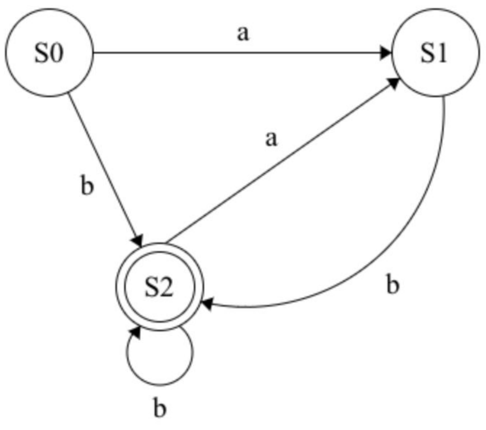
# ICS SHEET #11

## Problem 11.1 :

a) The FSM (Σ,S,s0,δ,F) with Σ = {a, b}, S = { S0, S1, S2 }, s0 = S0, F = {S2} and :

δ = {((S0, a), S1)
((S0, b), S2)
((S1, b), S2)
((S2, a), S1)
((S2, b), S2)
}

### b) Drawing



### c) Haskell Code :

```haskell
data State = S0 | S1 | S2

accepts :: State → String → Bool
accepts S0 ('a' : xs) = accepts S1 xs
accepts S0 ('b' : xs) = accepts S2 xs
accepts S1 ('b' : xs) = accepts S2 xs
accepts S2 ('a' : xs) = accepts S1 xs
accepts S2 ('b' : xs) = accepts S2 xs
accepts S2 [ ] = True
accepts _ _ = False

determine :: String → Bool
determine = accepts S0
```

### d) Grammar definition :

In order to define the regular grammar that generates L, I tried to fully grasp the rule that set our FSM.

First, let's define it in a couple of examples:

| rule | | example |
|---|---|---|
| strings over {a, b} where every a is followed b | ∈ | "babbbabbabab" / "bbbab" / "bab" |
| | ∉ | "Aaba" / "babbabaab" / "baab" |

Now, I wrote a function (C++) that would do the same as the finite state machine we just designed, except it prints "Success" if the string follows the rule and prints "Fail" if it doesn't :

```
#include <iostream>
#include <string>

using namespace std;

int main(){
   string str = "babbbabbab";
   for (int i = 0; i < str.length(); i++){
      if ( str[i] == 'b' ){
         // do nothing
      } else if ( str[i] == 'a' ){
            if ( str[i+1] == 'b' ){  // do nothin }
            else if ( str[i+1] == 'a' || i + 1 == str.length() ){
             // second condition is to check that the string doesn't end with an a
             cout << "Fail" << endl;
             exit(1);
            // exits the function because this is not valid
         }
      }
   }
   cout << "Success" << endl;
   return 0;
}
```

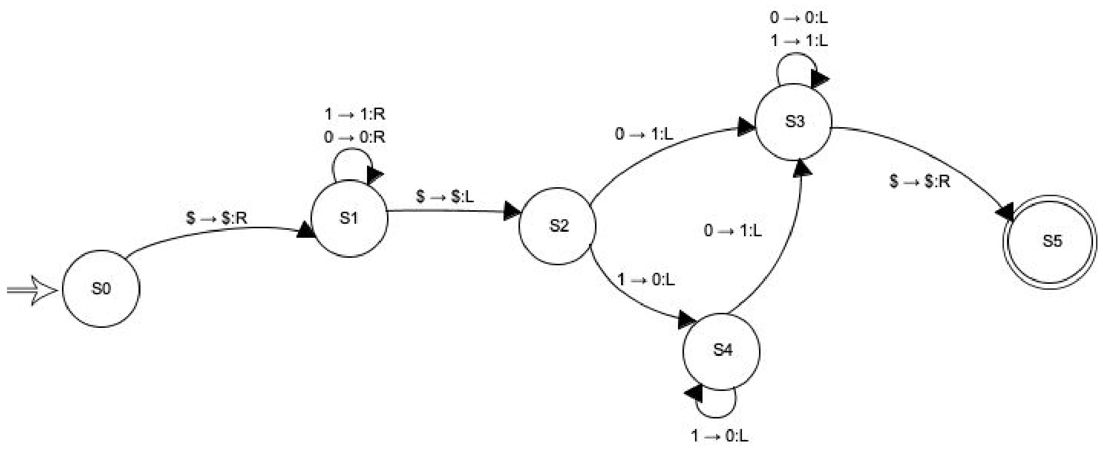
Now to generate strings over {a, b} in a way where a is always followed by b, let S0|S1|S2 be the symbols in our finite state machine (b) :

$$
\text{(GR)}\quad
\begin{aligned}
S0 &\rightarrow a \mid S1\\
S0 &\rightarrow b \mid S2\\
S1 &\rightarrow b \mid S2\\
S2 &\rightarrow a \mid S1\\
S2 &\rightarrow b \mid S2
\end{aligned}
$$

Thus grammar (GR) generates L.

