Hjemmeeksamen 1 i INF3110/4110

Innleveringsfrist: fredag 24. oktober kl. 1500

Innlevering

Hele besvarelsen skal leveres skriftlig på papir i IFI-ekspedisjonen innen fredag 24. oktober kl. 1500. Merk besvarelsen med navn, samt ditt brukernavn ved IFI.

Programkoden fra Del 1 skal også sendes per e-post innen den samme fristen. Hvis du er på gruppe *x*, så sender du e-post til `inf3110-`*x*`@ifi.uio.no`. Programkoden skal sendes som vedlegg og ha navnet *brukernavn*`.sml` hvor *brukernavn* er ditt brukernavn (!) ved IFI.

Fristen er absolutt; det gis ingen utsettelse uten gyldig grunn. Det gis heller ingen mulighet for å ta denne hjemmeeksamen på nytt etter innlevering.

Samarbeid og egenerklæring

Denne oppgaven er ment som en individuell oppgave, siden resultatet av oppgaven inngår som en del av den samlede karakteren for kurset, men det er også tillatt å 澳门葡京手机版app下载e to og to. I så tilfelle skal begge levere hver sin oppgave, men skrive klart og tydelig på forsiden hvem de har 澳门葡京手机版app下载et med. Begge vil da få samme uttelling for besvarelsen.

Alle som leverer må også fylle ut og underskrive en egenerklæring som leveres sammen med besvarelsen. Dette er et krav for å kunne gå opp til eksamen. Skjemaet kan du laste ned og skrive ut herfra:
Norsk versjon: `http://www.ifi.uio.no/studinf/skjemaer/erklaring.pdf`
Engelsk versjon: `http://www.ifi.uio.no/studinf/skjemaer/declaration.pdf`

Vurdering

Eksamensform i dette kurset består av to hjemmeeksamener og en avsluttende 3-timers eksamen. Begge hjemmeeksamenene teller 20%; den avsluttende eksamen teller 60%. For å gjøre regningen noe enklere vil du kunne oppnå 20 *poeng* på hver hjemmeeksamen, samt 60 poeng på avsluttende eksamen. Til slutt vil din poengsum bli regnet om til en bokstavkarakter. Manglende innlevering av hjemmeeksamen gir 0 poeng.

Annen informasjon

Filen med programkode skal være *kjørbar* i SML/NJ, dvs. kunne hentes inn ved å bruke `use "`*brukernavn*`.sml";`.

Kommenter programkoden slik at den blir forståelig!

Hvis du ikke klarer alt i en oppgave, så forklar hvordan du har tenkt og hvor langt du har kommet.

Lykke til!

Del 1 - Programmering i ML

I denne oppgaven skal du lage funksjoner som gjør operasjoner på trær. Følgende datatype skal benyttes:

```
datatype 'a tre = Node of 'a | Tre of 'a * ('a tre list);
```

I Figur 1 finner du eksempler på trær av heltall (tegnet med roten nederst) og hvordan disse kan representeres som verdier av typen over. Du kan bruke disse til å teste funksjonene dine.

Et minstekrav til funksjonene dine er at de gjør det de skal. Utover det oppfordres du til å skrive enkle og elegante funksjoner. Du står fritt til å benytte alle de deler av språket vi har gått igjennom til nå. Husk å kommentere det du gjør.

(1 poeng) Oppgave 1 `dybde : 'a tre -> int`

Skriv en ML-funksjon `dybde` av type `('a tre -> int)` som gir lengden på den lengste grenen i treet. Vi sier at en enkelt node har dybde 0.

Eksempler:

`dybde(tre1)` gir 0	`dybde(tre2)` gir 1	`dybde(tre3)` gir 1
`dybde(tre4)` gir 2	`dybde(tre5)` gir 3	`dybde(tre6)` gir 2

(1 poeng) Oppgave 2 `forgrening : 'a tre -> int`

Skriv en ML-funksjon `forgrening` av type `('a tre -> int)` som gir størrelsen på den største forgreningen i treet.

