Informasjon

UNIVERSITETET I OSLO
Det matematisk-naturvitenskapelige fakultet

Eksamen i: INF3110 Programmeringsspråk
Eksamensdato: 29. november, 2019
Eksamenstdispunkt: 14:30, 4 timer
Tillattte hjelpemidler: Alle trykte og skrevne hjelpemidler, inkludert læreboka.

Denne oppgaven består av 3 spørsmål som kan besvares uavhengig av hverandre. Dersom du synes spørsmålsteksten er uklar, så må du gjøre dine egne tolkninger og antakelser. Skriv ned disse som en del av svaret.

I dette oppgavesettet har du mulighet til å svare med digital håndtegning (oppgave 1c, 1e og 2a). Du bruker skisseark du får utdelt. Det er anledning til å bruke flere ark per oppgave. Se instruksjon for utfylling av skisseark på pult.

Det er IKKE anledning til å bruke digital håndtegning på andre oppgaver enn oppgave 1c, 1e og 2a. Det blir IKKE gitt ekstratid for å fylle ut informasjonsboksene på skisseark (engangskoder, kand.nr. o.l.).

De 3 spørsmålene vektes som følger:

Runtime-systemer, syntaks og semantikk: 40 %
ML: 40 %
Prolog: 20 %

Lykke til!

1(a) 1a

Programmeringsspråk er forskjellige på mange måter, og en av disse forskjellene er hvordan typesystemet/typesjekkingen fungerer. Vanligvis skiller vi mellom statisk typede og dynamisk typede språk. Forklar kort hva forskjellen på disse konseptene er, og hvilke fordeler begge varianter kan gi.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 8

1(b) # 1b

Et programmeringsspråk kan også ha statisk eller dynamisk skop. Forklar kort hva forskjellen på disse er. Videre: er det slik at et språk med dynamisk skop alltid er dynamisk typet, og et språk med statisk skop alltid er statisk typet? Argumenter for ditt svar.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 8

1(c) # 1c

I denne oppgaven skal du svare med digital håndtegning. Bruk eget skisseark (utdelt). Se instruksjon for utfylling av skisseark nederst på skjermen.

Følgende Java-program viser om et gitt år er et skuddår eller ikke. Det er skuddår hvert 4. år, unntatt år som kan deles på 100, såfremt det ikke også kan deles på 400.

```
public class LeapYear {
   int[] years = { 1999, 2000, 2001, 2004, 2100 };

   public static void main(String[] argv) {
      for (int year : years) {
         System.out.println(year + ": " + isLeapYear(year));
      }
   }

   public static boolean isLeapYear(int year) {
      boolean leapYear =
         (year % 100 == 0) ?
         (year % 400 == 0) :
         (year % 4 == 0);

      // HERE

      return leapYear;
   }
}
```

Tegn kjøretids-stakken («the runtime stack») med aktiveringsblokker slik vi har gått gjennom på forelesning, når funksjonen **isLeapYear** har blitt kalt med parameteren 2004, og eksekveringen har kommet til punktet «**HERE**» merket i koden.

Maks poeng: 8

1(d) # 1d

Anta nå at vi ønsker å utvide Java slik at man kan velge å bruke dynamisk skop i språket, når man selv ønsker det. Dynamisk skop skal implementeres ved at man kan deklarerer en funksjon med nøkkelordet **dynscope**, f.eks:

public static dynscope boolean isLeapYear(…)

Dette betyr da at inne i funksjonskroppen til **isLeapYear**, så brukes dynamisk skop for å se etter deklarasjoner av variable og metoder.

(i) Skriv funksjonen **isLeapYear** på en slik måte at den gjør bruk av dynamisk skop.

(ii) Skriv en EBNF-gramatikk for funksjonsdefinisjoner som tillater nøkkelordet **dynscope**. Du trenger ikke skrive EBNF for kroppen til funksjonen, og kan anta at vanlige ting som identifikatorer, parameterlister, etc, allerede er definert og tilgjengelig for bruk.

(iii) Det å innføre dynamisk skop i et språk som Java er selvsagt ganske dramatisk, og får en del konsekvenser. Forklar kort hvordan dette kunne blitt implementert, med fokus på kjøresystemet. Nevn også noen viktige ting i språket for øvrig som ville blitt påvirket av en slik endring.

Skriv ditt svar her...

Format | B I U x_2 x^2 I_x | Ω | Σ

Words: 0

Maks poeng: 8

1(e) 1e

I denne oppgaven skal du svare med digital håndtegning (i) og i tekstboksen under (ii). Bruk eget skisseark (utdelt) til håndtegningen. Se instruksjon for utfylling av skisseark nederst på skjermen.