## Problem 11.2 :

a) Tinc (Σ,S,s0,Γ,b,δ,F) with Σ = {0,1},
S = {S0,S1,S2,S3,S4,S5}, s0 =S0, Γ={0,1, $} ,b= $ ,F ={S5},and
δ = { (S0, $, S1, $, R), (S1, 0, S1, 0, R), (S1, 1, S1, 1, R), (S1, $,S2, $, L),
(S2, 0, S3, 1 ,L), (S2, 1, S5, 0, L), (S3, 1, S3, 1, L), (S3, 0, S3, 0, L),
(S3? $, S5, , $, R), (S4, 1, S4, 0, L), (S4, 0, S3, 1, L)
}



**Textual description of $T_{inc}$:**

This turing machine starts at state S0, this state simply reads the enclosing symbol '$' and shift to the right of it using the state transition. It then moves to state 2 (S1) which has 2 loopback transitions. The transitions read 1 or 0 from the tape, keep their value and move to the right, this keeps happening until it reaches the left enclosing symbol and shifts back to the left to move to the third state S2 keeping the enclosing symbol unchanged. This state (S2) reads the value to the left, if it's a 0 (case 1) then it turns into a 1, shifts to the left and moves to S3 otherwise if it's a '1' (case 2) then it substitutes it with a 0, moves to the left and moves to S4. Let's discuss both cases.
Case 1 :
In this case we're in S3. It simply travels back to the left enclosing bracket by leaving the 1's and 0's unchanged using two loopback transitions. When it reaches the left enclosing bracket, it transitions to S5 by keeping the enclosing symbol the same and moving to the right. The turing machine has now reached it's acceptance state.
Case 2 :
The transition happens from S2 to S4 by changing the 1 to 0 and moving to the left. S4 has a loopback transition that turns 1's to 0's while moving to the left. This will keep happening until it reaches a 0. We're sure it'll find a 0 because we assumed the first digit from the left will always be 0 to avoid overloading. Once that digit is reached, it will be changed back to 1 and the turing machine will transition to S3. S3 will either travel back to the left of the input if any digits are remaining or will directly find the enclosing bracket and transition to the acceptance state by keeping the enclosing symbol unchanged and moving to the right.

**Execution of “$0110$”, a symbol in brackets indicates the head of the turing machine :**

| | | | | | | |
|---|---|---|---|---|---|---|
| S0 : | **[$]** | 0 | 1 | 1 | 0 | $ |
| S1 : | $ | **[0]** | 1 | 1 | 0 | $ |
| S1 : | $ | 0 | **[1]** | 1 | 0 | $ |
| S1 : | $ | 0 | 1 | **[1]** | 0 | $ |
| S1 : | $ | 0 | 1 | 1 | **[0]** | $ |
| S1 : | $ | 0 | 1 | 1 | 0 | **[$]** |
| S2 : | $ | 0 | 1 | 1 | **[0]** | $ |
| S3 : | $ | 0 | 1 | **[1]** | 1 | $ |
| S3 : | $ | 0 | **[1]** | 1 | 1 | $ |
| S3 : | $ | **[0]** | 1 | 1 | 1 | $ |
| S3 : | **[$]** | 0 | 1 | 1 | 1 | $ |
| S5 : | $ | **[0]** | 1 | 1 | 1 | $ |

