REPORT
ON

Application Of Continuous Time DFA

Siddheshwari C. Sisodia	101081032
Supriya B. Narute	101081024
Dhanashree N. Desale	101081038
Jaydeep R. Pawar	111080972
Shantanu B. Jain	071080022

Introduction

Deterministic Finite Automata

In DFA the machine can exist in only one state at any given time. For an input it goes to one and only one state. DFA works on Discrete time.

Discrete Time DFA

DFA were all working in "discrete time". This means that the inputs came one by one, and after each input the DFA produced the output, updated the state, and then waited for the next input.

Continuous Time DFA

If we assume that the inputs arrive every second then it is like a short burst of activity at every tick of the second-hand followed by no activity until the next tick. In the world of hardware, however, time is continuous. Inputs and outputs are voltages through wires and every wire has some voltage or other *all the time.*

A little silly game

Here is a small role playing adventure game. The storyline is like this: You are a gunman figting against a tank. Your gun is strong enough for the encounter, but the tank is behind a moving shield-like armour that makes it immune to gunfire. However, you also have a disarmer which deactivates the armour. After you have disarmed the tanker you can try to kill it with your gun. You will 006Eeed two firings for that, as the first bullet will only injure the tank, but not kill it. If you do not succeed in killing the tnk within three steps (*i.e.*, three button clicks) then the tank will kill you.

Here is the transition diagram. We have 7 states (numbered 0 to 6). initial state. State 6 is the final state (where either the player or the tank is destroyed). All the other states record two info: the elapsed time and the condition of the tank (healthy, disarmed or injured). The arrows are labelled with only the inputs (d for disarm, f for fire).

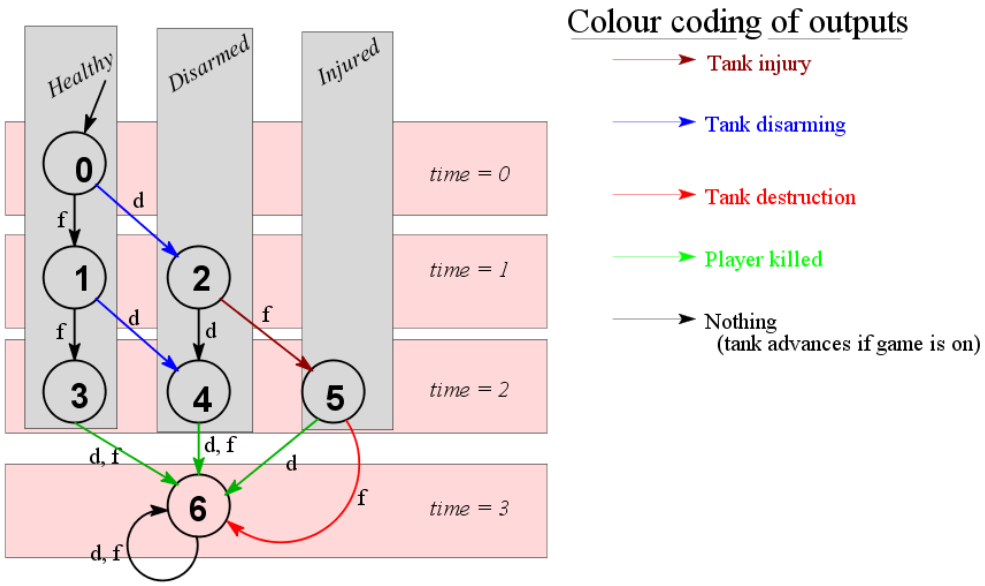


Figure: Transition Diagram

This game dosen't create the same feeling as you would feel if you faced a tank charging straight at you. Here it is obvious that you are merely playing a game against the tank where the MOVES ALTERNATE. Howsoever powerful the tank might be, it has to wait patiently until you make your move. That is to say time is discrete here. It is marked by ticks that correspond to your pressing a button. Nothing can happen *between* two ticks.

