

Ricardo Deodutt

My simple and unique code will be attached. The main objective of my code is read a users input and match it with the predertmined DFA. I felt the predetermined DFA string should be my name, "Ricardo". Before I get into details, please take a look at the DFA which is shown below in Figure 1. In figure 1 there are lots of vertices and edges. This graph is specifically an directed graph. There are also self loops in important parts of the graph. This DFA accepts the language $L = \{ \text{Ricardo, ricardoo, ricardooo, ricardooo,} \}$.

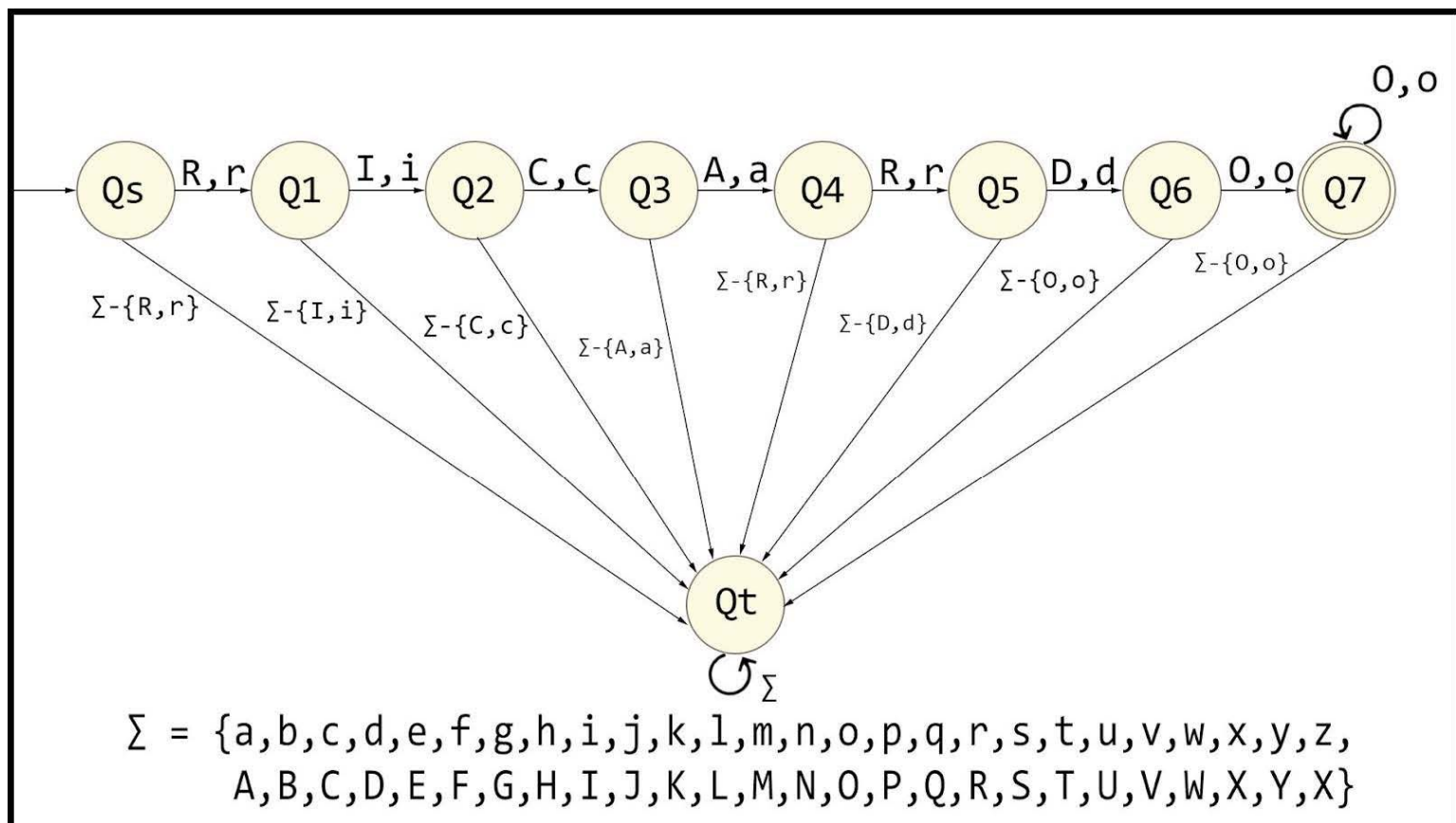


Figure 1

This DFA is specifically designed for my name, which is Ricardo. I thought this was unique and different because most students in my class are doing numbers such as 0010, 1010, etc. I felt this was more of a challenge than random numbers. The best part of my program that the characters are case sensitive. I made it so lettering does not matter. Before I did this project I only knew how to either make it capital letters only or lowercase. These are just small details that I liked to add to my program. In Figure 1, Σ is a finite input alphabet. In my case, I used the whole alphabet in capital letters and lowercase. There are 3 major states. Q_s is a start state. Q_7 is the accepted state/ final state. Q_t is a trap/dead state. In my DFA, the only string that is accepted is "Ricardo". The first few letters has to include my name. Once my name is inputted, the user can add any amount of the letter 'o'. For example, if a user enters "Ricardoooooooo", the string will be accepted. Once the user enters the last digit, 'o', they are sent to the accepted state which has the loop of letter o's. While in the accepted state, if a user enters anything that's not the letter 'o', then the string would be denied and the next state would be the trap state. In figure 1, the symbol Σ is portrayed a lot. Essentially Σ is every letter in the alphabet that is capitalized and lowercase. In figure 1, at the start state, the user has two paths to go. The user can enter either 'R,r' or " $\Sigma - \{R,r\}$ ". R,r means it can either be two entries which will be uppercase R or lowercase r. The term " $\Sigma - \{R,r\}$ " means any letter in the alphabet that's not 'R' or 'r'. Another example in figure 1 is in the trap state. When in the trap state, there is a loop that has the symbol ' Σ '. That basically means, any letter in the alphabet the user enters is sent back into the trap state.

