Ghiorghita Carina-Ioana

Group 933

Lab 4 Documentation

Finite Automata

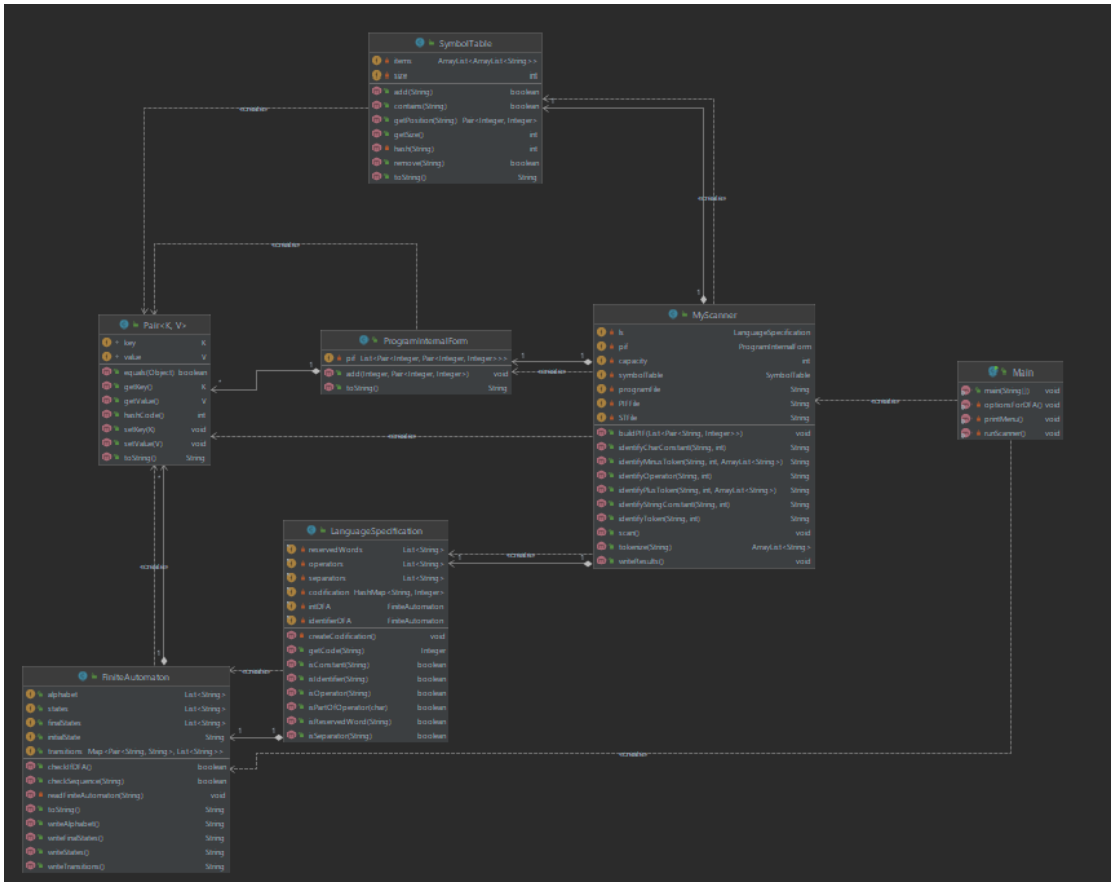
Link to GitHub repository:
https://github.com/carinaghiorghita/UBB_Sem5_FLCD/tree/main/Lab4/src/ubb/flcd

Requirement: Write a program that reads a FA from a file; display the contents of the FA and, for a DFA, verify if a sequence is accepted by the DFA. Integrate the FA in the Scanner program.

- Input: the file FA.in contains the elements of the FA; for the Scanner integration, intDFA.in contains the elements for a FA identifying the integer constants, while identifierDFA.in contains the elements for a FA corresponding to the identifiers of the program. The program is read from file P1.txt
- Output: the elements of the FA, as well as the check that a sequence is accepted by the FA are printed in the console; for the Scanner integration, the PIF and Symbol Table will be printed in files PIF1.txt and ST1.txt

Analysis and Design:

Class Diagram (for the Scanner integration):



Finite Automaton: It is structured as a class, having fields for the initial state, final states, alphabet, all states, and transitions. The initial state is represented as a single String. The states, alphabet and final states are each stored in a different set of Strings. The transitions are kept in a map, having as keys Pairs of two String values (the source state and the value through which it accesses the destination state), each mapped to a set of String values, representing the destination states. In the case of a DFA, such a set will contain only one value. The FA will be read from a file; duplicated transitions, as well as transitions containing invalid states or characters, will be ignored.

- Checking that the FA is a DFA: using the sets of values provided by the transitions map, we check that each key has as value a set of length not longer than 1, returning true or false accordingly.
- Checking that a sequence is accepted by the DFA: starting from the initial state, we go through each character of the given sequence and we check that the Pair formed by the current state and the value given by the current character in the sequence is mapped to a set containing exactly one value; this value will be the new current state in the next iteration. If the Pair is not mapped to any set, then the sequence is not accepted by the DFA and the algorithm stops, returning false. When we reach the end of the sequence, if the last state belongs to the set of final states, then the sequence is accepted by the DFA and the method returns true.