A real enemy, however, will not wait for you to take action. It will move at its own pace unless you manage to thwart its attack in time. In other words, in real world time is continuous. At each state we emit some particular message on the screen, and then we *wait* for your response. It is this waiting that blocks everything else until you click on a button. We refer to this type of behavior as **blocking**. So to make it real it becomes necessary to apply continuous time DFA.

Here is a graphics version of the same game. You must first click the "START" button to start the game.

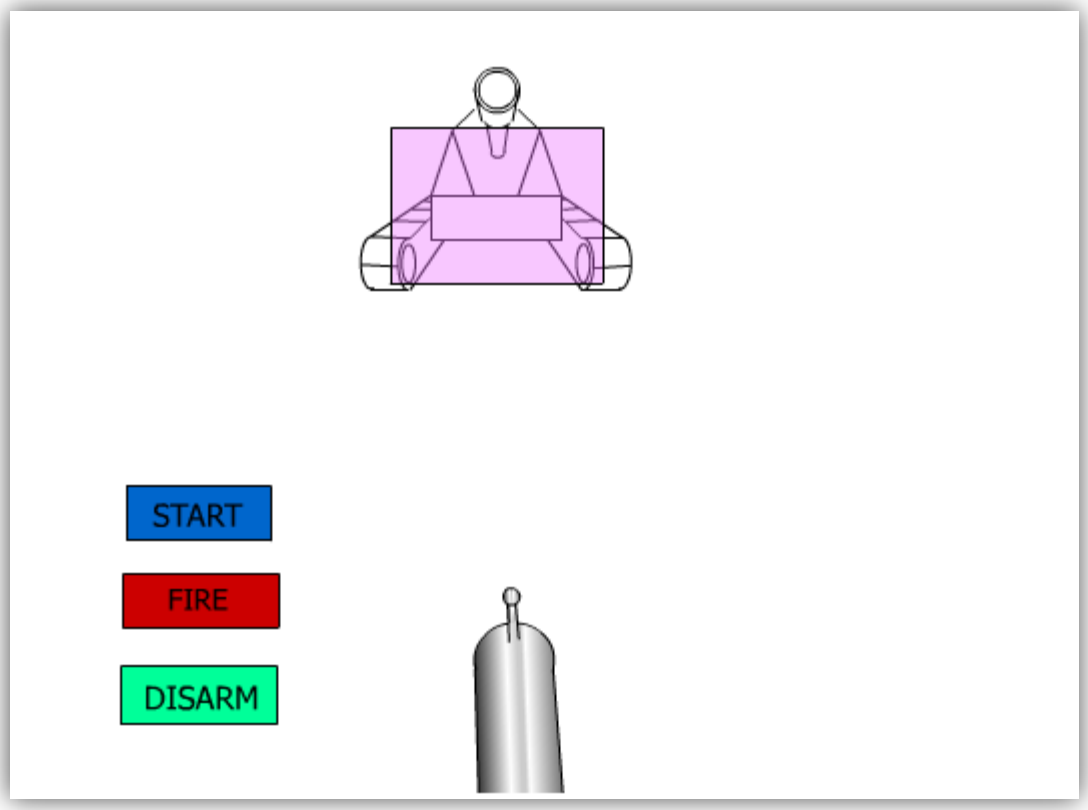


Figure: Graphics Version

This game is much closer to reality. The tank seems to have a life of its own. Even if we do not actively do anything the tank now continues to approach, *i.e.*, the output is produced in *continuous* time. However, at the same time it is also continuously listening for any input from the user, since the instant we click "DISARM" the armor vanishes.

