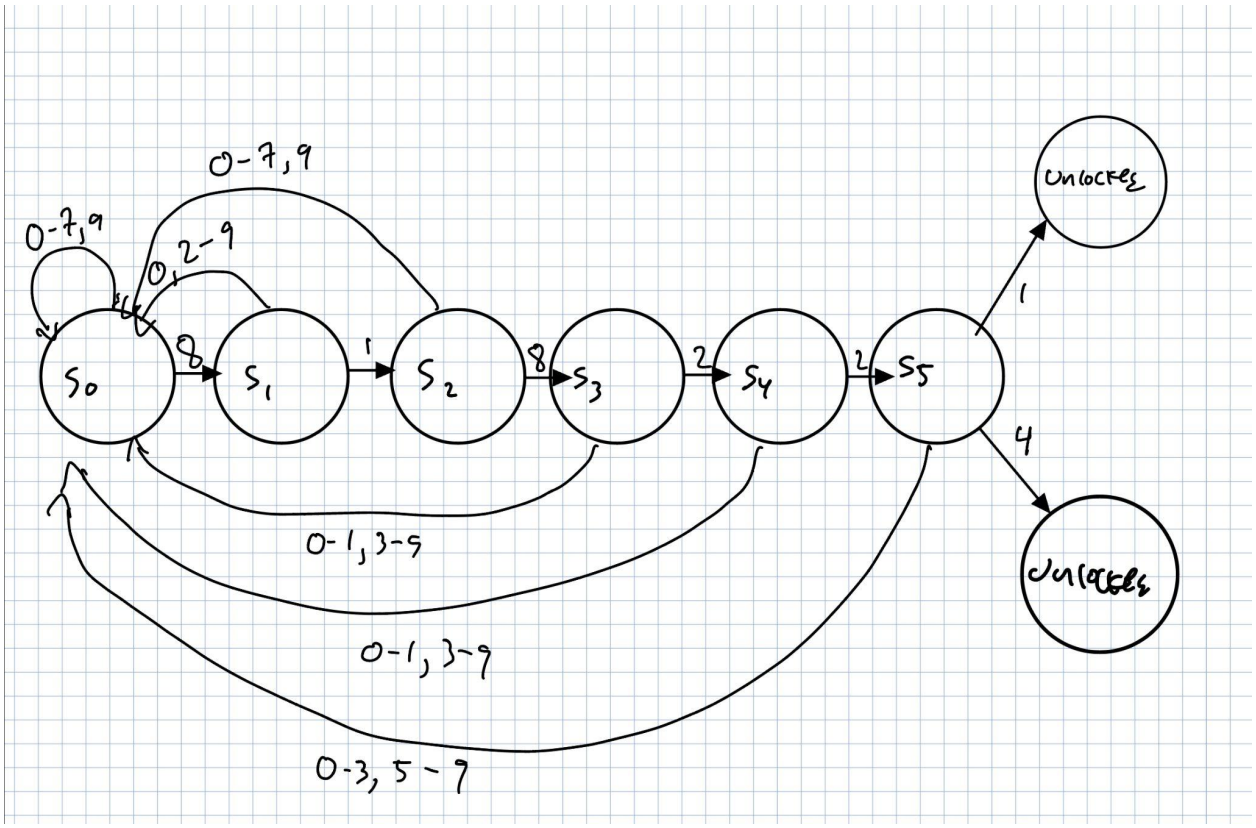
FSM Security Device

By Braulio Aguilar Islas

At a glance



The machine is implemented in python and stores its current state in a list which requires the correct sequencing of the input characters. It transitions through states as characters are added to the input string as it iterates through the input which is taken as one digit at a time, much like a real keypad which can only take one keystroke at a time. It then adds the newly added keystroke to the state and then it progresses until it has a correct sequence for unlock or lock. The code can be established at the beginning of the program in the sample FSM provided above the code is 81822 and then 1 to unlock and 4 to lock

The machine accepts digits 0-9 and anything else is ignored and does not affect the state of the machine as it is intended to simulate a keypad with no letters. The only

things that affect the state of the machine are missed digit inputs (breaking the sequence of the numbers in the key)

The Regular expression of the FA is represented by:

$$R: (‘0’ + ‘1’ + ‘2’ + ‘3’ + ‘4’ + ‘5’ + ‘6’ + ‘7’ + ‘8’ + ‘9’)^*.$$

Part 1:

The program implements the FSM by a series of steps:

Getting user’s ID

Firstly we have to configure the user’s Id number by asking the user for the input. The code checks that it is a valid string of 5 digits which represents the id. It then stores two copies of the correct key, one for locking and one for unlocking,

Asking the user for a keystroke

Now we ask the user for one keystroke at a time. Much like a real keypad the user inputs one digit at a time, anything else will be ignored if it's not a digit with len of 1. It will then progress through the states as the correct sequence its inputted. The machine stores the last6 digits to be inputted. Any wrong inputs technically resets the machine as it breaks the sequence of input preventing the array in which the inputs are stored to resemble the key array

