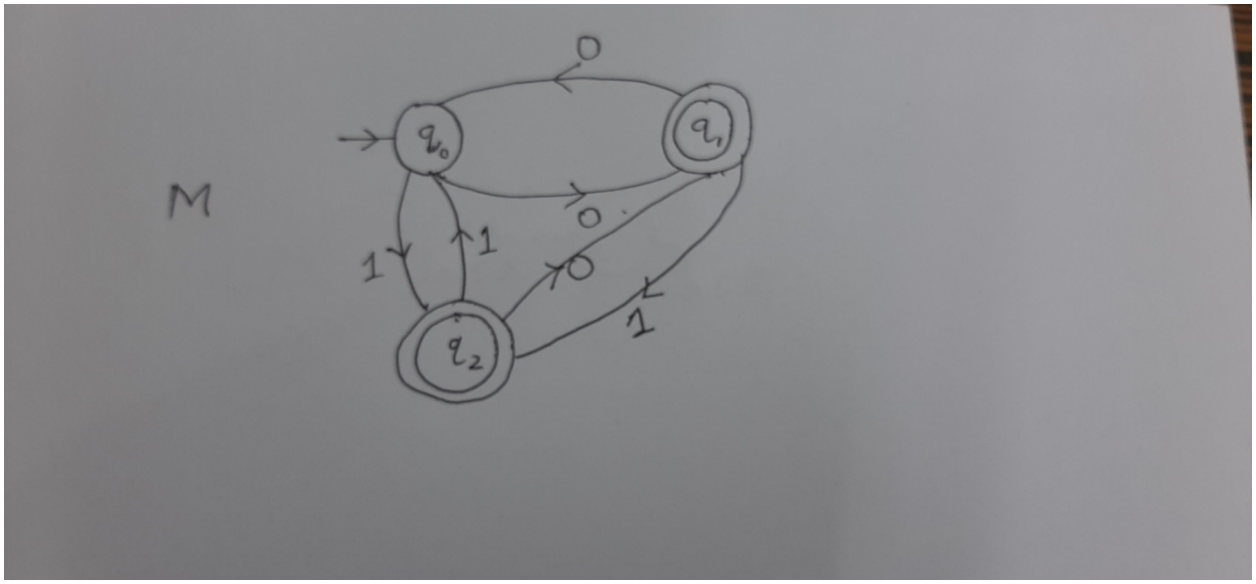
Deterministic Finite Automaton (DFA)

An Example



A DFA can be given simply by a Transition diagram which is a directed graph with edges labeled by symbols of the alphabet. The vertices are called the states. There is a start state marked with an incoming arrow (here q_0). There are some final states which are circled twice (there may not be any). (Here q_1 and q_2) Here the alphabet is {0,1}. Unless otherwise specified the alphabet will always be assumed to be {0,1}.

Edges will represent transitions. If there is an edge from state q to state q' with a label a, there is a transition from state q to q' when the machine encounters the symbol a. An edge may be labeled by more than one symbols. A DFA is essentially given by a transition function δ.

$\delta(q,a) = q'$ if there is an edge from q to q' with label a. Thus a DFA M can be formally specified by a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

Q is a finite set of states

Σ alphabet

$\delta : Q \times \Sigma \rightarrow Q$ transition function

$q_0 \in Q$ start state

F a subset of Q : set of final states

Thus the above DFA is given by M = ($\{q_0,q_1,q_2\}$, $\{0,1\}$, δ, $\{q_1, q_2\}$) where $\delta(q_0,0) = q_1$, $\delta(q_0,1)=q_2$, $\delta(q_1,0)=q_0$, $\delta(q_1,1)=q_2$, $\delta(q_2,0)=q_1$, $\delta(q_2,1) = q_0$. There is a third method called the Transition Table method of representing a DFA. Here the DFA is represented by a table whose rows are named by the states and the columns are named by the symbols of the alphabet. The start state is marked by an arrow -> and the final states are marked by *. The row q and column a entry is δ(q,a). Thus the above DFA is represented by the table

M	0	1
$\to q_0$	q_1	q_2
$* q_1$	q_0	q_2
$* q_2$	q_1	q_0

Transition diagrams are convenient to design and to understand the properties of a DFA and for almost all DFA's that we will discuss we will give the transition diagrams. But in a textual presentation the functional specification is the most convenient and this will be used in examinations. The tabular form is also useful in some work. Conversion from one form to another is straightforward and students should have practice.

For every state and every symbol there is precisely one outgoing edge. So the next state is completely determined by the current state. Hence – deterministic.

