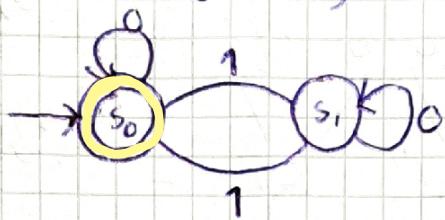


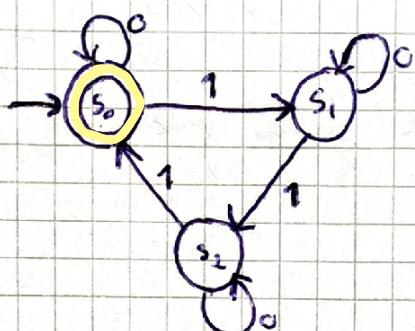
# Øving 11

DM TDAT 2005

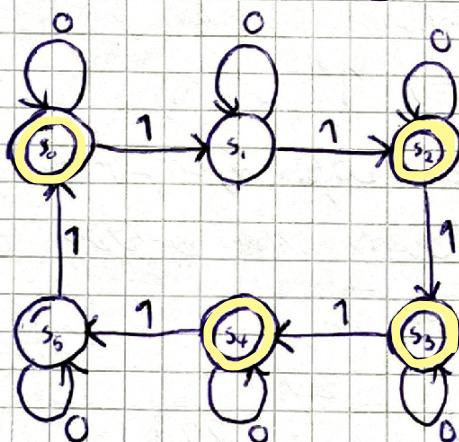
① a)  $L_1 : 0^*(10^*1)^*0^*$



$L_2 : 0^*(10^*10^*1)^*0^*$



b)



$$0^*(10^*1)|(10^*10^*1)0^*$$

② a)

$$V = \{S, 0, 1\}$$

$$S \rightarrow \epsilon$$

$$\Sigma = \{0, 1\}$$

$$S \rightarrow 1S1$$

$$S \rightarrow 0S0$$

$$S \rightarrow 0$$

$$S \rightarrow 1$$

b) dette er ikke et regulert  
fordi det ikke finn  
en endelig automat

som aksepterer spesifik ettersom automaten krever  
en form for minne for å vite om den andre  
halvdelene er spesieltsett av den første.

