Jacob Itz

Formal Languages and Computability

Pablo Rivas

Project Milestone

The purpose of this project is to show how one would use a Deterministic Finite Automaton in a Tic Tac Toe game. The idea is that each move made by either player will result in a change to the gamestring. And after the gamestring is altered it will be fed to a DFA which will determine if a player has won, or if it is the next player's turn. In addition to validating each turn, a DFA will decide determine which moves the computer will make while trying to defeat the Player.

As I tried to workout how to construct a diagram for a DFA which validates a Tic Tac Toe board layout, I realized that my original plan for a language consisting of 9 characters would not suffice because of the number of states it would require. I broke the issue down and decided to build two DFAs: one to check if Player 1(X) had won, and a second to check if Player 2(O) had won. So my new language has 3 symbols(0, 1, 2). The board is represented as a nine-digit string where the index of each character corresponds to a square on the board. The first three digits are the first row, the second three are the second row, and the final three represent the third row. In this string, a 0 is an empty space, 1 is a space with an X on it, and 2 is a space with an O on it.

Once I had decided on the language, I had to figure out which strings should result in wins(a.k.a. Accepting states). There are 8 patterns that result in a win for each player, 6 straight lines(full row or column of Xs) and 2 diagonal lines that should be accepted.

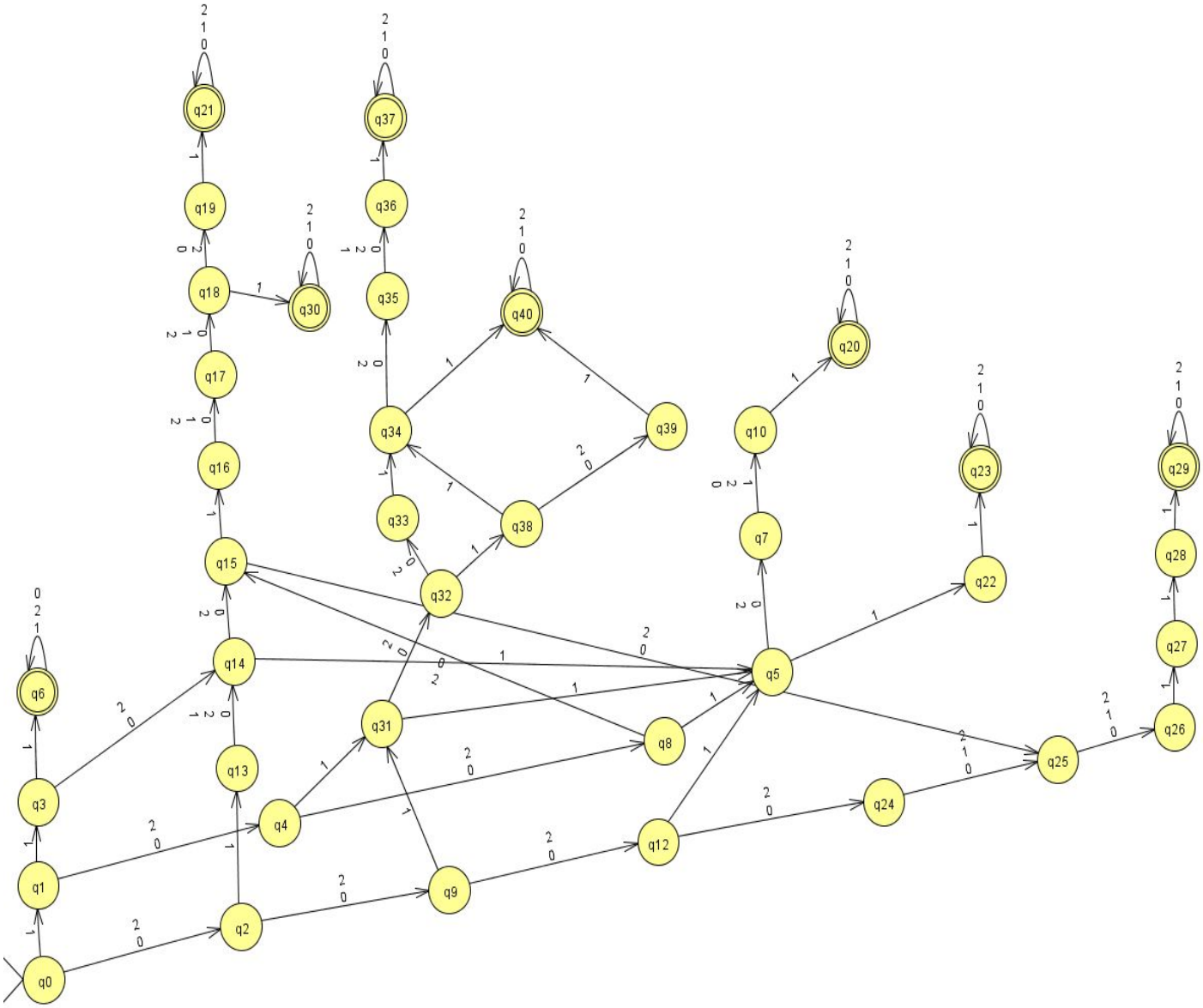
- 111###### - Top row
- ###111### - Middle row
- ######111 - Bottom row
- 1##1##1## - Left column
- #1##1##1# - Middle column
- ##1##1##1 - Right column
- 1###1###1 - Top left to bottom right diagonal
- ##1#1#1## - Top right to bottom left diagonal

Using a DFA building tool called JFLAP, I constructed the X DFA where each state had one transition for 1(the symbol representing an X) and one transition for 2 and 0. I then tested a variety of game string because JFLAP provides functionality for that. To construct the O DFA I took the X DFA and swapped the transitions for each state, one transition for 2 and one transition for 1 and 0 at each state.