Given an input string w, a DFA M processes it by starting in the start state. Then the symbols of w are consumed one by one and M keeps changing state as dictated by the transition function. This way M lands in a state q_f when the whole input string is consumed. The machine then halts and accepts w if q_f is in F, and rejects w if q_f is not in F. For example our DFA M with inputs 01001 and 10100 carries out the following computations :

$(q_0 , 01001)$ $(q_0, 10100)$

$\downarrow$ $\downarrow$

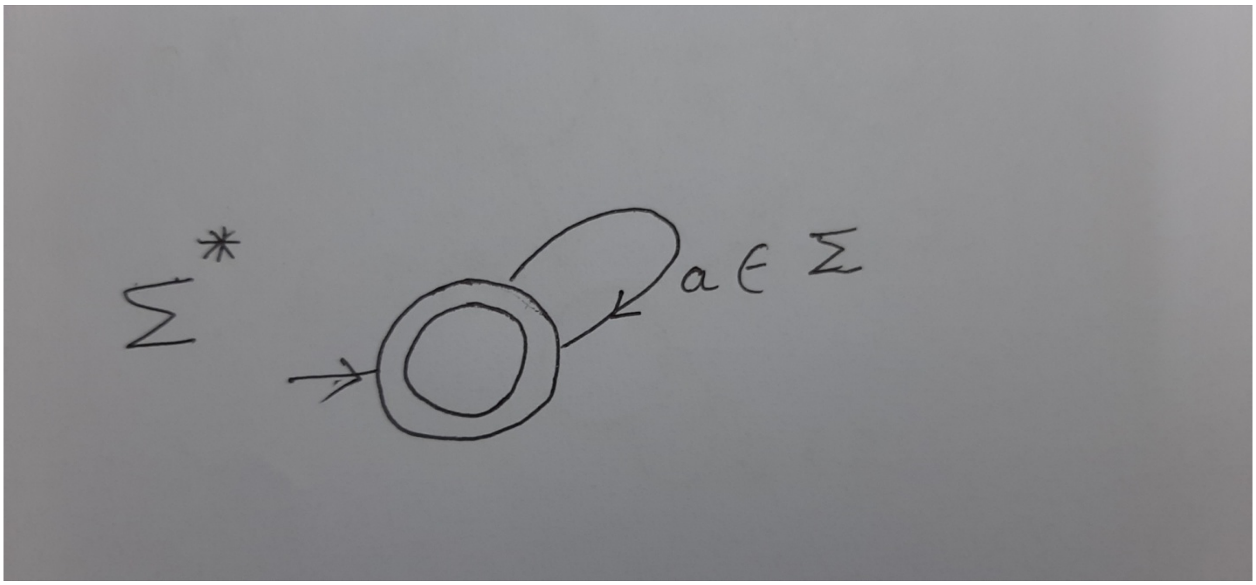
$(q_1 , 1001)$ $(q_2, 0100)$

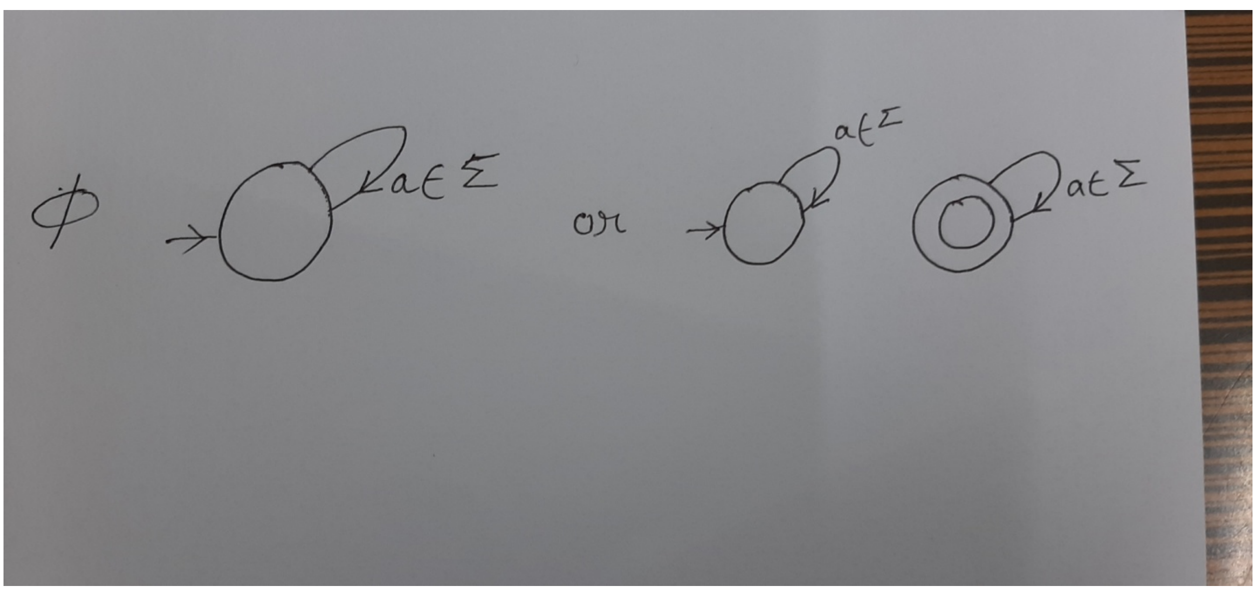
$$
\begin{array}{cc}
\downarrow & \downarrow \\
(q_2, 001) & (q_1, 100) \\
\downarrow & \downarrow \\
(q_1, 01) & (q_2, 00) \\
\downarrow & \downarrow \\
(q_0, 1) & (q_1, 0) \\
\downarrow & \downarrow \\
(q_2, \varepsilon) & (q_0, \varepsilon) \\
\text{Accepted} & \text{Rejected}
\end{array}
$$

