

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

NORSK

Eksamen i	:	INF-121A Funksjonell programmering
Dato	:	5 desember 2014
Tid	:	9:00 – 12:00
Antall sider	:	2
Tillatte hjelpemidler	:	ingen

- Løsninger av delproblemer som du ikke har besvart kan antas gitt dersom de trenges i andre delproblemer.
- Prosentsatsene angir *kun omtrentlig* vektning ved sensur og forventet tidsforbruk
- Programmer og forklar alle hjelpefunksjoner som du selv innfører.
- Korrekthet er viktigst, men du får ekstra uttelling for konsis og lesbar kode.

1 Grammatikk (20%)

1.1. Skriv en entydig grammatikk for et språk L over alfabetet $\{a, b, c\}$ som inneholder alle (og kun) strenger med minst to substringer “ab”. F.eks., følgende strenger er med i L :

– abab, ccbabbacabcaaa, abcabab, ...

mens følgende er ikke med i L :

– aba, abadb, aabb, ...

1.2. Programmer en Haskell funksjon `trans :: String -> String` som returnerer, for input `s`

1. en streng som inneholder ”FEIL!”, dersom `s` ikke er med i L
2. `s` med alle forekomster av “ab” erstattet med “d”, dersom `s` er med i L .

F.eks., `trans ‘‘abcabba’’ = ‘‘dcdba’’`, `trans ‘‘abcab’’ = ‘‘dcd’’`, mens `trans ‘‘abc’’` gir en streng “...FEIL...”, der “...” kan avhenge av input og din implementasjon.

2 Kombinasjoner og typer (40%)

Spesifiser eventuelle antakelser om typer av funksjoner (f.eks. at de er `Eq`, `Ord`, el.l.). Du kan anta at inputlister ikke inneholder noen duplikater, men si om du bruker denne antakelsen eller ikke.

2.1. Programmer Haskell funksjon `tails :: [a] -> [[a]]` som returnerer alle suffiksene av inputlisten, f.eks. `tails [1,2,3] = [[1,2,3], [2,3], [3], []]`.

2.2. Programmer Haskell funksjon `com :: Int -> [t] -> [[t]]` som returnerer alle mulige valg av n elementer (n gitt som første argumentet) fra en liste gitt i andre argumentet. Elementer kan velges gjentatte ganger og i forskjellige rekkefølger. F.eks.

`com 1 [1,2,3] = [[1], [2], [3]]`

`com 2 [1,2,3] = [[1,1], [1,2], [1,3], [2,1], [2,2], [2,3], [3,1], [3,2], [3,3]]`

2.3. Programmer funksjon `combi :: Int -> [t] -> [[t]]` som genererer alle *forskjellige* kombinasjoner av n *forskjellige* elementer fra listen, dvs. valgene som nå returneres, ulikt valgene fra

`com`, (1) aldri inneholder noen repetisjoner av elementer (som `[1,1]`) og (2) aldri inneholder to lister med samme elementer (som `[1,2]` og `[2,1]`). F.eks.:

```
combi 2 [1,2,3] = [[1,2],[1,3],[2,3]]
```

```
combi 2 [1,2,3,4] = [[1,2],[1,3],[2,3],[1,4],[2,4],[3,4]]
```

[Hint: Funksjonen `com` eller `tails` kan komme til nytte.]

2.4. Angi typen til Haskell funksjon `f` definert på følgende måten: `f x y = x y y`.

3 Gray sekvenser

(40%)

En Gray sekvens er en sekvens av alle bitstrenger (over $\{0,1\}$) av samme lengde, der to suksessive strenger har nøyaktig én bit forskjellig. Argumentet til `gray` angir lengden på strengene i sekvensen (vi dropper “...” rundt strenger for å øke lesbarhet):

```
gray(1) = [0,1]
```

```
gray(2) = [00,01,11,10]
```

```
gray(3) = [000,001,011,010,110,111,101,100]
```

.....

