Marist College

Deterministic Finite Automata

Solitaire

Andrew DiBella
Formal Languages CMPT440
Professor Rivas
15 May 2020

Abstract:

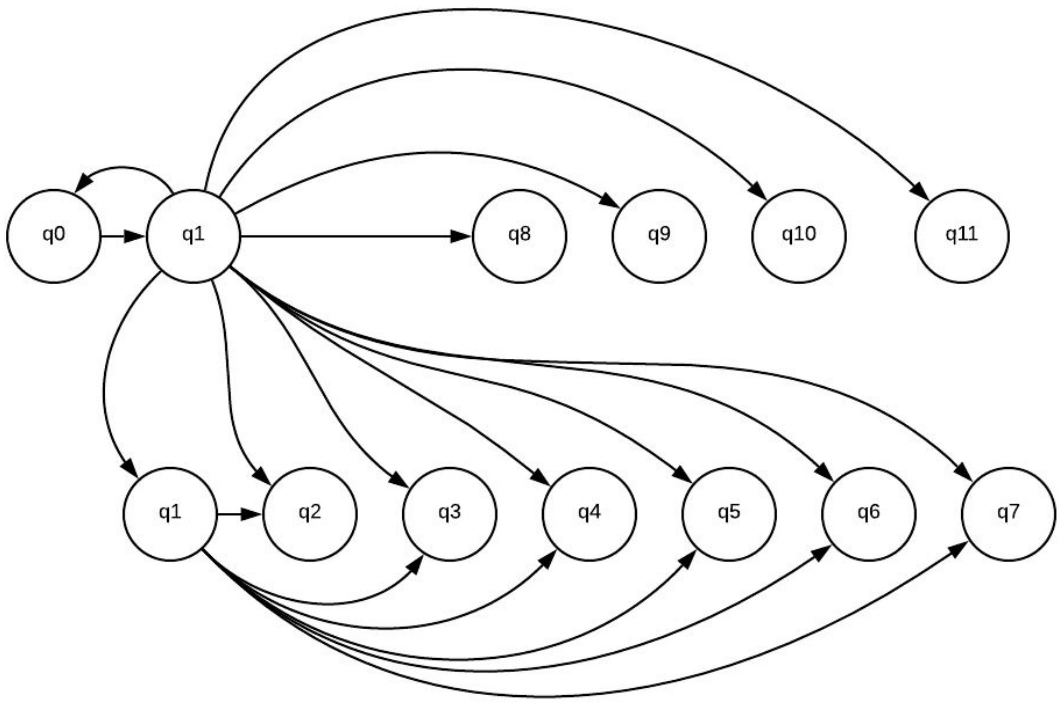
A Deterministic Finite Automata(DFA) is anything that can be described as having a defined number of states with a final accepting state as the end goal. Now any card game has an infinite number of possibilities depending on the shuffling of the alphabet in this game being the 52 cards in a full deck. Since this is the case I am going to abstract a DFA from the game of Solitaire and validate if the set of cards has a potential DFA based on the algorithm I created. Having an alphabet of 52 cards would make the DFA almost impossible to create; instead there will be three states and the alphabet will still be the cards but the diagram won't depict each individual card moving from state to state. Considering Solitaire is not easy to win, I have created my algorithm to follow basic strategy rules and if a game is won it will return the number of moves and the time it took to finish from a sample of differently shuffled decks. A shuffled deck will be considered a DFA if each suit is ordered from ace to king in the foundation state. Cards in the hand can only be placed if their preceding card is 1 greater than the previous value and the previous card is the opposite color of the card trying to be placed. Once the deal state and the hand state are both empty there is a possible DFA for that specific deck.

Introduction:

Solitaire is a famous card game designed for one player and a full deck of cards. Although many people find it boring and lengthy, it actually takes a fair amount of skill to win a game. Yes of course, online you are able to undo moves until you finally win, but where's the fun in that. Solitaire was one of the first card games I learned to play as a young kid, so I thought to create a DFA based algorithm that will perform the same actions to win a game of solitaire. Since it is not possible to win every game of Solitaire, I will be testing numerous differently shuffled decks on the algorithm I have created. The motivation for this project comes from my enjoyment from playing card games and I thought it would be interesting to combine that with my coding interests and implement a feasible single player game in Java.

