Final Write-up

John DeFalco

May 2019

1 Abstract

This write-up covers my semester project of building a Definite Finite Automaton. The introduction covers the overview of the project, mentioning what the project topic is, and what differences I have noticed between what I expected and what the actual outcome has been. The detailed system description includes a diagram of the DFA, along with the formal definition of the automaton. This section also covers what the system does, and what it is used for. The requirements section covers what materials are necessary to practically apply this system into the real world. The literature survey goes over how standard parking meters work in the real world, and how this system is both similar and different to those other practical applications. The user manual covers how one would interact with this system, and how they may solve any errors they could possibly encounter when using the system. Finally, the conclusion goes over the material required.

2 Introduction

The project I have created is a DFA-implementation of a standard parking meter.

The motivation behind this idea was that a parking meter is a practical system that I've encountered before, and I figure it would help me to more easily understand how finite automata work. The parking meter is rather simple: enter coins into the machine until a minimum amount is reached, and then stop entering coins. The monetary value entered reflects a time period allotted, for example 25 cents can be 30 minutes of parking. Using this idea, I built what I believe to be a comprehensive and concise deterministic finite automaton to represent this system.

3 Detailed System Description

I have formally defined the DFA to be implemented as follows:

- $Q = \{0, 0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00, Payment, Error\}$
- $\Sigma = 0.25, 0.50, Enter$
- δ : *See Table 3.1*
- $q_0 = 0$
- $F = Payment$

State	Input (0.25)	Input (0.50)	Input (*Enter*)
0	0.25	0.50	*Error*
0.25	0.50	0.75	*Error*
0.50	0.75	1.00	*Payment*
0.75	1.00	1.25	*Payment*
1.00	1.25	1.50	*Payment*
1.25	1.50	1.75	*Payment*
1.50	1.75	2.00	*Payment*
1.75	2.00	2.00	*Payment*
2.00	2.00	2.00	*Payment*
Payment	*Payment*	*Payment*	*Payment*
Error	*Error*	*Error*	*Error*

Table 3.1: Transition Function

Since the midterm report, I've implemented methods to contain values for the time allotted and the monetary value inserted. These help by reporting the money inserted to the user every time they enter a new input, and show the user how much time they have purchased once they confirmed their payment.

4 Requirements

The requirements the user would have to have to use this machine is just enough money, in quarters or half-dollars, that is at the last 50 or more cents. One could argue the user would also need to have a vehicle to park in the space that the meter covers, however if a user so chooses they could hypothetically use the meter without the vehicle.

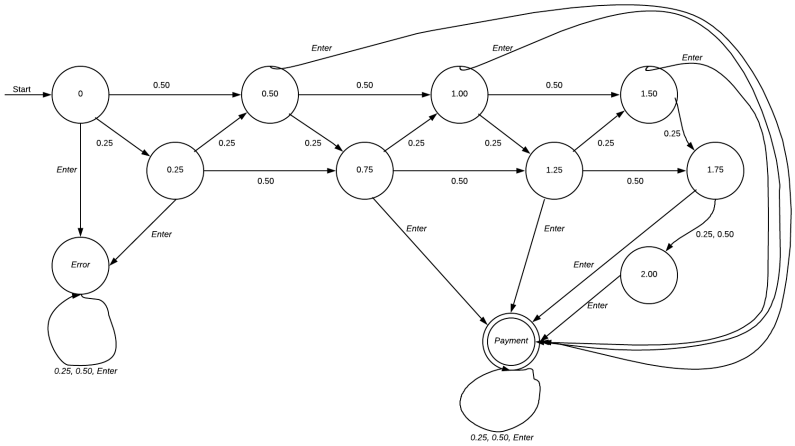

Figure 3.1: Drawing of DFA defined by the 5-tuple

5 Literature Survey

From my personal experience, this system is extremely similar to any other parking meter. There are some exceptions, meters that can accept credit/debit cards, and a number of meters that can accept more than 2 dollars, however my intention with this project was not to create a completely unique system.

Most parking meters are within an allotted period of time, for example the parking meters within the city of Poughkeepsie typically run from 8:00am to 6:00pm on weekdays, and outside of that time period parking is free for the user. This is not something I implemented into the project as it is outside of the scope of the DFA, but it is something to consider. If this was implemented into a real world application, this is a component that would have to be added in so the user would not have to pay at all hours of the day [1].

6 User Manual

The user would park their vehicle in the parking spot, and approach the meter covering the spot.

On the meter, they would see that 25 cents is equal to 30 minutes of parking space. The minimum is an hour of parking, or 50 cents, and the machine will accept either quarters or half-dollar coins. The maximum amount allowed is 2 dollar, or 4 hours of parking time. If the user attempts to enter more than 2 dollars, the machine will not stop the user from entering the money, however the meter **will not go further past 2 dollars.** If the user tries inserts 25 cents

into the machine and attempts to select *Enter*, the machine will not accept the payment, and prompt the user to enter more money until they pass the 50 cent threshold.

The user will enter their coins until their desired amount of time, and then select *Enter* to finalize the amount entered. From their, the meter will run until the allotted time is depleted, where the user will either have to enter more money into the meter, or move their vehicle.

7 Findings

Through this project, I came to understand how a *DFA* is useful for string validation. I came to realize that implementing a different project, for example one for checking input, would probably be more comprehensive as *DFA*s are used for input validation, typically within the lexical analyzer in a compiler. However, through this project I was able to use a *DFA* in an nontraditional manner. This allowed me to understand how to use the *DFA* more abstractly with the rest of my program, for example indicating what value and what time each state represents.

References

[1] N. City of Poughkeepsie. Article ix parking meters. pages 236–249, 2013.