3

```

class Automat
{
    private final char[] alfabet;
    private final int[][] neste_tilstands_funksjon;
    private int tilstand;
    private int[] aksepterende_tilstander;

    private Automat(String alfabet , int ant_tilsatnder , int[] aksepterende_tilstander)
    {
        this.alfabet = alfabet.toCharArray();
        this.neste_tilstands_funksjon = new int[ant_tilsatnder][this.alfabet.length];
        this.aksepterende_tilstander = aksepterende_tilstander;
    }

    public static void main(String[] args)
    {
        println_Color( color: "blue" , tekst: "\n\nOppgave 8a: bitstreng som starter på 0, og inneholder nøyaktig 1 ener");
        int[] aksepterende_tilstander_8a = { 2 };
        Automat oppgave_8a = new Automat( alfabet: "01" , ant_tilsatnder: 4 , aksepterende_tilstander_8a);

        /*
        legger manuelt inn overganger slik da jeg synes det er mye mer oversiktlig, kunne også gitt en 2-dimensjonal array
        i konstruktøren men syntes det var vanskelig å se for seg de forskjellige overgangene mellom tilstandene slik
        */
        oppgave_8a.leggTilFunksjon( fra_tilstand: 0 , tegn: '0' , til_tilstand: 1 );
        oppgave_8a.leggTilFunksjon( fra_tilstand: 0 , tegn: '1' , til_tilstand: 3 );
        oppgave_8a.leggTilFunksjon( fra_tilstand: 1 , tegn: '0' , til_tilstand: 1 );
        oppgave_8a.leggTilFunksjon( fra_tilstand: 1 , tegn: '1' , til_tilstand: 2 );
        oppgave_8a.leggTilFunksjon( fra_tilstand: 2 , tegn: '0' , til_tilstand: 2 );
        oppgave_8a.leggTilFunksjon( fra_tilstand: 2 , tegn: '1' , til_tilstand: 3 );
        oppgave_8a.leggTilFunksjon( fra_tilstand: 3 , tegn: '0' , til_tilstand: 3 );
        oppgave_8a.leggTilFunksjon( fra_tilstand: 3 , tegn: '1' , til_tilstand: 3 );

        oppgave_8a.sjekkInput_print("");
        oppgave_8a.sjekkInput_print("100");
        oppgave_8a.sjekkInput_print("010");
        oppgave_8a.sjekkInput_print("111");
        oppgave_8a.sjekkInput_print("010110");
        oppgave_8a.sjekkInput_print("001000");

        println_Color( color: "blue" , tekst: "\n\nOppgave 8b: strenger som starter med ab");
        int[] aksepterende_tilstander_8b = { 3 };
        Automat oppgave_8b = new Automat( alfabet: "ab" , ant_tilsatnder: 5 , aksepterende_tilstander_8b);

        oppgave_8b.leggTilFunksjon( fra_tilstand: 0 , tegn: 'a' , til_tilstand: 1 );
        oppgave_8b.leggTilFunksjon( fra_tilstand: 0 , tegn: 'b' , til_tilstand: 2 );
        oppgave_8b.leggTilFunksjon( fra_tilstand: 1 , tegn: 'a' , til_tilstand: 4 );
        oppgave_8b.leggTilFunksjon( fra_tilstand: 1 , tegn: 'b' , til_tilstand: 3 );
        oppgave_8b.leggTilFunksjon( fra_tilstand: 2 , tegn: 'a' , til_tilstand: 3 );
        oppgave_8b.leggTilFunksjon( fra_tilstand: 2 , tegn: 'b' , til_tilstand: 4 );
        oppgave_8b.leggTilFunksjon( fra_tilstand: 3 , tegn: 'a' , til_tilstand: 3 );
        oppgave_8b.leggTilFunksjon( fra_tilstand: 3 , tegn: 'b' , til_tilstand: 3 );
        oppgave_8b.leggTilFunksjon( fra_tilstand: 4 , tegn: 'a' , til_tilstand: 4 );
        oppgave_8b.leggTilFunksjon( fra_tilstand: 4 , tegn: 'b' , til_tilstand: 4 );

        oppgave_8b.sjekkInput_print("abbb");
        oppgave_8b.sjekkInput_print("aaah");
        oppgave_8b.sjekkInput_print("babab");
        oppgave_8b.sjekkInput_print("bbba");
    }

    //legger inn en overgang fra en tilstand til en annen utifra et gitt tegn
    private void leggTilFunksjon(int fra_tilstand , char tegn , int til_tilstand)
    {
        neste_tilstands_funksjon[fra_tilstand][finnIndeks(tegn)] = til_tilstand;
    }

    //finner hvilken indeks tegnet har i neste_tilstands_funksjon tabellen
    private int finnIndeks(char tegn)
    {
        int tegn_indeks = -1;
        for(int i = 0 ; i < this.alfabet.length ; i++)
        {
            if(this.alfabet[i] == tegn)
            {
                tegn_indeks = i;
            }
        }
        return tegn_indeks;
    }

    //printer ut resultatet på en ryddig måte
    private void sjekkInput_print(String input)
    {
        if(sjekkInput(input))
        {
            String res = string_Color( color: "green" , tekst: " \tTRUE");
            println_Color( color: "yellow" , tekst: "\\" + input + "\\ " + res + "\n");
            return;
        }
        String res = string_Color( color: "red" , tekst: " \tFALSE");
        println_Color( color: "yellow" , tekst: "\\" + input + "\\ " + res + "\n");
    }
}

```

```
//returnerer om en streng er akseptert eller ikke av Automaten
private boolean sjekkInput(String input)
{
    this.tilstand = 0;
    char[] inputArray = input.toCharArray();

    for(char c : inputArray)
    {
        if(finnIndeks(c) == -1)
        {
            //input strengen inneholder en char som ikke finnes i det gitte alfabetet
            return false;
        }
        //hvis tegnet er gyldig, endres tilstanden
        this.tilstand = funksjon(c);
    }

    //etter å ha gått igjennom hele input strenger sjekker vi om tilstanden vi nå er i, er en aksepterende tilstand
    for(int aksepterende_tilstand : aksepterende_tilstander)
    {
        if(this.tilstand == aksepterende_tilstand)
        {
            return true;
        }
    }
    return false;
}

//returnerer den nye tilstanden når vi leser tegnet 'tegn' fra får nåværende tilstand
private int funksjon(char tegn)
{
    // neste tilstanden ligger lagret i posisjonen til hvilken tilstand vi er i, og indeksen til tegnet vi leser
    int ny_tilstand = neste_tilstands_funksjon[this.tilstand][finnIndeks(tegn)];
    //System.out.println("tilstand: " + this.tilstand + "\t\tleser " + tegn + "\t\tny tilstand: " + ny_tilstand);
    return ny_tilstand;
}
```

