

**Løsningsforslag**  
**Skriftlig eksamen**

**6. januar 2010**

Version 3, 2010-01-20

**Spørgsmål 1**

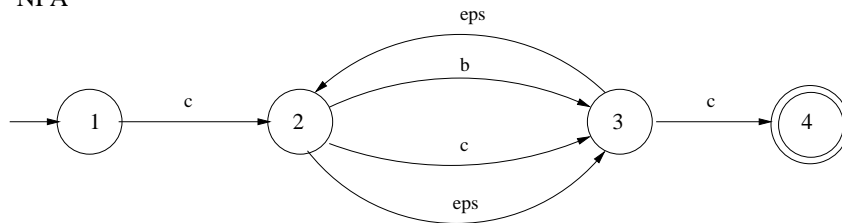
**Spørgsmål 1.1**

Nogle eksempler på strenge der genkendes er  $cc$ ,  $cbc$ ,  $ccc$ ,  $cbbc$ ,  $cccc$ ,  $cbcc$ ,  $ccbc$ , ...

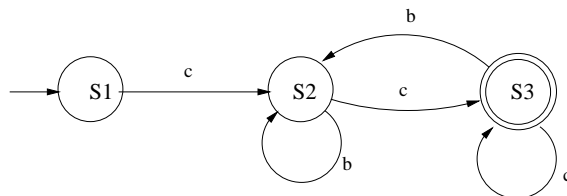
Mere generelt sagt beskriver udtrykket netop de sekvenser der starter og slutter med  $c$  og som derimellem har en vilkårlig sekvens af nul eller flere  $b$ 'er og  $c$ 'er.

**Spørgsmål 1.2 og 1.3**

NFA



DFA



**Spørgsmål 1.4**

Mængden af strenge der opregnes består tilsyneladende af alle strenge der består udelukkende af  $a$ 'er, dog mindst ét; samt alle strenge der består udelukkende af et ulige antal  $b$ 'er.

Den mængde kan beskrives med det regulære udtryk  $aa^*|b(bb)^*$ , eller mere eksplicit  $(aa^*)|(b(bb)^*)$ .

## Spørgsmål 2

### Spørgsmål 2.1

```
let rec rollMany n s =  
  if n=0 then [] else rollOne s :: rollMany (n-1) s;;
```

### Spørgsmål 2.2

```
let rollSum s n = List.fold (fun x y -> x+y) 0 (rollMany s n);;
```

Opgaven kan selvfølgelig også løses ved at skrive en funktion `sum : int list -> int` og kombinere den med `rollMany`, eller skrive en analog til `rollMany` som direkte summerer i stedet for at opbygge en liste af resultater; eller bruge funktionen `List.sum` hvis man kan overbevise sig om at den findes ved at slå op i en bog.

### Spørgsmål 2.3

```
let rec count p xs =  
  match xs with  
  | [] -> 0  
  | x::xr -> if p x then 1 + count p xr else count p xr;;
```

Opgaven kan også løses ved at bruge `List.filter` og `List.length`, eller ved at bruge `List.fold`.

### Spørgsmål 2.4

```
let rec take n xs =  
  match xs with  
  | [] -> []  
  | x::xr -> if n=0 then [] else x :: take (n-1) xr;;  
  
let smallest n xs = take n (List.sort xs);;
```

Bemærk at `take n xs` returnerer den tomme liste hvis `xs` er den tomme liste. Det betyder at `smallest n xs` returnerer den tomme liste hvis `n` er større end længden af `xs`.

### Spørgsmål 2.5

```
let largest n xs = take n (List.rev (List.sort xs));;
```

### Spørgsmål 3

#### Spørgsmål 3.1

```
Assign(AccVar "x", Prim2("+", Access(AccVar "x"), Prim2("*", CstI 2, Access(AccVar "y"))))
3 * (x + y)
```

#### Spørgsmål 3.2

```
While(Prim2(">", Access(AccVar "x"), CstI 0),
      Expr(Assign(AccVar "x", Prim2("-", Access(AccVar "x"), CstI 1))))

While(Prim2("!=" , Access(AccIndex(AccVar "arr", Access(AccVar "i"))),
        CstI 0),
      Expr(Assign(AccVar "i", Prim2("+", Access(AccVar "i"), CstI 1))))
```

#### Spørgsmål 3.3

Stak	Instruktion
(tom)	CSTI 5
5	CSTI 3
5 3	SUB
2	CSTI 4
2 4	CSTI 6
2 4 6	MUL
2 2 4	ADD
2 6	

#### Spørgsmål 3.4

Vi bruger oversættelseskemaerne fra slides til forelæsning 8, eller PLCSD kapitel 8.6-8.8, hvilket giver noget i denne stil (hvis man ikke optimerer koden ved at fjerne `CSTI 0 ; ADD` og den slags):

```
L1: GETBP
    CSTI 0
    ADD
    LDI
    CSTI 0
    SWAP
    LT
    IFZERO L2
    GETBP
    CSTI 0
    ADD
    GETBP
    CSTI 0
    ADD
    LDI
    CSTI 1
    SUB
    STI
    INCSP -1
    GOTO L1
L2:
```

## Spørgsmål 4

### Spørgsmål 4.1

```
10 + sum 4d6
```

```
Plus(Sum(Roll(4,6)), Count(2, Lt, Roll(3,10)))
```

### Spørgsmål 4.2 og 4.3

```
Expr:
  CSTINT                { CstI $1          }
|  ROLL CSTINT          { Roll(1, $2)      }
|  CSTINT ROLL CSTINT  { Roll($1, $3)     }
|  SUM Expr             { Sum $2           }
|  LARGEST CSTINT OF Expr { Largest($2, $4)  }
|  SMALLEST CSTINT OF Expr { Smallest($2, $4) }
|  COUNT CSTINT GT Expr { Count($2, Gt, $4) }
|  COUNT CSTINT LT Expr { Count($2, Lt, $4) }
|  Expr PLUS Expr      { Plus($1, $3)     }
|  LPAR Expr RPAR      { $2               }
;
```

Dette giver en masse skift-reducer konflikter hvis ikke man giver PLUS lav præcedens.

### Spørgsmål 4.4

```
let rec size e : int =
  match e with
  | CstI _ -> 1
  | Roll(n, s) -> n
  | Sum e -> let res = size e
             in 1
  | Largest(n, e) -> let em = size e
                    in if n<=em then n
                      else failwith "Size error in largest"
  | Smallest(n, e) -> let em = size e
                    in if n<=em then n
                      else failwith "Size error in smallest"
  | Count(_, _, e) -> (size e; 1)
  | Plus(e1, e2) ->
    let em1 = size e1
    let em2 = size e2
    in if em1=em2 then em2
       else failwith "Size error in plus";;
```

I tilfældet `Sum e` har resultatet naturligvis altid størrelse én, men udtrykket er kun velformet hvis `e` er velformet. Derfor beregnes størrelsen af `e`, men resultatet af dette ignoreres, og der returneres 1.