50.051 Programming Language Concepts

W5-S3 Practical on FSMs and RegEx

Matthieu De Mari

Practice Activities

- In today's session, we have prepared a few practice activities for today, regarding FSMs and RegEx.
- Some of these activities' answers will be discussed together in class. *(But probably not all of them!)*
- We advise to take them in order.

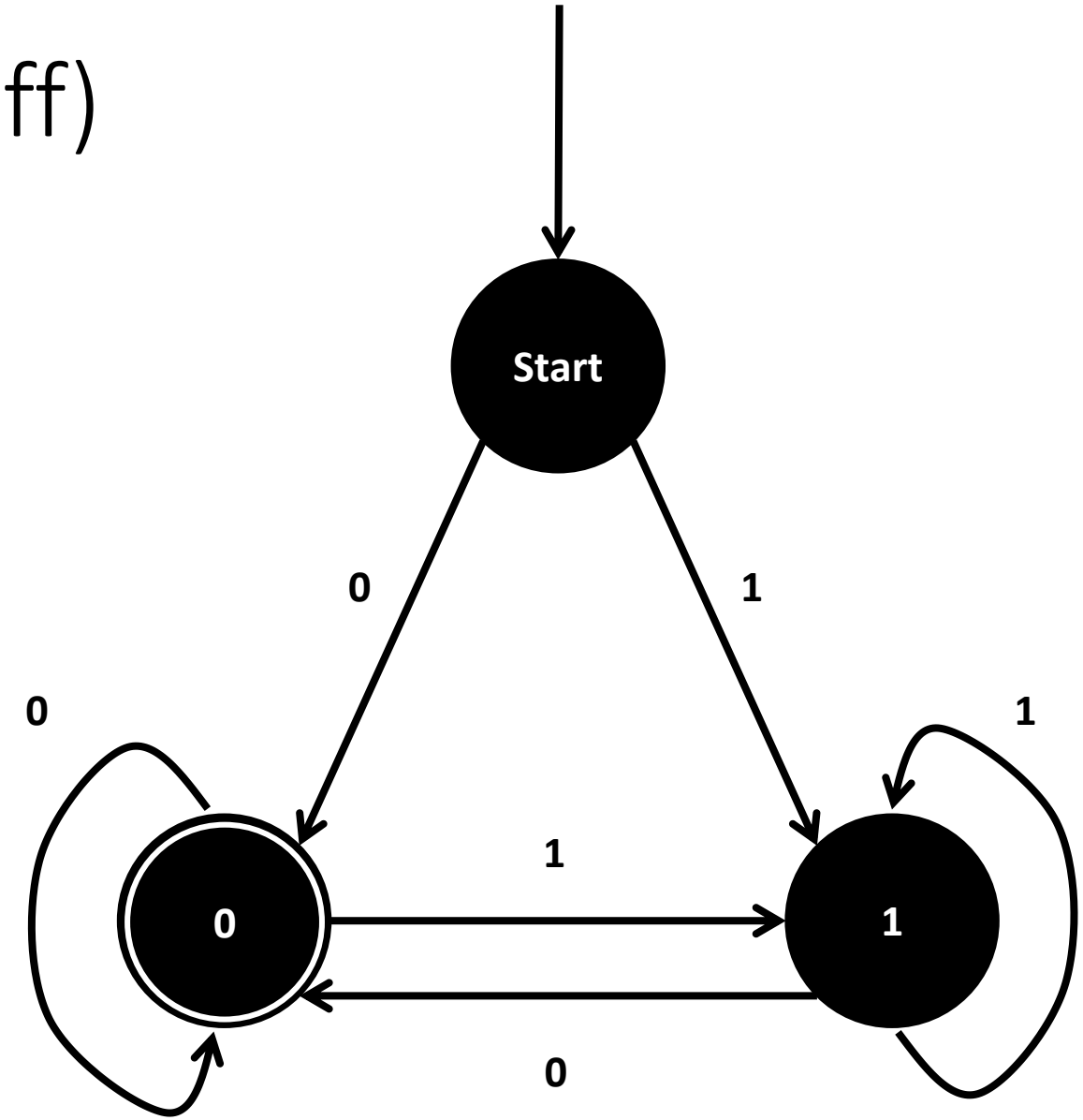
- **Some of these practice activities (i.e. activities 1, 2 and 3) require a checkoff, call the instructor/TA when <u>all your solutions are ready.</u>**
- Once done, additional practice that is not part of the checkoff.