Gitt $gray(n) = [x_1, x_2, \dots, x_{2^n}]$, kan du generere neste sekvens $gray(n+1)$ på din egen måte (som du i så fall må forklare), men en mulig oppskrift er som følger:

1. dupliser hvert element i $gray(n)$, $dup(n) = [x_1, x_1, x_2, x_2, \dots, x_{2^n}, x_{2^n}]$
2. konkatener passende antall sekvenser $[0,1,1,0]$, slik at resulterende sekvens har samme lengde som $dup(n)$ fra 1 (f.eks., har $dup(n)$ lengde 8, blir resultatet $[0,1,1,0]++[0,1,1,0]$, har den lengde 16, blir resultatet $[0,1,1,0]++[0,1,1,0]++[0,1,1,0]++[0,1,1,0]$, osv.)
3. legg sammen de to sekvensene ved å konkatenerer, elementvis, hvert element fra sekvensen 2 etter tilsvarende element fra $dup(n)$, dvs. ($x0$ betegner her Haskells konkatenering av strenger `x++‘0’`, osv.)

$$\begin{aligned} gray(n+1) = & [x_10, x_11, x_21, x_20, \\ & x_30, x_31, x_41, x_40, \\ & \dots \\ & \dots, x_{2^n}1, x_{2^n}0] \end{aligned}$$

Som eksempel, se de siste tegnene i $gray(3)$ sekvens: `[000, 001, 011, 010, 110, 111, 101, 100]`.

3.1. Programmer Haskell funksjon `gray::Int->[String]` (samt alle hjelpefunksjoner) for å generere Gray sekvensen for tallet oppgitt som argument.

3.2. Programmer Haskell funksjon `diffen::[String]->Bool` som sjekker om inputsekvensen oppfyller kravet om at hvert par av suksessive strenger har nøyaktig én bit forskjellig. Den skal altså være slik at `diffen(gray n) == True` for enhver `n` men, f.eks.,

`diffen [‘00’, ‘01’, ‘10’, ‘11’] == False`, siden to suksessive strengene `‘01’` og `‘10’` har forskjellige bitene i begge posisjonene.

Lykke til!
Michał Walicki

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

NORSK

Eksamen i	: INF-121 Programmeringsparadigmer
Dato	: 5 desember 2014
Tid	: 9:00 – 12:00
Antall sider	: 2
Tillatte hjelpemidler	: ingen

- Løsninger av delproblemer som du ikke har besvart kan antas gitt dersom de trenges i andre delproblemer.
- Prosentsatsene ved hver oppgave angir *kun omtrentlig* vektning ved sensur og forventet tidsforbruk
- Korrekthet er viktigst, men du får ekstra uttelling for konsis og lesbar kode.

1 Grammatikk (10%)

1.1. Skriv en entydig grammatikk for et språk L over alfabetet $\{a, b, c\}$ som inneholder alle (og kun) strenger med minst to substrenger “ab”. F.eks., følgende strenger er med i L :

– abab, ccbabbacabcaaaa, abcabcab, ...

mens følgende er ikke med i L :

– aba, abadb, aabb, ...

1.2. Programmer en Haskell funksjon `trans :: String -> String` som returnerer, for input s

1. en streng som inneholder ”FEIL!”, dersom s ikke er med i L
2. s med alle forekomster av “ab” erstattet med “d”, dersom s er med i L .

F.eks., `trans ‘‘abcabba’’ = ‘‘dcdba’’`, `trans ‘‘abcaab’’ = ‘‘dcd’’`, mens `trans ‘‘abc’’` gir en streng “...FEIL...”, der “...” kan avhenge av input og din implementasjon.

2 Haskell (40%)

Spesifiser eventuelle antakelser om typer av funksjoner (f.eks. at de er `Eq`, `Ord`, el.l.). Du kan anta at inputlister ikke inneholder noen duplikater, men si om du bruker denne antakelsen eller ikke.

2.1. Programmer Haskell funksjon `tails :: [a] -> [[a]]` som returnerer alle suffiksene av inputlisten, f.eks. `tails [1,2,3] = [[1,2,3], [2,3], [3], []]`.

2.2. Programmer Haskell funksjon `com :: Int -> [t] -> [[t]]` som returnerer alle mulige valg av n elementer (n gitt som første argumentet) fra en liste gitt i andre argumentet. Elementer kan velges gjentatte ganger og i forskjellige rekkefølger. F.eks.

