Exercise 5 - Solutions

1. (a) All strings over the alphabet $\Sigma = \{a, b\}$ that

- start with 0 or more letters "a",
- continue with 1 or more letters "b",
- followed by exactly 1 "a".

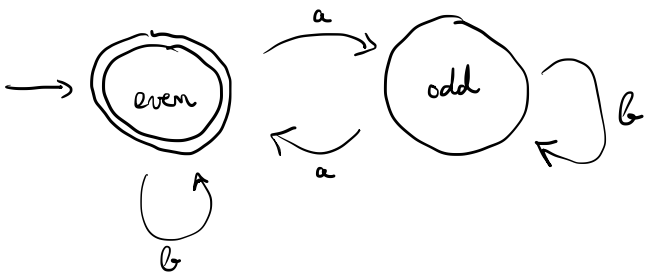
Remark: Regex is `a*b+a` (classical syntax)
`^a*b+a$` (Python regex)

(b) Any binary string that ends in "101".

Remark: Regex is `[01]*101` (classical syntax)
`^[01]*101$` (Python regex)

Why?
- Reading "101" from any state puts the FSA into the accepting state S_3
- There is no other way to reach the accepting state S_3
- The FSA cannot "get stuck"

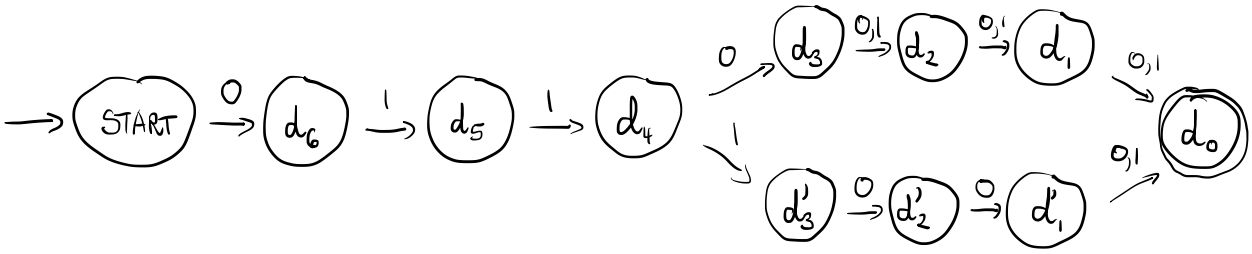
2.



3. Number digits in ASCII are 0x30 to 0x39, in 7-bit binary 011 0000 to 011 1001, so:

- start with 011
- if next bit is 0, it's a number digit for sure
- if next bit is 1, it's a number digit only iff followed by two 0.

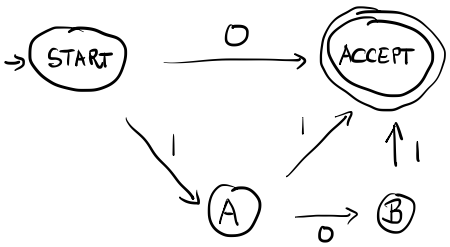
FSA:



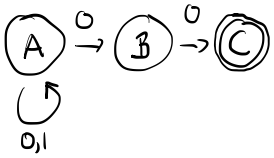
(We take the convention that if the machine get "stuck", we reject the input string.)

4. (a) $0 \mid (11) \mid (101)$ alternatively: $0 \mid 1(1 \mid 01)$

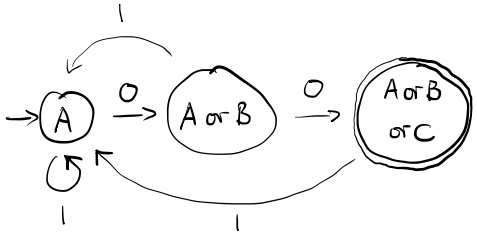
FSA:



(b) $[01]*00$ FSA:

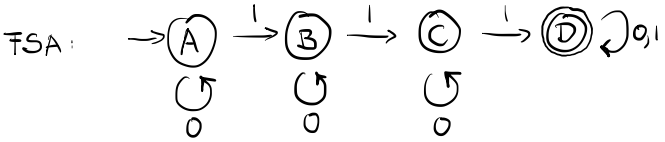


alternative deterministic version:



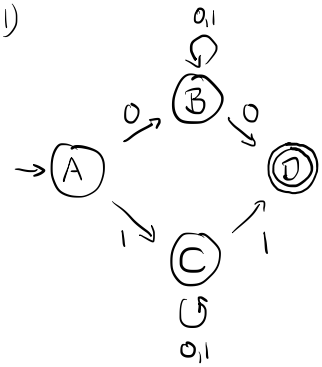
(c) $0* \mid 0* \mid 0* \mid [01]*$

FSA:

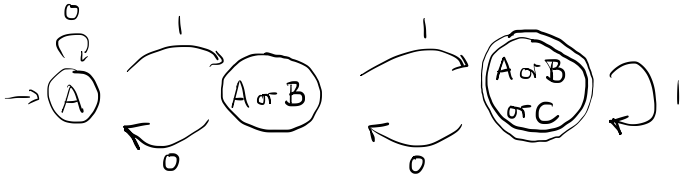


(d) $(0[01]*0) \mid (1[01]*1)$

FSA:



5. (a)



(b) $0* \mid (0+1 \mid 1+0)* 1+$, $0* \mid (0+1)* \mid (0\,(0+1)*1)*1*$ is also a solution

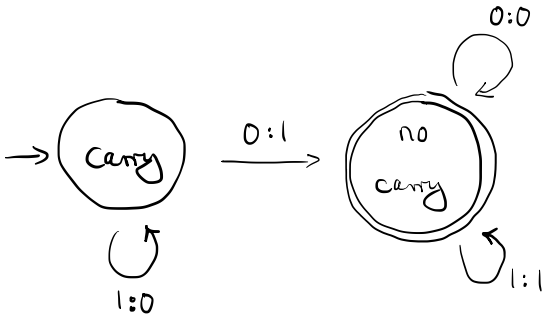
6. (a) The transducer outputs a zero first, then outputs the previous input.

⇒ Can be seen as a shift of the input sequence

(b) If a binary number is input left-to-right, it performs a binary division by 2 (the remainder is in the final state of the transducer: A=0, B=1)

If a binary number is input right-to-left, it performs binary multiplication by 2. In this case, it is necessary to pad the number with a leading 0 to get the most significant digit out of the transducer.

7. Feed the digits of the binary number right-to-left into the following transducer:



Note: If the final state is "carry", this signals an overflow.
It can be avoided by padding the number with a leading 0.