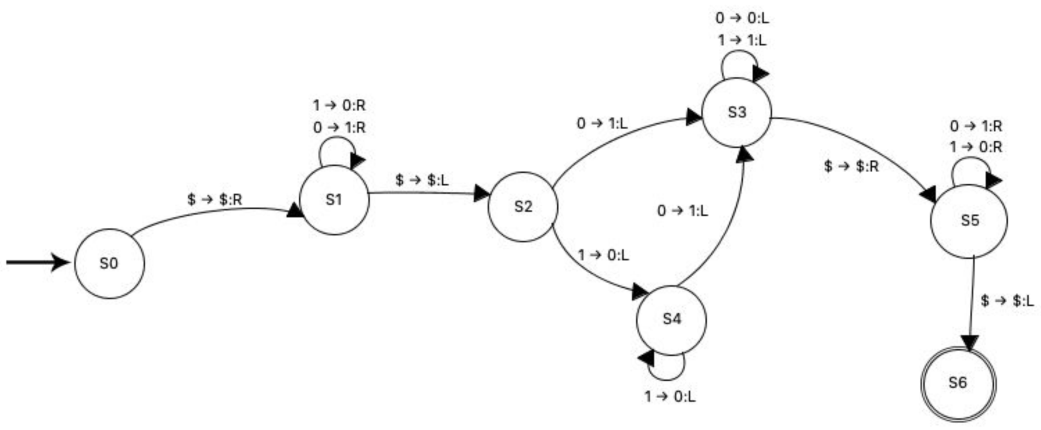
So the result of adding 1 to 0110 is 0111, which is true.

**Turing machine Haskell code (file ‘tinc.hs’) :**

**b**)

$T_{dec}$ (Σ,S,s0,Γ,b,δ,F) with Σ = {0,1},
S = {S0,S1,S2,S3,S4,S5, S6}, $s_0$ =S0, Γ={0,1, $} ,b= $ ,F ={S6}, and

δ = {(S0, $, S0, $, R), (S1, 1, S1, 0, R), (S1, 0, S1, 1, R), (S1, $, S2, $, L),
(S2, 0, S3, 1, L),(S2, 1, S4, 0, L), (S3, 1, S3, 1, L), (S3, 0, S3, 0, L),
(S3, $, S3, $, R), (S4, 1, S4, 0, L),(S4, 0, S3, 1, L),
(S5, 1, S5, 0, R), (S5, 0, S5, 1, R), (S5, $, S6, $, L),
}



**Textual description of $T_{dec}$:**

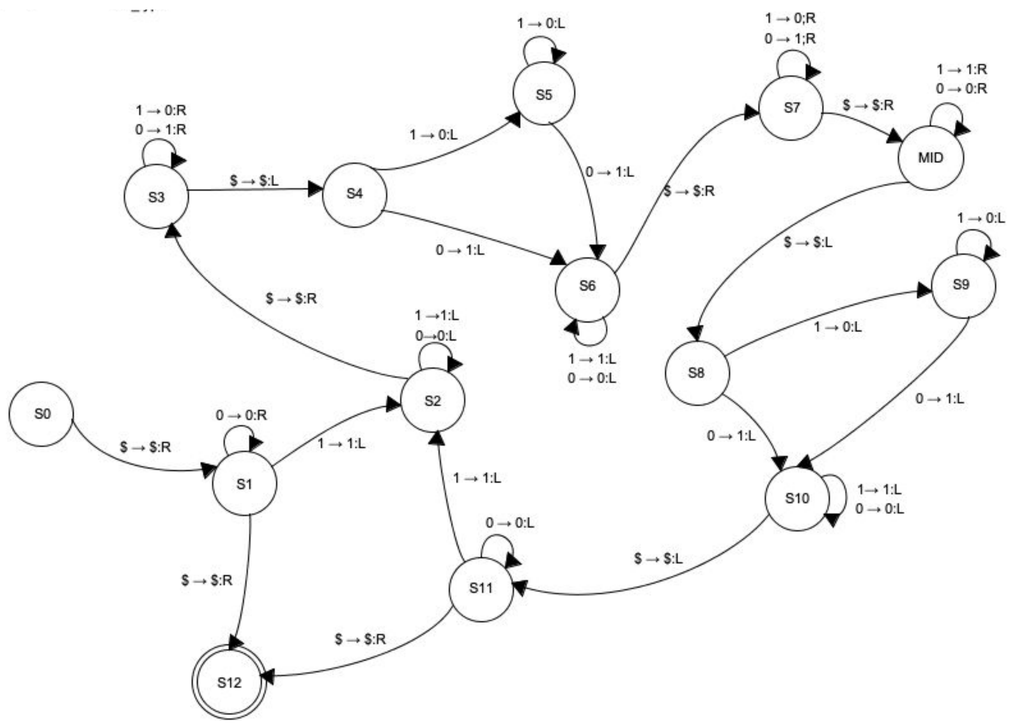
$T_{dec}$ turing machine first starts from the initial state S0 and since the turing machine starts with the head located at the $ sign left of the number, then the first transition it makes is a movement to the right without changing the tapes initial symbol ‘$’ since it encloses our binary number. It then moves to the second state, this state constantly changes 0’s to 1’s and vice-versa while moving to the right until it reaches the rightmost enclosing symbol ‘$’. It then moves to the next state, from now it’s very similar to $T_{inc}$ until it reaches S5, this is where things change. This state transitions through all the binary digits again and changes the 1’s to 0’s and 0’s to 1’s and once it reaches the ‘$’ enclosing symbol and finally moves to the accepting state S6.