This processing can be formulated by defining a configuration and the change of configuration in one step of computation. Configuration is (q,w’) where q is the current state and w’ is the remaining part of the input. In one step of computation (q,aw') changes to (yields in one step) $(\delta(q,a),w')$ [$(q,aw') \rightarrow (\delta(q,a),w')$]. Configuration c yields c' ($c \rightarrow^* c'$) if $c = c'$ or $c = c1 \rightarrow c2 \rightarrow \ldots\ldots cn = c'$. Starting configuration is (q_0,w) for input w and ending configuration is (q_f,ε). w is accepted if q_f is in F and rejected if q_f is not in F. Thus every string is either accepted or rejected. The set of strings accepted by a DFA M is called the language L(M) accepted (recognized) by M. Formally $L(M) = \{ w \in \sum^* \mid (q_0,w) \rightarrow^* (q_f, \varepsilon) \text{ for some } q_f \in F\}$

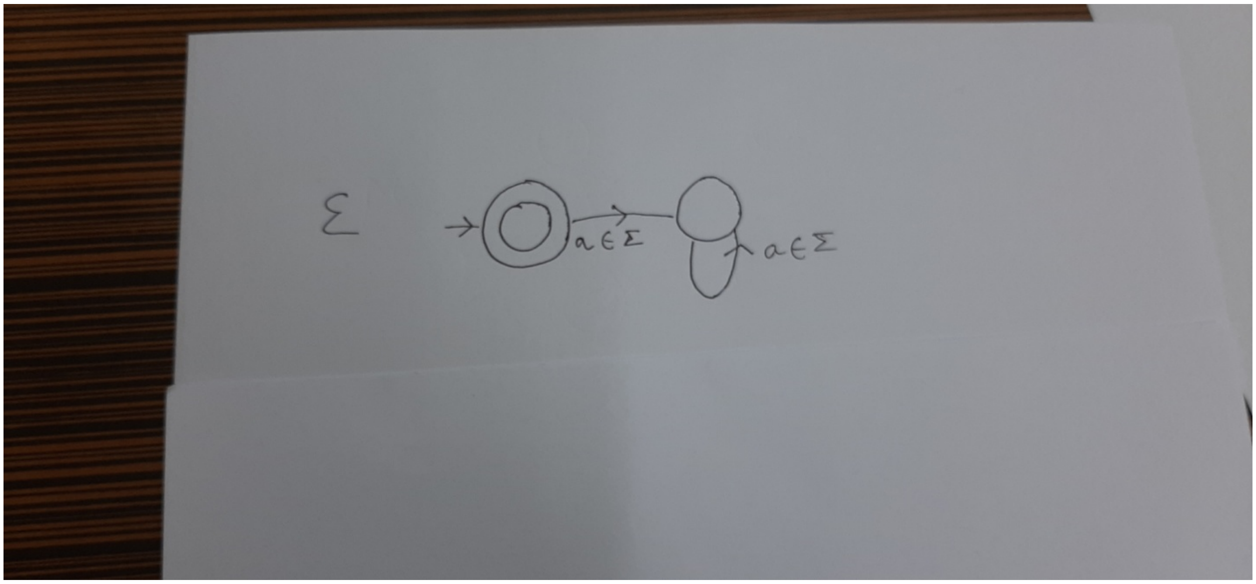
We can also define an extended transition function $\delta_e : Q \times \sum^* \rightarrow Q$ inductively by $\delta_e(q,\varepsilon) = q$, and $\delta_e(q,aw) = \delta_e(\delta(q,a),w)$. Thus $\delta_e(q,a) = \delta_e(\delta(q,a),\varepsilon)=\delta(q,a)$, $\delta_e(q,a1a2)=\delta_e(\delta(q,a1),a2) =\delta(\delta(q,a1),a2)$ etc. If $\delta_e(q,w) = q'$, M goes to state q' from q after processing w. In terms of δ_e, $L(M) = \{ w \in \Sigma^* \mid \delta_e(q_0,w) \in F\}$

