EKSAMEN I FAG
TDT4165 PROGRAMMERINGSSPRÅK
Lørdag 15. mai, 2004, 0900–1300

Hjelpemidler: D (Ingen trykte eller håndskrevne hjelpemidler er tillatt. Godkjent lommekalkulator er tillatt.)

Les hele oppgavesettet før du begynner å besvare det. Svar kort og klart: Er svaret uklart eller lengre enn nødvendig, trekker dette ned.

Sensuren faller 5. juni 2004.

Oppgaven er kvalitetssikret av faglærer og en ekstern kontrollør.

Per Holager (faglærer):

Håvard Engum (ekstern kontrollør):

Del 1 Flervalg (hver underoppgave teller 2.86%)

- Denne oppgaven skal besvares på vedlagt avkrysningsskjema.
- Sett bare ett kryss for hver oppgave. Oppgaver med mer enn et kryss vil få null poeng. På avkrysningsskjemaet er det flere svaralternativer enn i oppgavesettet. Pass på å ikke krysse av for alternativer som ikke finnes i oppgavesettet.
- Kryss av med svart eller blå kulepenn. La krysset gå helt ut til hjørnet av ruta, men ikke utenfor den. Hvis du krysser feil, kan krysset slettes ved å fylle hele ruta med blekk. Når du skal slette et kryss på denne måten er det viktig at ruta blir helt dekket slik at ikke noe av det hvite papiret synes inne i ruta. Korrekturlakk er forbudt.
- Ikke brett svarskjemaet på noen som helst måte.
- Skriv studentnummeret pent to ganger.
- Ikke skriv noe i feltet merket "Eventuell ekstra ID".
- For hver oppgave, velg det ene svaret du mener er mest riktig.

1) Hvilke deler består en enkel kompilator av?

   A: En tolker og en parser.

   B: En leksikalsk analysator (eng: *tokenizer*, *lexical analysator*), en parser og en kodegenerator.

   C: En prekompilator og en parser.

   D: En prekompilator, en parser, en leksikalsk analysator og en virtuell maskin.

2) Denne funksjonen bruker mønstergjenkjenning (eng: *pattern matching*) for å finne det første elementet i en liste. Hvilken linje er det korrekt å sette inn?

```
fun {First Xs}
   % Sett inn riktig linje her. <---
end
```

A: `case Xs of Xs.1 then true end`

B: `case Xs of Xs.1 then Xs.1 end`

C: `case Xs of X|_ then X end`

D: `case X of Xs.1 then X end`

3) Hva ville være en konsekvens av å forandre enkelt-tilordningslageret (eng: *single-assignment store*) i det deklarative kjernespråket (DKL) slik at det tillot mer enn én tilordning (eng: *assignment*) til en variabel?

A: Høyere ordens programmering ville bli umulig.

B: Objekt-orientert programming ville bli umulig.

C: Programmering av deklarative komponenter ville bli umulig.

D: Programmerte komponenter ville ikke lenger være garantert deklarative.

4) Hva skjer i den deklarative beregningsmodellen i (Oz når en semantisk setning `(raise <x> end, E)` utføres? (I hele denne oppgaven bruker vi samme notasjon for utførelsestilstand som i tolkerøvingen.)

A: Ingenting. Den semantiske setningen er ugyldig.

B: Utførelsen består av følgende handlinger:

- Hvis stakken er tom, stopp utførelsen med en feilmelding. Ellers, popp neste element av stakken. Hvis dette elementet ikke er en `catch`-setning, stopp utførelsen med en feilmelding.
- La `(catch <y> then <s> end Ec)` være `catch`-setningen som ble funnet.
- Dytt `(<s>,Ec+{<y>->E(<x>)})` på stakken.

C: Utførelsen består av følgende handlinger:

- Hvis stakken er tom, stopp utførelsen med en feilmelding. Ellers, popp neste element av stakken.
  - Hvis dette elementet ikke er en `catch`-setning, dytt `(raise( <y> end, E)`, hvor `<y>` er elementet som ble poppet, på stakken.
  - Hvis dette elementet er en `catch`-setning `(catch <z> then <s> end Ec)`, dytt `(<s>,Ec+{<z>->E(<x>)})` på stakken.