**Execution of “$0111$”, a symbol in brackets indicates the head of the Turing Machine :**

```
S0 :  [$]  0    1    1    1    $
S1 :   $  [0]   1    1    1    $
S1 :   $   1   [1]   1    1    $
S1 :   $   1    0   [1]   1    $
S1 :   $   1    0    0   [1]   $
S1 :   $   1    0    0    0   [$]
S2 :   $   1    0    0   [0]   $
S3 :   $   1    0   [0]   1    $
S3 :   $   1   [0]   0    1    $
S3 :   $  [1]   0    0    1    $
S3 :  [$]  1    0    0    1    $
S5 :   $  [1]   0    0    1    $
S5 :   $   0   [0]   0    1    $
S5 :   $   0    1   [0]   1    $
S5 :   $   0    1    1   [1]   $
S5 :   $   0    1    1    0   [$]
S6 :   $   0    1    1   [0]   $
```

**Turing machine Haskell code ( file ‘tdec.hs’) :**

## c)



$T_{add}$ (Σ,S,s0,Γ,b,δ,F) with Σ = {0,1},
S = {S0,S1,S2,S3,S4,S5, S6, S7, S8, S9, S10, S11, S12, MID},
$s_0$ = S0, Γ={0,1, $} ,b = $ ,F ={S12},

δ = {
(S0, $, S1, $, R), (S1, 0, S1, 0, R), (S1, $, S12, $, R), (S1, 1, S2, 1, L), (S2, 0, S2, 0, L), (S2, 1, S2, 1, L), (S2, $, S3, $, R), (S3, 1, S3, 0, R), (S3, 0, S3, 1, R), (S3, $, S4, $, L), (S4, 1, S5, 0, L),
(S4, 0, S6, 1, L), (S5, 1, S5, 0, L), (S5, 0, S6, 1, L), (S6, 1, S6, 1, L), (S6, 0, S6, 1, L),
(S6, $, S7, $, R), (S7, 1, S7, 0, R), (S7, 0, S7, 1, R), (S7, $, MID, $, R),(MID, 1, MID, 1, R)
(MID, 0, MID, 0, R), (MID, $, S8, $, L), (S8, 1, S9, 0, L), (S8, 0, S10, 1, L), (S9, 1, S9, 0, L),
(S9, 0, S10, 1, L), (S10, 1, S10, 1, L), (S10, 0, S10, 0, L), (S10, $, S11, $, L),
(S11, 0, S11, 0, L), (S11, 1, S2, 1, L), (S11, $, S12, $, R)
}

**Textual description of $T_{add}$:**

$T_{add}$ is an adjusted combination of both $T_{inc}$ and $T_{dec}$.
First, S0 transitions to S1 by keeping the enclosing symbol the same and moving to the right. S1 then runs a check using a loopback transition by checking the bits moving rightwards to see if the first binary number is not a 0. This gives us two cases. First one is that the first binary number is 0, in this case once the middle enclosing symbol is reached the state transitions to S12 (acceptance state). Second case is when there's a 1 in the binary number (number is not null). In this case, the turing machine transitions to S2 by keeping the 1 unchanged and going back to the left. S2 then runs two loopback transitions which essentially go back to the leftmost enclosing symbol, it then transitions again to the right, keeping $ the same, to S3. From now on the turing machine works similarly to $T_{dec}$ Until it reaches S7. S7 inverts all the bits using two loopback transitions and transitions to MID by moving to the right and keeping the $ unchanged. MID then runs two other loopback transitions in order to travel to the rightmost enclosing symbol and once it does it transitions to S8 by moving to the left and leaving the enclosing bracket $. This part also runs similarly to $T_{inc}$ therefore we can skip directly to S10. S10 then travels to the middle enclosing symbol using two loopback transitions and transitioning from S10 to S11 by moving to the left. S11 then runs a similar check to S1 using a similar

loopback transition, expect; this one travels from the left. This also gives us two possible cases. First one is the first binary number is null, in this case S11 transitions to S12 by moving to the right. S12 is the acceptance state, therefore the turing machine stops. Second case is it finds a 1, which will lead to a transition from S11 to S2 by keeping the one unchanged and following up with the leftward movement of the loopback transition in S11. This loop then iterates as many times as necessary until the first binary number is null and the turing machine reaches its finite state.