Siden vi er Seriøse Programvareutviklere™, så ønsker vi nå å refaktorere og generalisere funksjonen isLeapYear fra 1c, slik at man kan sjekke et år for flere forskjellige ting. Vi innfører derfor en metode som sjekker et år, og som tar inn en funksjon som parameter, hvor sistnevnte funksjon utfører den faktiske sjekken. For å få til dette, så later vi som om Java har støtte for funksjonsparametere og deklarasjon av indre funksjoner:

```
public class LeapYear {
   public static void main(String[] argv) {
      int[] years = { 1999, 2000, 2001, 2004, 2100 };
      for (int year : years) {
         System.out.println(year + ": " + isLeapYear(year));
      }
   }

   public static boolean checkYear(boolean checker(int), int year) {
      bool result = checker(year);
      System.out.println(result ? "yes!" : "no...");
      // HERE
      return result;
   }

   public static boolean isLeapYear(int year) {
      boolean leapYear(int y) {
         return (y % 100 == 0) ?
           (y % 400 == 0) :
           (y % 4 == 0);
      }

      checkYear(leapYear, year);

      return leapYear;
   }
}
```

(i) Tegn kjøretidsstakken til dette programmet, når eksekveringen har kommet til punktet merket "**HERE**" i koden, og året som sjekkes er 2004.

(ii) Anta at funksjonen **leapYear** nå blir deklarert som følger med **dynscope** fra 1d:

```
boolean dynscope leapYear() {
    return (year % 100 == 0) ? (year % 400 == 0) : (year % 4 == 0);
}
```

Hvilke konsekvenser får dette for resten av programmet? Dersom man skulle tegne kjøretidsstakken til det modifiserte programmet, ville det blitt noen vesentlige forskjeller fra den du tegnet i punkt (i)? Er det noe generelt som du tenker ville blitt gjort annerledes for

funksjonsverdier («closures») på kjøretidsstakken om de benytter dynamisk skop?

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 8

2(a) Type Inference

I denne oppgaven skal du svare med digital håndtegning. Bruk eget skisseark (utdelt). Se instruksjon for utfylling av skisseark nederst på skjermen.

Antar som gitt funksjonen

mkPair : L → R → (L,R)

som bygger opp en tupel. Beregn typen til det følgende uttrykket i henhold til MLs algoritme for typeinferens:

fn f => (mkPair (f 3)) (f "x");

Utled de tilhørende ligningene ved hjelp av en parseringsgraf ("parse graph"), og løs det resulterende ligningssettet.

Maks poeng: 10

2(b) Recursion

Endre følgende enkle funksjon slik at den benytter halerekursjon ("tail recursion"). Legg også til en ny "wrapper-funksjon", som har det samme antall parametere som den opprinnelige **pow**!

```
(* val pow = fn : int -> int -> int *)
fun pow b 0 = 1
|   pow b e = b * pow b (e-1);
```

Skriv ditt svar her...

```
1
```

Maks poeng: 5

2(c) Simple expressions (1/2)

Evaluer følgende ML-uttrykk, eller svar med begrunnelse hvorfor en *type checking*- eller *runtime feil* oppstår:

map (((fn x => x>42)) o (foldl (op +) 0)) [[1,2,3],[100,99,98]];

o = fn : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b er SML sin *infix* komposisjon av funksjoner, som anvender første argument til resultatet av anvendelse av andre argument til noen verdi.

Skriv ditt svar her...

```
1
```

Maks poeng: 5

2(d) Simple expressions (2/2)

Evaluer følgende ML-uttrykk, eller svar med begrunnelse hvorfor en *type checking*- eller *runtime feil* oppstår:

exception E;
let val x = raise E in 42 handle E => 0 end;

Skriv ditt svar her...

```
1
```

Maks poeng: 5

2(e) Simple programming

Definér funksjonen **elem = fn : “a → “a list → bool** som blir sant når første argument forkommer minst én gang i listen gitt som andre argument.

I tillegg, skriv en funksjon **noDup = fn : ‘’a list → bool** som blir sant når listen inneholder ingen duplikater, fortrinnsvis ved hjelp av “**elem**”.

Skriv ditt svar her...

```
1
```

Maks poeng: 3

2(f) Data structures

Vi modellerer et enkelt konferanse-håndteringssystem i ML. En konferanse er organisert som et antall sesjoner. Hver sesjon har et emne («topic»), blir tilordnet et rom (angitt med et heltall) og et starttidspunkt (vi bruker et heltall for dette også).

Et emne («topic») kan være en presentasjon, som vi modellerer med navnet på den som holder presentasjonen, og presentasjonens tittel. Det finnes to typer presentasjoner, vanlige presentasjoner, og såkalte «keynotes». De viktigste typene av sesjoner er selvsagt kaffe- og lunsjpausene!