Practice 1 (part of checkoff)

Question 1: How would you modify this FSM with accepting states, so that it considers as acceptable inputs any **string *x* of 0 and 1, that have an even number of zeroes?**

Question 2: Show a C code that runs the FSM you have produced for Q1. You may reuse the template from the W5S1 lecture.

Question 3: Can this pattern be described by using a RegEx?

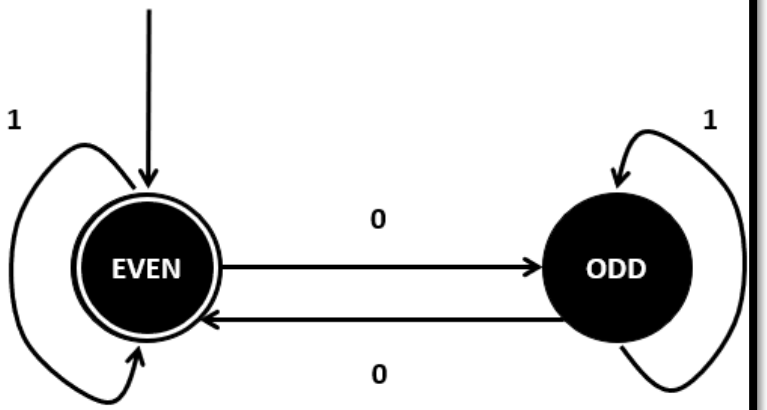


Practice 1 (solution)

Question 1: The answer was already given in the W5S1 slides on Monday.

Practice 4

Consider the FSM with a single stopping state, that considers as acceptable inputs **any string *x* of 0 and 1, that have an even number of zeroes.**



Practice 1 (solution)

Question 2: Should be fairly easy by reusing the W5S1 code template.

Question 3: Lots of possible RegEx can be proposed here.
The most common one was ^(1*|1*01*01*)*$.

Practice 2 (part of checkoff)

Consider a vending machine and draw its **FSM with outputs**.

- States will represent the current balance (in terms of money that has been inserted so far).
- This FSM will have three possible actions/inputs.
  - **“A0.5”:** insert a 50 cents coin,
  - **“A1”:** insert a 1 dollar coin,
  - **“PB”:** press the machine button.

It also has four possible outputs:

- **“B0.5”:** give back a 50 cent coin to the user,
- **“B1”:** give back a 1 dollar coin to the user,
- **“C”:** give a chocolate bar to the user,
- **“N”:** do nothing.

Practice 2 (part of checkoff)

We would like to define a vending machine that has the following logic.

- **Rule 1:** Whenever a coin is inserted by the user, the total balance is updated (can be represented as a state of some sort?).
- **Rule 2:** If the user has inserted 1.5 dollars in total and presses the button, a bar will be given to the user, and the balance will go to 0.
- **Rule 3:** If the user presses the button but the balance is not exactly 1.5 dollars, nothing happens.
- **Rule 4:** If the user inserts a coin and the new balance exceeds (strictly) the maximal allowed balance of 1.5 dollars, then the machine will return the last coin the user has inserted.

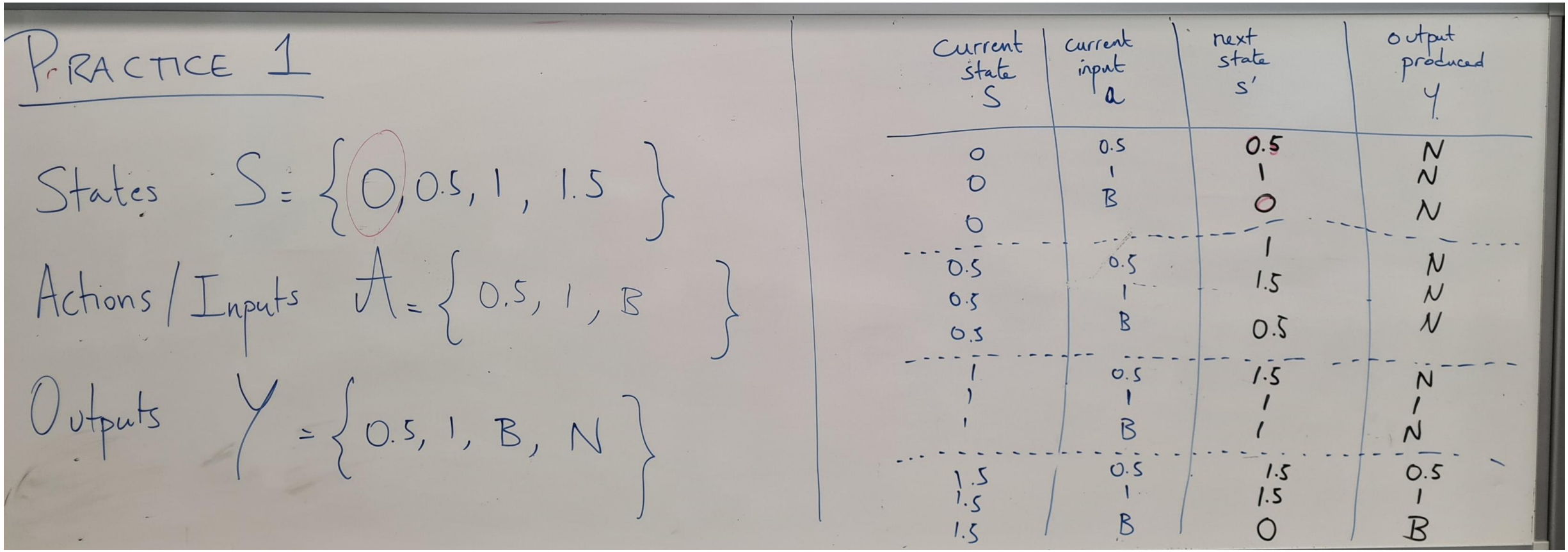
Practice 2 (part of checkoff)

Question 1: What could the possible states for this FSM be?

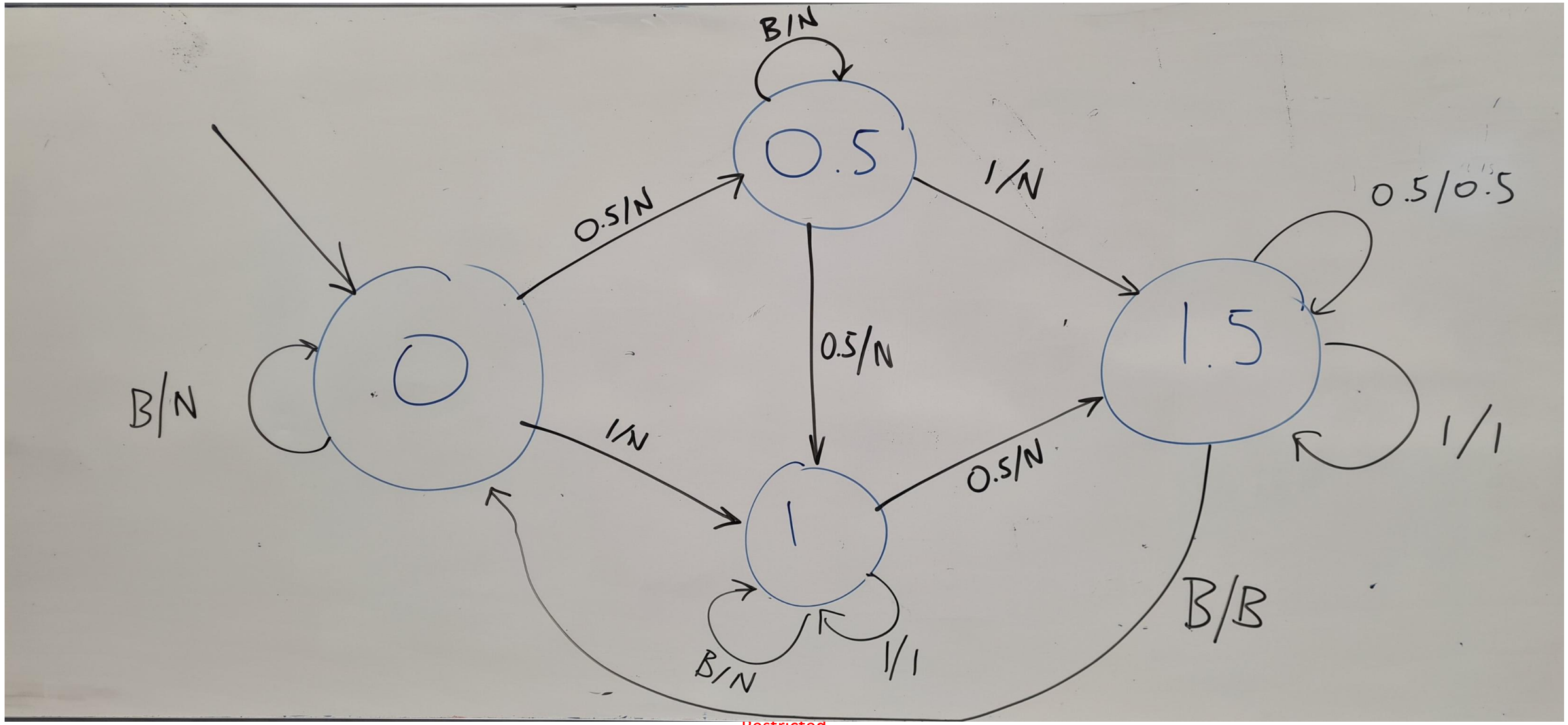
Question 2: Draw the state diagram for this FSM.

Question 3: What if we wanted to break rule 4 and asked for the machine to keep track of any amount of money we would have inserted. The machine would then return the change only after the button has been pressed.
Could this be represented with a simple FSM as before?
If yes, show the FSM in question.
If not, briefly explain why.

Practice 2 (solution)



Practice 2 (solution)



Practice 2 (solution)

Question 3: No, because it would require an infinite number of states to do so, and this cannot be done with a finite state machine.

Practice 3 (part of checkoff)

Question 1: What is the RegEx describing a **valid int number**?

It should accept “0”, “123”, “-154”, “+2”, etc.
It should not accept “0127”.

(Note: we will not care about upper-lower limits for these int/float that are due to memory sizes.)

Question 2: What is the RegEx describing a **valid float number** (in Python)?

It should accept “0”, “123”, “-154”, “12.7”, “+0.5”, “5.000”, “.5”, etc.

It should not accept “01.07”, “--0.2”, or “1.7.2”

(Note: for both questions, ignore exponential notations, i.e. “5.6e3”.)

Practice 3 (part of checkoff)

Question 1: Again, many possible answers here.

The most common one was ^[+-]?(0|[1-9]\d*)$.

Question 2: Again, many possible answers here.

The most common one was ^[+-]?(0|[1-9]\d*)(\.\d+)?$.

It might have to be modified into ^[+-]?(0|[1-9]\d)?(\.\d+)?$ to accept numbers such as ".57".*

You are done with the checkoff activities, congrats

But there are more tasks for you to practice with.

Practice 4

We would like to write an FSM with stopping states and no outputs.

It will take as inputs, strings *x* consisting of combinations of four characters: Z, A, and M.

Possible inputs include “MAZ”, “AMAZ”, “ZZZZAM”, etc.

The only acceptable input should be exactly “ZAMZAM”.

The string “ZAMZAMZAMZAM” is not valid.

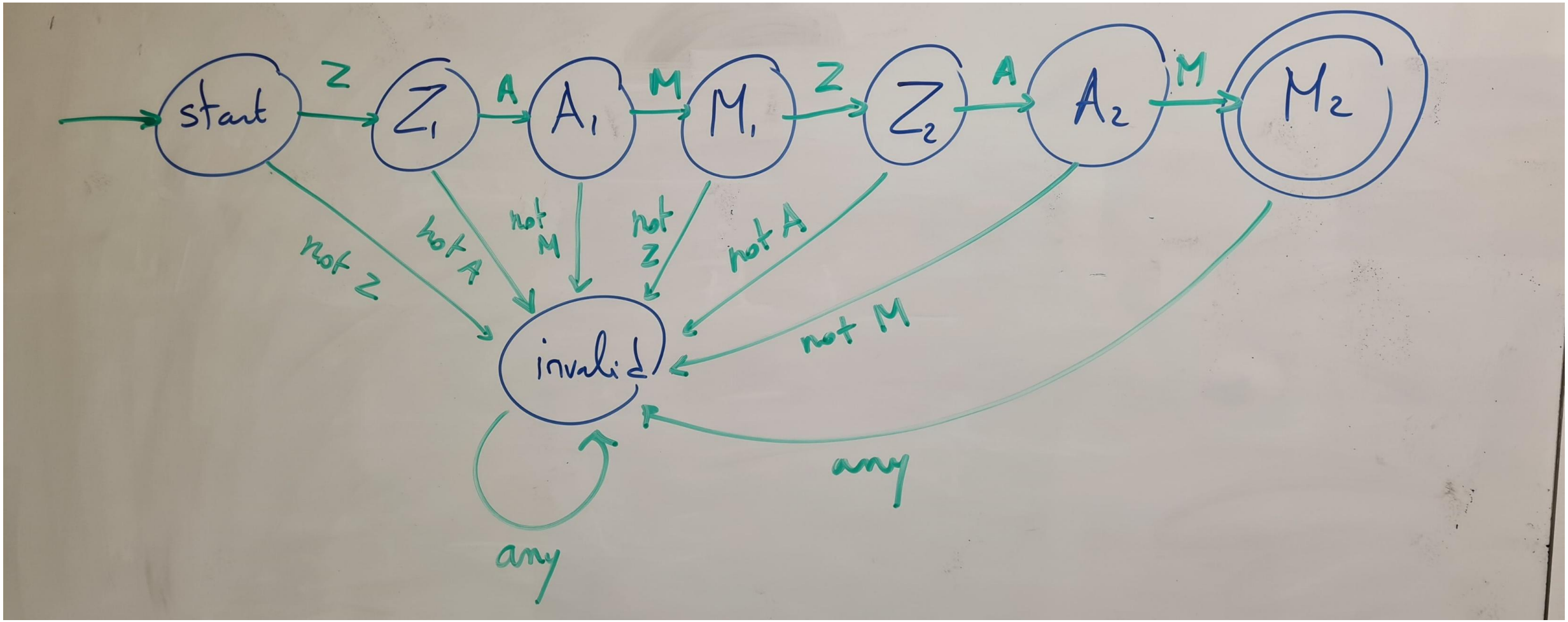
Practice 4

Consider a FSM state diagram, which:

- Has 8 States (Start, Z1, A1, M1, Z2, A2, M2, Invalid),
- Has 3 possible Actions (Z, A, M),
- Has the Start state defined as the starting state,
- Has only one stopping/accepting state,
- Has the FSM stop in this stopping/accepting state, if and only if *x* is exactly “ZAMZAM”; otherwise, it stops in another state.

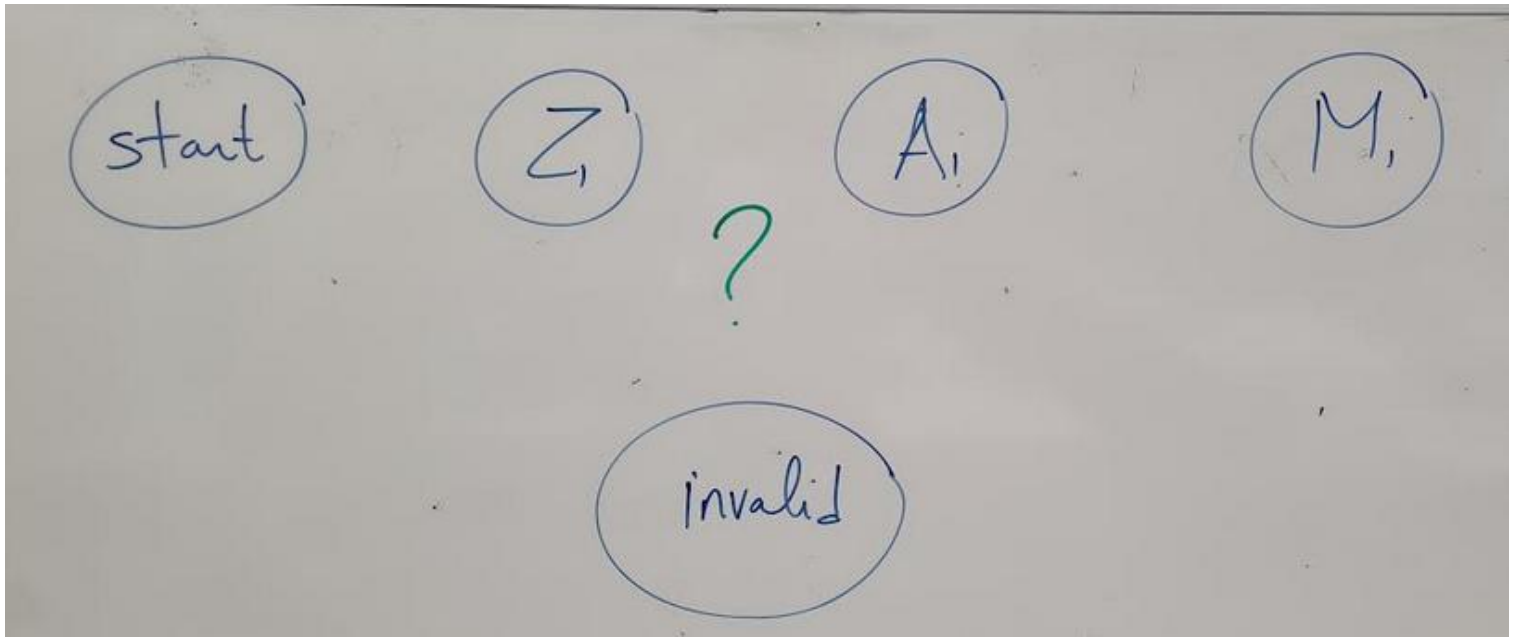
This FSM is shown in the next slide.

Practice 4



Practice 4

Question: What would be the minimal number of states for this FSM? Do we really need 8 states (Z1, A1, M1, Z2, A2, M2, start and invalid) as shown in the previous slide? Or can it be done with less states, for instance 5 states (Z, A, M + start and invalid) by adding some nicely chosen transitions to the state diagram below?

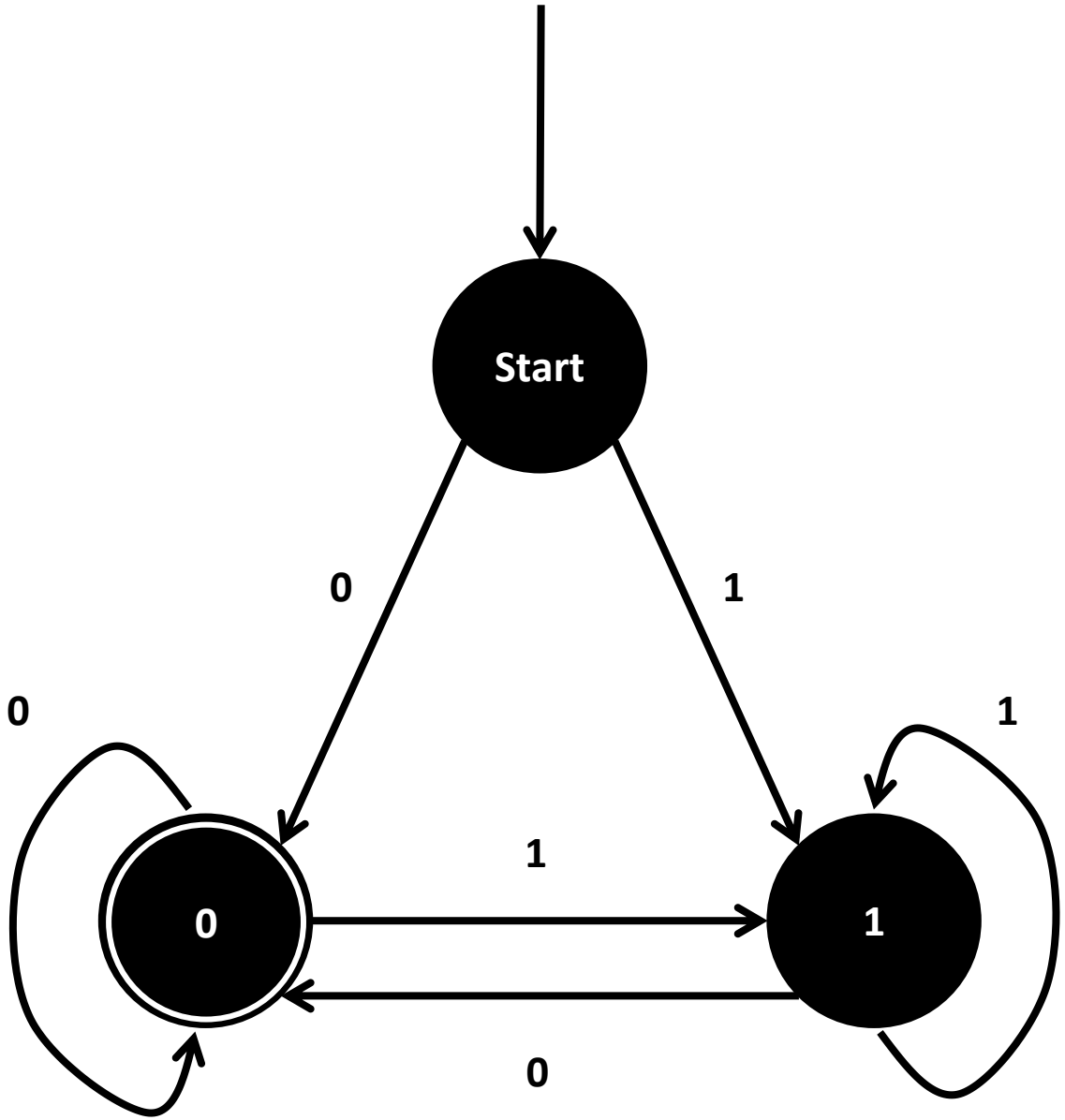


Practice 5

We are now interested to modify this FSM with stopping states, so that it considers as acceptable inputs **any string *x* of 0 and 1, that have the same number of zeroes and ones.**

Question 1: Can this be described using an FSM of some sort?

Question 2: Can this be described as a RegEx of some sort?



Practice 6

Question 1: Design an FSM that detects whether a binary input string *x* consists of an alternating bit pattern (e.g., "01010101" or "10101010").

The FSM should end in an accepting state if and only if the entire input string follows an alternating pattern.

As possible test cases, note that “0”, “1”, “101”, “01010”, “10”, and “0101” are valid according to this pattern.

Question 2: Can this pattern be written as a RegEx?

Practice 7

Q1: What is the RegEx for checking that a string corresponds to a valid **int/float number written in exp form**? (e.g. it should accept 1e5, +4.1e7, -2.3e4 but not 3.5e2.4).

Q2: What is the RegEx for checking that a string corresponds to a valid **Singaporean phone number?** (e.g. it should accept “63036600”, “+6563036600”, “006563036600”, etc.)

Q3: What is the RegEx for checking that a string corresponds to a valid **email address?** (e.g. it could be name_stuff@something.something, myname.thing@sutd.edu.sg, etc.)

Remember to use full string matching for these tasks!

Practice 8

Consider input strings that are combinations of 0 and 1.
A string should be considered acceptable **if and only if it does not contain two successive zeroes**.

Question 1: Draw an FSM with accepting states for this task.

Question 2: Can the pattern be described as a RegEx? If so, what is it?

Some additional practice

From last year’s homeworks, already mentioned in W4S2.

And as extra, to keep you busy…

For each of the previous activities involving FSMs:

- Code said FSM in C.
- Most FSM problems can be resolved using an FSM with accepting states, or by using an FSM with outputs. Try implementing both versions.

In the case of RegEx:

- Code the RegEx using the W5S2 template and then check that it has the correct behavior.
- If you RegEx uses advanced notations (such as ? or + or [] or \d or \w, etc.), rewrite said regex using only the three fundamental operations (concatenation, choice and Kleene * symbol)

Final boss: Let us consider strings consisting of 0s and 1s only.

We would like to check for strings that have the same digit in their first, third, fifth, etc. location. Have a look at the table below for some examples of acceptable and not acceptable strings.

Question: Your friend Chris claims that it is impossible to describe this pattern using a RegEx. Do you agree with him, or can you provide a RegEx that works for this task?

String	Acceptable?
1	Yes
101	Yes
01000101	Yes
100110	No
101011100	No