828 PROJECT REPORT

CAMERON CUNNING

1. Intro

The project for CSCE 828 is to write a program that can perform two functions:

(1) Determine if a string s will be accepted or rejected by a provided deterministic finite automaton **(DFA)**
(2) Determine if two provided DFA's are equivalent; i.e. $M_1 \equiv M_2 \iff L(M_1) = L(M_2)$

2. Program Description

I chose to write this program in the Ruby language. The lack of a need to compile/build along with the functions provided by the Ruby core API made Ruby a great choice for this project. Plus, I haven't used Ruby in awhile and just felt like going back to it. The program is spread across three different files: **test.rb**, **DFA.rb**, and **setnode.rb**

2.1. **test.rb.** This is the entry point for the program. This file looks at the command line arguments. There are two options for command line arguments:

(1) **{filename}.dfa {a string}** - String can be empty
(2) **{filename}.dfa {filename}.dfa**

The first option constructs a DFA class (contained in DFA.rb), which is a class containing fields for the states, input alphabet, start state, accept states, transition function, and current state of a DFA. The DFA then executes the string provided as the second argument by iterating over the string, and for each character, changing the current state of the DFA based on the transition function. Once this iteration is complete, if the current state is in the list of accept states, the machine **accepts**, otherwise it **rejects**.

The second option constructs two DFA objects ($\in$ DFA.rb) and tests to see if they are equivalent using the DFA.equivalent?(m1, m2) method.

2.2. **DFA.rb.** This file contains the class used to mode the DFA, as well as the **equivalent?** "static" method and the **run** "instance" method. The run method has been described previously. The **equivalent?** method checks to see if two DFA's are equivalent, and, i they are not, provides a "witness", i.e. a string that will pass on one machine and fail on another. I first discovered the algorithm I used in an MS thesis by Norton [4], but this algorithm originally appeared in a publication by Hopcroft and Karp in 1971 [3]. This algorithm performs the check in linear time. For brevity's sake, the algorithm will not be discussed in detail. For a detailed description of the algorithm, read the source code and

the cited references. The high level description is that we follow the transition function for each input, starting with the starting states, for both machines simultaneously. When the status of the states we are in differs, i.e. one is accepting and the other is not, then we know the machines are not equivalent. This algorithm makes clever use of disjoint set forest and hash map data sets to achieve this computation in linear time.

2.3. **setnode.rb.** This file contains an implementation of the disjoint-set forest data structure, exactly as described in the ubiquitous Introduction to Algorithms textbook [2]. The Wikipedia page [1] also has a good description.

References

[1] Disjoint-set forests. `https://en.wikipedia.org/wiki/Disjoint-set\_data\_structure#Disjoint-set\_forests`. Accessed: 2017-04-17.
[2] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms second edition, 2001.
[3] John E Hopcroft and Richard M Karp. A linear algorithm for testing equivalence of finite automata. Technical report, Cornell University, 1971.
[4] Daphne Norton. Algorithms for testing equivalence of finite automata, with a grading tool for jflap. 2009.