Finite Automata - documentation

Theory

A **finite automaton** (FA) is a 5-tuple M = (Q,Σ,δ,q0,F), where:

- Q - finite set of states (|Q|<∞)
- Σ - finite alphabet (|Σ|<∞)
- δ – transition function : δ:Q×Σ→P(Q)
- q0 – initial state q0 ∈ Q
- F⊆Q – set of final states

A word (or sequence of symbols from the alphabet) is said to be accepted by the FA if a final state can be obtained by applying transition operations, beginning with the final state, using the symbols of the sequence (sequentially).

Implementation

The FA was implemented as a class, containing a field for each element of the tuple (from the definition):

- states: set of possible states (set of strings)
- alphabet: set of the alphabet (set of strings)
- transitions: Python dictionary. The keys are tuples of the form <state, alphabet_symbol> and the values are lists of states, representing transitions from <state> to other possible states using <alphabet_symbol>
- initial_state: the initial state (string)
- final_states: set of final states (set of strings)

The FA can be read from a file (file format is described below) and can verify a sequence (if the FA is a Deterministic Finite Automaton). The verifying algorithm works as follows:

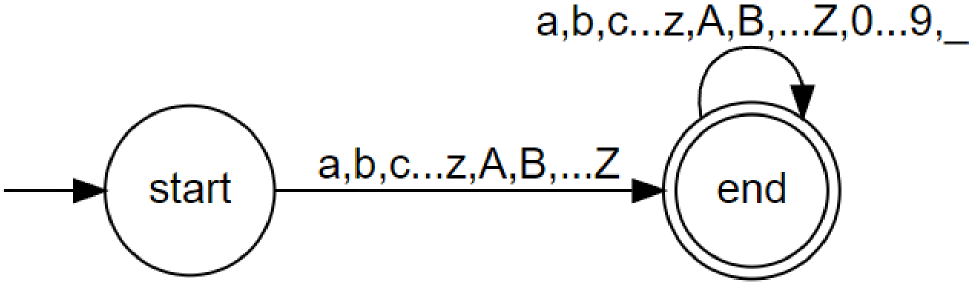
- Check if the FA is a DFA (if false, stop the process)
- Set the current_state as the initial_state
- For each symbol in the sequence, check if there is a transition from the current_state to another state, using the respective symbol. Repeat this operation until we reach the end of the sequence (or we cannot find a transition. In this case, the FA does not accept the sequence)
- If the current_state is a final state, then the FA accepts the sequence

The data file

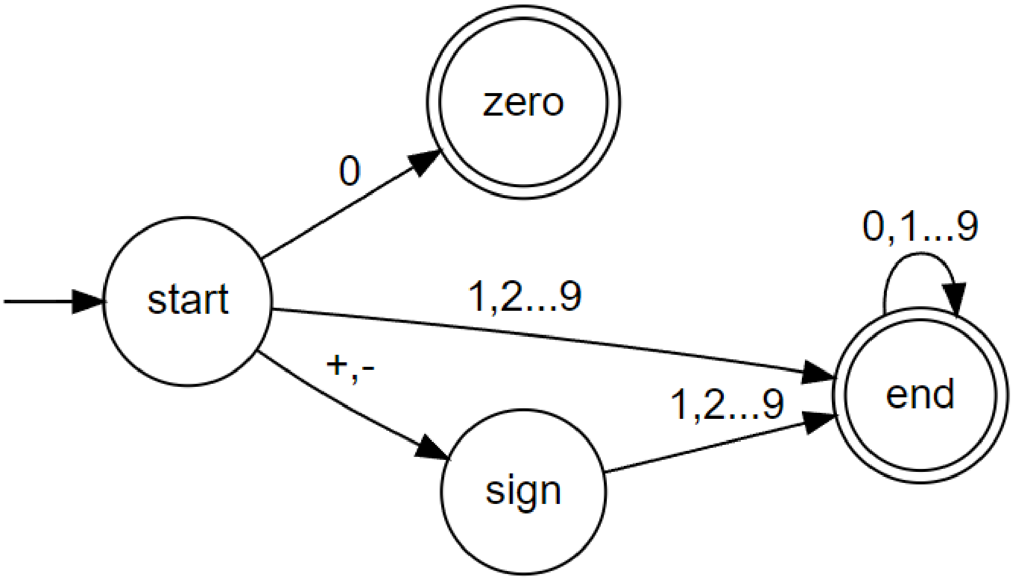
The EBNF for the file format is:
fa_file = states alphabet initial_state final_state transitions.
states = sequence{(sequence)}.
alphabet = sequence{(sequence)}.
initial_state = sequence.
final_state = sequence{(sequence)}.
transitions = sequence sequence sequence.

symbol = "0" | "1" |...| "9" | "A" | "B" | ... | "Z" | "a" | "b" | ... | "z".
sequence = symbol{symbol}.

