git link: https://github.com/alexandra-murariu/FLCD

Documentation

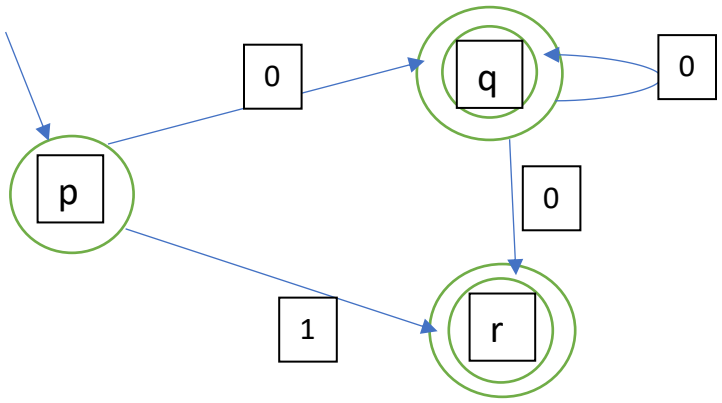
The FiniteAutomata class object gets a filename on creation, from which it receives the FA's definition.

The class has the following methods:

- read: it opens the file given through the constructor and splits the lines by spaces. The transition lines are first being split by arrows, which separate the pair of initial state and symbol with the result of the transition. Then, after saving all information regarding the states, alphabet, initial state, final states and transitions, some validations are performed. First, we check if the final states are part of the states' list. Then, we check if the initial state is also part of the valid states list. After that we iterate through the transition dictionary keys (represented by the pair starting state-symbol) and see if all states are part of the valid states list, if all symbols are from the alphabet and if all values of the dictionary (represented by the destination state of the transition) are part of the valid states list.
- is_dfa: in this method we check to see if in the dictionary of transitions all values are represented by a single state (the values in the dictionary represent the destination state/states of the transition described by the mathing key in the dictionary, which contains the starting state and the symbol for that transition). Basically, it checks to see if in the rhs of each transition there can be only one destination state.
- is_seq_accepted: this method first checks if the FA is a DFA, if it is not it returns False. Then, it iterates through the sequence symbols and checks to see if it is a valid path going through the transitions dictionary. Then, it checks if the final state is part of the final states' list.

Input file NFA

```
p q r states
0 1 symbols
p starting state
p r final states
(p,0) -> q transitions
(p,1) -> r
(q,0) -> q
(q,0) -> r
```

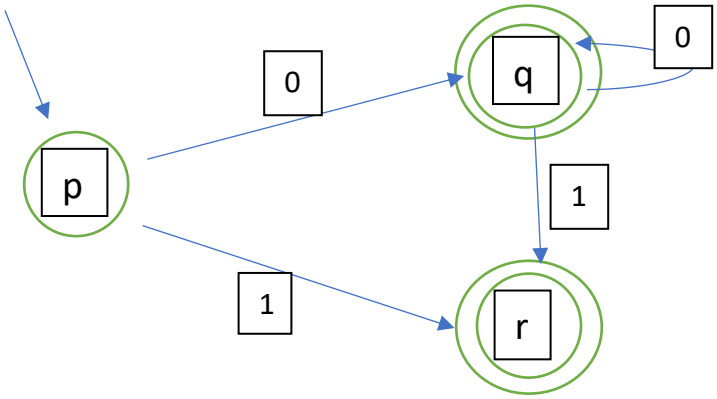


We can see that the sequence generated by this NFA (in EBNF) is:

NFA ::= (“0”{“0”}) | “1”

Input file DFA

```
p q r states
0 1 symbols
p starting state
q r final states
(p,0) -> q transitions
(p,1) -> r
(q,0) -> q
(q,1) -> r
```



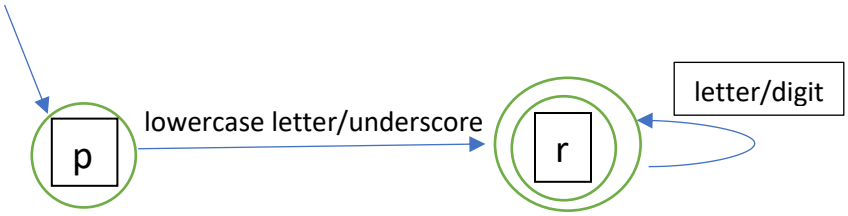
We can see that the sequence generated by this DFA (in EBNF) is:

DFA ::= (“0” {“0”} [“1”]) | “1”