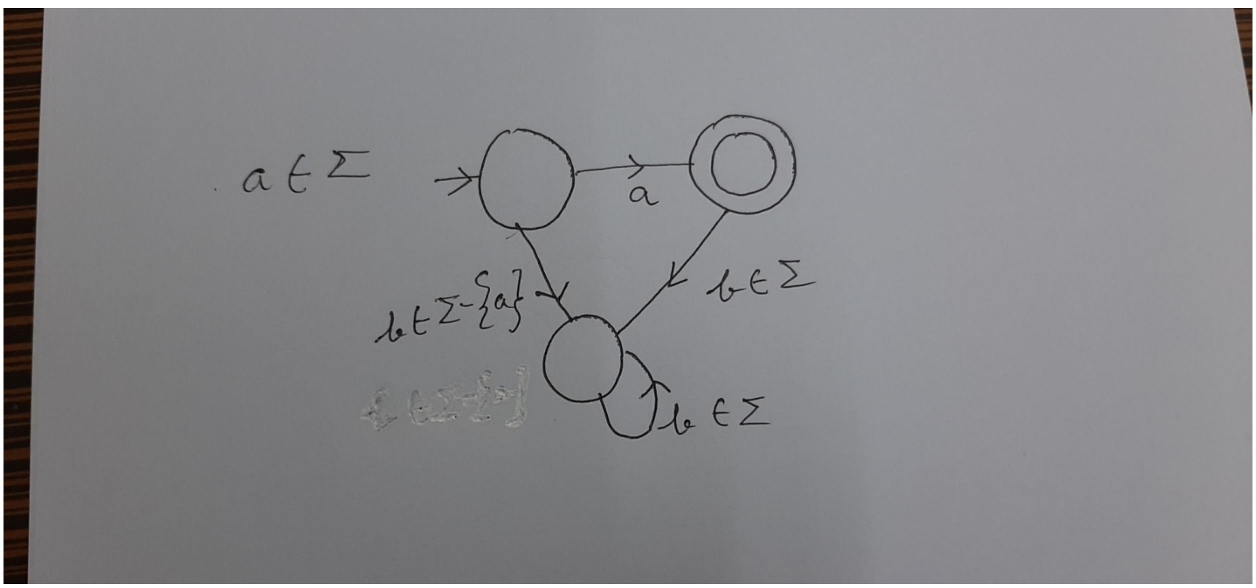
DFA's for some simple languages :





No final state final states unreachable





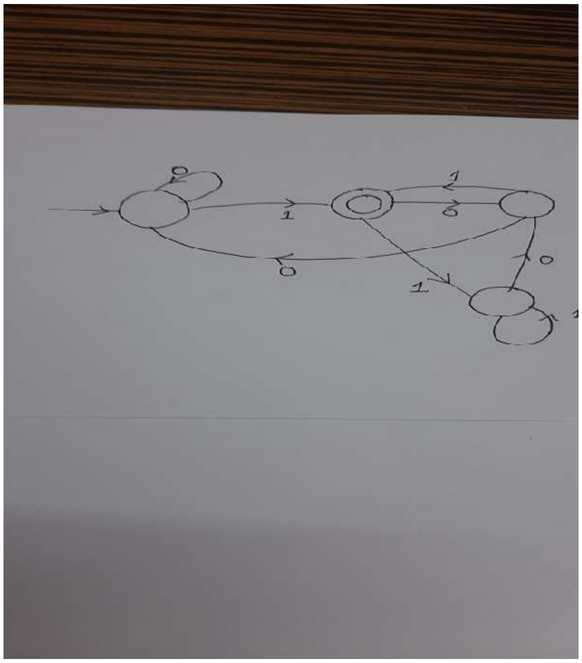
DFA's M1 and M2 are called equivalent if L(M1) = L(M2). We will get an algorithm to test equivalence.

Also given a DFA M we can obtain an equivalent DFA M_0 with a minimum number of states. M_0

