NTNU

Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i **TD4165 Programmeringsspråk**

Retteforslag

Faglig kontakt under eksamen: Lars Bungum [a], Øystein Nytrø [b]

Tlf: [a] 92046135 , [b] 91897606

Eksamensdato: 5. august 2014

Eksamenstid (fra–til): 09.00 – 13.00

Hjelpemiddelkode/Tillatte hjelpemidler: C: Ingen trykte og håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.

Annen informasjon:
Oppgaven har 4/18 deloppgaver/punkter. Alle har samme vekt. Skriv tydelig, kort og presist. Gardering og ulne formuleringer trekker ned.

Om du er i tvil, forklar dine antakelser. Alle programmer skal skrives i Oz.

Målform/språk: bokmål

Antall sider: 10

Antall sider vedlegg: 0

Kontrollert av:

Dato Sign

Retteforslag *Retteveiledningen er ikke noe komplett løsningsforslag!* △

Oppgave 1 Lat og annen programmering i Oz (5 punkter)

a) Hva er hensikten med lat programmering? Forklar med et eksempel. Vis hvordan et uttrykk med syntakssukkeret `lazy` blir oversatt til `ByNeed` i kjernespråket.

Retteforslag *Hensikten er å*

- *muliggjøre behov- eller datadrevet utførelse.*
- *kunne bruke uendelige/rekursive datastrukturer.*
- *gjøre kontrollflyt implisitt, - som del av uttrykksevaluering*
- *spare minne og beregning ved bare å evaluere det som faktisk etterspørres*

Eksempel:

```
fun lazy {SumSquares N1 N2}
   N1*N1+N2*N2
end
```

blir i kjernespråket:

```
SumSquares = proc {$ N1 N2 ?Result}
  local Compute
    in Compute = proc {$ ?R}
      local M1 in local M2 in
         M1 = {Number.'*' N1 N1}
         M1 = {Number.'*' N2 N2}
         R =    {Number.'+' M1 M2}
    end end end
  {ByNeed Compute Result}
  end
end
```

△

b) Bruk lat utførelse til å implementere `ListOddIntegers` som genererer alle odde heltall fra og med 1.

Retteforslag

```
fun {ListOddIntegers}
  fun lazy {Enumerate N}
    N | {Enumerate N+2}
  end
in
  {Enumerate 1}
end
```

△

c) Implementer `fun {PickListItem Itemno List}` som plukker element nummer `Itemno`, nummeret fra 1, fra listen `List`.

Retteforslag

```
fun {PickListItem Itemno List}
  case List
  of Head|Tail then
    if Itemno == 1 then Head
    else {ListItem Tail Itemno-1}
    end
  end
end
```

△

d) Hva blir resultatet av `Browse`-kallene eller å evaluere uttrykkene vist under? Forklar.

```
OI1={ListOddIntegers}
OI2={ListOddIntegers}

{Browse local N in
     {PickListItems N OI1}
  end}

{Browse local H|T=[10 100] in
     T
     end}

local LowerBound Y C in
   Y = 5
   proc {LowerBound X ?Z}
      if X>=Y then Z=X else Z=Y end
```

```
   end
   {LowerBound 1 C}
   {Browse C}
end

{Browse local Y in
     local X=[1 2] in
        X.1=Y.1
     end
      end}

local Dodat X = 2 Y in
   proc {Dodat Z} Z = 1 end
   {ByNeed Dodat X}
   {Browse Y}
   Y = X + 1 end
```

Retteforslag

1. *Ingenting*
2. *100*
3. *5*
4. *Suspends*
5. *3*

△

e) Implementer en funksjon som returnerer n-fakultet ($n!$) av et gitt tall.

Retteforslag

```
local Factorial in
     fun {Factorial N}
      fun {Factorial N F}
 if N < 2 then F
 else {Factorial N-1 F*N} end end in
      {Factorial N 1} end
end
{Browse {Factorial 5}}
```

△

Oppgave 2 Programmeringsparadigmer og språkteori (5 punkter)

a) Forklar hva komposisjonalitet betyr med hensyn til definisjon av et programmeringsspråk. Gi eksempler på språkegenskaper som er komposisjonelle (angi hvilket språk du bruker i eksemplet). Hva er fordeler og ulemper med komposisjonalitet i programmeringsspråk?