Checking if the correct sequence was inputted

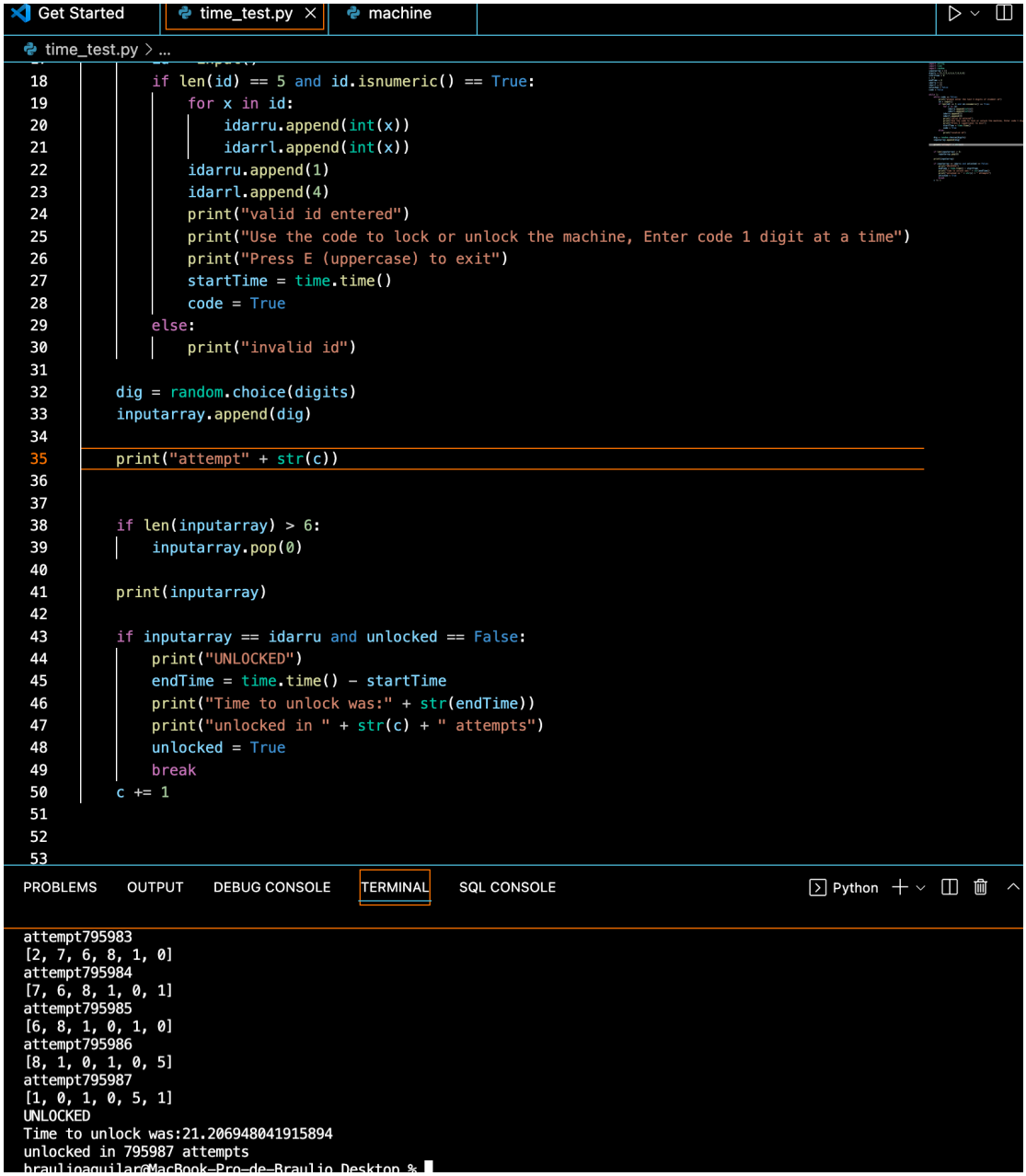
We check after each keystroke if the machine has progressed through the whole path into one of the two states. It does this by checking the input array, which stores the last 6 keystrokes and checks if it resembles any of the key arrays, it then checks the state of the lock and performs the action of locking or unlocking the door.

Part 2:

Estimating the time it will take to guess the code

In this case since our machine works by processing one keystroke at a time, I worked a guesser in time_test.py which guesses one digit from 0-9 and keeps pushing “keys” on the keypad until the machine is unlocked. Since the machine requires the guesser to guess 6 numbers in a row. We can estimate that in the worst case scenario they have a 1/1000000 chance of guessing the right sequence of digits. Which means that in the worst case scenario where they have to try all 1000000 combinations and we have to wait 1 second per attempt then it would take about 11 ½ days to guess the right code. Yet as determined by the time_test.py program the time is much shorter when we can attempt 10000 keystrokes in a second,

In the time_test.py program we program a random chooser to pick a keystroke at random and then we register how many keystrokes the attacker made and we use python's time library in order to register how much actual time it takes the program to break the security.



AS demonstrated by the picture, the time for that trial was around 21.28 seconds and there were 795987 keystrokes before security was broken.

In an average of 10 attempts this were the statistics:
Time: 15.01 seconds
Attempts: 775,549