Løsningsforslag
til eksamen i tdt4165 Programmeringsspråk
Fredag 6.august 2004

Oppgave 1

a)

6

b)

3

c)

3

d)

2

e)

3

f)

4

g)

1

h)

5

i)

1

j)

5

k)

4

l)

4

Oppgave 2

a)

Når en metode definert for en baseklasse redefineres for en av dens subklasser (ny metode med samme navn), skal denne redefinisjonen virke på samme måte som den som var definert for baseklassen. 'Samme måte' betyr her at en programmerer som kjenner baseklassen, metoden på den og 'meningen med' subklassen, skal kunne gjette hva subklassens utgave av metoden gjør. Et eksempel som bryter med utbyttbarhet:

```
class Number
    attr val
    meth incr  val := @val + 1  end
end

class Complex from Number
    attr imVal
    meth incr  imVal := @val + @imVal  end
end
```

b)

Dette angår programmer med tråder. Hvis verdien som bindes til en variabel kan bli forskjellig om trådene utføres i forskjellige rekkefølger, har man ikke-determinisme. Et eksempel på et ikke-deterministisk program (fenomenet blir synlig ved en feilmelding om at man prøver å binde 2=1):

```
declare X
thread X=1 end
X=2
```

c)

Unifisering er prosessen som binder udefinerte deler av to datastrukturer når man forteller at de to strukturene skal være like. Har man for eksempel

```
declare A B
    D=m([A 1 B] 3)
    C=m([A A B] B)
C=D
```

så blir A=1, B=3, C=D=m([1 1 3] 3)

d)

En kontekst-fri grammatikk er et sett atomer/leksikal-symboler, et sett omskrivingsregler og et startsymbol; omskrivingsreglene kan referere hverandre rekursivt. En regulær grammatikk er det samme bortsett fra at rekursjon ikke er tillatt. En kontekst-avhengig grammatikk kan defineres som en kontekst-fri grammatikk der det i tillegg er regler som begrenser hva man kan velge for en substitusjon gitt hvilke substitusjoner man har valgt andre steder.

```
<sum> ::= <tall> | <sum> + <tall>
<tall> ::= <siffer> | <tall> <siffer>
<siffer> ::= 0 | 1
```

Oppgave 3

a)

```
fun {FromDec T R}
    case T
    of D|Tr then {FromDec Tr R*10+D}
    else R
    end
end
```

b)

```
fun {MakeFrom B}
    fun {Frm T R}
        case T
        of D|Tr then {Frm Tr R*B+D}
        else R
        end
    end
in
    Frm
end
```

