CPSC 439-01

Project 2

by Christos Bakis, Michael Clausen, Yao Lin, Ben Martinez, Long Vu, and Santiago Zavala

Introduction

For this project, we used two different platforms, the interpreter from the textbook run on a Google Colab notebook for part two, and Haskel run on a Replit for part three. Links to both are included below.

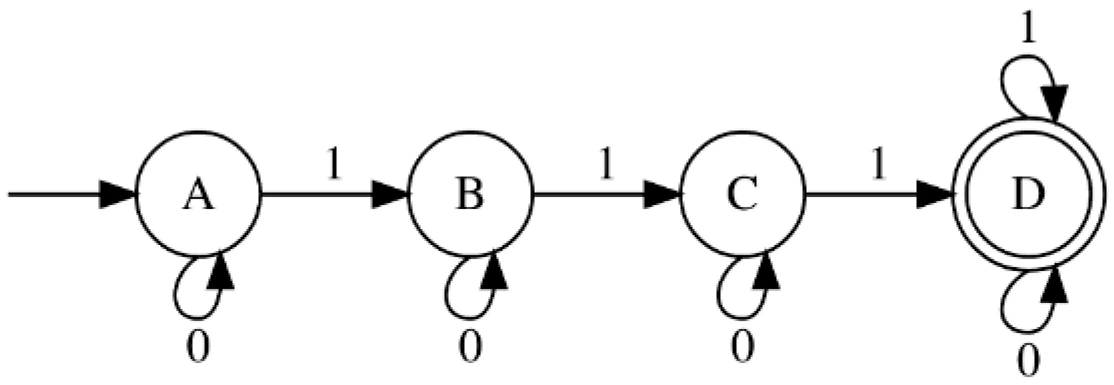
Part 1

Regular Language

For our regular language, we chose:

L = {w: w contains at least three 1s}

Below is the corresponding DFA:



Non-Regular Language

For our non-regular language, we chose:

L = {w: w = 1^n0^n for n >= 1}.

In other words:

L = {w : w contains at least one 1s, followed by the same amount of 0s}

Part 2

We used the Python interpreter from the textbook to implement our regular language in the enhanced lambda calculus. Our solution, along with multiple sample outputs, are included in this Google Colab:

https://colab.research.google.com/drive/1NRgmOf9CQH6jIYRfPlGadwIAD3H8CQhk

The main difficulty was recursively iterating through two different lists, the input list and the list of states, at different times using a single lambda expression. We accomplished this by only passing the “tail” of the list to the recursive call when we desired to iterate to the next element. We passed the head of the list or the list itself otherwise.

Part 3

Our part 3 was written in Haskell, for a challenge. The toughest challenge was following the enhanced lambda calculus limitations, as it restricted us from several Haskell standard library functions that could have made it easier. After attempts with various reduce methods, using recursion to iterate through the list until a non consecutive character was found to count the number of that character was done, and from that logic alongside the total size, we can recognize our language.

https://replit.com/@uygwfd254/project-2