Eksempler:

`forgrening(tre1)` gir 0	`forgrening(tre2)` gir 2	`forgrening(tre3)` gir 3
`forgrening(tre4)` gir 4	`forgrening(tre5)` gir 3	`forgrening(tre6)` gir 3

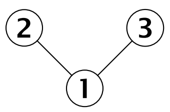
(1 poeng) Oppgave 3 `antall : 'a tre -> int`

Skriv en ML-funksjon `antall` av type `('a tre -> int)` som gir antall elementer i treet.

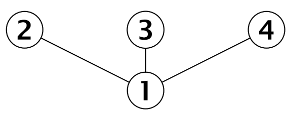
Eksempler:

`antall(tre1)` gir 1	`antall(tre2)` gir 3	`antall(tre3)` gir 4
`antall(tre4)` gir 6	`antall(tre5)` gir 8	`antall(tre6)` gir 13

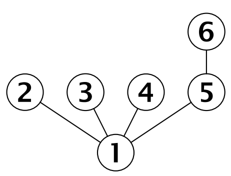
```
val tre1 = Node(1);
```



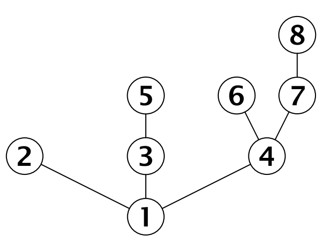
```
val tre2 = Tre(1,[Node(2), Node(3)]);
```



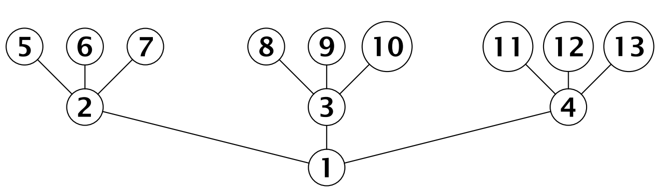
```
val tre3 = Tre(1,[Node(2), Node(3), Node(4)]);
```



```
val tre4 = Tre(1, [Node(2),
                   Node(3),
                   Node(4),
                   Tre(5, [Node(6)])]);
```



```
val tre5 = Tre(1, [Node(2),
                   Tre(3, [Node(5)]),
                   Tre(4, [Node(6),
                           Tre(7, [Node(8)])])]);
```



```
val tre6 = Tre(1, [Tre(2, [Node(5),
                           Node(6),
                           Node(7)]),
                   Tre(3, [Node(8),
                           Node(9),
                           Node(10)]),
                   Tre(4, [Node(11),
                           Node(12),
                           Node(13)])]);
```

Figur 1: Eksempler på trær

(1 poeng) **Oppgave 4** `flatD : 'a tre -> 'a list`

Skriv en ML-funksjon `flatD` av type (`'a tre -> 'a list`) som *flater ut* et tre ved å gå gjennom treet *dybde først* fra venstre mot høyre. Elementene i listen skal bestå av elementene i treet, og rekkefølgen skal svare til å gå gjennom treet *dybde først*.

Eksempler:

`flatD(tre5)` gir `[1,2,3,5,4,6,7,8]`.
`flatD(tre6)` gir `[1,2,5,6,7,3,8,9,10,4,11,12,13]`.

(2 poeng) **Oppgave 5** `flatB : 'a tre -> 'a list`

Skriv en ML-funksjon `flatB` av type (`'a tre -> 'a list`) som *flater ut* et tre ved å gå gjennom treet *bredde først* fra venstre mot høyre. Elementene i listen skal bestå av elementene i treet, og rekkefølgen skal svare til å gå gjennom treet *bredde først*.

Eksempler:

`flatB(tre5)` gir `[1,2,3,4,5,6,7,8]`.
`flatB(tre6)` gir `[1,2,3,4,5,6,7,8,9,10,11,12,13]`.

(2 poeng) **Oppgave 6** `trelik : ''a tre -> ''a tre -> bool`

Skriv en ML-funksjon `trelik` av type (`''a tre -> ''a tre -> bool`) som sjekker om to trær er like. For at to trær skal være like, så må alle elementene som de er bygget av være like og de må ha samme struktur.