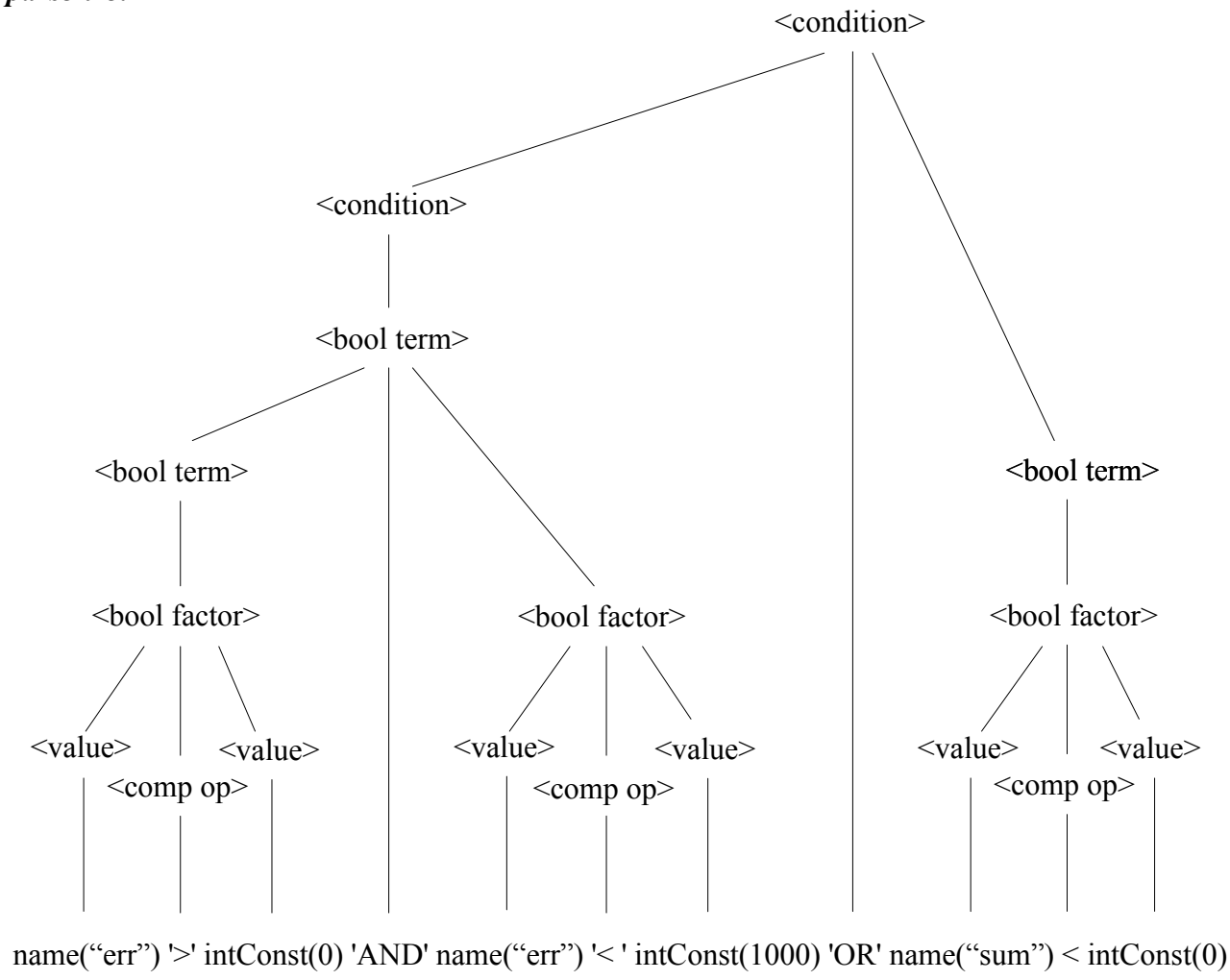
c)

```
fun {FromHex T} {{MakeFrom 16} T 0} end
```

