THE CODE

The FSMD's goal is to execute instructions at a state then checking conditions to move to the next state. The goal of the complete code written is to read an xml file, executing the instructions in each state and transition through states accurately based on the condition of transition. The code given is able to read the xml file and find the number of states and it understands the instructions of each state and the conditions to leave the states. Therefore, the goal of this assignment is to write a program that executes the instructions and transitions to the next state on repeat.

The following code does this:

```
1   while cycle<iterations:
2       for transition in fsmd[state]:
3           if (evaluate_condition(transition["condition"])==True):
4               execute_instruction(transition["instruction"])
5               state=transition["nextstate"]
6               cycle += 1
```

The first line is a loop that will go on until the maximum number of cycles has been reached, the second line is another loop that will go through the transition options in the state. First, the loop will check the conditions and once it reaches one that is true it will execute the instruction and move to the next state. Once moved to the next state, the program will increment the number of cycles.

Furthermore, print statements are added to the code to show the user the progression of each cycle. The code will print out the cycle number, current and next state, the transition condition that is true, the instruction executed and the variables at that point in the cycle. The output to the console will look like the following picture.

```
Cycle: 29
Current state:  PRIME
True is true
Executing instruction: NOP
Next state: PRIME
At the end of cycle 30 execution, the status is:
var: 7
var_th: 3.5
i: 4
------------------------------------------------
```

Next is the addition of input to the FSMD. The above code will work only for one set of inputs stated in the same xml file as the states. Therefore, using the code snippet given with the assignment will read a second xml file that will set inputs at a specific cycle number.

Moreover, the second snippet can be used to identify when reaching the end state. using the second xml file and adding the name of the end state to it, the code snippet will compare it to the name of the current state to the name of the state in the second xml file and if it is the same the program will print out a “End -state reached.” Statement.

XML FILES

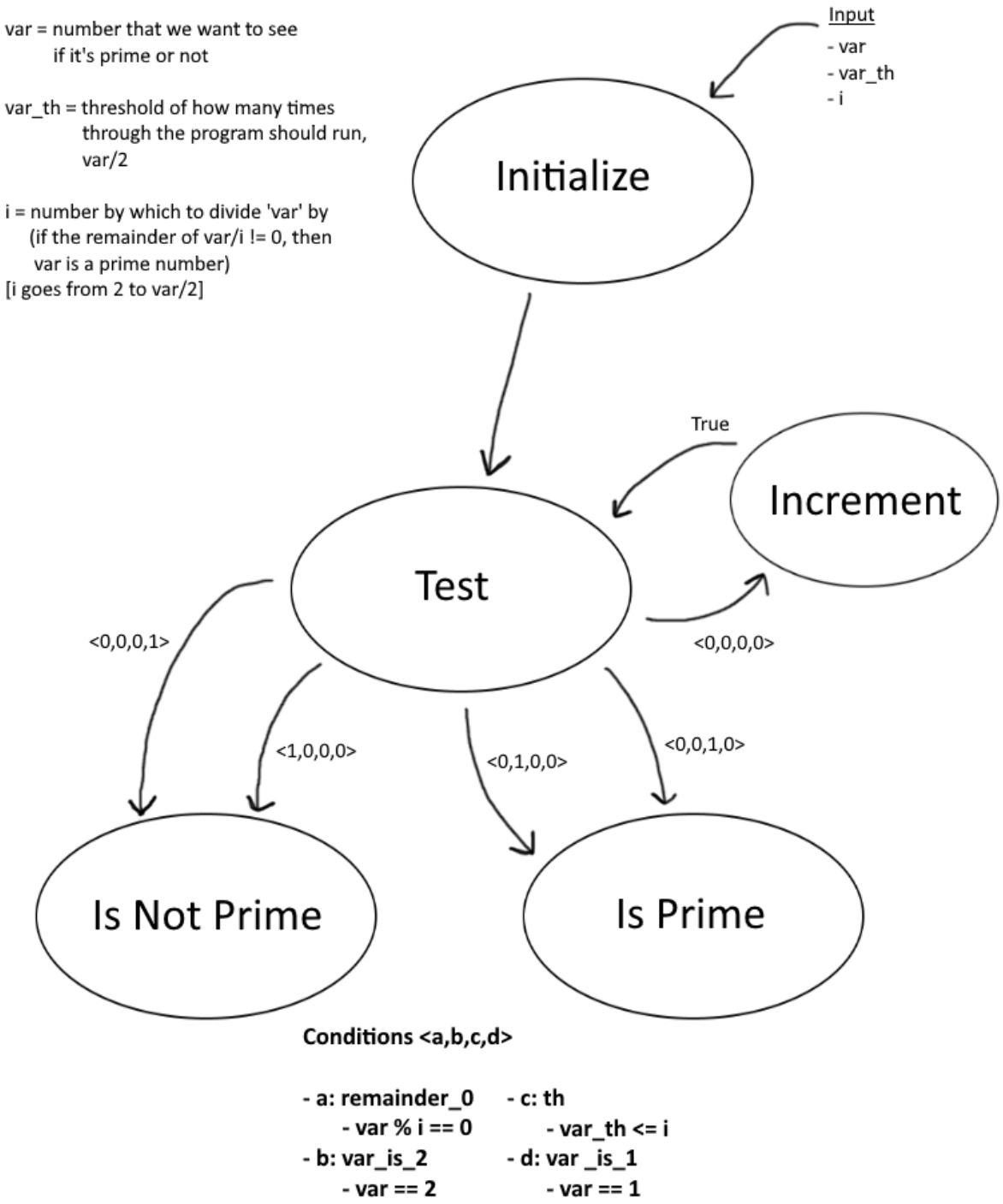
One of the ways that we can show that our machine is an FSMD is by feeding it an XML file which provides the computer with finite states to operate within.