D: Utførelsen består av følgende handlinger:

- Elementer poppes av stakken på leting etter en `catch`-setning.
  - Hvis en `catch`-setning `(catch <y> then <s> end Ec)` ble funnet, popp den fra stakken. Dytt så `(<s>,Ec+{<y>->E(<x>)})` på stakken.
  - Hvis stakken ble tømt uten at en `catch`-setning ble funnet, stopp utførelsen med en feilmelding.

5) Hvilke av programmeringspråkene Java, Prolog, det deklarative kjernespråket i Oz (DKL) og kjernespråket i Oz med eksplisitt tilstand (KLES) bruker statisk typesjekking?

A: DKL og KLES.

B: Alle.

C: Alle unntatt Prolog.

D: Java.

6) Studer følgende program:

```
local X Y Z in
   fun {X Y} Y+Z end
   Y = 2
   Z = 3
   local Y Z in
      Y = 5
      Z = 7
      {Browse {X Y}}
   end
end
```

Hva ville vises i Browser-vinduet hvis Oz brukte dynamiske skop-regler? (Oz bruker til vanlig statiske skop-regler.)

A: 5

B: 8

C: 9

D: 12

7) Hva vil vises i Browser-vinduet etter at følgende buffer er blitt matet inn i Emacs-grensesnittet til Oz?

```
declare

fun {FoldL L F U}
   case L
   of nil then U
   [] X|L2 then
      {FoldL L2 F {F U X}}
   end
end

{Browse {FoldL [1 2 3] fun{$ X Y} Y-X end 0}}
```

A: -6

B: 2

C: 6

D:
```
%************************ syntax error ************************
%**
%** expression at statement position
%**
%** in file ``Oz'', line 11, column 34
%** ------------------ rejected (1 error)
```

8) Hvilke av de følgende to funksjonene vil kjøre med konstant stakk-størrelse?

```
fun {F1 A B}
   if B<0 then raise domainError end
   elseif B==0 then 1
   else A*{F1 A B-1}
   end
end
```

```
fun {F2 A B C}
   if B<0 then raise domainError end
   elseif B==0 then C
   else {F2 A B-1 A*C}
   end
end
```

A: Ingen av dem

B: F1 men ikke F2.

C: F2 men ikke F1.

D: Begge.

9) Hvilket utsagn om følgende produsent/konsument-program er sant?

```
declare
fun {Produce N Limit}
   if N<Limit then
      N|{Produce N+1 Limit}
   else nil end
end
fun {Consume Xs A}
   case Xs
   of X|Xr then {Consume Xr A+X}
   [] nil then A
   end
end
local Xs S in
   thread Xs={Produce 0 500} end
   thread S={Consume Xs 0} end
   {Browse S}
end
```

A: Dette er et synkront program siden den ene tråden utføres etter den andre.

B: Variabelen `Xs` er delt mellom trådene, noe som gjør programmet uforutsigbart.

C: Konsument-tråden er synkronisert med produsent-tråden ved at konsumentens case-setning blokkerer sin utførelse inntil det neste strømelementet ankommer.

D: Dette programmet kommer ikke til å terminere.

10) Kappløpsforhold (eng: *race conditions*) kan oppstå i parallellitetssmodellen (eng: *concurrency model*) til Oz hvis følgende er tilfelle:

A: Automatisk søppeltømming brukes.

B: Automatisk søppeltømming brukes ikke.

C: Eksplisitt tilstand brukes.

D: Eksplisitt tilstand brukes ikke og lat uførelse brukes.

11) Hvilken metode for overføring av parametere simuleres i følgende prosedyre?

```
proc {IncrementAndBrowse A}
   B={NewCell A}
in
   B:=@B+1
   {Browse B}
end
```

A: "Kall-ved-variabel" (eng: *Call by variable*)

B: "Kall-ved-verdi" (eng: *Call by value*)

C: "Kall-ved-verdi/resultat" (eng: *Call by value-result*)

D: "Kall-ved-behov" (eng: *Call by need*)

12) Hvilket av de følgende klassehierarkiene er lovlig?