At the level of milliseconds there are actually 6 inputs:

1. Timer on, "Disarm" released, "Fire" released: (abbrev: tdf),
2. Timer on, "Disarm" released, "Fire" pressed: (abbrev: tdF),
3. Timer on, "Disarm" pressed, "Fire" released: (abbrev: tDf),
4. Timer out, "Disarm" released, "Fire" released: (abbrev: Tdf),
5. Timer out, "Disarm" released, "Fire" pressed: (abbrev: TdF),
6. Timer out, "Disarm" pressed, "Fire" released: (abbrev: TDf),

Since you have only one mouse pointer, you cannot press both the buttons simultaneously! So there is no DF case. Also, because of the same reason the input dF and the input Df cannot occur side by side. You'll

need some space of time to take the mouse pointer from one button to the other, thus inserting the input tdf or Tdf in-between.

The transition diagram for such case can be as follows:

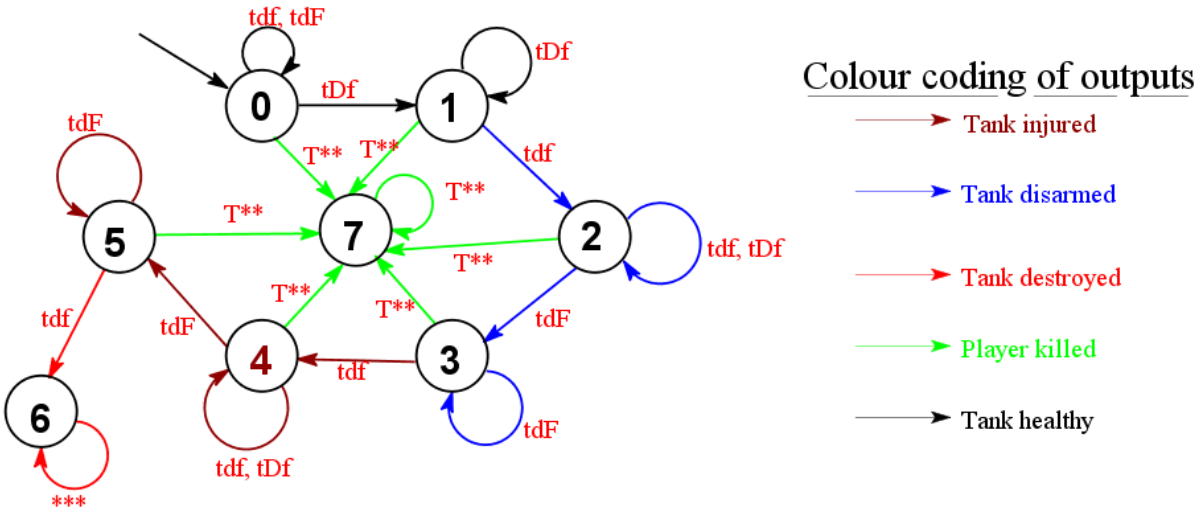


Figure: State Diagram

We have used some abbreviations here: T** stands for all inputs starting with a T (*i.e.*, Tdf, TdF, TDf). Similarly, *** denotes the set of all possible inputs.
Here 0 is the initial state.

	tDf	tdF	tdf	TDf	TdF	Tdf
0	1	0	0	7	7	7
1	1	-	2	7	7	7
2	2	3	2	7	7	7
3	-	3	4	7	7	7
4	4	5	4	7	7	7
5	-	5	6	7	7	7
6	6	6	6	6	6	6
7	7	7	7	7	7	7

Figure: Transition table

Implementation:

There are many ways to implement such DFA in programming; one of the simplest is by using conditional calls or conditional branching

i.e.

Each state can be defined as a function and each transition can be defined as a function call

e.g.

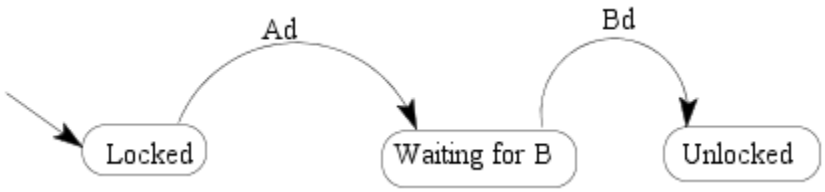
```
if(input==a)
        Injured();
else if(input==b)
        destroyed();
```

```
else
        healthy();
```