**Oppgave 8a: bitstreng som starter på 0, og inneholder nøyaktig 1 ener**

**Oppgave 8b: strenger som starter med ab**

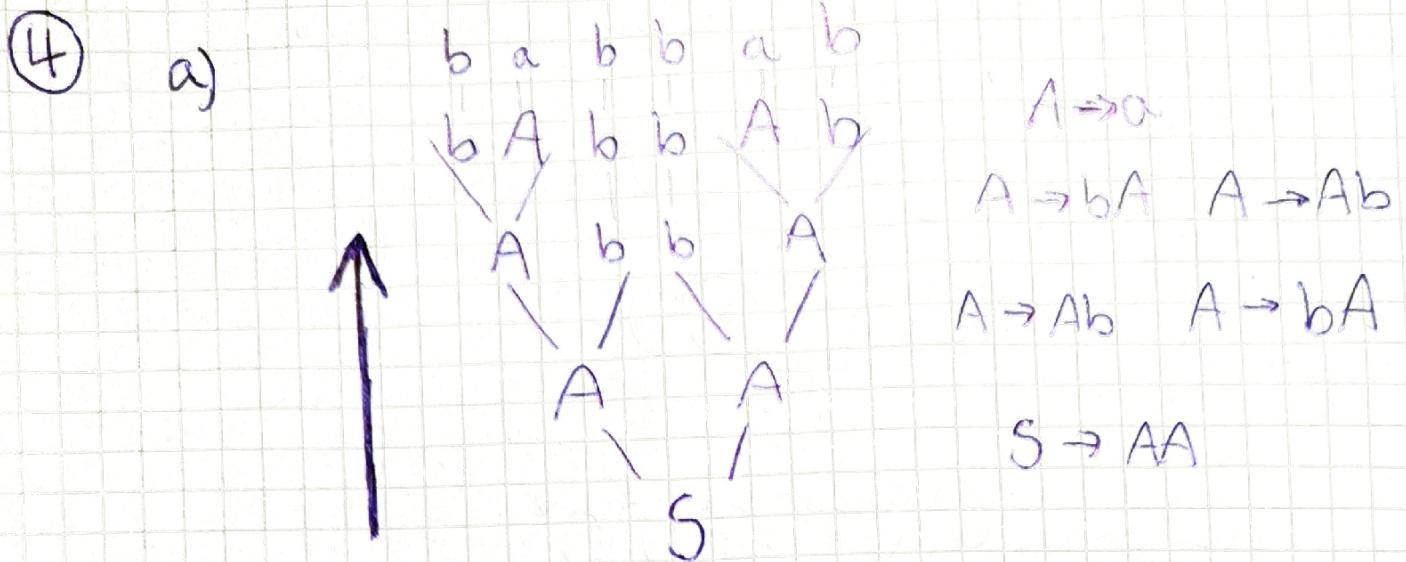
'abbb' TRUE

'aaab' FALSE

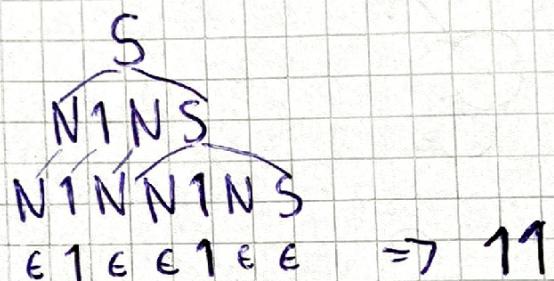
'babab'      TRUE

'bbba' FALSE

Process finished with exit code 0



b) kan vi se dette med et eksempel:



altså kan vi få bitstringer med partells antall enere. Samt ikke passar med spesifik med oddetalls enere.

(5)  $\Sigma = \{ p, q, r, (,), !, ||, \&\& \}$

$$V = \{ S, B, Q, O, E, p, q, r, (,), !, ||, \&\& \}$$

(1)  $S \rightarrow B$

(2)  $B \rightarrow \{ (B), !B, BQ \}$

(3)  $B \rightarrow \{ p, q, r \}$

(4)  $Q \rightarrow \{ O, E \}$

(5)  $O \rightarrow \&\&B$

(6)  $E \rightarrow ||B$