Finite automata

We defined a finite automaton according to the mathematical definition:

a set of states

a set of inputs (the alphabet)

an initial state

a set of final states

a state of transitions – pairs of (State, Input) and List[State], meaning all the states that can be reached from a state through an input. If the length of the list is greater than one, the FA is nondeterministic.

```
std::vector<State> states;
std::vector<Input> alphabet;
State initialState;
std::vector<State> finalStates;
std::map<Transition, std::vector<State>> transitions;
```

We also created the following aliases in order for our definitions to be more intuitive:

```
using State = std::string;
using Input = std::string;
using Transition = std::pair<State, Input>;
```

An input file for a FA is of the form:

numStates numInputs numFinalStates numTransitionPairs

state[1] state[2] … state[numStates]

input[1] input[2] … input[numInputs]

initialState

finalState[1] finalState[2] … finalState[numFinalStates]

transition[1] transition[2] … transition[numTransitionPairs]

where transition is defined as

numTransitions

startState endState input[1] input[2] … input[numTransitions]

Or in EBNF (for the purpose of identifiers and integers):

digit ::= 1 | … | 0

integer ::= 0 | digit integer

character ::= a | … | z | A | … | Z | 0 | … | 9 | _ | '

state ::= character | character state

input :: character | character input

state_sequence ::= state | state state_sequence

input_sequence ::= input | input input_sequence

transition ::= integer state state input_sequence

transition_sequence ::= transition | transition transition_sequence

definition ::= integer integer integer integer state_sequence input_sequence state state_sequence transition_sequence

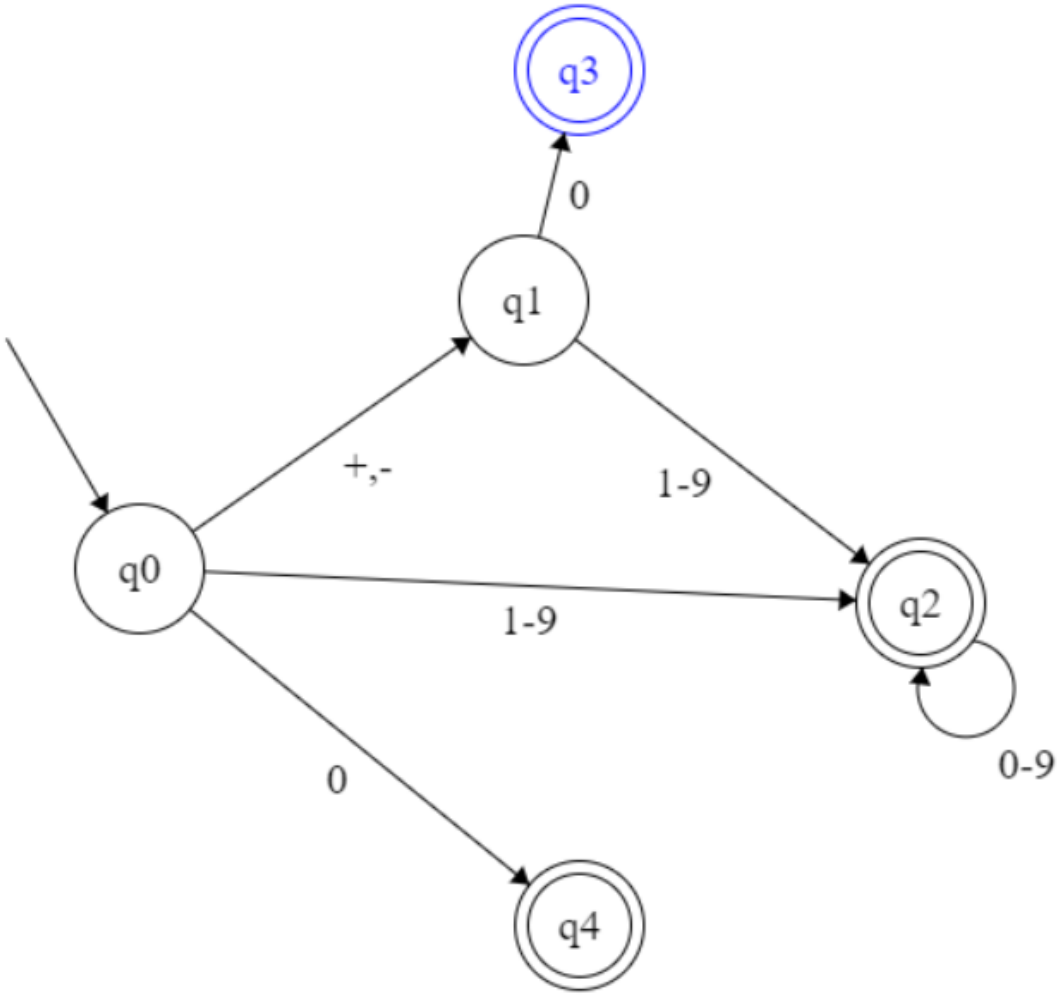
Checking the determinism of an AF is trivial – if the length of output states for any transition is greater than 1, it means that the FA is nondeterministic:

```
bool isDeterministic() const
{
    for (auto& pair : transitions)
    {
        if (pair.second.size() > 1)
        {
            return false;
        }
    }
    return true;
}
```