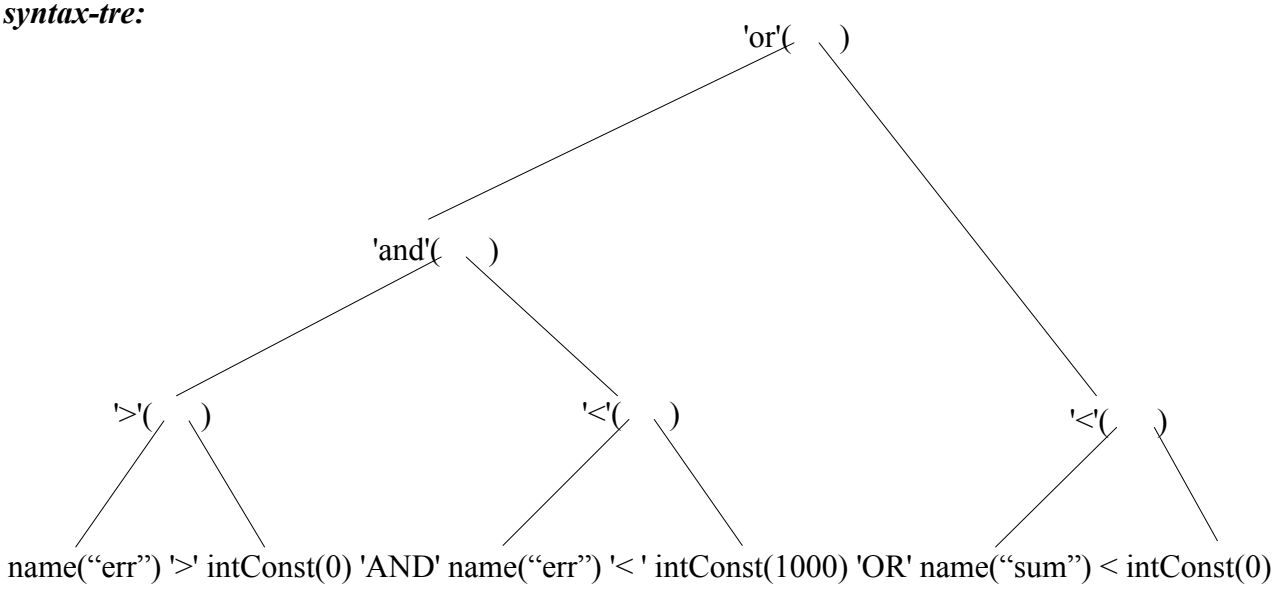
Oppgave 4

a)

parse-tre:



syntax-tre:



b)

Grammatikken har venstre-rekursive definisjoner for <condition> og <bool term>, dette egner seg ikke. Man bruker den vanlige transformasjonen for å få bort disse, og får da:

```
<condition>    ::= <bool term> <condition rest>
<condition rest> ::=   | 'OR'  <bool term> <condition rest>
<bool term>    ::= <bool factor> <bool term rest>
<bool term rest> ::=   | 'AND' <bool factor> <bool term rest>
<bool factor> ::= <value> <comp op> <value> | '(' <condition> ')'
<comp op>      ::= '<' | '<=' | '>' | '>=' | '=' | '<>'
<value>        ::= <name> | <int const>
```

her kan vi eventuelt videre skrive

```
<condition rest> ::=   | 'OR'  <condition>
<condition>    ::= <bool term> (   | 'OR'  <condition>)
```

og tilsvarende for <bool term rest> og <bool term>

c)

```
fun {Value Inn Ut}
    case Inn
    of name(N) | V  then Ut=V  name(N)
    [] intConst(C) | V  then Ut=V  intConst(C)
    end
end


fun {CompOp Left Inn Ut}
    Right
in
    case Inn
    of '<' | V   then Ut=V  '<'(Left Right)
    [] '<=' | V  then Ut=V  '<='(Left Right)
    [] '>' | V    then Ut=V  '>'(Left Right)
    [] '>=' | V  then Ut=V  '>='(Left Right)
    [] '=' | V    then Ut=V  '='(Left Right)
    [] '<>' | V  then Ut=V  '<>'(Left Right)
    end
end


fun {FirstBoolFac Inn Ut}
    V1  V2
    P={CompOp {Value Inn V1} V1 V2}
in
    P.2={Value V2 Ut}
    P
end
```

```
fun {BoolFactor Inn Ut}
    case Inn
    of name(_) | _  then  {FirstBoolFac Inn Ut}
    [] intConst(_) | _  then  {FirstBoolFac Inn Ut}
    [] '(' |V  then  B={Condition nil V  ')' |Ut}
    end
end
```

d)

```
fun {BoolTerm  Left Inn Ut}
    V1  F
in
    if  Left==nil then  F={BoolFactor Inn V1}
    else F='and'(Left {BoolFactor Inn V1})
    end
    case V1
    of 'AND' |V2 then  {BoolTerm F V2 Ut}
    else Ut=V1  F
    end
end


fun {Condition  Left Inn Ut}
    V1  F
in
    if  Left==nil then  F={BoolTerm nil Inn V1}
    else F='or'(Left {BoolTerm nil Inn V1})
    end
    case V1
    of 'OR' |V2 then  {Condition F V2 Ut}
    else Ut=V1  F
    end
end
```