Assignment 1: Implementation of an FSMD simulator in Python

Dušana Milinković (s194741), Emma Andreea Chirlomez (s194780) and Paulo Lima (s194729)

02135 Introduction to Cyber Systems E20
Danmarks Tekniske Universitet
September 27, 2020

1 Introduction

In this project we designed and implemented a FSMD simulator, which is able to run the given tests in the assignment and the test designed by ourselves. The designed simulator is the cycle based program, where after every step a list of certain commands or expressions are being executed.

2 Describing the machine

In the beginning of our code we are setting up the environment we want to work in, by initialising variables and then getting the end state from the stimulus file (if there is any). We also initialize a variable "stim_per_cycle" of *dict* type that will later contain all the input expressions for the respective cycles.

```python
# State stores the state the fsmd is in.
state = initial_state
# We define the max_cycle as the number of iterations.
max_cycle = iterations
# End state is the state we want to break out of.
end_state = ""
if 'fsmdstimulus' in fsmd_stim and 'endstate' in fsmd_stim['fsmdstimulus']:
    end_state = fsmd_stim['fsmdstimulus']['endstate']

# Getting the stimulus for each cycle.
stim_per_cicle = dict()
for c in range(max_cycle + 1):
    stim_per_cicle[c] = []
```

In the next part of our code we need to extract the provided information from the files of the specific test. The first thing we need to do is to check whether there is a stimulus file or not. When the stimulus file is present, we are just going to update the variable "stim_per_cycle" by adding the cycle number as the key and the expression for that cycle as a value. In the case when there is a single stimulus we are just extracting the dictionary directly, otherwise we are using a loop to extract every element from the list of dictionaries.

```python
if 'fsmdstimulus' not in fsmd_stim:
    fsmd_stim['fsmdstimulus'] = {}
if 'setinput' not in fsmd_stim['fsmdstimulus']:
    fsmd_stim['fsmdstimulus']['setinput'] = []
fsmd_stim = fsmd_stim['fsmdstimulus']['setinput']
def ExtractStim(elem):
    stim_per_cicle[int(elem['cycle'])].append(elem['expression'])
if isinstance(fsmd_stim, list):
    for elem in fsmd_stim:
        ExtractStim(elem)
else:
    ExtractStim(fsmd_stim)
```

This is the main part of our simulator. We use a **for loop** to define one of the two stop conditions of the simulator. For the other stop condition we check if the current state is equal to the end state, which is stated in the stimulus file. In each cycle we print information about the variables, inputs and states using the **PrintStateInfo** function, that we defined previously.

Now we iterate through the available transitions, defined in the description file, and match it with the correct transition of the state that we are in. There are two possible scenarios; when we have only one possible transition and then we have multiple of them. In case we have only one possible transition we simply execute the corresponding instruction. On the other hand, when we have multiple possible transitions we must evaluate the condition of each one of them, and only execute the instruction if the condition evaluates to **True**. After executing the instruction we update the variable **state** with the next state defined in the transition.

```
for cycle in range(max_cycle):
    for stim in stim_per_cicle[cycle]:
        execute_setinput(stim)

    PrintStateInfo()

    # If state = end_state, we need to exit.
    if state == end_state:
        print(f"Exited because the current state ({state}) is the End State!")
        break

    # Iterating through the transitions available to find a matching one.
    for transition in fsmd[state]:
        # This is the valid transition
        if evaluate_condition(transition['condition']):
            PrintTransitionInfo(transition)
            execute_instruction(transition['instruction'])
            state = transition['nextstate']
            # We changed state so we need to get out of the loop
            # to avoid undefined behaviour.
            break
    else:
        # If we get here it means that no condition matched.
        # This state is illegal.
        print("Unexpected state!")
        exit()
```

3 Test 3

We created a test which determines if a given number is a prime or not. In order to do so, we defined the set of rules which our simulator must follow in the description file. We created 2 files corresponding to the test, one representing the description of our FSMD and the other one the stimulus file.

In this test we have a boolean flag storing whether the number P is prime or not and an integer variable x iterating from 2 to $\sqrt{P}$. If the variable P is divisible by x, we simply enable the boolean flag described above.

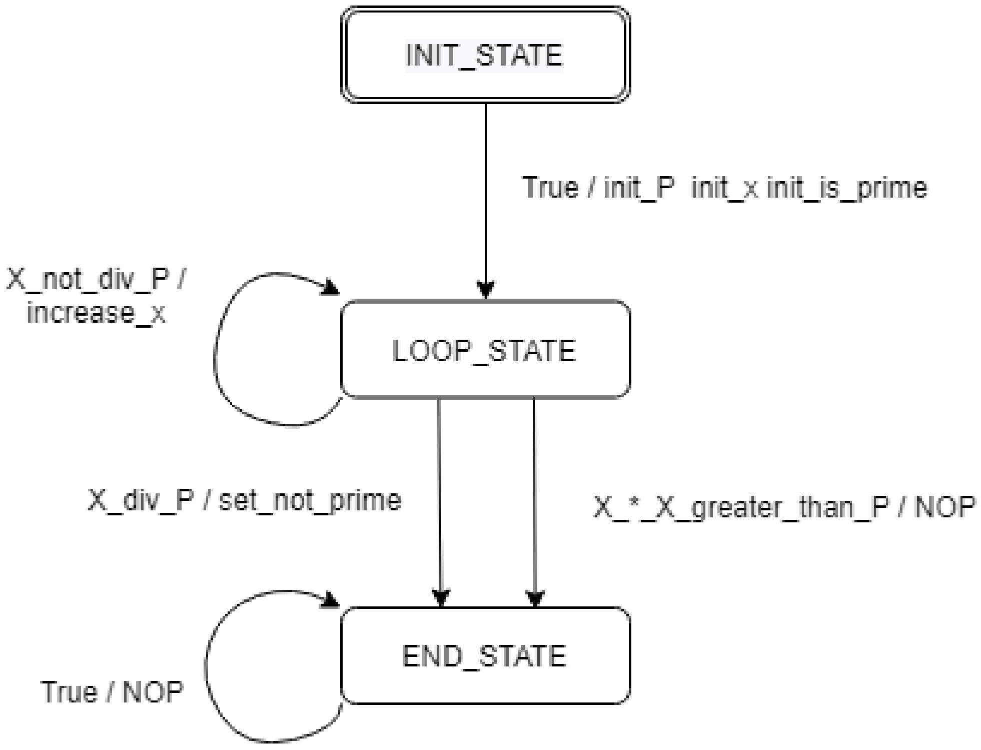

Figure 1: FSMD diagram for test 3

Our FSMD is composed of 3 states and works on 3 variables, "init_P", "init_x" and "init_is_prime". After every cycle x is incremented by 1 and the condition is checked again. This process is repeated until either we find the number x that divides P or until the condition "X_*_X_greater_than_P" is fulfilled (Trial division method). More details about the test (variables, states, conditions, etc.) can be found in the attached file named "TEST3.md".