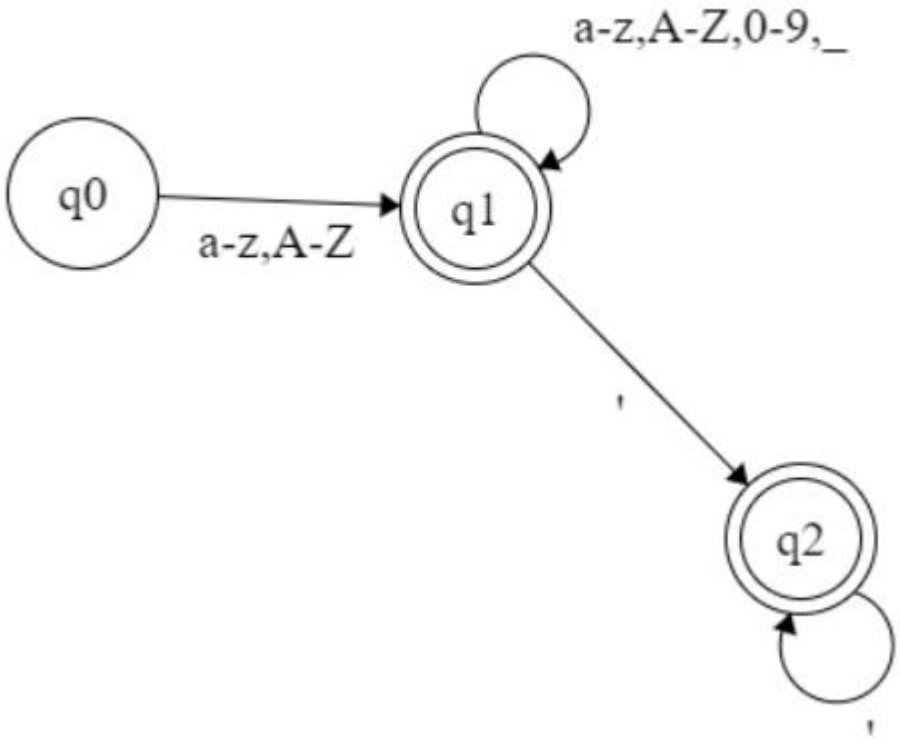
Checking if a sequence is accepted by the FA is also simple. First we check whether the input is deterministic. If it is, we set the current state to the initial state, iterate through all the characters of the string and check whether there is a transition from the current state through the current character to another state. If there is, we set the current state to that state and move on until there's no more characters left.

```cpp
bool accepted(const std::string& input)
{
    if (!isDeterministic())
    {
        return false;
    }
    State currentState = initialState;
    for (char c : input)
    {
        if (transitions.find(Transition(currentState, std::string(1, c))) != transitions.end())
        {
            currentState = transitions[Transition(currentState, std::string(1, c))][0];
        }
        else
        {
            return false;
        }
    }
    return true;
}
```

FA for integers



FA for identifiers



Class diagram