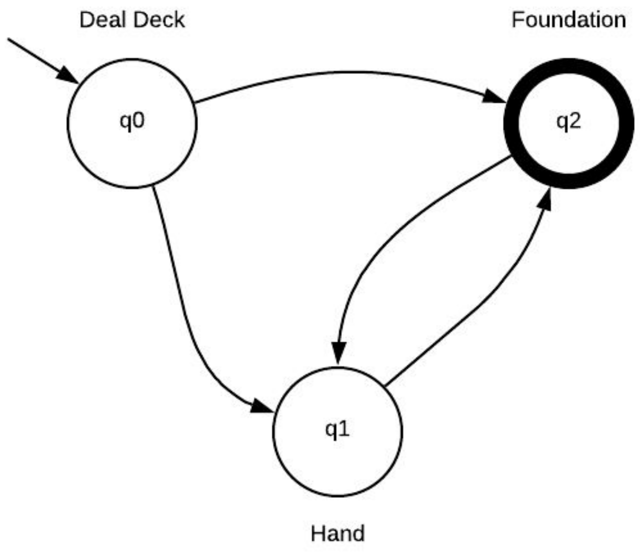
Detailed System Description:

My initial idea was to make a DFA for all of the possibilities for all 11 states and 52 cards. Looking back, I see now that this was a naive approach. The mere number of states and a large deck of cards as the alphabet would make the DFA extremely convoluted and impossible to comprehend.



This DFA is only the start of my initial plan and doesn't even consist of half of the transitions it should in the diagram. Each of the bottom states need transitions to and from each of the other states which you can see would be nearly impossible to implement as a DFA. Because of this I decided to divert my original plan and create an abstraction of this original DFA and make it something easier for a person to understand.

The new DFA consists of three states and the only transitions necessary from state to state is the mere absence of cards from the deal state, the initial state(q0), and its transfer to the player hand(q1), and the foundation being the accepting state (q2).



Even though this is a simple diagram with only 4 transitions, the algorithm I have created will still validate each card movement from state to state as described in the more intricate diagram. The program will take a randomly shuffled deck of cards and attempt to win a game of solitaire. If the program reaches an accepting state then each of the cards moved from the initial deal state and transferred either directly to the foundation, or from the hand to the foundation. Once a card has been moved from the deal state it will only be able to transfer back and forth from the hand and the foundation. Considering it is possible to lose a game of solitaire depending on the moves you make and the deck given, there will not always be a corresponding DFA for each game.

My implementation of solitaire will include 4 java classes

Card.
CardStack
CardStackNode
Solitaire: Driver

The program will utilize several different CardStack objects for each state shown in diagram1 which represents each CardStack. I chose to use a stack data structure to organize the data due to its First In Last Out to represent a physical deck of cards. This will make it easy to only access the top of each stack which is what is used in the game of Solitaire.

Requirements:

Literature Survey:

User Manual:

If you are interested on testing this DFA yourself on different decks follow these instructions:

1. Open a command line
2. Clone the Repository-
   ```
   $ git clone https://github.com/andrewdibs/cmpt440dibella.git
   ```
3. Change directory path to -
   ```
   $ cd cmpt440dibella/writeup/code
   ```
4. Compile all java files -
   ```
   $ javac <filename>.java
   ```
5. To run program -
   ```
   $ java Solitare <   <inputFile>.txt
   ```

Input files should follow the structure of "value suit" -

```
1 H
5 S
11 D
12 C
```

This input should consist of a total of 52 cards in any order with no duplicates.Notice the number inputs for 11 - 14 represent Jack through King. In most card games Ace is represented as either 1 or 15; in the game of Solitaire Ace is always defined as a value of 1.

Results:

Conclusion:

References: