50.051 Programming Language Concepts

W4-S3 Some more practice on FSMs and RegEx

Matthieu De Mari

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

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.
- More practice on FSMs and RegEx will follow in Lab 2 next week!

Practice 1

Consider a vending machine and describe it as a FSM with outputs.

States will represent the current balance (in terms of money).

It takes three possible actions.

- “0.5”: insert a 50 cents coin,
- “1”: insert a 1 dollar coin,
- “B”: press the machine button.

It also has four possible outputs:

- “0.5”: give back a 50 cent coin to the user,
- “1”: give back a 1 dollar coin to the user,
- “B”: give a chocolate bar to the user,
- “N”: do nothing.

Practice 1

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

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

Question: What could the possible states for this FSM be? Draw a state diagram for this FSM.

Practice 2

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.

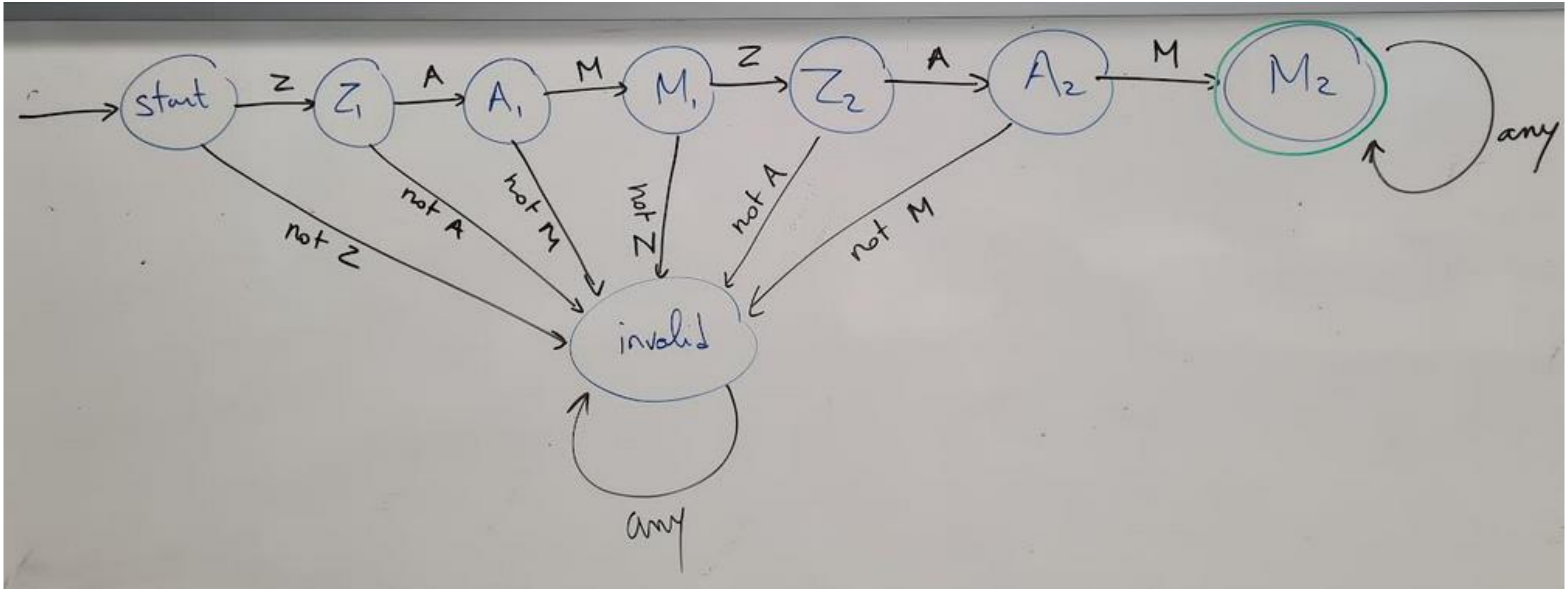
Practice 2

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 state,
- Has the FSM stop in this state, if and only *x* is exactly "ZAMZAM"; otherwise, it stops in another state.

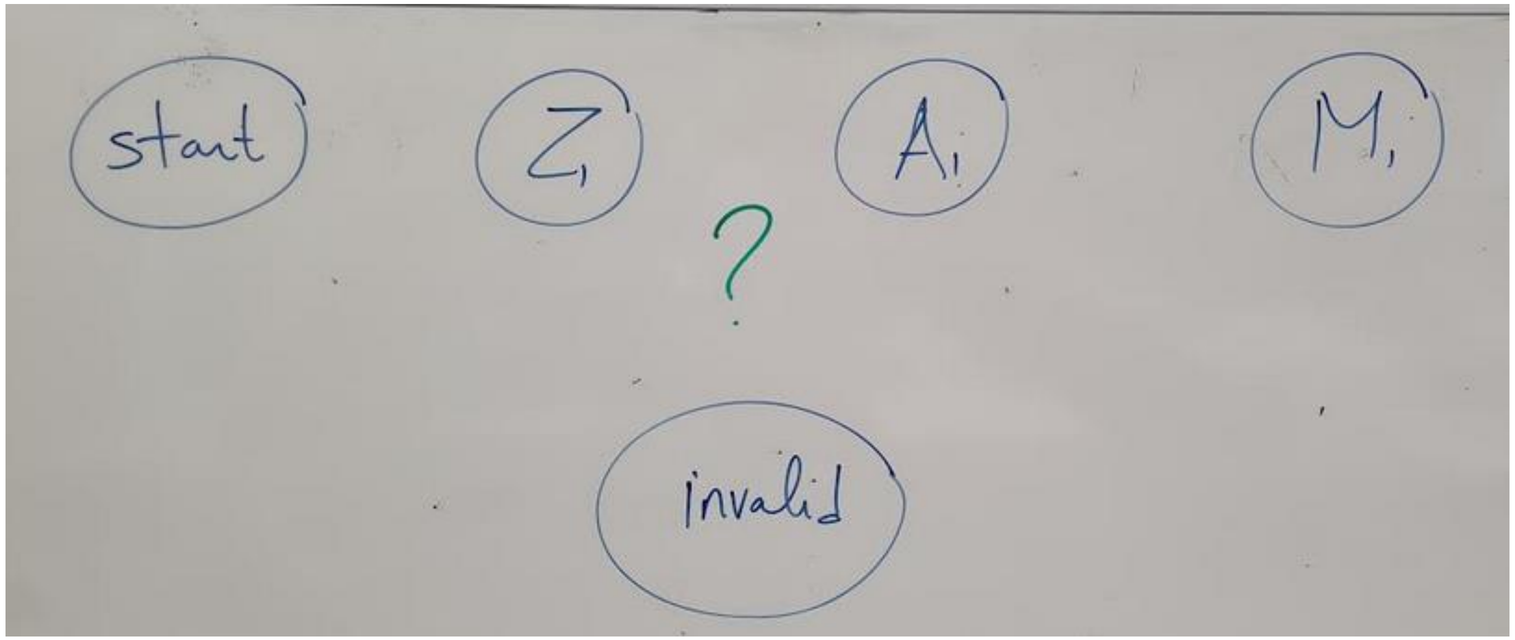
This FSM is shown in the next slide.

Practice 2


start
Z
Z1
A
A1
M
M1
Z
Z2
A
A2
M
M2
any
not Z
not A
not M
not Z
not A
not M
invalid
any

Practice 2

Question: What would be the minimal number of states for this FSM? Do we really need 8 states (Z1, A1, M1, Z2, A2, M2 and maybe others like start and invalid) as shown in the previous slide? Or can it be done with 3 states (Z, A, M + start and invalid) by adding some nicely chosen transitions to the state diagram below?



Practice 3

We would like to write an FSM with stopping states and no outputs that will take strings *x* consisting of combinations of four characters: Z, A, and M.

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

Draw a FSM state diagram, which:

- Has possible States, which you are free to decide,
- Has 3 possible Actions (Z, A, M),
- Has the Start state defined as the starting state,
- Has one stopping state,
- Has the FSM stop in this state, if and only *x* **contains** the string “ZAM”; otherwise, it stops in another state.

Practice 4

Can the FSMs in Practice 2 and 3 be described as RegExs?

If so, what would be a valid RegEx for these FSMs?

Can you implement them in C using the regex library?

Practice 5

Q1: What is the RegEx describing a **valid int number**?