Eksempler:

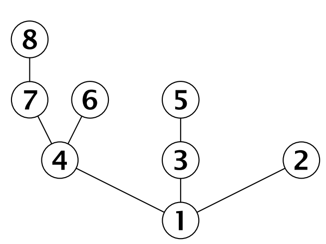
`trelik tre2 (Tre(1,[Node(2), Node(3)]))` gir `true`
`trelik tre2 (Tre(1,[Node(3), Node(2)]))` gir `false`
`trelik tre1 tre2` gir `false`

(2 poeng) **Oppgave 7** `speil : 'a tre -> 'a tre`

Skriv en ML-funksjon `speil` av type (`''a tre -> ''a tre`) som *speilvender* et tre.

Eksempel:

`speil tre5` gir følgende tre:




```
val tre51 = Tre(1, [Tre(4, [Tre(7, [Node(8)]),
                            Node(6)]),
                    Tre(3, [Node(5)]),
                    Node(2)]);
```

Del 2 - Syntaks

(3 poeng) Oppgave 8

a) Definer en regulær grammatikk (i klassisk BNF) over alfabetet {1,2,3} som kun tillater ikke-tomme strenger hvor alle forekomster av 3 kommer *til venstre for* alle forekomster av 2 og alle forekomster av 2 kommer *til venstre for* alle forekomster av 1.

Eksempler på *tillatte* strenger: 1, 2, 21, 321, 33333222, 33311, 3111, 3211, 333

Eksempler på *ikke-tillatte* strenger: 12, 13, 11113, 2223

b) Definer den samme grammatikken i *utvidet* BNF.

c) Gi en endelig, deterministisk automat som godkjenner nøyaktig setningene i dette språket.

d) Utvid grammatikken (og alfabetet) over slik at den også tillater strenger med tegnet + hvor tegnet *umiddelbart til høyre for* + må være 1, 2 eller 3. (Tenk på + som et tegn som "nullstiller" rekkefølgen, slik at man kan begynne på et større tall igjen.)

Eksempler på *tillatte* strenger: 1, 2, 21+1, 21+331, 1+1+1+31

Eksempler på *ikke-tillatte* strenger: 12, 13, 1+13, 331+, +21

e) Definer den samme grammatikken i *utvidet* BNF.

f) Gi til slutt en endelig, deterministisk automat som godkjenner nøyaktig setningene i dette språket.

(2 poeng) Oppgave 9

Se på følgende grammatikk over alfabetet {(,)}, dvs. de eneste tegnene er venstreparentes og høyreparentes:

$\langle U \rangle \rightarrow$ **()** | **(**$\langle U \rangle$**)** | $\langle U \rangle\langle U \rangle$

a) Vis at grammatikken er *flertydig*.

b) Er grammatikken regulær? Hvis ja, hvorfor? Hvis nei, hvorfor ikke?

c) Modifiser grammatikken slik at den ikke lenger er flertydig. Den skal fortsatt gi *nøyaktig* samme setninger som den opprinnelige.

(3 poeng) **Oppgave 10**

a) Definer en grammatikk over alfabetet {a, b} slik at alle setninger har like mange a'er som b'er.

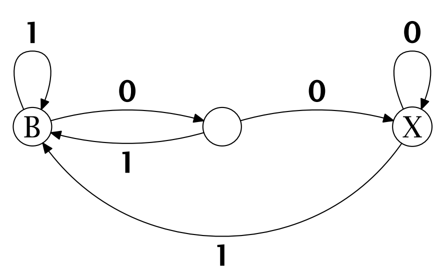
Eksempler på *tillatte* strenger: ε, ab, aabb, aabbba, abab

Eksempler på *ikke-tillatte* strenger: a, bba, aaa

b) Er det mulig å gi en regulær grammatikk for dette språket? Hvis ja, gi den regulære grammatikken. Hvis nei, gi en *kort* begrunnelse for hvorfor det ikke går.

(2 poeng) **Oppgave 11**

Gitt følgende endelige, deterministiske automat:



B angir starttilstanden og X angir slut-tilstanden/aksepterende tilstand.

a) Finn en kort og intuitiv beskrivelse av språket som denne automaten definerer.

b) Skriv et uttrykk i utvidet BNF som uttrykker nøyaktig samme språk.

SLUTT