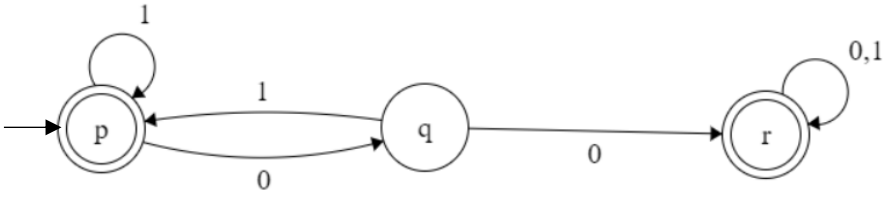
Integration with Scanner: In the LanguageSpecification class, the regex used for identifying integer constants and the one used for matching identifiers will each be replaced with a DFA which will check that the given sequence is accepted, i.e. is an integer constant/identifier. The two FAs are kept in the files intDFA.in and identifierDFA.in, and will be read and constructed when the LanguageSpecification class is constructed.

Implementation:
https://github.com/carinaghiorghita/UBB_Sem5_FLCD/tree/main/Lab4/src/ubb/flcd

Tests:

A Deterministic Finite Automaton read from file FA.in:

```
p q r
0 1
p
r p
q 0 r
r 1 r
p 1 p
p 0 q
q 1 p
r 0 r
```



letter ::= 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'Z'

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

alphabetCharacter ::= letter | digit

state ::= letter

transition ::= state alphabet state

states ::= {state }+

alphabet ::= {alphabetCharacter }+

initialState ::= state

finalStates ::= {state }+

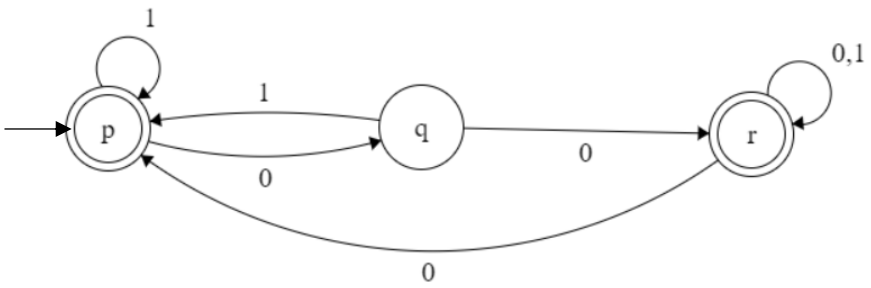
FAfile ::= states '\n' alphabet '\n' initialState '\n' finalStates

```
1. Test DFA
2. Scanner
Your option:
1
FA read from file.
1. Print states.
2. Print alphabet.
3. Print final states.
4. Print transitions.
5. Check if sequence is accepted by DFA.
0. Exit
Your option:
1
States: p q r

1. Print states.
2. Print alphabet.
3. Print final states.
4. Print transitions.
5. Check if sequence is accepted by DFA.
0. Exit
Your option:
2
Alphabet: 0 1

1. Print states.
2. Print alphabet.
3. Print final states.
4. Print transitions.
5. Check if sequence is accepted by DFA.
0. Exit
Your option:
3
Final states: r p

1. Print states.
2. Print alphabet.
3. Print final states.
4. Print transitions.
5. Check if sequence is accepted by DFA.
0. Exit
Your option:
4
Transitions:
<q,0> -> [r]
<r,1> -> [r]
<p,0> -> [q]
<q,1> -> [p]
<p,1> -> [p]
<r,0> -> [r]
```

```
Your option:
5
Your sequence:
0011
Sequence is valid

1. Print states.
2. Print alphabet.
3. Print final states.
4. Print transitions.
5. Check if sequence is accepted by DFA.
0. Exit
Your option:
5
Your sequence:
0110
Invalid sequence

1. Print states.
2. Print alphabet.
3. Print final states.
4. Print transitions.
5. Check if sequence is accepted by DFA.
0. Exit
Your option:
5
Your sequence:

Sequence is valid
```

For a Non-Deterministic Finite Automaton, when checking if a sequence belongs to the DFA, we will be alerted that this is not a DFA:

```
p q r
0 1
p
r p
q 0 r
r 1 r
p 1 p
p 0 q
q 1 p
r 0 r
r 0 p
```




```
FA read from file.
1. Print states.
2. Print alphabet.
3. Print final states.
4. Print transitions.
5. Check if sequence is accepted by DFA.
0. Exit
Your option:
5
FA is not deterministic.
```

Integration with the Scanner: The FA used for integer constants and the one for identifiers are on GitHub under the Resources folder. The program will work as previously.

Integer constants:

integer_const ::= [sign] non_zero_digit {digit} | '0'

sign ::= '+' | '-'

non_zero_digit ::= '1' | '2' | ... | '9'

Identifiers:

identifier ::= letter {(letter | digit | _)}

letter ::= 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'z'

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