https://github.com/AliceHincu/Formal-Languages-and-Compiler-Design

Finite Automaton

The **Finite Automaton** is structured as a class with 5 main fields:

- Q – the states are kept in a list of strings
- E (Σ) - the alphabet is kept in a list of strings
- q0 – the initial state kept as a string
- F – the final states are kept in a list of string
- S - the transitions are kept in a Map:
    - The source state is mapped to another Map, that has as key an input (=element from the alphabet) and as value a list with destination states.

Methods:

- isDFA() - checks that the FA is a DFA
    - In DFA, there is only one path for specific input from the current state to the next state.
    - Checking that the FA is a DFA is done by going through all the lists, and looking if there's any list with a length greater than 1
    - ```
      public boolean isDFA() {
          for (String src : S.keySet()) {
              Map<String, List<String>> inputs = S.get(src);
              for (String input : inputs.keySet())
                  if (inputs.get(input).size() > 1) return false;
          }
          return true;
      }
      ```

- isAccepted(String sequence) - checks that the given sequence is a valid one
    - Goes through each symbol from the sequence and checks that it can be reached by the given FA's transitions.
    - The obtained state needs to be in the final states list.
    - ```
      public boolean isAccepted(String sequence) {

              if (!this.isDFA()) return false;

              String currentState = this.q0;
              for (int i = 0; i < sequence.length(); i++) {
                  String input = sequence.charAt(i) + "";
                  if (S.containsKey(currentState) &&
              S.get(currentState).containsKey(input))
                      currentState = S.get(currentState).get(input).get(0); // get
              first transition
                  else
                      return false;
              }
              return F.contains(currentState);

      }
      ```

BNF or EBNF format the form in which the FA.in file should be written

The FA files from which the FA reads the nodes and the rules are in the following format:

- Line 1: the states (separated by space)
- Line 2: the alphabet with every character (separated by space)
- Line 3: starting node
- Line 4: final states(separated by space)
- Line 5: mention of the transition function (usually “S =”)
- Line 6+: the transition functions. It is of the format ‘(A, 0) -> B’ where A is the state where the transition begins, 0 is an input from the alphabet, and B is the state where the transition ends

EBNF File format:

- letter ::= “A” | “B” | “C” | … | ”Z” | “a” | “b” | … | “z”
- digit ::= “0” | “1” | … | “9”
- symbol ::= “+” | “-” | “_” | lettter | digit
- alphabet ::= symbol
- state ::= letter {letter|digit}
- states ::= state {state}
- initialState ::= state
- finalStates ::= states
- transition ::= “(“ state “,” alphabet “)” “->” state

Integration with Scanner lab

The regex matching for constants and identifiers from the last lab is replaced with checking that the FA accepts a given sequence, namely one which represents an identifier/a constant, like 12, abc, -100 a.s.o.

The FAs are kept in the ‘fa-identifier.in’ file for **identifiers** and ‘fa-int.in’ & ‘fa-string.in’ files for **constants**, and read on start up.

Example of FA here:

https://github.com/AliceHincu/Formal-Languages-and-Compiler-Design/blob/main/code/src/files/fa1.in

EBNF for Identifiers:

- IDENTIFIER ::= letter {letter | digit | underscore}
- letter ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"
- digit ::= "0" | non_zero_digit
- non_zero_digit ::= "1" | ... | "9"

- underscore ::= "_"

EBNF for Constants:

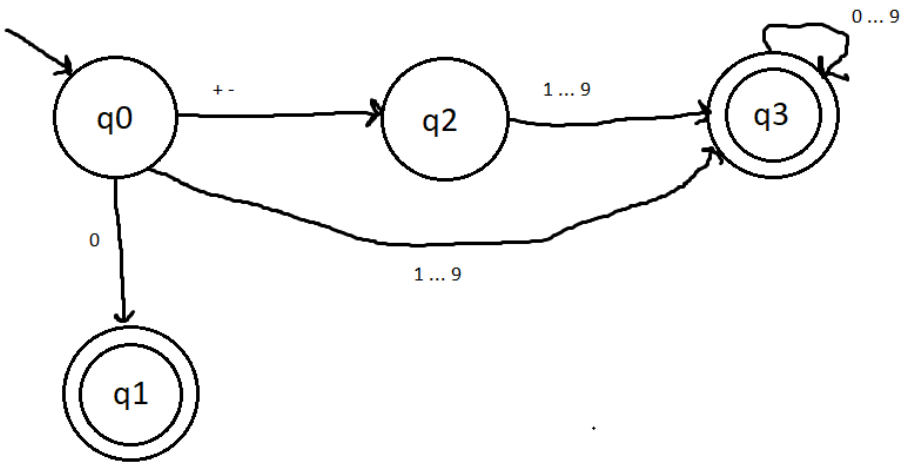
- CONSTANT = integer | character | string
- integer = "0" | ["+" | "-"] non_zero_digit{digit}
- character = ‘letter’ | ‘digit’ | ‘underscore’
- string = "{character}"

These are also specified in Lexic.txt:

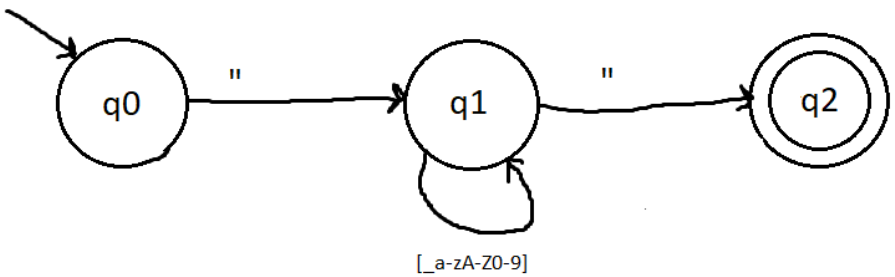
- https://github.com/AliceHincu/Formal-Languages-and-Compiler-Design/blob/main/Lexic.txt

FA for constants is split in 2: for integers and for strings

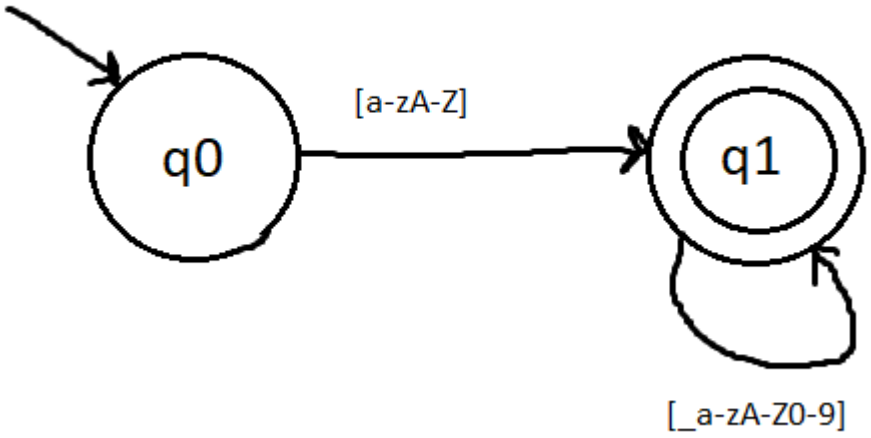
- FA for integers:



- FA for strings:



FA for identifiers:


q0
[a-zA-Z]
q1
[_a-zA-Z0-9]