Compilers: Assignment #2

Due on Sunday, November 29, 2015

Resubmit: Task 1, 2

Mirza Hasanbasic

Indhold

Task 1

a)

$$([og][og])^*|([og][og][og])^*$$

Her vil vi have et alfabet, hvor sekvensen af længden er delelig med 2 eller 3.

b)

$$T \rightarrow c$$
$$R \rightarrow cT$$
$$T \rightarrow aTb$$

Vil producerer en context-free grammer, hvor c er tilladt til at være alene, c vil altid være mellem a og b og der vil, for hvert a være et b.

c)

i)

Når man har *% prec letprec*, så specificerer det en regel, som er associativ med *letprec* i dette tilfælde. I *% nonassoc* har man indikeret at *letprec* ikke er associativ, altså vores *LET* og *IN*.

ii)

FIX:

Det som *let* gør, er at, der gives et eller flere udtryk, som deklareres en værdi, hvor disse værdier så udgør et resultet til sidst.

Hvor den første variabel består af en streng, et udtryk og så positionen af angivelsen. Altså, hvis så kigger, på `(Dec (#1 $2, $4, $3), $6, $1)`, så har vi, at #1 $2 vil refere til den første parameter og andet symbol, ID, som er en streng der holder navnene på variablene.

$4 vil så refere til parameteren af det fjerde i udtrykket som det vil blive båndet til. Hvor $3 så er symbolet EQ, da det er på denne position, og dermed vil det være til et lighedstegn, som den vil blive båndet til.

For at kigge på de sidste to, kigger vi på hele udtrykket, altså `Let (Dec (#1 $2, $4, $3), $6, $1)`. Her har vi at $6 referer til hvilken erklæring af variablen, (Dec) vil blive brugt og hvor $1 referer til positionen af let symbolet

Task 2

a)

Task 2 continued on next page...

For filter, vil det altså være

$$\forall\alpha \quad ((\alpha \rightarrow \texttt{bool}) * [\alpha]) \rightarrow [\alpha]$$

FIX:

b)

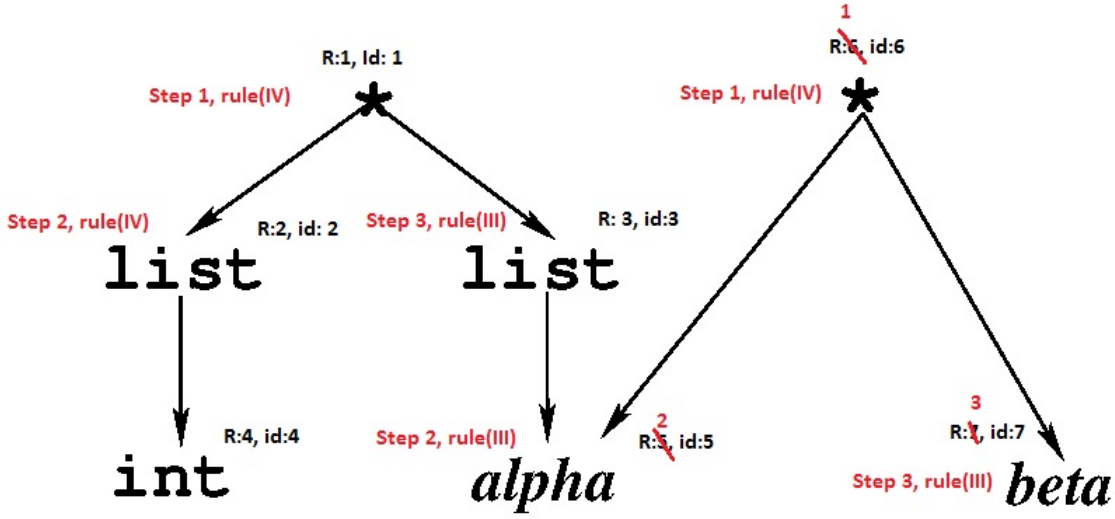
```
1  Check_Exp(Exp, vtable, ftable) = case Exp of
2
3      filter(p, arr_exp) ⇒
4          t_arr = Check_Exp(arr_exp), vtable, ftable)
5          t_el = case t_arr of
6                      Array(t_1) → t_1
7                  | other → error()
8
9          t_f = lookup(ftable, name(p))
10         case t_f of
11             (t_in → t_out) ⇒ if t_in = t_el and t_out = bool
12                                 then t_arr
13                                 else error()
14           | _ ⇒ error()
```

c)

```
1  Check_Exp(Exp, vtable, ftable) = case Exp of
2
3      filter( Lambda(ret_type, id_type_lst, body), arr_exp) ⇒
4          t_body = Check_Exp(body, vtable, ftable)
5          t_arr = Check_Exp(arr_exp), vtable, ftable)
6          t_el = case t_arr of
7                      Array(t_1) → t_1
8                  | other → error()
9          t_in = case id_type_lst of
10                     [(_, t_2)] → t_2
11                 | other → error()
12         if ret_type = bool and t_body = bool and t_el = t_in
13         then t_arr
14         else error()
```

Task 3

Figuren under er et eksempel på hvordan man kan bruge MGU algoritmen




```
1  I     if( find(m) = find(n) )                          then return true;
2  II    else if( m and n are the same basic type)  then return true;
3  III   else if( m or n represent a type variable) then union(m, n);
4        return true;
5  IV    else if( m and n are the same type constructor
6                with children m_1,...,m_k and n_1,...,n_k,∀k) then
7                union(m, n); return unify(m_1,n_1) and ... and unify(m_k,n_k);
8  V     else return false;
```