In figure 1, the DFA starts off with the Q_s (start state). Whenever a user a letter, there are only 2 paths it could go. If a user enters 'R' or 'r', then the next state will be Q_1 . In the case the user enters any other letter that's not 'R' or 'r', then the next state would be the trap state. Once the user is in a trap state, there is essentially no escape. Anything the user enters in the trap state will be looped back into the trap state.

In the code attached, the program does exactly what my DFA is asking for. I found it was pretty simple to use a bunch of nested if statements. By using nested if statements, it allowed the specific string that I wanted, which is "Ricardo". The main perk of my code is that, it is case sensitive. When I originally did this project, there was only one specific case lettering. I spent

time researching how to allow case sensitive characters. At first I had a char list with all the possible entries for the 1st, 2nd, 3rd, ... , 7th letter. Now I have the if statements that check for both character versions. My code looks much better than it did before.

My code starts off by asking a user to enter a letter. The letter is then inputted by the user and that value is compared in an if statement. If the letter matches any of the predetermined values, another value is asked and the value is changed. That new value is then compared in another if statement, and if that matches a new value is asked by the program. In line 40 of my code the 7th letter is checked in an if statement, and if it matches a loop location is set after that line. The reason I set a loop location there is because after the 7th letter, its essentially a loop. There will only be one letter accepted by the program. This program matches in DFA in figure 1. In figure 1, the Q7 state is the final accepted state. In that final state, you can see there are two paths. If a user enters 'O' or 'o', the DFA is basically looped back into the same state. In my code the same thing is done in line 41-46. Between those lines, a user is asked to enter any letter, if that letter is 'O' or 'o', then the program will use a goto statement and send the program back to line 41. Once the program is back in line 41, it will ask the user for another letter and continue to loop until a user enters a wrong value or a symbol *. The symbol * is added in order to put an end to the program. If this part was not added, the program would loop forever or until something wrong is entered. In lines 56-100, there are a bunch of else statements to match the if statements. These else statements basically tell the user the string was denied and ends the program.

