Non Deterministic Finite Automaton (NFA) Tester

Project. Part I

Arturo Burela

This program reads from a file the elements that define an NFA and indicates if the input string is accepted by the automaton. Also prints all the accepted transitions.

The file must be defined as follows:

- First row: `Set of states`
- Second row: `Alphabet symbols`
- Third row: `Initial state`
- Fourth row: `Set of final states`
- Fifth row and up: `Transition function`

`Set of states` indicates all the states separated by commas. No whitespace. Each state may have a name of variable length. Example:

```
q0,A,longerName,q3,q4
```

`Alphabet symbols` indicates the alphabet symbols separated by commas. No whitespace. (This program actually ignores this line) Example:

```
a,b,c
```

`Initial state` indicates the initial state. Example:

```
q0
```

`Set of final states` indicates the set of final or acceptance states. Example:

```
q4,q0
```

`Transition function` indicates the transition function in the following format: currentState,input:destinationState. Example:

```
q0,a:q2
```

This means that when being in 'q0' state and the next input is 'a' will take us to 'q3' state.

`Epsilon Transitions` are also supported. Epsilon is represented by '&' character. Example:

```
q0,&:q3
```

For the previous functionality '&' can not be used as another element of `Alphabet symbols`.

Usage instructions

This program has been tested on Debian 9 and MacOS 10.12. It is necessary to have a C++11 compiler. There is a makefile to facilitate the process, clone the project in any desired location, enter to the project directory and type `make` in the terminal to compile the program. If you do not have make installed visit https://www.gnu.org/software/make/ (https://www.gnu.org/software/make/) . You may also compile the source yourself.

To use the program, first create the NFA file as described previously. Here is an example of a full NFA file:

```
q0,A,longerName,q3,q4
a,b,c
q0
q4
q0,a:A
q0,a:longerName
q0,b:longerName
q0,b:q3
A,a:q3
A,b:q3
longerName,a:longerName
longerName,b:q3
q3,&:q4
q3,a:q4
```

The program receives the NFA filename as the first argument and the string to test as second argument. For example to use the nfa file 'nfa1.txt' and test string 'aab' the program must be run like this:

```
./nfa nfa1.txt aab
```

The program will make the test and print the results. This approach facilitates testing different strings with different automatons.

Program Explanation

This program is written in c ++. It consists of 2 classes in 2 cpp files and a main function in a third file that makes use of them. They are divided as follows:

- main.cpp: `Main function`
- nfa.cpp: `NFA Class`
- state.cpp: `State Class`

`Main function` only contains a main function that checks for two arguments and passes them to NFA class constructor. This file includes nfa.cpp.

`NFA Class` defines the NFA structure. Contains all the states, the string to test, a pointer to the initial state and the paths of an accepted string. This class has the following functions:

- `test`: Calls explore recursive function on initial state, receives the paths and prints if the string is valid or not along with the paths.
- `findState` Search a state by name and returns a pointer to the it.
- `logStates` Logs all states data.
- `logPaths` Logs all valid paths.
- `loadFile` Reads the NFA file and fills NFA structure. Including states names, states links, initial and final states.

`State Class` defines states structure. It has attributes such as name, if it is final and a vector of 'link', a struct that contains the input and a pointer to the destination state. This class has the following functions:

- `addLink` Receives a link struct and adds it to the link vector.
- `getName` Returns the state's name.
- `setFinal` Marks the state as final.
- `explore` Recursive function that returns a vector of paths. Each path is conformed by a vector of links. The function receives the test string and use the next input to call itself at every link it matches. If the string is empty checks if the state is final and returns the appropriate value. All the epsilon links are always called first. If the received value is valid it appends the new link to the path and returns.
- `logData` Log all the state's data.

For more details see code and comments on each file. Find this project at https://github.com/ArturoBurela/ndfa-tester (https://github.com/ArturoBurela/ndfa-tester)