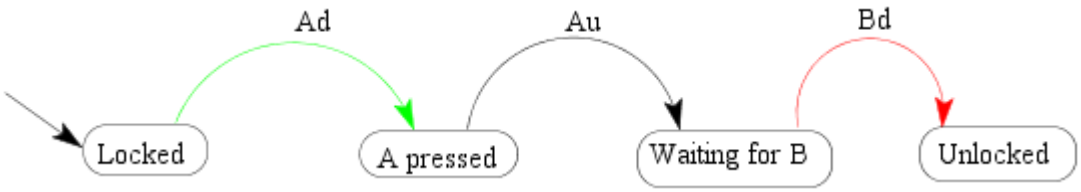
Each of the function themselves contain different conditional function calls.
In such way the whole DFA can be implemented as a program.

Mobile keypad unlock

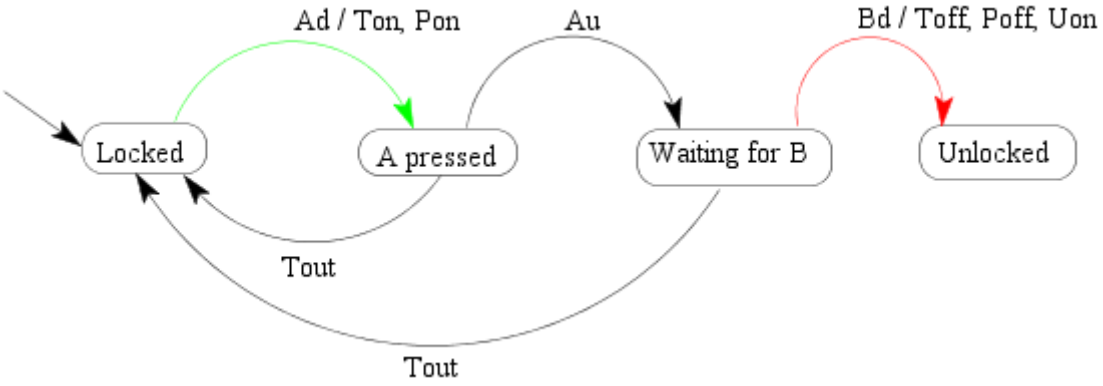
Most mobile phones feature a keypad locking facility that automatically locks the keypad if it lies idle for a specific amount of time. The typical unlocking technique is to press two particular keys (*A* and *B*, say) in rapid succession. If any key other than *A* is pressed a message like "Press A and then B" is displayed. After *A* is pressed the message "Press B now" is displayed for some time. If the user fails to press *B* during this display, the screen reverts to the blank display. If the use presses *B* before the display vanishes, the keypad is unlocked, and the message "Keypad unlocked" is displayed.
We want to write a microcontroller program that emulates this feature. Specifically, we shall have three buttons *A,B* and *C* (the last one representing the rest of the keypad). We start by constructing a DFA.
First consider only the buttons *A* and *B*. We shall denote by *Ad* the pressing of button *A*, while *Au* is for releasing it. Similarly for *Bd, Bu*. Ignore the timing issue. Then we have the state transitions:



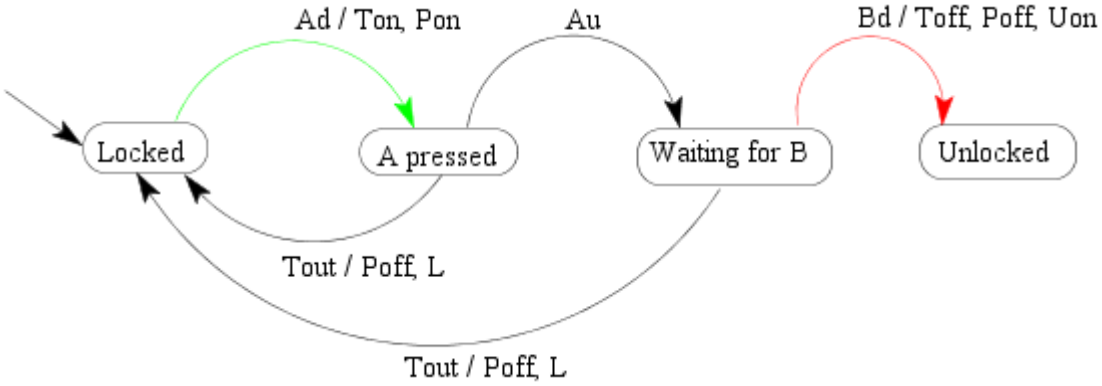
Notice that in this first version we have not cared about the release of button *A* before pressing *B*. In my mobile, however, this does matter. You must release *A* before pressing *B*. So we update the diagram to the following.