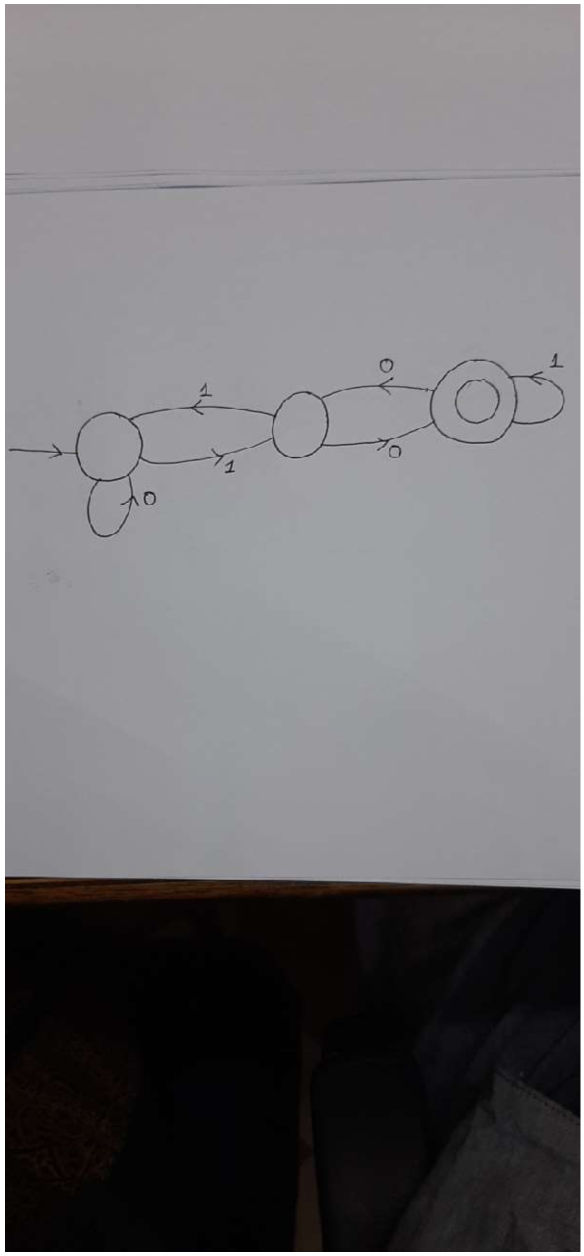
is unique except for a possible renumbering of the states. The process of obtaining M_0 from M is called the minimization of the DFA M. We will get an algorithm for minimization of a DFA.

DFA's for some more languages:

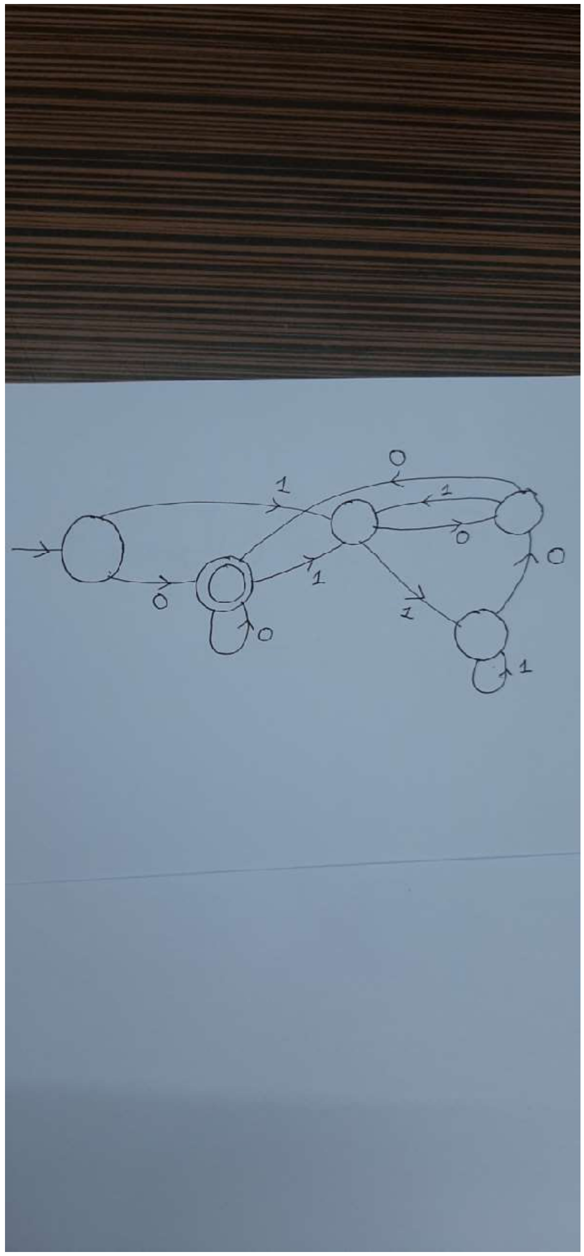
Binary strings which are 1 mod 4



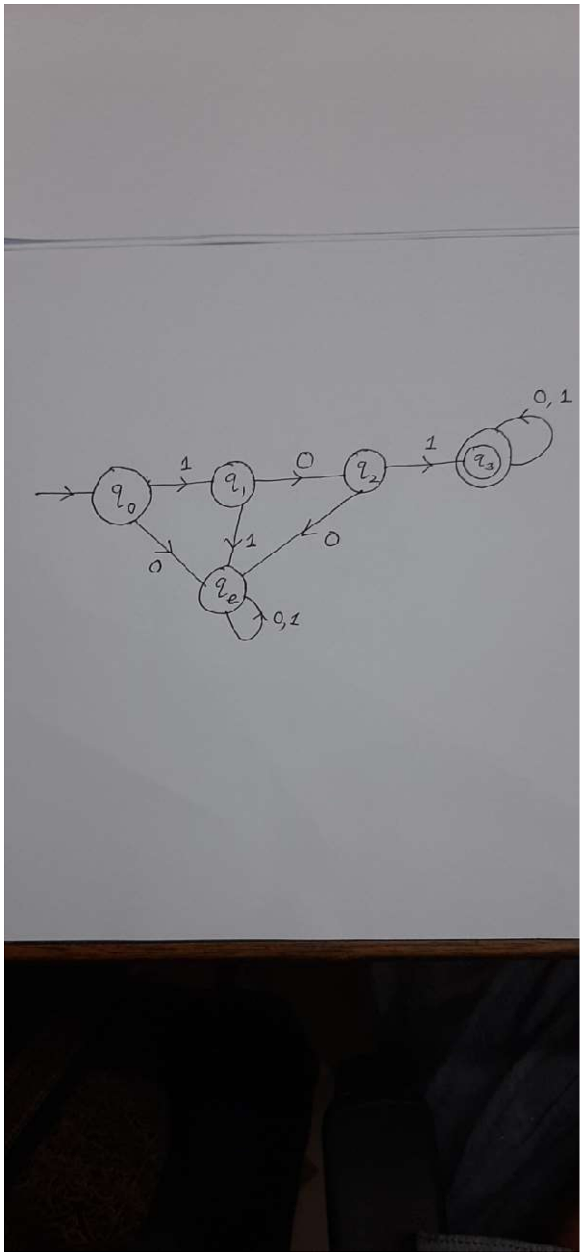
Binary strings which are 2 mod 3



Nonempty binary strings which are 0 mod 4

