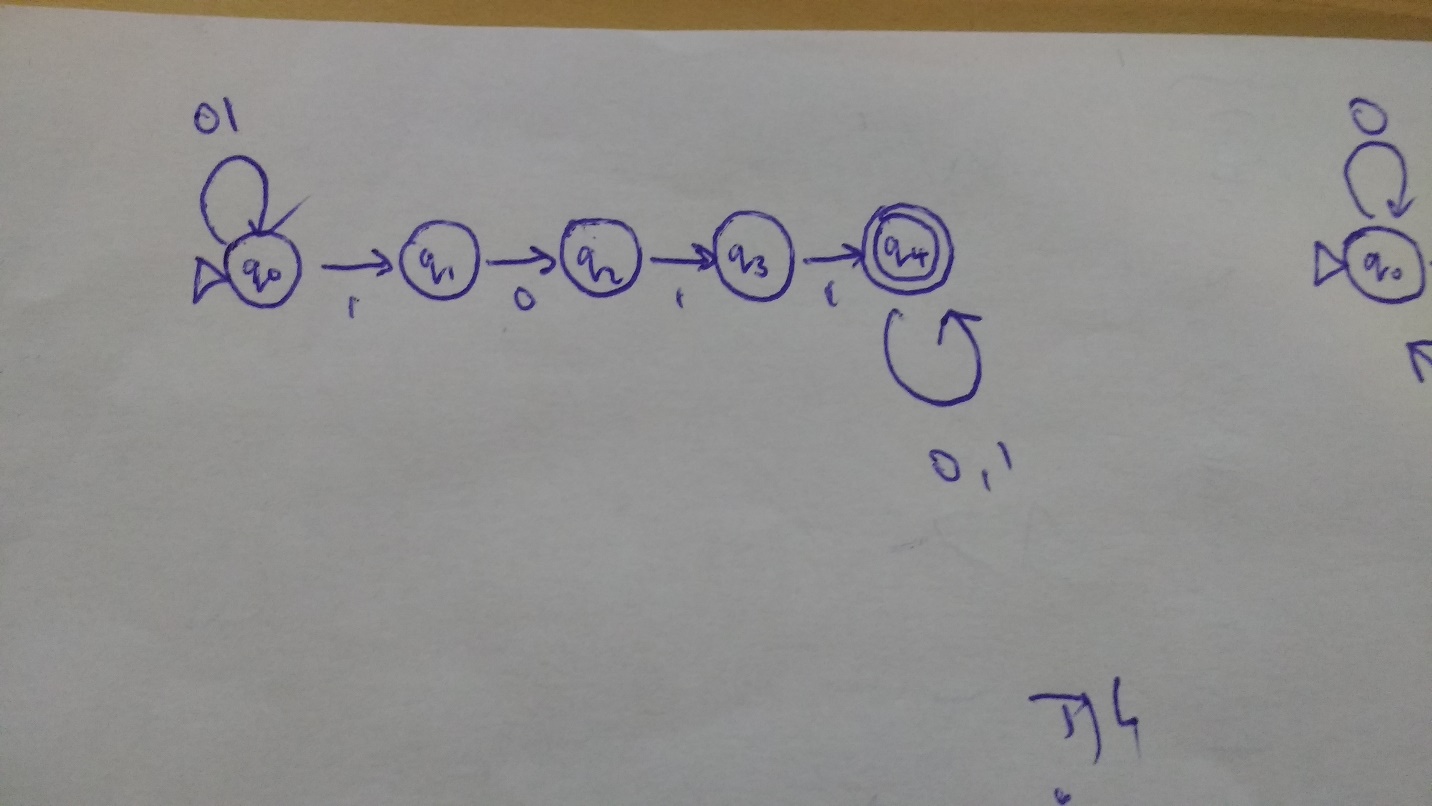
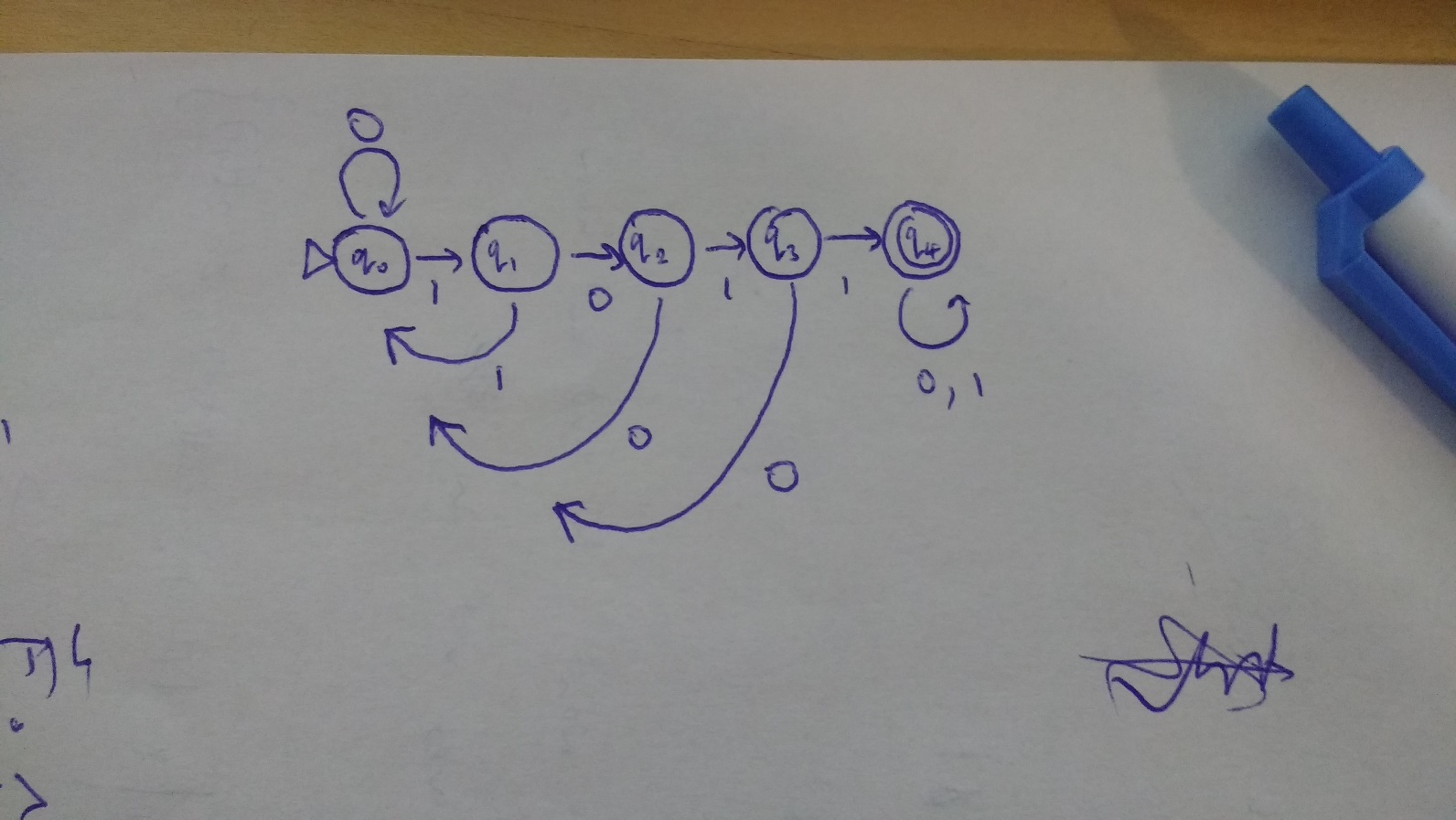
FOCS Homework 6 – Leon Lam

