Credit Card Validation Using DFA and Luhn's Algorithm

Miguel Mateo Osorio Vela (U96987884)

Evan Persad (U28675784)

John Hightower (U05320833)

Sofia Cobo Navas (U43878127)

April 20 2025

Introduction

Credit card numbers appear random, but they follow rules defined by major card issuers. These numbers are not only restricted to specific starting digits and lengths but also to a mathematical checksum, typically validated using the Luhn algorithm, to detect input errors. Traditional methods validate credit card numbers by sequentially verifying issuer patterns followed by checksum computation. Modeling these rules through formal automata provides an efficient validation. This project presents a Deterministic Finite Automaton (DFA) capable of validating credit card numbers by applying rules based on each issuer's unique digit sequence and calculating the Luhn checksum during transitions. By integrating structural and mathematical validation into a single framework, this system can quickly accept or reject a credit card number with minimal computation. Through testing on valid and invalid numbers, our solution demonstrates accuracy and practicality, offering insights on how automata theory can be used to solve real-world problems.

Methodology

The credit card validator was designed to model verification as a DFA with integrated Luhn checksum computation. The methodology followed several stages:

1) The DFA was constructed to model credit card structure based on issuer-specific prefixes and length requirements:
    - Start State: Reads the first digit to identify potential issuer.
    - Visa Path: If the first digit is '4', the DFA transitions into states expecting either 13 or 16 total digits.
    - MasterCard Path: If the first two digits are between '51' and '55', the DFA follows the MasterCard validation path for 16 digits.

- American Express Path: If the first two digits are '34' or '37', the DFA follows the Amex path expecting 15 digits.
- Reject States: If an unexpected digit or length violation occurs, the automaton transitions to a reject state.

2) As each digit is processed, the automaton updates two separate sums according to the Luhn algorithm:

- Every second digit is doubled, and if the result exceeds 9, its digits are summed.
- The sum of these processed digits is added to the sum of the untouched digits.
- At the final state, the validator accepts the number only if:

$$c \bmod 10 = 0$$

3) Implementation was conducted using Python within a Jupyter Notebook environment. while Plotly was used to display the resulting diagrams. The validator accepted user inputs, determined the potential issuer based on starting digits, tracked the number length, and confirmed validity based on the final checksum condition.

4) To make sure everything was working correctly, we tested the validator with both valid and invalid card numbers.

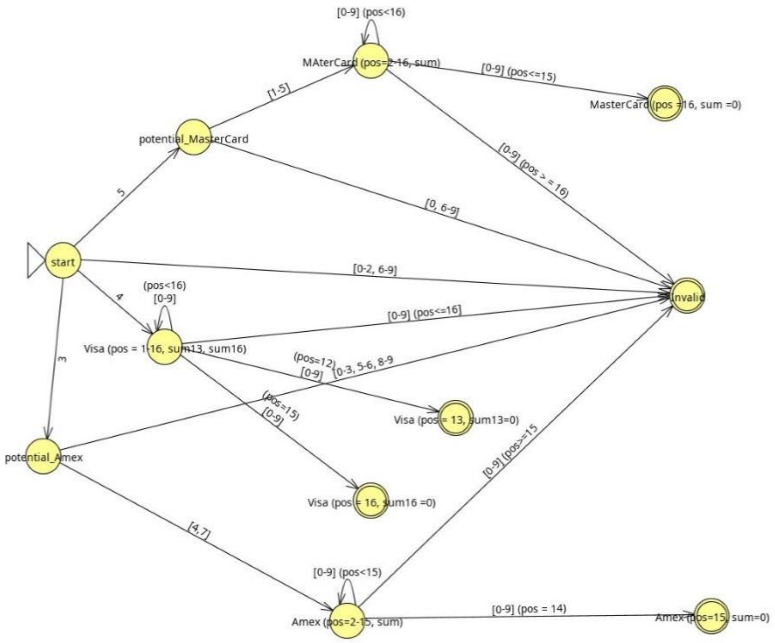
Results

The validator performed as expected classifying numbers based on issuer-specific rules and checksum integrity. For valid card numbers, the DFA accepted inputs that met the required starting digit patterns and length constraints while also satisfying the Luhn checksum condition. Sample valid inputs included 16-digit Visa and MasterCard numbers beginning with 4 and 5 respectively, and 15-digit American Express numbers starting with 34 or 37. For invalid card numbers, the system was able to reliably reject inputs that didn’t follow the correct structure or failed the checksum.

```
Testing 378282246310005: Expected American Express, Got AMEX
Testing 371449635398431: Expected American Express, Got AMEX
Testing 378734493671000: Expected American Express Corporate, Got AMEX
Testing 30569309025904: Expected Diners Club, Got INVALID
Testing 6011111111111110: Expected Discover, Got INVALID
Testing 6011000990139420: Expected Discover, Got INVALID
Testing 3530111333300000: Expected JCB, Got INVALID
Testing 3566002020360500: Expected JCB, Got INVALID
Testing 5555555555554444: Expected Mastercard, Got MASTERCARD
Testing 5105105105105100: Expected Mastercard, Got MASTERCARD
Testing 4111111111111111: Expected Visa, Got VISA
Testing 4012888888881881: Expected Visa, Got VISA
Testing 4999991111111113: Expected Visa, Got VISA
Testing 5199999999999991: Expected Mastercard, Got MASTERCARD
Testing 4999992222222229: Expected Visa, Got VISA
Testing 5299999999999990: Expected Mastercard, Got MASTERCARD
Testing 123456789101112: Expected Fake, Got INVALID
```

Table 1.- Validation results for sample credit card numbers

To better understand and explain how the system works, we visualized the entire DFA using Jflap. The diagram made it easier to trace how numbers progressed through different states and where invalid inputs were caught. This was especially helpful for debugging and verifying that each transition matched the intended logic.



Graph1.- DFA used for credit card Validator

Conclusion

This project showed how a deterministic finite automaton (DFA) combined with the Luhn algorithm can efficiently validate credit card numbers. By encoding issuer-specific digit patterns and length rules directly into the DFA, we were able to flag invalid inputs and recognize valid ones like Visa, MasterCard, and AmEx. We integrated the Luhn algorithm into the state transitions so the checksum could be calculated in real time, making the process faster than doing it in separate steps. Visualizing the DFA with Plotly helped us confirm that the logic was working correctly. The validator handled different inputs well, even catching common errors like swapped digits or wrong lengths. Overall, this project showed how automata theory can be used for real-world tasks like input validation. In the future, the tool could be expanded to include more issuers and extra checks for better fraud detection.

References

1. Harvard University. (2025). *CS50, Problem Set 1: Credit – Problem to solve*. Cambridge, MA.
2. Python Software Foundation. (2023). *time — Time access and conversions*. Python Documentation. https://docs.python.org/3/library/time.html
3. Plotly Technologies Inc. (2015). *Collaborative data science*. https://plot.ly
4. PayPal. (n.d.). *Standard test cards*. PayPal Developer Documentation. https://developer.paypal.com/api/nvp-soap/payflow/integration-guide/test-transactions/#standard-test-cards
5. Jupyter Widgets Community. (2025). *ipywidgets: Interactive HTML widgets for Jupyter Notebooks* (Version 8.1.6) [Software]. Jupyter Project. https://github.com/jupyter-widgets/ipywidgets