`com 1 [1,2,3] = [[1], [2], [3]]`

`com 2 [1,2,3] = [[1,1], [1,2], [1,3], [2,1], [2,2], [2,3], [3,1], [3,2], [3,3]]`

2.3. Programmer funksjon `combi :: Int -> [t] -> [[t]]` som genererer alle *forskjellige* kombinasjoner av n *forskjellige* elementer fra listen, dvs. valgene som nå returneres, ulikt valgene fra

`com`, (1) aldri inneholder noen repetisjoner av elementer (som `[1,1]`) og (2) aldri inneholder to lister med samme elementer (som `[1,2]` og `[2,1]`). F.eks.:

```
combi 2 [1,2,3] = [[1,2],[1,3],[2,3]]
```

```
combi 2 [1,2,3,4] = [[1,2],[1,3],[2,3],[1,4],[2,4],[3,4]]
```

[Hint: Funksjonen `com` eller `tails` kan komme til nytte.]

2.4. Angi typen til Haskell funksjon `f` definert på følgende måten: `f x y = x y y`.

3 Prolog

(25%)

3.1. Definer Prolog relasjon `trans(+A,B)` som returnerer `true` hvis og bare hvis `A` er (en liste med tegn tilsvarende) en streng med i språket L fra oppgave 1, og `B` er en liste med tegn der hver forekomst av `a,b` (dvs. en `a` etterfulgt umiddelbart av en `b`) i `A` er erstattet med `d`. F.eks., skal `trans([a,b,c,a,b,b,a],[d,c,d,b,a])` og `trans([a,b,c,a,b],[d,c,d])` gi `true`. Det andre argumentet må kunne genereres: spørringen `trans([a,b,a,b,a,b,c,b],B)` skal gi `B=[d,d,d,c,b]` (og helst ingen flere svar).

3.2. Definer Prolog relasjon `prod(+A,+B,P)` som holder hvis og bare hvis listen `P` er kartesisk produkt av lister `A` og `B`, dvs. en liste med alle par `X:Y` der `X` forekommer i `A` og `Y` i `B`, f.eks., `prod([1,2],[3,4],[1:3,1:4,2:3,2:4])`. Du kan anta at hverken `A` eller `B` inneholder noen duplikater.

3.3. Hva blir resultatet av spørringen `X is 5+3, Y=X+X`. gitt til Prolog?

4 Søketrær

(25%)

Vi betrakter følgende Prolog klausuler:

```
nabo(a,b).  
nabo(a,c).  
nabo(b,d).  
nabo(X,Y) :- nabo(Y,X).
```

4.1. Tegn en representativ del av SLD-treet (ev. forklar hvordan resten av treet vil se ut) for spørringen `nabo(a,X)`. Hva blir svarene fra Prolog (ved gjentatte ;) til denne spørringen?

4.2. Vi forandrer nå siste klausulen til `nabo(X,Y) :- nabo(Y,X), !`. Tegn SLD-treet til spørringen `nabo(a,X)`. Hva blir svarene fra Prolog (ved gjentatte ;) til denne spørringen?

Lykke til!
Michał Walicki

Problem 1 – solution

1.1. Grammatikken kan være (S er startsymbolet):

```
S := aB | bS | cS
B := aB | bC | cS
C := aD | bC | cC
D := aD | bF | cC
F := aF | bF | cF | e
```

1.2. Vi definerer `trans s = tr s 0` der andre argumentet til `tr` teller antall ab'er:

```
tr "" n = if n>1 then "" else "FEIL!"
tr ('a':'b':s) i = 'd': tr s (i+1)
tr ('a':s) i = 'a': tr s i
tr ('b':s) i = 'b': tr s i
tr ('c':s) i = 'c': tr s i
tr (_:s) i = "FEIL!"
```

Problem 2 – solution

2.1.

```
tails [] = [ [] ]
tails (x:xs) = (x:xs) : tails xs
```

2.2.

```
com 0 xs = [[]]
com n xs [x:ls | x <- xs, ls <- com (n-1) xs]
```

2.3. En mulighet er å programmere alt stykkvis, f.eks.:

```
remDup [] = []
remDup (x:xs) = if elem x xs then remDup xs else x: remDup xs
combi n xs = let li = map sort (com n xs); nod = map remDup li in
              filter (\x -> length x == n) (remDup nod)
```

men med `tails` kan det bli mer elegant:

```
combi 0 _ = [ [] ]
combi n xs = [ y:ys' | y:xs' <- tails xs, ys' <- combi (n-1) xs' ]
```

2.4. Typen av funksjonen `fun f x y = x y y` er `f :: (a->a->b) -> a -> b`.

Problem 3 – solution

3.1. Teller, som i Haskell, antall 'ab'er på veien:

```

trans(S,T) :- trans(S,T,0).
trans([],[],N) :- N > 1.
trans([a,b|H],[d|R],N) :- M is N+1, !, trans(H,R,M).
trans([c|H],[c|R],N) :- trans(H,R,N).
trans([a|H],[a|R],N) :- trans(H,R,N).
trans([b|H],[b|R],N) :- trans(H,R,N).