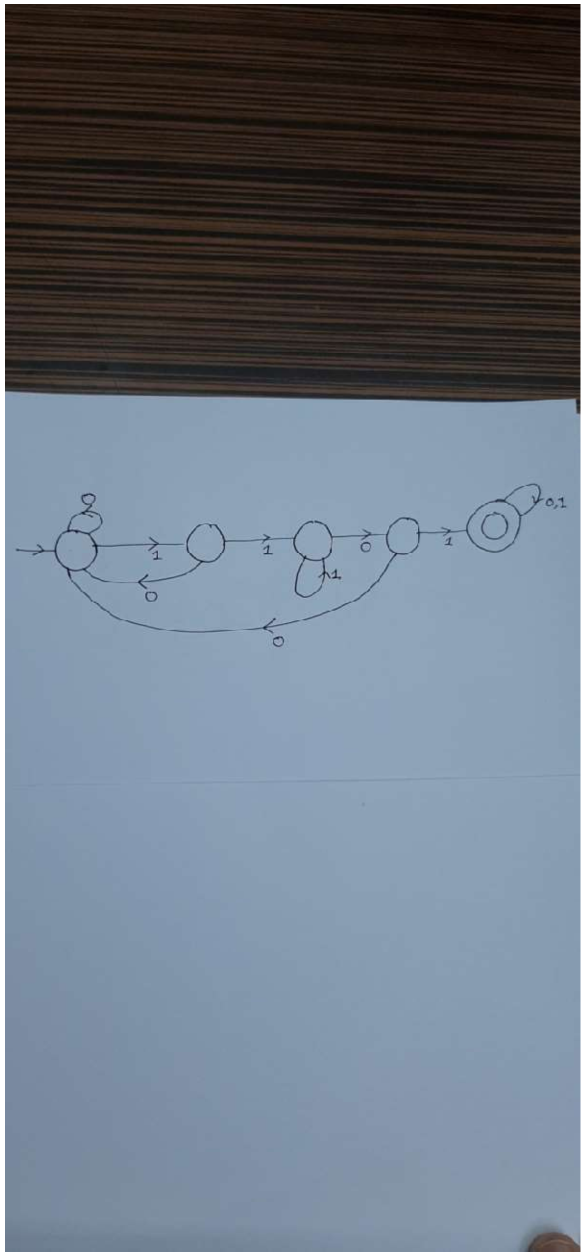


Set of all strings over {0,1} beginning with 101

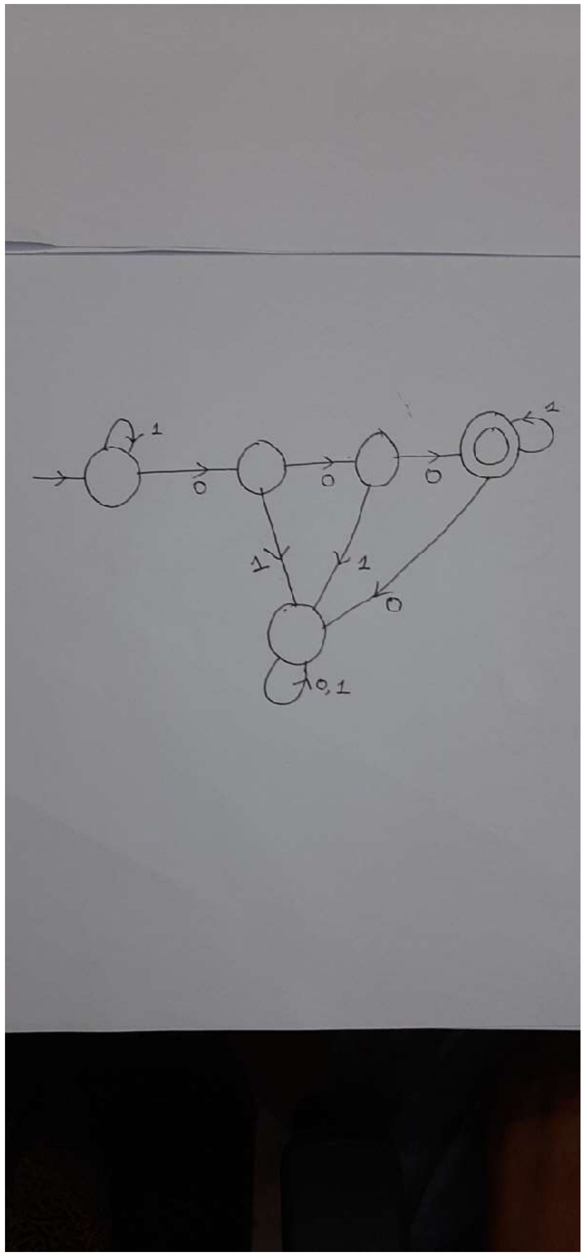


Here q_e behaves like an error state.