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

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

Q2: 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” or “1.7.2”

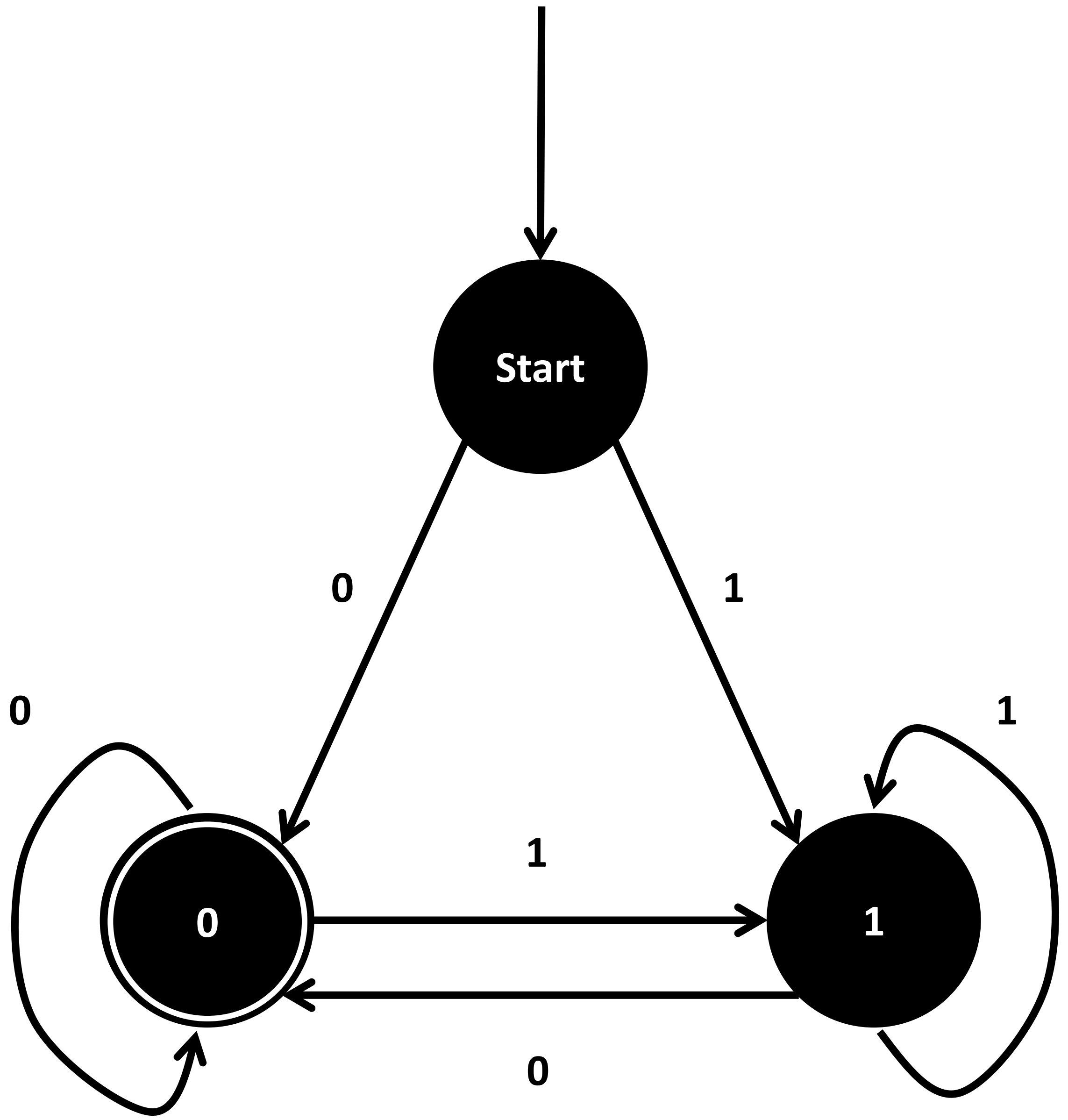
Q3: How about **int/float numbers written in exp form**? (e.g. 1e5).

Remember to use full string matching for these tasks!