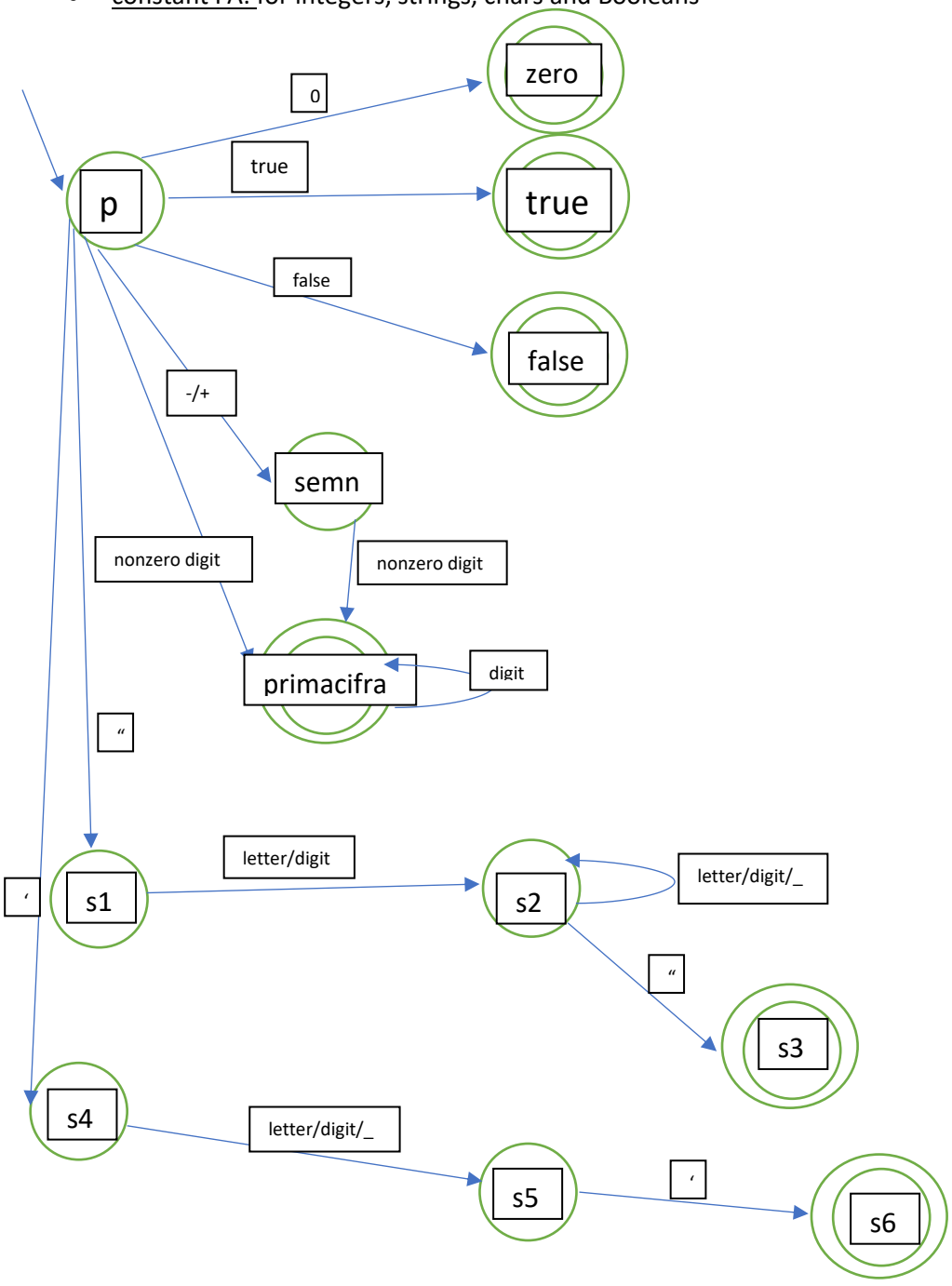
Scanner implementation of FA

The Scanner contains 2 FAs: one for identifiers and the other one for constants.

- identifier FA: This identifier is defined only with 2 states.



- constant FA: for integers, strings, chars and Booleans



These are used inside the Scanner in the following way:

```
elif self.identifier_fa.is_seq_accepted(elem):
    self._st.add(elem)
    self._pif.append(('identifier', self._st.find(elem)[0].key))

elif self.constant_fa.is_seq_accepted(elem):
    self._st.add(elem)
    self._pif.append(('constant', self._st.find(elem)[0].key))
```

Class diagram