Briefly, the XML uses it's very clear and definite encoding rules to help us to understand and define different states that are independent from each other, while being easily readable to both us humans as well as computers. The files provide *S number of states* within a statelist, *V number of variables* in a variable list, and *O number of operations* in an operations list. The FSMD model is also defined within the XML file and has conditions for each state defined together with a small description, which allow certain true/false criteria to be associated with each state. These are compiled within XML notation, and the python code has been written such that it can interpret and separate one state from another based on the layout of the XML. In this way we can write a code that takes state S as input, check which variable V is true, and based on that V and the associated true/false information, compute the true output O that comes together with the state S. Output O will always lead to a next state S, which the computer will thus continue to compute from that state.

We have tested our FSMD on three different XML files that give states for an FSMD: the two provided tests, and one self-designed test, which we call Optimus Prime, our FSMD prime number detector. This self-made test shows that our FSMD computer does indeed fulfill the requirements of an FSMD, by giving it specific states to process, and giving only one true output.

Intro to Cyber Systems - Assignment 1: Implementation of an FSMD simulator in Python
David Philip Wakeman, Mustafa Jawad El-Madani, Luca Francisco Pirovano

THE TEST



A bubble chart representing Optimus Prime, our custom-made test for checking whether a number is prime or not.

Intro to Cyber Systems - Assignment 1: Implementation of an FSMD simulator in Python
David Philip Wakeman, Mustafa Jawad El-Madani, Luca Francisco Pirovano

We made Optimus Prime with the goal of checking whether a variable “var” is a prime number or not. The input also consists of a threshold “var_th”, which is $\frac{var}{2}$ and is used to determine how many times the loop should run before determining if the number is prime or not, and a number “i”,which is the number used to divide “var” in order to check its remainder, and this number increases by 1 every time the FSMD reaches the Increment state, from 2 all the way to $\frac{var}{2}$.

After it finishes the Initialize state, it goes on to test whether the remainder of $\frac{var}{i}$ is 0 or not. If it is, then that’s because the current “i” is a factor of “var”, and that makes condition “a” to be True, therefore confirming that “var” belongs in the “Is Not Prime” state.

If not, it goes on to check the next “i”, all the way up until “i” is equal or greater than $\frac{var}{2}$, which is the largest factor of a composite number. If this still does not give a remainder of 0, then that’s because “var” is prime, and condition “c” is True. It then sets the state to be “Is Prime”.

There’s a few odd cases in which this process does not work, and we’ve made sure to address those accordingly. There are still two conditions that I’ve not mentioned that I will do so now: as this process does not work with the numbers 1 and 2, we made separate conditions to accommodate for that; if “var” is 1, then condition “d” is true and it is not prime, as 1 is neither prime nor composite, and if “var” is 2, then condition “b” is True and it is prime. It’s as simple as that.