A:
```
class AX meth m(...) ... end end
class BX meth n(...) ... end end
class A from AX meth m(...) ... end end
class B from BX end
class C from A B end
```

B:
```
class AX meth m(...) ... end end
class BX meth m(...) ... end end
class A from AX meth m(...) ... end end
class B from BX end
class C from A B end
```

C:
```
class AX meth m(...) ... end end
class BX meth m(...) ... end end
class A from AX end
class B from BX end
class C from A B end
```

D:
```
class AX meth m(...) ... end end
class BX meth n(...) ... end end
class A from AX meth n(...) ... end end
class B from BX meth o(...) ... end end
class C from A B end
```

13) Studer følgende utsagn om ren Prolog (eng: *pure Prolog*) og det relasjonelle kjernespråket i Oz (RKL).

i): Alle programmer i ren Prolog kan oversettes til RKL.

ii): Alle programmer i RKL kan oversettes til ren Prolog.

iii): Prolog og RKL inneholder de samme programmeringskonseptene og er bare forskjellige i hvordan konseptene opptrer syntaktisk.

Hvilke av dem er sanne?

A: Kun i).

B: Kun ii).

C: Kun iii).

D: Alle tre.

14) Studer følgende utsagn om den relasjonelle beregningsmodellen i Oz.

i) Det er mulig å kjøre flere søk samtidig.

ii) Det er mulig å kjøre et søk inne i et annet søk.

iii) Det er mulig å innkapsle et søk slik at resten av programmet blir beskyttet fra det.

Hvilke av dem er sanne?

A: Kun i).

B: Kun ii).

C: Kun iii).

D: Alle tre.

Del 2 Spørsmål (hver underoppgave teller 6%)

Skriv ikke mer enn en side i normal skriftstørrelse som svar på denne delen.

a) Hva er lat utførelse (eng: *lazy execution*)? Nevn to mulige fordeler ved å bruke lat uførelse.

b) Hva er en strøm (eng: *stream*)? Nevn to ting som kan gjøres med strømmer som ikke enkelt kan gjøres uten dem.

c) Hva er syntaktisk sukker? Nevn en fordel eller en ulempe ved bruk av syntaktisk sukker.

d) Hva er fordelen ved å gjøre feilhåndtering ved hjelp av unntak (eng: *exceptions*) framfor å gjøre det ved å kalle bruker-definerte prosedyrer når feil oppstår?

Del 3 Høyere ordens programmering (teller 6%)

Skriv en funksjon `Map` som tar som argumenter en liste `L` og en funksjon `F` og returnerer en liste hvor hvert element `E` i `L` er erstattet med {`F E`}.

For eksempel skal {`Map [1 2 3] fun{$ X} X+1 end`} returnere lista `[2 3 4]`.

Del 4 Semantikk (teller 6%)

Det finnes minst ett program i det deklarative kjernespråket (DKL) som vil bli utført på den abstrakte maskinen i 13 utførelsessteg

$$(ST_0, \sigma_0) \rightarrow (ST_1, \sigma_1) \rightarrow \cdots (ST_{13}, \sigma_{13}),$$

hvor utførelsestilstandene (eng: *execution states*) etter noen av trinnene er de samme som i Tabell 1. Skriv et slikt program. (Tabell 1 hopper over enkelte steg. Steg som er resultat av oppdeling av sekvensiell sammensetting (eng: *sequential composition*) er hverken med i Tabell 1 eller i opptellingen som ga tallet 13.)

Del 5 Grammatikker og parsing (hver underoppgave teller 6%)

I denne oppgaven skal du skrive i Oz en rekursiv nedstigningsparser for et lite subsett av `SELECT`-setninger i SQL definert ved følgende grammatikk:

```
<select stmt> ::= 'select' <name list> 'from' <name list> 'where' <expr>
<name list>   ::= <name> | <name> ',' <name list>
<expr>        ::= <name> '=' <value>
```

Anta at `'select'`, `'from'`, `'where'`, `','`, og `'='` har blitt redusert til atomer av en leksikalsk analysator (eng: *tokenizer*, *lexical analysator*) som også har laget tupler `name(L)` for `<name>` og `val(V)` for `<value>`, hvor `L` er et atom som svarer til navnet som brukeren skrev og `V` er et heltall likt konstant-verdien.

