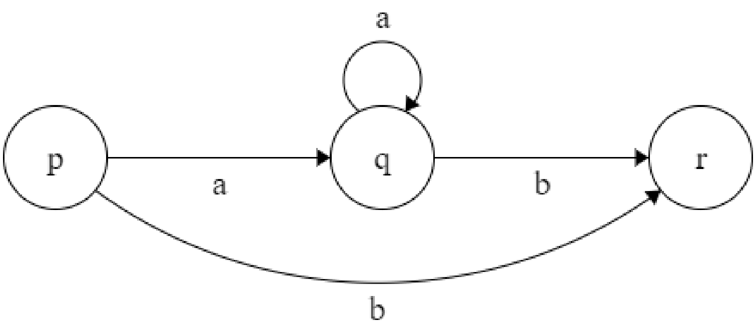
transitions ::= {identifier constno identifier}

initialState ::= identifier

finalStates ::= identifier {identifier}

Finite Automaton

c) Example:
p q r
a b
4
p a q
q a q
q b r
p b r
p
r



2. Implementation details

The class for treating the finite automaton is called *Parser*. All member variables and functions can be consulted at the bottom of the document in the final class diagram.

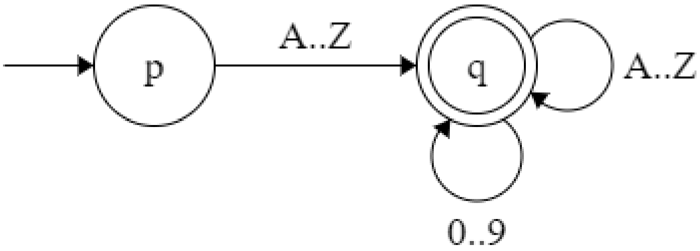
Method *readFA* is used to read the data from the *FA.in* file and store it accordingly in the RAM memory. Some error cases are treated such as:

- one of the transition terms (state 1, transition term, state 2) does not belong to the declared states / alphabet respectively
- the initial state does not belong to the declared states
- one of the final states does not belong to the declared states
- duplicate transitions (not an error, but treated nevertheless)

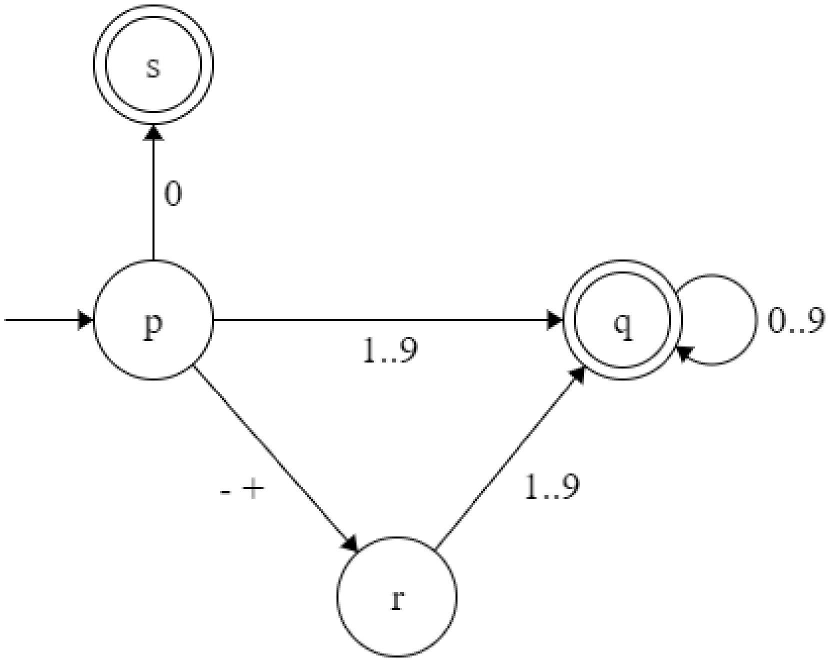
Method *verifySequence* checks whether a given sequence is accepted by the FA. This is done by simply using a for loop to cycle through the characters of the sequence and using a *currentState* variable to keep track of the current state. The method *move* is used to transition between states using the current symbol from the alphabet. In case the sequence could not be consumed entirely, an error occurs.

3. Integration with labs 1-3

Two finite automations were created for identifying identifiers and constants. The first one, namely *parserIdentifiers*, uses the *FA_identifiers.in* definition of a finite automation. The corresponding drawing can be seen below:

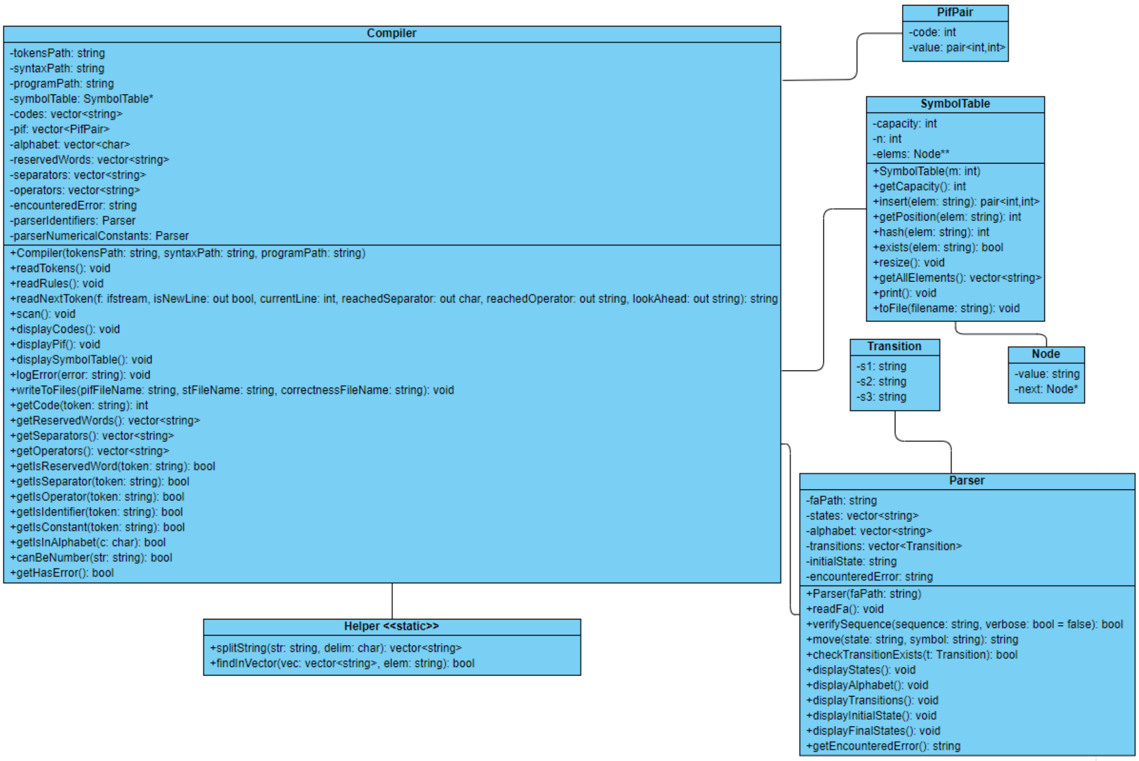


The second finite automaton, namely *parserNumericalConstants*, is read from the *FA_constants.in* file corresponding to the following finite automaton:



As for modifications performed to integrate the Finite Automata for identifiers and numerical constants, the implementations of methods *getIsIdentifier* and *canBeNumber* have been changed. Instead of performing manual "rudimentary" checks, the verifySequence method is used on the given token to check if it meets the requirements.

Updated class diagram:


Compiler
-tokensPath: string
-syntaxPath: string
-programPath: string
-symbolTable: SymbolTable*
-codes: vector<string>
-pif: vector<PifPair>
-alphabet: vector<char>
-reservedWords: vector<string>
-separators: vector<string>
-operators: vector<string>
-encounteredError: string
-parserIdentifiers: Parser
-parserNumericalConstants: Parser
+Compiler(tokensPath: string, syntaxPath: string, programPath: string)
+readTokens(): void
+readRules(): void
+readNextToken(f: ifstream, isNewLine: out bool, currentLine: int, reachedSeparator: out char, reachedOperator: out string, lookAhead: out string): string
+scan(): void
+displayCodes(): void
+displayPif(): void
+displaySymbolTable(): void
+logError(error: string): void
+writeToFiles(pifFileName: string, stFileName: string, correctnessFileName: string): void
+getCode(token: string): int
+getReservedWords(): vector<string>
+getSeparators(): vector<string>
+getOperators(): vector<string>
+getIsReservedWord(token: string): bool
+getIsSeparator(token: string): bool
+getIsOperator(token: string): bool
+getIsIdentifier(token: string): bool
+getIsConstant(token: string): bool
+getIsInAlphabet(c: char): bool
+canBeNumber(str: string): bool
+getHasError(): bool
Helper <<static>>
+splitString(str: string, delim: char): vector<string>
+findInVector(vec: vector<string>, elem: string): bool
PifPair
-code: int
-value: pair<int,int>
SymbolTable
-capacity: int
-n: int
-elems: Node**
+SymbolTable(m: int)
+getCapacity(): int
+insert(elem: string): pair<int,int>
+getPosition(elem: string): int
+hash(elem: string): int
+exists(elem: string): bool
+resize(): void
+getAllElements(): vector<string>
+print(): void
+toFile(filename: string): void
Transition
-s1: string
-s2: string
-s3: string
Node
-value: string
-next: Node*
Parser
-faPath: string
-states: vector<string>
-alphabet: vector<string>
-transitions: vector<Transition>
-initialState: string
-encounteredError: string
+Parser(faPath: string)
+readFa(): void
+verifySequence(sequence: string, verbose: bool = false): bool
+move(state: string, symbol: string): string
+checkTransitionExists(t: Transition): bool
+displayStates(): void
+displayAlphabet(): void
+displayTransitions(): void
+displayInitialState(): void
+displayFinalStates(): void
+getEncounteredError(): string