Now we bring the time consideration. Each arrow is like a time point. The red arrow must occur within a specified time gap after the green arrow. This means we have to start the timer in the green arrow, and turn it off in the red arrow. Also, we have to worry about what happens if the timer expires before reaching the red arrow. Well, we shall keep it simple by assuming that all time outs lead to the initial locked state.



Next the *C* button enters into picture. Since *C* does not play any role in the unlocking algorithm, any occurrence of the inputs *Cd* or *Cu* takes us to the locked state. We shall not clutter our diagram by explicitly showing this. Finally, we shall introduce the messages. We shall have three LEDs *L*, *P* and *U* corresponding to the messages: "Press A and then B", "Press B now" and "Keypad is unlocked". Some of the outputs are trivial to insert in the transition diagram.



Double click

This example is motivated partly by the double-click mechanism of a mouse, and partly by the keypad of a mobile phone. The problem is this: we have to make a device with a single button and two LEDs such that if we press the button then LED 1 will glow untikl we release the button. If, however, we "double click", (*i.e.*, press-release-press with the time gap between the two presses below a certain threshold) then LED 2 (and not LED 1) will glow. The LED will go off when we release the button.

We shall first design the DFA. Notice that we need to have a timer to measure the time gap between two successive presses. The timer can be in one of two states: on or off. Also the button can be in one of two states: up or down. Combining these possibilities we have 4 states in all:

1. button up, timer off
2. button up, timer on
3. button down, timer off
4. button down, timer on

A little trial-and-error now yields the following state diagram. Here the output S stands for turning LED 1 on, D for LED 2, and Z for turning both LEDs off. Also, the outputs "on" and "off" stands for turning the timer on and off.

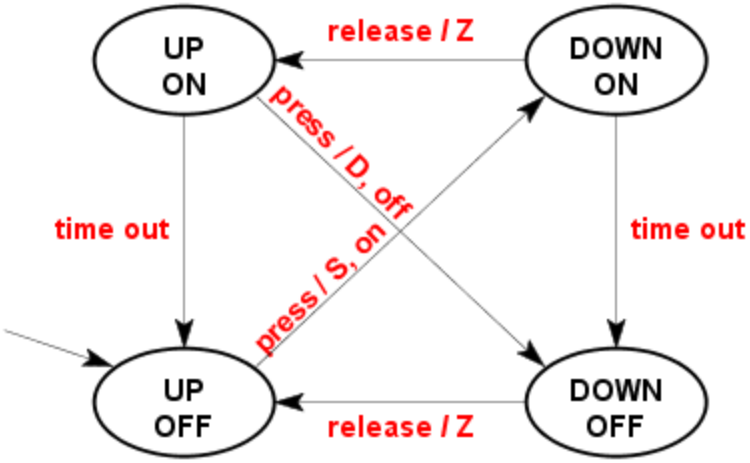


Figure: State Diagram of Mouse Double Click

	PRESS	RELEASE	TIMER (OUT)
→UP OFF	DOWN ON	-	-
DOWN OFF	-	UP OFF	-
DOWN ON	-	UP ON	DOWN OFF
UP ON	DOWN OFF	-	UP OFF

Figure: Transition Table of Mouse Double Click