Oversættere, ugeopgave 4

Anders Bjerg Pedersen (andersbp@me.com)

13. december 2009

Spørgsmål 1

Vi bliver i dette spørgsmål bedt om at udvide funktionen $Eval_{Exp}$, så den kan genkende anonyme funktioner. Vi skal altså tilføje de to produktioner

$$Exp \rightarrow \texttt{fn}\ \textbf{id}\ \texttt{=>}\ Exp \quad (1)$$
$$Exp \rightarrow Exp\ Exp \quad (2)$$

som cases i $Eval_{Exp}$. (1) skal returnere et *closure*, som er af typen `func`. Et closure består af argumentet til den anonyme funktion (**id** i (1)), kroppen af den anonyme funktion (Exp på højresiden i (1)) samt en symboltabel[1] hørende til den anonyme funktion.

Der er en del overvejelser i denne opgave:

- Den endelige værdi af den anonyme funktion kan ikke direkte beregnes ved første evaluering, da vi ikke umiddelbart har nogen mulighed for at sende argumentet videre til den i første gennemløb.
- Produktionen (2) er kun lovlig, når Exp_1 er af typen `func`, samt Exp_2 har typen `int`. Vi har ingen problemer med at typechecke og evaluere Exp_2.
- Anden gang, når vi evaluerer udtrykket returneret fra (1), kan vi rent faktisk evaluere værdien af den anonyme funktion.

Derfor gør (1) ikke meget andet end at slå **id** op og returnere vore closure, og (2) evaluerer Exp_1 ved at modtage vores closure, evaluere Exp_2 til en værdi og derefter evaluere udtrykket fra den anonyme funktion med den fundne værdi af Exp_2:

[1] Dina: nu skriver du i dine kommentarer til opgave 1, at det er **F**table, der skal sendes videre, men mener du ikke **V**table? Der skulle jo ikke gerne ske særlig meget med ftable...?

$Eval_{Exp}(Exp, vtable, ftable)$ = `case` Exp `of`	
$Exp_1 Exp_2$	$v_1 = (var, Exp_3, vtable') = Eval_{Exp}(Exp_1, vtable, ftable)$ $v_2 = Eval_{Exp}(Exp_2, vtable, ftable)$ *if* v_1 *is a* `func` *and* v_2 *is an* `int` *then* $vtable'' = bind(vtable', var, v_2)$ $Eval_{Exp}(Exp_3, vtable'', ftable)$ *else* **error**()
`fn` **id** `=>` Exp	$v = lookup(vtable, name(\textbf{id}))$ *if* $v = unbound$ *then* **error**() *else* $(v, Exp, vtable)$

Jeg er ikke helt sikker på, at *lookup* og efterfølgende fejl er nødvendige i nederste case, da **id** først bindes i den øverste case. Det er altså muligvis ikke et krav, at **id** eksisterer i symboltabellen, når vi evaluerer funktionsdefinitionen (men den skal selvfølgelig være der, når vi skal evaluere funktionens udtryk).

Dernæst skal vores liste Get_{TypeId} i figur 5.3 af mulige funktionstyper også udvides med vores nye `func`-type:

$Get_{TypeId}(TypeId)$ = `case` $TypeId$ `of`	
`int` **id**	($name$(**id**), `int`)
`bool` **id**	($name$(**id**), `bool`)
`func` **id**	($name$(**id**), `func`)

Spørgsmål 2

I dette spørgsmål skal vi udvide typecheckeren i figur 6.2, så den også kan godkende de anonyme funktioner defineret ovenfor.

$Check_{Exp}(Exp, vtable, ftable)$ = `case` Exp `of`	
$Exp_1 Exp_2$	$t_1 = Check_{Exp}(Exp_1, vtable, ftable)$ $t_2 = Check_{Exp}(Exp_2, vtable, ftable)$ *if* t_1 = `func` *and* t_2 = `int` *then* `int` *else* **error**(); `int`
`fn` **id** `=>` Exp	$t_1 = lookup(vtable, name(\textbf{id}))$ *if* $t_1 = unbound$ *then* $vtable' = bind(vtable, t_1, \texttt{int})$ $t_2 = Check_{Exp}(Exp, vtable', ftable)$ *else* $t_2 = Check_{Exp}(Exp, vtable, ftable)$ *if* t_1 = `int` *and* t_2 = `int` *then* `func` *else* **error**(); `func`

Til sidst skal vores liste Get_{TypeId} i figur 6.3 af mulige funktionstyper igen udvides med vores nye `func`-type:

$Get_{TypeId}(TypeId) =$ `case` $TypeId$ `of`	
`int` **id**	($name$(**id**), `int`)
`bool` **id**	($name$(**id**), `bool`)
`func` **id**	($name$(**id**), `func`)