**<u>Execution of “$0001$0101$”, a symbol in brackets indicates the head of the Turing Machine :</u>**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S0 : | [$] | 0 | 0 | 0 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S1 : | $ | [0] | 0 | 0 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S1 : | $ | 0 | [0] | 0 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S1 : | $ | 0 | 0 | [0] | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S1 : | $ | 0 | 0 | 0 | [1] | $ | 0 | 1 | 0 | 1 | $ |
| S2 : | $ | 0 | 0 | [0] | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S2 : | $ | 0 | [0] | 0 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S2 : | $ | [0] | 0 | 0 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S2 : | [$] | 0 | 0 | 0 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S3 : | $ | [0] | 0 | 0 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S3 : | $ | 1 | [0] | 0 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S3 : | $ | 1 | 1 | [0] | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S3 : | $ | 1 | 1 | 1 | [1] | $ | 0 | 1 | 0 | 1 | $ |
| S3 : | $ | 1 | 1 | 1 | 0 | [$] | 0 | 1 | 0 | 1 | $ |
| S4 : | $ | 1 | 1 | 1 | [0] | $ | 0 | 1 | 0 | 1 | $ |
| S5 : | $ | 1 | 1 | [1] | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S6 : | $ | 1 | [1] | 1 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S6 : | $ | [1] | 1 | 1 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S6 : | [$] | 1 | 1 | 1 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S7 : | $ | [1] | 1 | 1 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S7 : | $ | 0 | [1] | 1 | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S7 : | $ | 0 | 0 | [1] | 1 | $ | 0 | 1 | 0 | 1 | $ |
| S7 : | $ | 0 | 0 | 0 | [1] | $ | 0 | 1 | 0 | 1 | $ |
| S7 : | $ | 0 | 0 | 0 | 0 | [$] | 0 | 1 | 0 | 1 | $ |
| MID: | $ | 0 | 0 | 0 | 0 | $ | [0] | 1 | 0 | 1 | $ |
| MID: | $ | 0 | 0 | 0 | 0 | $ | 0 | [1] | 0 | 1 | $ |
| MID: | $ | 0 | 0 | 0 | 0 | $ | 0 | 1 | [0] | 1 | $ |
| MID: | $ | 0 | 0 | 0 | 0 | $ | 0 | 1 | 0 | [1] | $ |
| MID: | $ | 0 | 0 | 0 | 0 | $ | 0 | 1 | 0 | 1 | [$] |
| S8 : | $ | 0 | 0 | 0 | 0 | $ | 0 | 1 | 0 | [1] | $ |
| S9 : | $ | 0 | 0 | 0 | 0 | $ | 0 | 1 | [0] | 0 | $ |
| S10: | $ | 0 | 0 | 0 | 0 | $ | 0 | [1] | 1 | 0 | $ |
| S10: | $ | 0 | 0 | 0 | 0 | $ | [0] | 1 | 1 | 0 | $ |
| S10: | $ | 0 | 0 | 0 | 0 | [$] | 0 | 1 | 1 | 0 | $ |
| S11: | $ | 0 | 0 | 0 | [0] | $ | 0 | 1 | 1 | 0 | $ |
| S11: | $ | 0 | 0 | [0] | 0 | $ | 0 | 1 | 1 | 0 | $ |
| S11: | $ | 0 | [0] | 0 | 0 | $ | 0 | 1 | 1 | 0 | $ |
| S11: | $ | [0] | 0 | 0 | 0 | $ | 0 | 1 | 1 | 0 | $ |
| S11: | [$] | 0 | 0 | 0 | 0 | $ | 0 | 1 | 1 | 0 | $ |
| S12: | $ | [0] | 0 | 0 | 0 | $ | 0 | 1 | 1 | 0 | $ |

So the output of the addition after the execution is $0000$0110$
Which is true because 0001 + 0101 = 0110

**<u>Turing machine Haskell code (file ‘tadd.hs’)</u>**