Retteforslag *Komposisjonalitet defineres av læreboken (*s. 411*) som at*

> *Det skal være mulig å kombinere deler for å lage en ny del. Implisitt at vi kan abstrahere ved å navngi den nye delen.*
>
> *Definisjonen en del av flere egenskaper som har med abstraksjonsprinsippet å gjøre. Dette prinsippet går i korthet ut på at implementasjonen holdes adskilt fra spesifikasjonen, slik at det er mulig å bruke en komponent dersom man kjenner dens grensesnitt, uten å vite hvordan den er implementert. Et språk med komposisjonalitet er utvidbart: Det finnes uendelige utvidelsesmuligheter.*
>
> *I Oz kan komposisjonelle språkegenskaper være proserdyrer, funktorer og moduler, som kan administreres via Modul-systemet. I mange språk har man muligheten for å definere nye typer.*
>
> *Dersom verdier er fastsatt eksplisitt i en rutine, vil ikke komponenten kunne la seg kombinere med andre for løse generelle problemer der verdien er parameterisert.*
>
> *Mangel på komposisjonalitet gir et språk som bare består av enkle koder for uttrykk/utsagn. Fordelen er at oversetting, parsing blir trivielt. Også arbeidet med å lese og å forstå et program blir tilsvarende enkelt når alle navn/uttrykk/begrep er definert i språket og ikke av en programmerer.*

△

b) Hvilke deler har en komplett definisjon av et programmeringspsråk? Finnes det forskjellige måter å definere de forskjellige delene? Forklar!

Retteforslag

- *Syntaks - språkets form*
- *Semantikk - beregningsmodell som formaliserer betydning, utførelse og data.*
- *Pragmatikk - som omfatter programmering, verktøy, bibliotek, integrasjon osv.*

△

c) Kan du skissere hoveddelene av en komplett definisjon av kjernespråket for Oz (se punkt b)?

Retteforslag *Se lærebok.* △

d) Hva er et typesystem i et programmeringsspråk? Forklar de viktigste egenskapene i et typesystem.

Retteforslag *Et typesystem knytter mengder av verdier til uttrykk i programmeringsspråket. Slike mengder kan være forhåndsdefinert, komponerbare på forskjellige måter, hierarkisk organisert mv. Eksempler er positivt! Se side 50 i lærboken,*

En type eller datatype er en mengde verdier sammen med operasjoner på disse verdiene. Eks heltall og addisjon og substraksjon. △

e) Forklar hva abstraksjonene prosedyre, funksjon og tråd betyr (i Oz).

Retteforslag

- *prosedyre: en procedure binder en variabel til en prosedyreverdi. En prosedyreverdi inneholder et* statement *som gjør operasjoner på variabler tilgjengelig i prosedyredefinisjonen eller det øvrige miljøet.*
- *funksjon: en function er syntaktisk sukker som kan oversettes til en proceduredefinisjon hvor returverdien må angis eksplisitt i en variabel*
- *tråd: en tråd en oppdeling av et samtidig program, der ulike deler av det kjører i ulike tråder. Trådene eksekveres for seg, slik at man for operasjoner det er mulig å dele opp kan utnytte parallel hardware-arkitektur ved å gjøre flere ting samtidig.*

△

Oppgave 3 Tråder og tjenere (4 punkter)

Gitt følgende tre definisjoner i Oz:

```
/*
Function Reactive takes a procedure Procedure as argument,
and returns a value. This value, when called with a parameter
Message, immediately applies Procedure to Message.
*/
fun {Reactive Procedure}
    proc {$ Message}
        {Procedure Message}
    end
end

/*
Function Active takes a procedure Procedure as argument,
and returns a value. This value, when  called with a message
Message, appends Message to a stream by sending it to
the port; messages are retrieved from the stream
```

```
and Procedure is applied to them in a unique, separate thread.
*/
fun {Active Procedure}
    Stream
    Port = {NewPort Stream}
    proc {Process Message|Messages}
        {Procedure Message}
        {Process Messages}
    end
in thread {Process Stream} end
    proc {$ Message}
        {Send Port Message}
    end
end

/*
Function Hyperreactive takes a procedure Procedure as argument,
and returns a value. The value, when called with a message
Message, immediately starts a new thread in which
the Procedure is applied to Message.
*/
fun {Hyperreactive Procedure}
    proc {$ Message}
         thread {Procedure Message} end
    end
end
```

a) Hva slags verdier returnerer henholdsvis `Reactive`, `Active` og `Hypereactive`?
Retteforslag *Prosedyrer.* △

Du ønsker å lage forskjellige klienter som skal pinge[1] en tjener. I det følgende skal du anta dette brukseksemplet:

```
/* server names */
A = ask.ntnu.no
B = burle.ntnu.no

/* Start logging */
{Browse startpinging(A)}
/* Start pinging */
{Pinger A}
```

[1] PING er en internettprotkoll og tjeneste for å sjekke at en tjener gir respons.

```
/* Start logging */
{Browse startpinging(B)}
/* Start pinging */
{Pinger B}
```

Gitt

```
proc {Ping Host}
    proc {Do Attempt}
        if Attempt < 3 then
            {Delay 1000}
            {Browse ping(Host)}
            {Do Attempt+1} end
    end
in {Do 0} end
```

så kan `Pinger` implementeres på tre alternative måter:

1. `Pinger = {Reactive Ping}`
2. `Pinger = {Active Ping}`
3. `Pinger = {Hypereactive Ping}`

b) `Pinger` er implementert ved hjelp av `Reactive`. Hva blir resultatet (skrevet ut i `Browsevinduet`)? Er det flere mulige resultat? Ingen resultat? Deterministisk resultat?
Retteforslag *Intet retteforslag enda. Se ordinær eksamen 2007.* △Utskrift må bli:

```
pinging('ask.idi.ntnu.no')
ping('ask.idi.ntnu.no')
ping('ask.idi.ntnu.no')
ping('ask.idi.ntnu.no')
pinging('burle.idi.ntnu.no')
ping('burle.idi.ntnu.no')
ping('burle.idi.ntnu.no')
ping('burle.idi.ntnu.no')
```

c) `Pinger` er implementert ved hjelp av `Active`. Hva blir resultatet (skrevet ut i `Browsevinduet`)? Er det flere mulige resultat? Ingen resultat? Deterministisk resultat?
Retteforslag *fortsatt ordinær eksamen 2007* △

d) `Pinger` er implementert ved hjelp av `Hyperreactive`. Hva blir resultatet (skrevet ut i `Browsevinduet`)? Er det flere mulige resultat? Ingen resultat? Deterministisk resultat?
Retteforslag fortsatt ordinær eksamen 2007 △

Oppgave 4 Syntaks og parsing (4 punkter) Lisp[2] er et tidlig (stort sett) funksjonelt, interpreterende språk. Vi skal definere et nytt Lisp-liknende språk Epsum[3]. Et eksempel på et Epsum-program kan være:

```
;;; the value of one defined variable one to be 1
(define one 1)
;;; the value of qq defined as a function qq to be the result
;;; of applying  q on the result of applying q (whatever q
;;; may be) on (parameter) n
(define (qq n) (q (q n)))
;;; the value of applying qq to one
(qq one)
```

Epsum er et uttrykkksspråk, og alle uttrykk er applikasjoner, anvendelser, av funksjoner på et argument (verdier kan sees på som funksjoner uten argumenter). Syntaksen til Epsum er enkel:

- et program er en sekvens av én eller flere uttrykk;
- er uttrykk er enten en definisjon, en anvendelse, en identifikator eller et numerisk literal.
- en definisjon er enten en variabel-definisjon, `(define variable uttrykk)`, eller en definisjon av en unær funksjon `(define (function argument) uttrykk)`
- en anvendelse er et uttrykk på formen `(funksjonsnavn uttrykk)`.

a) Skriv en EBNF-grammatikk for Epsum utifra definisjonen gitt over slik at kodeeksemplene kan genereres av grammatikken. Husk å definere identifikatorer og tall (numeriske literaler) i grammatikken også! Hva er terminaler og ikke-terminaler i Epsum-grammatikken? Hva er startsymbolet?
Retteforslag *Likner sterkt på σ fra kont 2007 (august 2008).*
Grammatikk:

```
definisjon ::= (define <id> <uttrykk>)|
                (define (<id> <id>) <uttrykk>)
id         ::= {alfa}
alfa       ::= <a|b|.....y|z>
anvendelse ::= (<id> <uttrykk>)
uttrykk    ::= <definisjon> | <anvendelse> | <numlit>
numlit     ::= {<siffer>}
siffer     ::= 0|1|2|3|4|5|6|7|8|9
```

△

[2]List Processing language
[3]Et ProgrammeringsSpråk Uten stor Mening

b) Er grammatikken du har laget kontekstsensitiv? Forklar. Er den flertydig? Forklar.
Retteforslag *Intet retteforslag: Avhenger av grammatikken i forrige oppgave.* △

Anta at det finnes en leksikalsk analysator som omformer programteksten til en Oz-liste av symboler. Hver identifikator er representert som en post med etikett `id` og tall med etikett `num`. Kodeeksemplet over vil bli representert som:

```
[
   '(' 'define' id('one') num('1') ')'
   '(' 'define' '(' id('qq') id('n') ')'
        '(' id('q') '(' id('q') id('n') ')' ')' ')'
   '(' id('␣qq') id('one') ')'
]
```

c) Hvilke parsestrategier kan du bruke for å kontrollere syntaks for grammatikken? Skisser hvordan du kan bruke relasjonell utførelse for parsing.
Retteforslag *For den enkle grammatikken til Epsum er rekursiv nedstigningsparsing tilstrekkelig. I praksis ville man kanskje ha latt en parsergenerator (Bison...) lage en LR(0)-parser og inkludere evaluering av uttrykkene og symboltabellgenerering.* △

d) Implementer en samling prosedyrer, én for hvert ikke-terminal. Hver skal ha ett innparameter som inneholder en liste av symboler fra den leksikalske analysatoren som vist over. Hver prosedyre skal returnere `true` om den mottar en symbolliste som oppfyller ikke-terminalens definisjon i følge grammatikken din, `false` ellers. Ikke ta hensyn til feilretting eller feiltoleranse.
Retteforslag *Intet retteforslag enda.* △