UNIVERSITETET I BERGEN NORSK

Det matematisk-naturvitenskapelige fakultet

Eksamen i	:	INF-121A Funksjonell programmering
Dato	:	6 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* vekting 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 substrenger "ab". F.eks., følgende strenger er med i L:
– abab, ccbabbacabcaaa, 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 ‘‘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 èn 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 $gray(n) = [x_1, x_2, ..., x_{2^n}]$, 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), $dup(n) = [x_1, x_1, x_2, x_2, ..., x_{2^n}, x_{2^n}]$
2. lag en sekvens ved å konkatenere passende antall sekvenser [0,1,1,0] (slik at lengden av resultatet blir likt lengden av $dup(n)$ fra 1.)
3. legg sammen de to sekvensene ved å konkatenere, elementvis, hvert element fra sekvensen 2 etter tilsvarende element fra $dup(n)$, dvs. ($x\mathbf{0}$ betegner her Haskells konkatenering av strenger `x++''0''`, osv.)

$$\begin{aligned} gray(n+1) \quad = \quad & [x_1\mathbf{0}, x_1\mathbf{1}, x_2\mathbf{1}, x_2\mathbf{0}, \\ & x_3\mathbf{0}, x_3\mathbf{1}, x_4\mathbf{1}, x_4\mathbf{0}, \\ & \quad ... \\ & ..., x_{2^n}\mathbf{1}, x_{2^n}\mathbf{0}] \end{aligned}$$

Som eksempel, se de siste tegnene i $gray(3)$ sekvens: [00**0**, 00**1**, 01**1**, 01**0**, 11**0**, 11**1**, 10**1**, 10**0**].

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	:	6 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* vekting 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, ccbabbacabcaaa, 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 ``abcab'' = ``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 umidelbart 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 SLD traer (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