While testing my program I had to do a lot of changes and tweaks. Overall I like the way the program looks. In figure 2, this is the bare minimum the program can accept. Based off the DFA in figure 1, the letters that are required before the final state are "Ricardo". In figure 5, there is an example of how the loops work in the DFA's final / accepted state. A user can enter any amounts of letters as long as it's the letter 'O', 'o' or a '*' to end the program. A great test case is in figure 6. Figure 6 basically shows the test case where a user tries to end the program using the symbol '*'. The program will recognize this and deny the string. The program matches the DFA in figure 1 and requires the user to enter the predetermined string "Ricardo" first then the program can end with an '*'. In figure 7, the program illustrates what will happen if you

enter an letter that's not 'o' in the final state. The program does what is in figure 1, and sends DFA to the trap state. In figure 3 and 4, the program does what is expected in the DFA. The program denies anything that doesn't match the predetermined string which is my name "Ricardo". Also a side perk is the program accurately tells the user which letter number was wrong.

Σ is a finite set

Q is a finite set called the states

$\delta : Q \times \Sigma \rightarrow Q$ is the transition function.

$F \subseteq Q$ is the set of accept states.

Based off the DFA in figure 1,

$Q = \{Q_s, Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q_t\}$

$\Sigma = \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z, \\ A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,X\}$

δ = Transition Function below

$\delta(Q_s, R) = Q1$ $\delta(Q_s, r) = Q1$

- $\delta(Q_s, \Sigma - \{R,r\}) = Q_t$

$\delta(Q1, l) = Q2$ $\delta(Q1, i) = Q2$

- $\delta(Q1, \Sigma - \{l,i\}) = Q_t$

$$\delta(Q2, C) = Q3 \qquad \delta(Q2, c) = Q3$$

- $\delta(Q2, \Sigma - \{C, c\}) = Q_t$

$$\delta(Q3, A) = Q4 \qquad \delta(Q3, a) = Q4$$

- $\delta(Q3, \Sigma - \{A, a\}) = Q_t$

$$\delta(Q4, R) = Q5 \qquad \delta(Q4, r) = Q5$$

- $\delta(Q4, \Sigma - \{R, r\}) = Q_t$

$$\delta(Q5, D) = Q6 \qquad \delta(Q5, d) = Q6$$

- $\delta(Q5, \Sigma - \{D, d\}) = Q_t$

$$\delta(Q6, O) = Q7 \qquad \delta(Q6, o) = Q7$$

- $\delta(Q6, \Sigma - \{O, o\}) = Q_t$

$$\delta(Q7, O) = Q7 \qquad \delta(Q7, o) = Q7$$

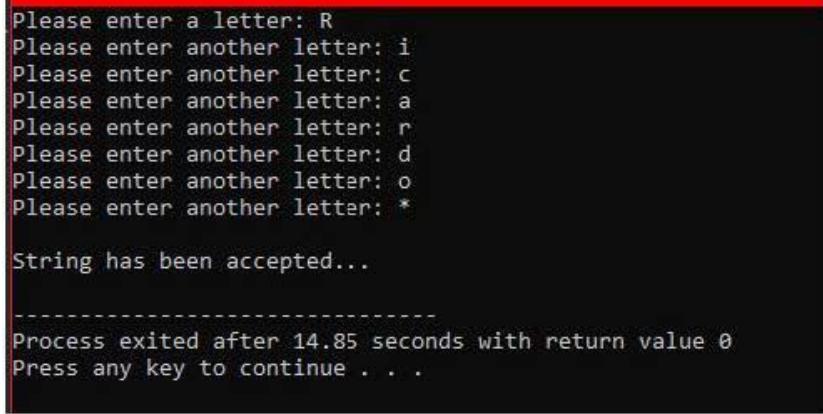
- $\delta(Q7, \Sigma - \{O, o\}) = Q_t$

- $\delta(Q_t, \Sigma) = Q_t$

Q_s is the start state, and $F = \{Q7\}$

Test Cases

Regular test case: Ricardo

A terminal window with a black background and white text. The text shows a sequence of prompts and inputs: 'Please enter a letter: R', 'Please enter another letter: i', 'Please enter another letter: c', 'Please enter another letter: a', 'Please enter another letter: r', 'Please enter another letter: d', 'Please enter another letter: o', and 'Please enter another letter: *'. This is followed by 'String has been accepted...', a dashed line separator, and then 'Process exited after 14.85 seconds with return value 0' and 'Press any key to continue . . .'.