Ikke bruk tid på feilhåndtering. Den innebygde feilhåndteringen i Oz er tilstrekkelig for denne oppgaven. Bruk unifikasjon og lignende teknikker der det kan gjøre programmet kortere.

Steg i	Miljø E_i		Lager σ_i
0			
1	`P1`	$\longrightarrow$	x_1
2	`P1`	$\longrightarrow$	$x_1 \rightarrow x_2 \rightarrow$ (`proc {$ T U V} T=U+V end`,$\emptyset$)
3	`X`	$\longrightarrow$	x_3
	`P1`	$\longrightarrow$	$x_1 \rightarrow x_2 \rightarrow$ (`proc {$ T U V} T=U+V end`,$\emptyset$)
$\vdots$	$\vdots$		$\vdots$
5	`X`	$\longrightarrow$	x_3
	`P1`	$\longrightarrow$	$x_1 \rightarrow x_2 \rightarrow$ (`proc {$ T U V} T=U+V end`,$\emptyset$)
	`P2`	$\longrightarrow$	$x_4 \rightarrow x_5 \rightarrow$ (`proc {$ T U V} T=X+V end`,{`X`$\rightarrow x_3$})
$\vdots$	$\vdots$		$\vdots$
10	`X`	$\longrightarrow$	$x_3 \rightarrow x_8 \rightarrow 1$
	`Y`	$\longrightarrow$	$x_6 \rightarrow x_9 \rightarrow 2$
	`Z`	$\longrightarrow$	x_7
	`P1`	$\longrightarrow$	$x_1 \rightarrow x_2 \rightarrow$ (`proc {$ T U V} T=U+V end`,$\emptyset$)
	`P2`	$\longrightarrow$	$x_4 \rightarrow x_5 \rightarrow$ (`proc {$ T U V} T=X+V end`,{`X`$\rightarrow x_3$})
11	`X, U`	$\longrightarrow$	$x_3 \rightarrow x_8 \rightarrow 1$
	`T`	$\longrightarrow$	x_7
	`V`	$\longrightarrow$	$x_6 \rightarrow x_9 \rightarrow 2$
12			$x_3 \rightarrow x_8 \rightarrow 1$
			$x_7 \rightarrow 3$
			$x_6 \rightarrow x_9 \rightarrow 2$

Tabell 1: Sekvens av utførelsestilstander for det hemmelige programmet.

a) Er strukturen i grammatikken egnet for en rekursiv nedstigningsparser? Forklar. Hvis den ikke er egnet, beskriv ekvivalens-bevarende transformasjoner som kan gjøres for å komme fram til en egnet grammatikk. Vis de endrede definisjonene med EBNF-notasjon.

b) Skriv i Oz en funksjon `{Expr In ?Out}` som kjenner igjen en instans av `<expr>` fra starten av leksem-lista (eng: *token list*) `In`, for eksempel lista `[name(jon) '=' val(4) ...]`. Funksjonen må binde resten av lista til `Out` og returnere et tuppel, for eksempel `equal(jon 4)`, som på en fornuftig måte inneholder informasjonen fra den delen av lista som ble gjenkjent.

c) Skriv i Oz en funksjon `{NameList In ?Out}` som kjenner igjen en instans av `<name list>` fra starten av leksem-lista (eng: *token list*) `In`, for eksempel lista `[name(f1) ',' name(f2) ...]`, binder resten av lista til `Out` og returnerer en liste av tupler, for eksempel `[name(f1) name(F2)]`, som på en fornuftig måte inneholder informasjonen fra den delen av lista som ble gjenkjent.

d) Skriv i Oz en funksjon `{SelectStmt In ?Out}` som kjenner igjen en hel instans av `<select stmt>`, fra leksem-lista (eng: *token list*) `In`, for eksempel

```
['select' name(f1) ',' name(f2)
 'from' name(tab)
 'where' name(f1) '=' val(4) ... ],
```

binder en parameter `Out` til resten av leksem-lista (markert ved `...`) og returnerer et syntakstre som inneholder informasjonen fra den delen av leksem-lista som ble gjenkjent, for eksempel `select([name(f1) name(f2)] [name(tab)] equal(f1 4))`.