CMPT 440 – Spring 2019: Credit Card DFA Writeup

Katie Bartolotta
Due Date: 5/15/2019

Abstract

This paper presents a system to process credit card numbers. Using an interface with a Java backend, an individual is able to enter a string of numbers in order to determine if they create a valid sequence. This system can not only validate the numbers, but also find which major credit card company the number belongs to. This system creates a straightforward and easy to use interface for a user to process credit card numbers.

Introduction

It has grown more and more uncommon for people to carry around paper money with them, replacing this with a plastic card. Credit cards have become an easy and efficient way to pay for items on a daily basis. There is no need to get change back or do mathematical calculations of any kind. Instead, one must only hand over their credit card and the money is taken out of their account automatically. Each credit card is unique to the holder, having its own fifteen or sixteen digit number. One individual may have multiple credit cards, but they will all have different numbers. There are four major credit cards that are used which include: Mastercard, Visa, Discover, and American Express. Each company uses their own way of generating numbers. Thus, it is necessary to figure out which type of credit card is being processed before evaluating the digits to see if they represent a valid card, a two part process that is more intricate than it seems.

DFA Diagram

See DFA.png attached

Detailed System Description

The system is composed of two Java files, DFA.java and driverDFA.java. The DFA.java file is responsible for defining and implementing the array that will represent the transitions of the credit card DFA. It also defines what states are accepted and rejected. This file also contains the methods that will reset the DFA to the start state as well as the method that signals when an accepted state

is reached. Finally, it defines the process method that allows for the system to recognize and numbers between zero and nine as valid input, otherwise throwing an out of bounds exception.

The driverDFA.java file is responsible for reading the user input and calling the methods from the DFA.java file to run the input through the credit card DFA. The interface that the user interacts with is a JFrame. When the file is run, a new Frame will open on screen and prompt the user to enter a credit card number. Once the user has hit the submit button, the input is saved. In the background, the file will be listening for the submit button to be pushed and then it will read the text in the text field. As long as the input was not an empty string, it will be sent through the process method mentioned previously. If the number is accepted, the text field will display that the number entered is valid along with what type of card it is. Otherwise, it will tell the user that the number entered is not a valid credit card number.

The DFA to validate credit card numbers is fairly large. It contains forty states including two accepting state and one error state. Beginning from the start state, q0, the DFA can go in one of six directions. The first digit of the input determines if the DFA will follow a Mastercard, American Express card, Visa card, Discover card, or enter an error state. Mastercards begin with 51-55, 222-229, 23-26, or 270-272. American Express card begin with 34-37. Visa cards only ever begin with a 4. Finally, Discover cards begin with either 65 or 6011. After the beginning sequences of these cards where the type of card is determined, all the cards can accept any digit zero through nine for the remainder of their digits. The only difference is that American Express cards have only 15 digits while all the other types of cards have 16 digits. While the first two to four digits of the credit card can be tricky, the following digits are straightforward.

Requirements

The system can run on either Mac or PC and requires a Java compiler.

Literature Survey

There has been a lot of research into the finite automata realm as well as responding to user input in a Java program. My system combines the two, using a Java system to take in user input and determine if this contains a solution to the deterministic finite automata created for credit cards. One journal article named, ***Using Java to Develop Discrete Event Simulations***, describes the

process behind a Java program that responds to events initiated by the user entering some data. It also lays out the benefits to using Java over other languages. Another source of information found was a chapter called ***String Acceptors*** from a book on Automata. It details system behaviors including accepting behaviors also named finite state machines. This chapter describes just how these determining sequences work, stating that the accepting states of a finite automata ultimately defines a type of language in which the elements each describe a path through the system to an end accepting state. Again, while there is no exact system created like the credit card validation system I have presented, it does encompass pieces of research as state above. By using both the deterministic finite automata and the Java programming language, the credit card number validation system is able to traverse through the entered credit card number and compare it to the deterministic finite automata created.

User Manual

In order to ensure smooth operation of the credit card validation system, the user must ensure that both the DFA.java and the driverDFA.java files are in the same folder. To run the program, the user must execute the driverDFA.java file. Following this execution of the program, the user will be presented with the user interface, or the JFrame that was created. As of right now, when entering a credit card number, the user must hit the submit button instead of just hitting the enter button on the keyboard. From there, the Java program will evaluate the input and return a response. If entering more than one string of numbers, the user must highlight the text in the textfield to delete it first before entering the second string, otherwise there will be an error in processing the input as there will be letters included in the string instead of numbers. Other than those key points, using the credit card validation system should be fairly straightforward.

Conclusion

In conclusion, this Java program to validate credit card numbers as well as decipher what type of card the number sequence belongs to, is a combination of both the strengths of a deterministic finite automata and a Java user interface. This work is significant in the exploration of user interaction with a Java interface, using a JFrame. It also presents a table-driven DFA through Java programming. Overall, this system presents a user-friendly approach to determining whether a credit card number is valid.

References

[1] *String Acceptors. Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach, by Robert P. Kurshan, Princeton University Press, 1994, pp. 6376. JSTOR, www.jstor.org/stable/j.ctt7ztp96.8.*

[2] *Pidd, M., and R. A. Cassel. Using Java to Develop Discrete Event Simulations. The Journal of the Operational Research Society, vol. 51, no. 4, 2000, pp. 405412. JSTOR, www.jstor.org/stable/254167.*