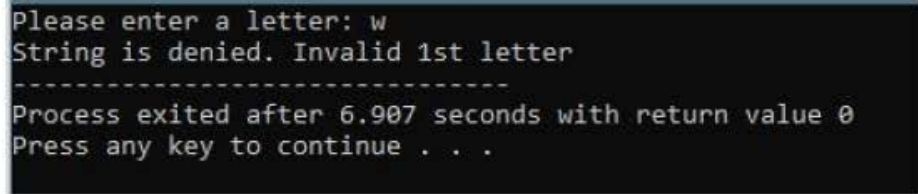
```
Please enter a letter: R
Please enter another letter: i
Please enter another letter: c
Please enter another letter: a
Please enter another letter: r
Please enter another letter: d
Please enter another letter: o
Please enter another letter: *

String has been accepted...

-----
Process exited after 14.85 seconds with return value 0
Press any key to continue . . .
```

Figure 2

Wrong First letter

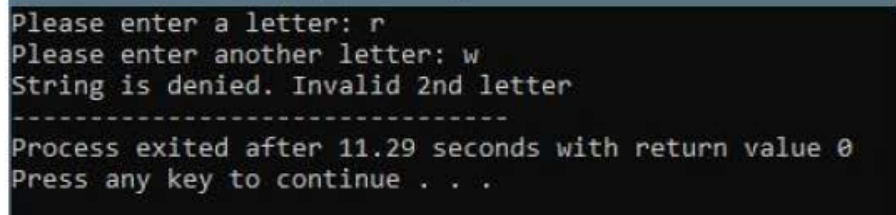
A terminal window with a black background and white text. The text shows a prompt 'Please enter a letter: w', followed by 'String is denied. Invalid 1st letter', a dashed line separator, and then 'Process exited after 6.907 seconds with return value 0' and 'Press any key to continue . . .'.

```
Please enter a letter: w
String is denied. Invalid 1st letter

-----
Process exited after 6.907 seconds with return value 0
Press any key to continue . . .
```

Figure 3

Correct First Letter, incorrect second letter

A terminal window with a black background and white text. The text shows a prompt 'Please enter a letter: r', followed by 'Please enter another letter: w', then 'String is denied. Invalid 2nd letter', a dashed line separator, and then 'Process exited after 11.29 seconds with return value 0' and 'Press any key to continue . . .'.

```
Please enter a letter: r
Please enter another letter: w
String is denied. Invalid 2nd letter

-----
Process exited after 11.29 seconds with return value 0
Press any key to continue . . .
```

Figure 4

Extended Test Case

```
Please enter a letter: R
Please enter another letter: i
Please enter another letter: c
Please enter another letter: a
Please enter another letter: r
Please enter another letter: d
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: *

String has been accepted...

-----
Process exited after 10.29 seconds with return value 0
Press any key to continue . . .
```

Figure 5

Trying to end string early with *

```
Please enter a letter: R
Please enter another letter: i
Please enter another letter: c
Please enter another letter: *
String is denied. Invalid 4th letter

-----
Process exited after 5.942 seconds with return value 0
Press any key to continue . . .
```

Figure 6

Invalid entry in the continuous loop state

```
Please enter a letter: R
Please enter another letter: i
Please enter another letter: c
Please enter another letter: a
Please enter another letter: r
Please enter another letter: d
Please enter another letter: o
Please enter another letter: o
Please enter another letter: o
Please enter another letter: w
String has been denied
-----
Process exited after 7.616 seconds with return value 0
Press any key to continue . . .
```

Figure 7

Handwritten graph and explanation of the DFA in the next page ----->

The handwritten DFA is the same as figure 1, but I personally liked the explanation that I wrote.