1.



2.

|  |  |  |
| --- | --- | --- |
| q0 | 0 | q0 |
| q0 | 1 | q1 |
| q1 | 0 | q2 |
| q1 | 1 | q0 |
| q2 | 0 | q0 |
| q2 | 1 | q3 |
| q3 | 0 | q0 |
| q3 | 1 | q4 |
| q4 | 0,1 | q4 |

**3**.

The machine would be able to handle complex input (over its larger language) roughly log2n (rounded up) times faster (meaning less steps required) than a binary machine. For example a machine with language {a,b,c,d} would be able to recognize a letter in a string in one step, but a binary machine would require the input (e.g. abbad) to be processed into binary (0001010011, given a = 00, b = 01, c = 10 and d = 11) and then processed bit by bit. Effectively, a larger language means the amount of information examined at each state transition is higher. The reverse (doing binary in a machine with language {a,b,c,d}) will take the same number of steps since you can let 0 = a and 1 = b.

In addition, a machine with a larger language would be able to move to more than 2 locations at each step without becoming nondeterministic. This means more choice: instead of building some sort of nested if-else structure where each state can lead to only 2 states, constructing a switch with n subsequent states is now possible.

**4**.

I feel like we could just flip the machine around, reversing all the arrows and state transitions? Would that work?

A little more formally: a nondeterministic machine that recognizes a certain substring could use the same structure as my answer to question 1. The beginning and the end (the parts we don’t care about) just loop on their own, and if I wanted to recognize a string containing 1101 instead of 1011 I could just make q4 the input and q0 the accepting state. I believe this will hold when expanded to full strings, since every string is the substring of (null) (string) (null). Thus if we have a certain nondeterministic machine that recognizes one string, flipping it entirely around should enable it to recognize the string’s reverse.

**6**.

0\*10\*. This means (zero to infinity 0s, but no 1s) + (1) + (zero to infinity 0s, but no 1s).

Matching all and only those strings which contain two 1s would be something like 0\*10\*10\*, I think. There can be zeroes anywhere in the string, but there must be two ones.