Practice 6

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

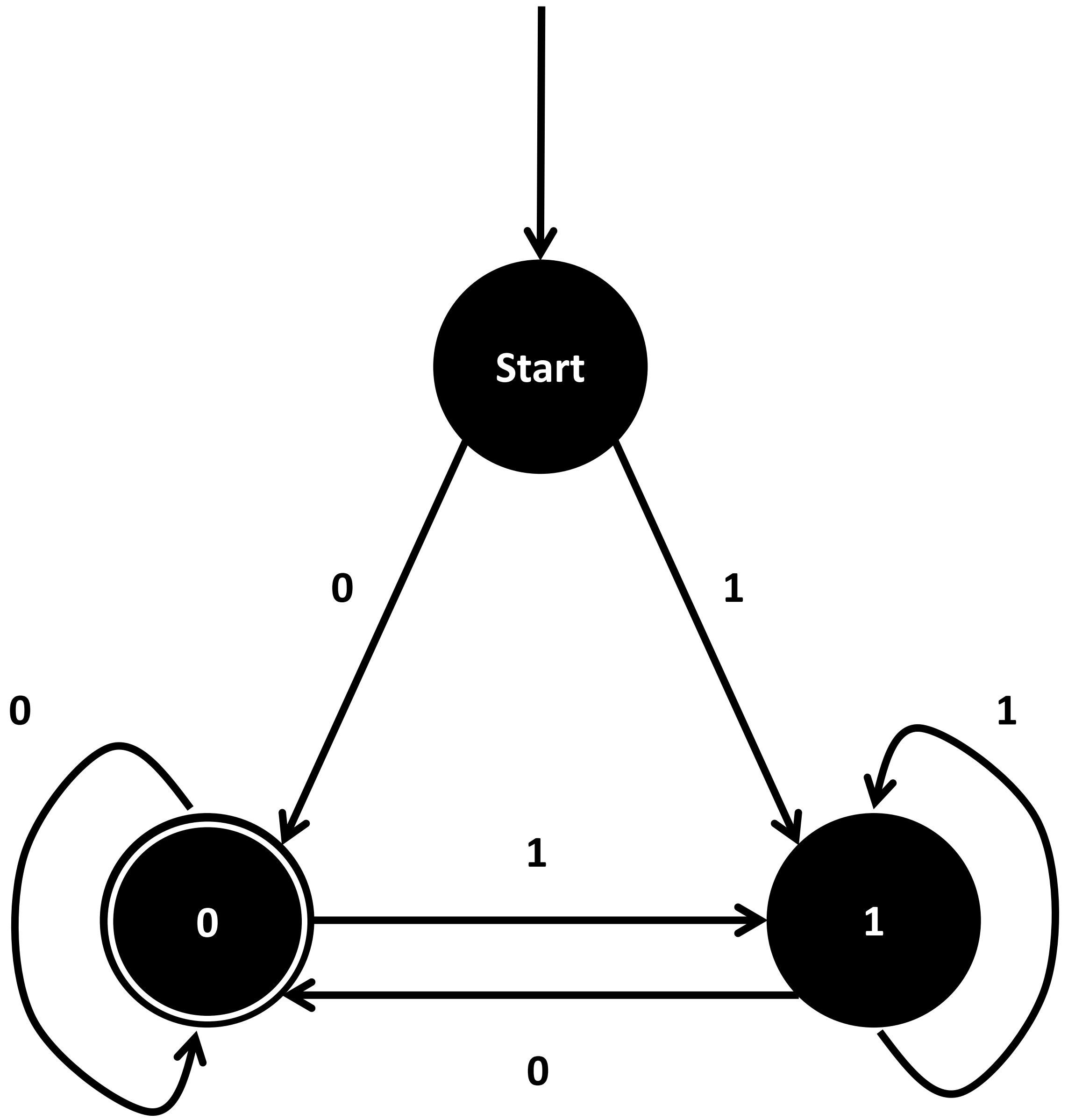
Question: Can this pattern be described by using a RegEx of some sort?



Practice 7

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



Practice 8

Q1: 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.

Keep in mind that “0”, “1”, “101”, “01010”, “10”, and “0101” are valid according to this pattern.

Q2: Can this pattern be written as a RegEx?

Practice 9

Let us go back to the Practice 1 – Vending Machine example.

Question: What if we wanted for the machine to keep track of any amount of money we would have inserted and return the change only after the button has been pressed?

Could this be represented with a simple FSM?

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 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 and verify 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 *)

You are working for a survey company in Singapore. You will be sending forms to a vast quantity of SG-based users, asking for their names and phone numbers, along with many survey questions regarding various topics. Every survey form will require the participants to enter their phone number, as a string of digits.

Some algorithm will be used to remove all the whitespaces in the string entered by the user for you, and you should expect one of the three string formats below after the string has been cleaned. After cleaning the strings will look like “63036600”, “+6563036600”, or “006563036600”. Possibly with other digits than those shown.

Question: Can you write a RegEx for checking if the phone number consists of eight digit and maybe a country code +65 or 0065 that might have been added to the phone number?

- We expect the users to enter only digits, no phone numbers with more than 8 digits (unless country codes are used), and no country codes other than +65 or 0065.
- A phone number not using 8 digits (before a country code is added) is invalid.
- Any other country code than +65 or 0065 should be rejected.
- We shall only check for valid characters: it is ok if our RegEx does not catch a phone number +6500000000, which is obviously fake.

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