```

Snitt er nødvendig for å utelukke feile svar, f.eks., uten snitt ville $\text{trans}([a,b,a,b,a,b],T)$ gi også $T = [d,d,a,b]$, mm.

3.2. Enkelt svar : $\text{prod}(A,B,X:Y) :- \text{member}(X,A), \text{member}(Y,B)$. gir kun noen få poeng, siden siste argument skal være hele listen av slike. Riktig svar er, f.eks.:

$\text{prod}(A,B,P) :- \text{setof}(X:Y, (\text{member}(X,A), \text{member}(Y,B)), P)$. Man kan også programmere det eksplisitt, f.eks.:

```

pr(A,[],[]).
pr(A,[B|T],[A:B|R]) :- pr(A,T,R).
prod([],_,[]).
prod(_,[],[]).
prod([A|As],Bs,Res) :- pr(A,Bs,R), prod(As,Bs,Rs), append(R,Rs,Res).
\end{cerbatim}

```

```

\delo {X=8, Y=8+8.}

```

```

\noo{ \sol{unif}\setcounter{section}{4}\setcounter{del}{0}
\delo {0occurs-check gir failure.}
\delo {X=8, Y=8+8.} }

```

$\text{\sol{SLD} \setcounter{section}{5}\setcounter{del}{0}}$ Det uendelige treet til venstre er for 5.1, mens snitt i 5.2 skj\ae rer det etter f\o rste vellykkede gren for det symmetriske delm\aa let i siste klausul, dvs. svarene blir kun $\text{\texttt{X=b;X=c;X=b;false.}}$

```

\[\begin{array}{c@{\hspace*{5em}}c}
\mathrm{xymatrix@R=0.5cm{ 5.1 \& \& nabo(a,X) \ar[dll] \ar[dl] \ar[d] \& \& \\
X=b \& X = c \& nabo(X,a) \ar[d] \& \& \\
\& \& nabo(a,X) \ar[dll] \ar[dl] \ar[d] \& \& \\
X=b \& X = c \& nabo(X,a) \ar[d] \& \& \\
\& \& }

```

&

```

\mathrm{xymatrix@R=0.5cm{ 5.2 \& \& nabo(a,X) \ar[dll] \ar[dl] \ar[d] \& \& \\
X=b \& X = c \& nabo(X,a), ! \ar[d] \& \& \\
\& \& nabo(a,X) \ar[dll] \& \& \% \ar[dl] \ar[d] \& \& \\
X=b \& \& \texttt{false.} \% X = c \& \& nabo(X,a) \ar[d] \& \& \\
\& \& }\vspace*{3ex}

```

```

\end{array}\]

```

%

```

\dotfill{INF-121-A}\dotfill

```

```

\sol{gray}\setcounter{section}{3}\setcounter{del}{0}

```

```

\delo Vi har at
$length(dup(gray(n))) = 2*2^n = 2^{n+1} = 4^n = (length(0110))^n$, s{\aa} vi
kunne bruke en spesifikk funksjon for {\aa} konkatenerer n kopier av
0110 og kalle
\texttt{(app "0110" n)} istedenfor \texttt{(concat (repeat "0110"))}. Men denne l{\o}sning
\begin{verbatim}
gray 1 = ["0","1"]
gray n = let g = gray (n-1) in
          map (\(xs,x) -> xs++[x]) (zip (dup g) (concat (repeat "0110")))
dup [] = []
dup (x:xs) = x:x:dup xs

```

3.3. `diff` bytter ut hver differanse mellom argumentene med en 1 og fjerner alle likheter. `diffen` sjekker etterpå om alle elementene i sekvensen `zip xs (tail xs)` er “1”.

```

diff("", "") = ""
diff ((x:xs),(y:ys)) = if x==y then diff(xs,ys)
                        else '1':diff(xs,ys)

difen xs = all (=="1") (map diff (zip xs (tail xs)))

```