Dette gir oss følgende typer så langt:

```
type Room = int;
type Start = int;
type Person = string;

datatype BreakType = (* Fullføres under punkt (i) *)
datatype Topic =     (* Fullføres under punkt (i) *)

datatype Session = Session of Topic * Room * Start;
```

Hver sesjon har en sesjonsleder («session-chair»), som er en person som har ansvar for at alt det praktiske rundt sesjonen flyter godt. En tilordning av sesjonsledere til sesjoner kaller vi en **DutyRoster**, og vi modellerer dette gjennom en funksjon som gitt en sesjon, returnerer personen som leder sesjonen:

```
type DutyRoster = Session → Person
```

Når vi setter dette sammen, så får vi at en konferanse kan modelleres som følger:

```
datatype Conference = Conference of Session list * DutyRoster;
```

i. Definér datatypen **Topic** med ulike konstruktører for en keynote-presentasjon, en vanlig presentasjon, og en pause. Begge presentasjonstypene skal ha med en streng som angir navnet på den som presenterer, og presentasjonens tittel. Definér også en datatype **BreakType** som angir om pausen er en kaffepause eller lunsjpause.
ii. Skriv en funksjon **checkConf = fn : Conference → bool** som sjekker at ingen rom blir brukt samtidig for flere sesjoner. Du kan (og burde!) bruke funksjonen noDup fra del a) i dette og andre spørsmål.
iii. Skriv en funksjon **checkDuty = fn : Conference → bool** som sjekker at ingen personer er på flere presentasjoner med det samme starttidspunktet.
iv. Skriv en funksjon **updateDuty = fn : Session * Person → DutyRoster → DutyRoster** som oppdaterer en DutyRoster med en ny sesjonsleder. Med andre ord, den nye DutyRoster-en er en funksjon som returnerer denne personen når man spør om denne sesjonen, og ellers, det vil si når man spør om en hvilken som helst annen sesjon, refererer til den opprinnelige DutyRoster-en.

Skriv ditt svar her...

```
1
```

Maks poeng: 12

Prolog: introduction

I denne oppgaven skal vi forsøke å kode roboten fra obligen i Prolog. Vi antar et fastsatt rutenett med celler 0..42 på begge aksene (X = horisontal, Y = vertikal, posisjon (0,0) er nederst til venstre). Vi definerer et predikat move/3, som representerer om vi kan flytte i ett enkelt steg fra en posisjon i en gitt retning til en ny posisjon, eller kræsje dersom vi er på kanten av rutenettet.

```
move(pos(0,_Y), w, crash).
move(pos(_X,0), s, crash).
move(pos(42,_Y), e, crash).
move(pos(_X,42), n, crash).
move(pos(X,Y), w, pos(Z,Y)) :- Z is X-1.
move(pos(X,Y), n, pos(X,Z)) :- Z is Y+1.
move(pos(X,Y), e, pos(Z,Y)) :- Z is X+1.
move(pos(X,Y), s, pos(X,Z)) :- Z is Y-1.
```

I de første fire reglene bruker vi den spesielle posisjonen "crash" for å indikere at vi falt ut av rutenettet.

Disse reglene ser i utgangspunktet ut til å fungere godt: ved å flytte oss nordover fra posisjon (3, 3) kommer vi til posisjon (3,4):

```
?- move(pos(3,3),n,P).
P = pos(3,4) ? ;
no
```

Men, dersom vi tester videre ser vi et problem - vi får uventede svar i tillegg:

```
| ?- move(pos(3,0),s,P).
P = crash ? ;
P = pos(3,-1)
yes
```

I de kommende spørsmålene vil vi diskutere denne løsningen, og forsøke å forbedre den.

3(a) Query evaluation

Forklar hvordan Prolog finner de to løsningene på spørringen (query) angitt over.

Skriv ditt svar her...

Format B I U x_2 x^2 I_x Ω Σ

Words: 0

Maks poeng: 5

3(b) Limiting search

Bruk "cut"-operatoren ("**!**") til å fikse programmet over slik at spørringene gir riktig resultat, det vil si, ikke returnerer posisjoner utenfor rutenettet når vi har en "crash".

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 5

3(c) Simple programming (1/2)

Skrive et predikat **safe/1** som tar en posisjon og er sant dersom en forflytning i en vilkårlig retning ikke vil resultere i at roboten går utenfor rutenettet ("crash")

Skriv ditt svar her...

```
1
```

Maks poeng: 5

3(d) Simple programming (2/2)

Utvid predikatet **move/3** fra før med mulighet for å flytte i retning nord-øst. Du skal ikke beregne den nye posisjonen direkte, eller teste koordinatene for start-posisjonen, men istedet komme frem til den nye posisjonen ved å først flytte nordover, og deretter østover. Det kan hende at du må lage et eget "case" for å indikere en "crash" dersom det første steget tar deg utenfor rutenettet.

Her er en eksempel-sesjon hvor det nye predikatet er brukt:

```
| ?- move(pos(41,41),ne,Q).
Q = pos(42,42) ? ;
no
| ?- move(pos(42,41),ne,Q).
Q = crash
yes
| ?- move(pos(41,42),ne,Q).
Q = crash
yes
```

Skriv ditt svar her...

```
1
```

Maks poeng: 5