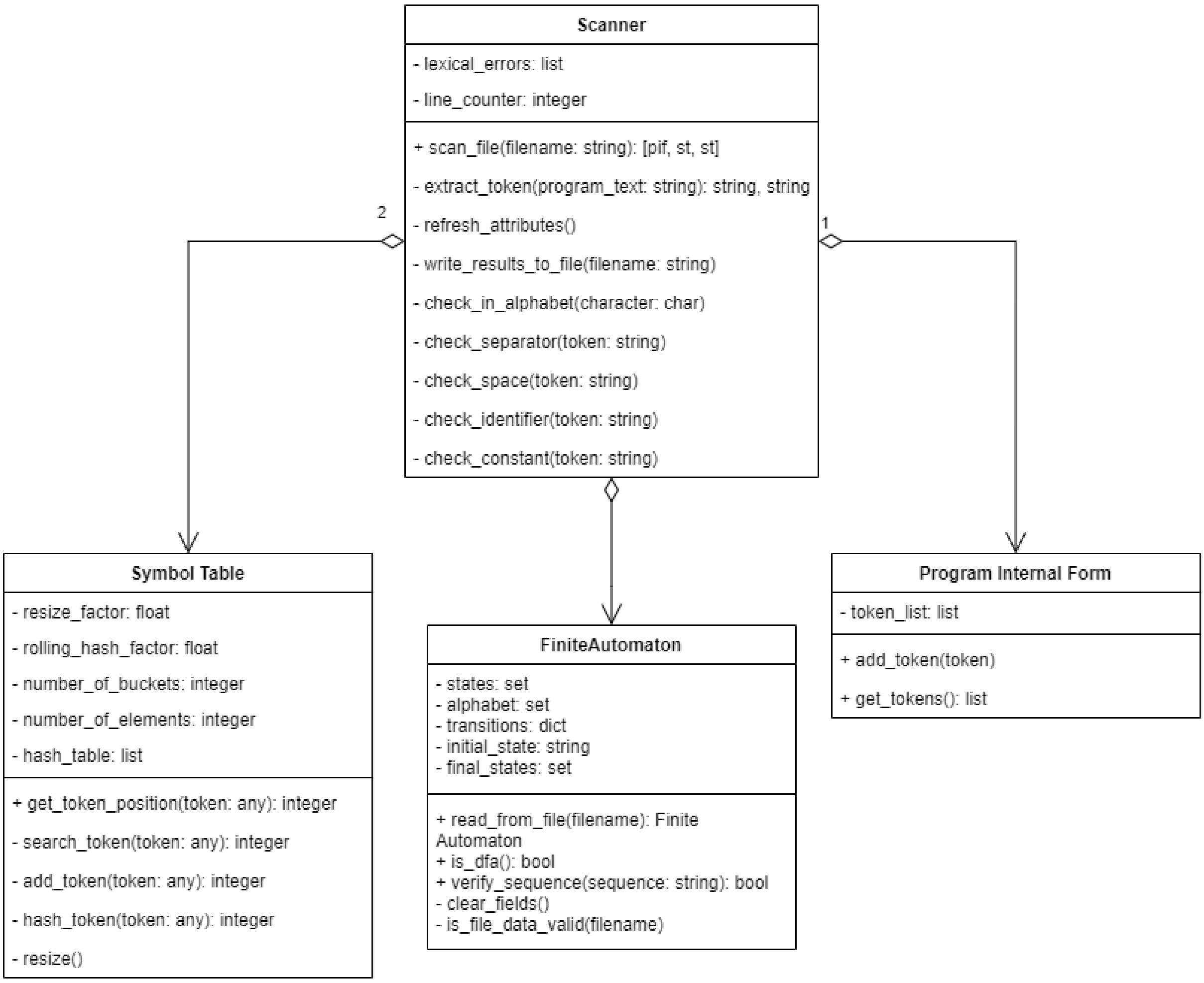
Finite Automaton for identifiers:



Finite Automaton for integer constants:



Class diagram:


Scanner
- lexical_errors: list
- line_counter: integer
+ scan_file(filename: string): [pif, st, st]
- extract_token(program_text: string): string, string
- refresh_attributes()
- write_results_to_file(filename: string)
- check_in_alphabet(character: char)
- check_separator(token: string)
- check_space(token: string)
- check_identifier(token: string)
- check_constant(token: string)
2
1
Symbol Table
- resize_factor: float
- rolling_hash_factor: float
- number_of_buckets: integer
- number_of_elements: integer
- hash_table: list
+ get_token_position(token: any): integer
- search_token(token: any): integer
- add_token(token: any): integer
- hash_token(token: any): integer
- resize()
FiniteAutomaton
- states: set
- alphabet: set
- transitions: dict
- initial_state: string
- final_states: set
+ read_from_file(filename): Finite Automaton
+ is_dfa(): bool
+ verify_sequence(sequence: string): bool
- clear_fields()
- is_file_data_valid(filename)
Program Internal Form
- token_list: list
+ add_token(token)
+ get_tokens(): list

Github link: https://github.com/livcristi/FLCD/tree/main/Lab%204