Set of all strings over {0,1} containing 1101 as a substring

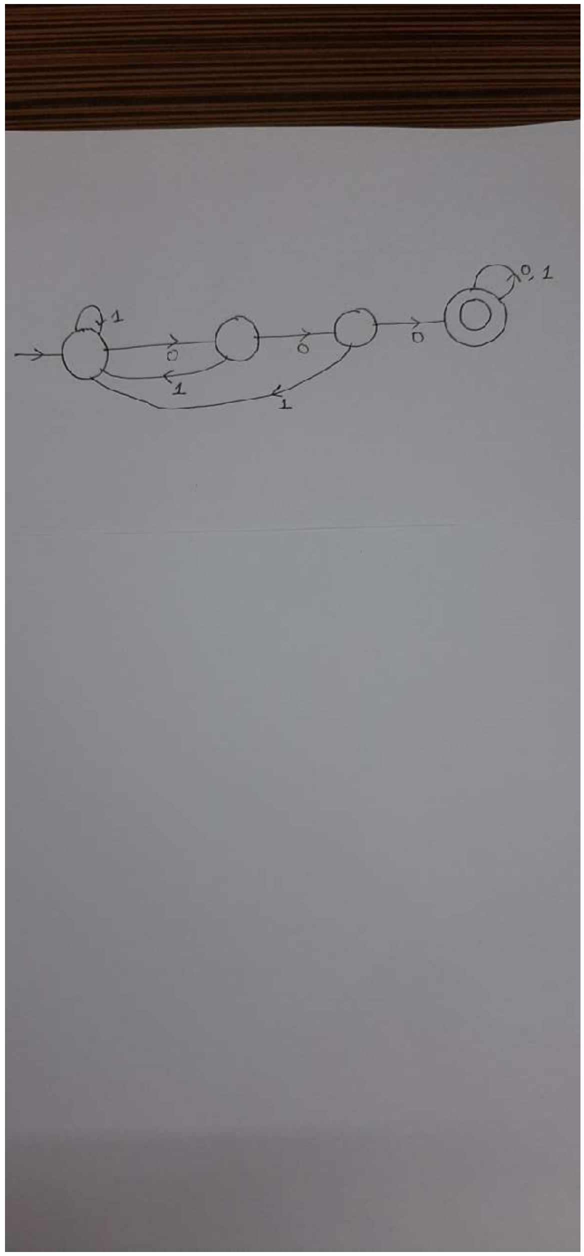


Set of all strings over {0,1} with exactly 3 0's which are consecutive

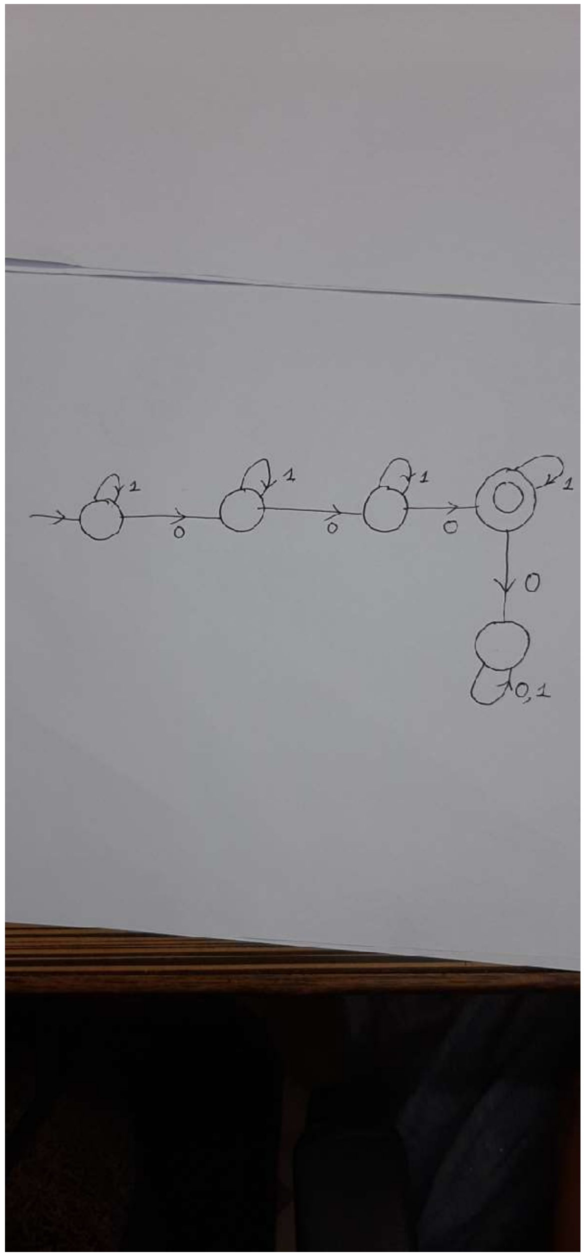


If not mentioned alphabet will be {0,1}

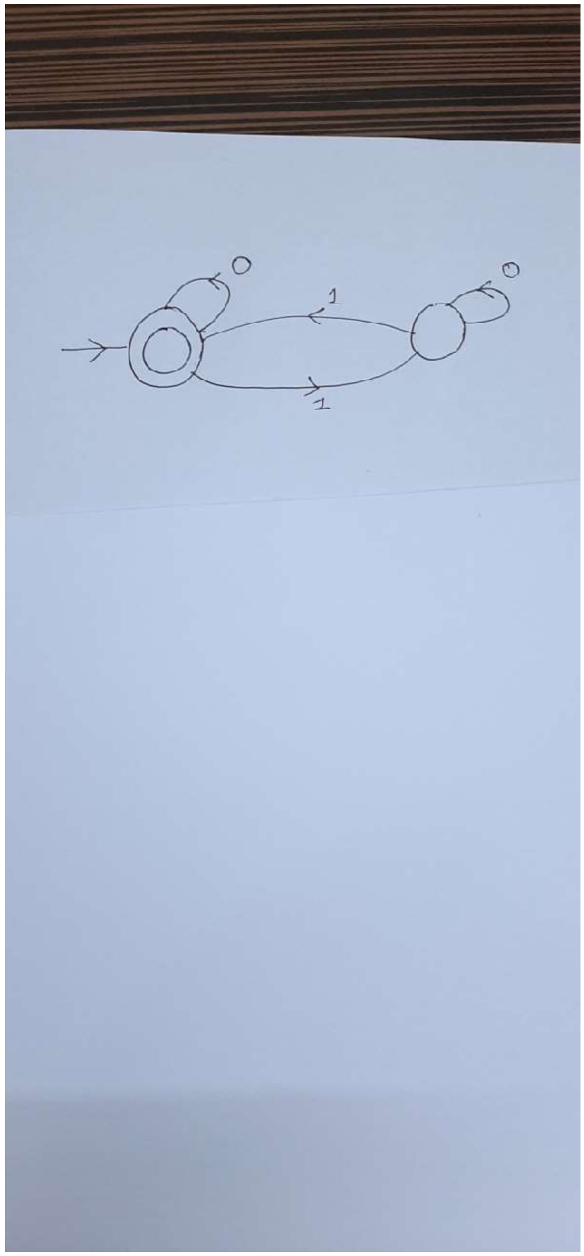
With 3 consecutive 0’s



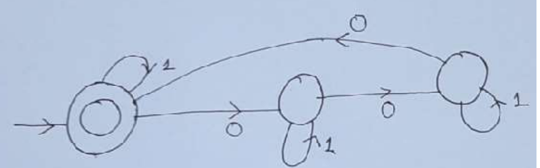
With exactly 3 0’s



Number of 1’s is even



Number of 0’s is a multiple of 3


0
1
0
0
1
1