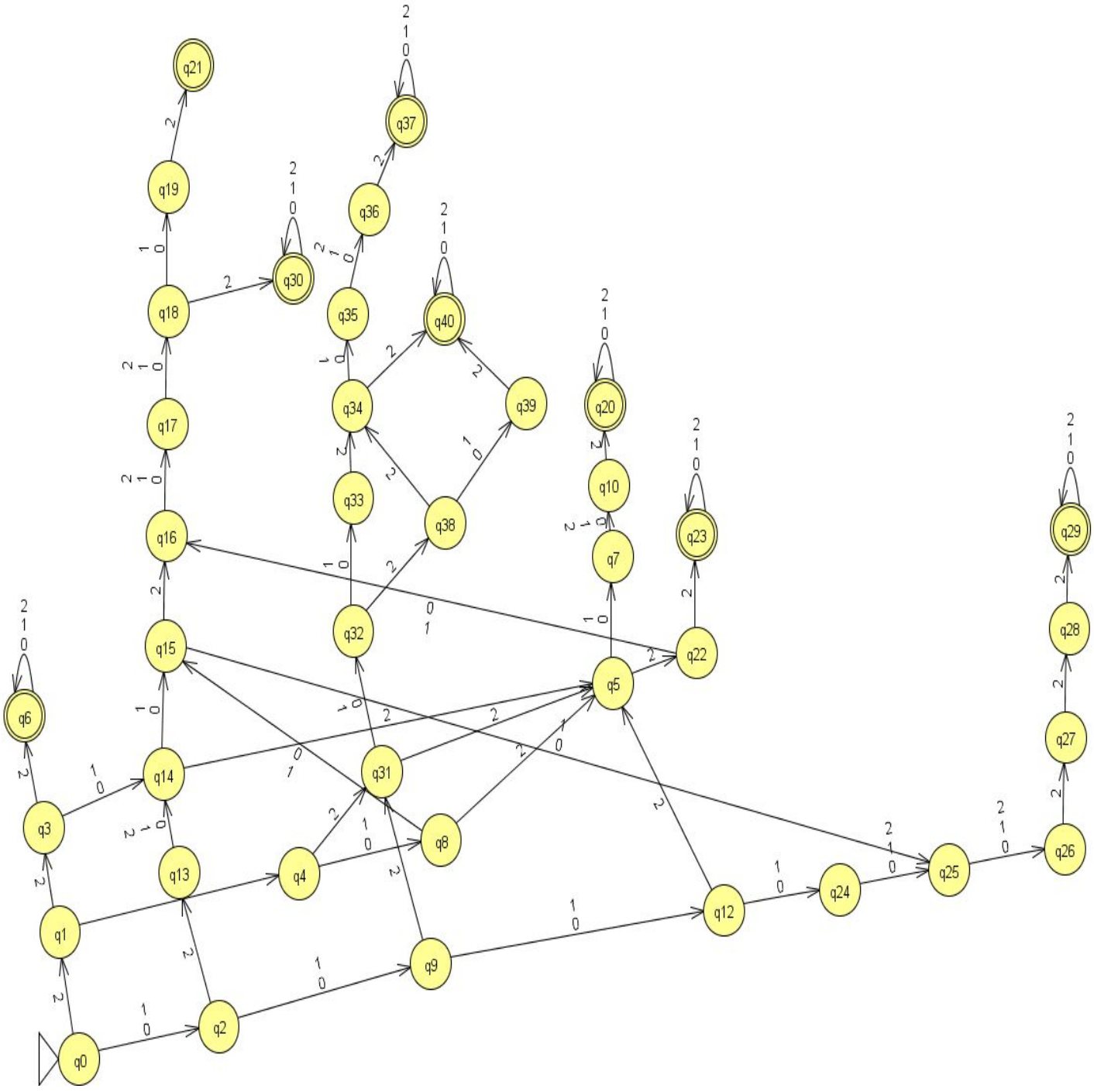
The next step will be to implement an algorithm which uses the DFA to decide which space to put an O in based on Player 1's moves. I am not sure if it will be able to beat the player, but it will at least be able to block player 1's chances of winning.

Diagrams for each DFA can be found on the following pages, as well as in the milestone folder in my repository.

Player 1(1/X):


q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q12
q13
q14
q15
q16
q17
q18
q19
q20
q21
q22
q23
q24
q25
q26
q27
q28
q29
q30
q31
q32
q33
q34
q35
q36
q37
q38
q39
q40

Player 2(2/O)


q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q12
q13
q14
q15
q16
q17
q18
q19
q20
q21
q22
q23
q24
q25
q26
q27
q28
q29
q30
q31
q32
q33
q34
q35
q36
q37
q38
q39
q40