G-Opgave

Michael Thulin & Philip Munksgaard

17. december 2010

1 Tilføjelser til Lexer.lex

Vi har tilføjet de manglende keywords og tokens til Lexer.lex.

2 Parser.grm

Her startede vi med at tilføje de manglende tokens i grammatikken. Derefter tilføjede vi de manglende typer og så de manglende produktioner. Så endte vi op med mere end 15 shift/reduce konflikter. For at slippe af med nogle af disse, eliminerede vi venstre-rekursion i adskillige af produktionerne, heriblandt Types, Pats og Exps. I nogle produktionerne, f.eks. Dec og Match, valgte vi at elmiminere tvetydighed ved at sætte associativitet korrekt. Den eneste shift/reduce-fejl der voldte os nævneværdige problemer var tvetydighed omkring $LPARExpRPAR$, som vi valgte at løse ved at dele produktionen $Exp \rightarrow LPARExpsRPARCOLONID$ op i to produktioner, $Exp \rightarrow LPARExpRPARCOLONID$ og $Exp \rightarrow LPARExpCOMMAExpsRPARCOLONID$, jvf. bilag 1.

Parser.grm oversætter nu uden fejl.

3 Type.sml

Vi har lavet de nødvendige tilføjelser i checkType, checkPat, checkExp. Derudover tilføjede vi kode til at hente type-deklarationer ud af tyDecs.

Herunder ses pseudokode for checkExp, checkDec og checkTuple, som er de dele af Type.sml vi har lavet mest om i.

checkExp(Exp, vtable, ftable, ttable) = case Exp of	
true	bool
false	bool
null	t = lookup x ttable if t = unbound then error() else TyVar s
id	t = lookup x vtable if t = unbound then error() else t
Exp1 = Exp2	t1 = checkExp(Exp1, vtable, ftable, ttable) t2 = checkExp(Exp1, vtable, ftable, ttable) if t1 = int and t2 = int then bool else error()
if Exp1 then Exp2 else Exp3	t1 = checkExp(Exp1, vtable, ftable, ttable) t2 = checkExp(Exp2, vtable, ftable, ttable) t3 = checkExp(Exp3, vtable, ftable, ttable) if t1 = bool and t2 = t3 then t2 else error()
Exp1 < Exp2	t1 = checkExp(Exp1, vtable, ftable, ttable) t2 = checkExp(Exp1, vtable, ftable, ttable) if t1 = int and t2 = int then bool else error()
not Exp1	t1 = checkExp(Exp1, vtable, ftable, ttable) if t1 = bool then bool else error()
Exp1 and Exp2	t1 = checkExp(Exp1, vtable, ftable, ttable) t2 = checkExp(Exp1, vtable, ftable, ttable) if t1 = bool and t2 = bool then bool else error()
Exp1 or Exp2	t1 = checkExp(Exp1, vtable, ftable, ttable) t2 = checkExp(Exp1, vtable, ftable, ttable) if t1 = bool and t2 = bool then bool else error()
let id = Dec1 in Exp1	t1 = checkDec(Dec1, vtable, ftable, ttable) checkExp(Exp1, vtable, ftable, ttable)
(Exps) : id	ty = lookup s ttable if ty = unbound then error() else checkTuple Exps ty vtable ftable ttable TyVar id
case Exp1 of in Match1 end	ty = checkExp(Exp1, vtable, ftable, ttable) checkMath m ty vtable ftable ttablee pos

checkDec (decs, vtable, ftable, ttable) = case decs of	
(pat, e) vtable1 @ vtable	ty = checkExp e vtable ftable ttable vtable1 = checkPat p ty ttable
(pat, e) :: decs	ty = checkExp e vtable ftable ttable vtable1 = checkPat pat ty ttable td = checkDec (decs, (vtable1 @ vtable), ftable, ttable) vtable1 @ td @ vtable

checkTuple(e, ty, vtable, ftable, ttable) = case (e, ty) of	
(e, ty)	If checkExp e vtable ftable ttable = ty then [ty] else error()
(e :: es, ty :: tys)	if checkExp e vtable ftable ttable = ty then ty :: checkTuple es tys vtable ftable ttable else error()

4 Compiler.sml

Nedenfor kan ses pseudokode for hvordan koden for de nye konstruktioner genereres.

CompilePat (p, v, vtable, fail) = case p of	
TrueP	[place := 1]
FalseP	[place := 0]
NullP	[place := 0]
TupleP	($code_1$, $vtable_1$) = compilePats(pats, v, vtable, fail, 0) if (v = 0) then Error() else (code1, vtable)

CompilePats (pats, v, vtable, fail, offset) = case pats of	
([], [])	([], [])
(pat :: pats)	x = newvar() $code_1$ = LW(x, v, offset) ($code_2, vtable_1$) = compilePat(pat, x, vtable, fail) ($code_3, vtable_2$) = compilePats(pats, v, ($vtable_1$ @ $vtable_2$), fail, (offset + 4)) ($code_1$ @ $code_2$ @ $code_3$, $vtable_2$ @ $vtable_1$ @ $vtable$)

compileDec(dec, vtable, fail) = case dec of	
[]	([], [])
(p, e) :: ds	t := newvar() code1 := compileExp e vtable t (code2, vtable1) := compilePat p t vtable fail (code3, vtable2) := compileDec ds (vtable1 @ vtaale) fail (code1 @ code2 @ code2, vtable2 @ vtable 1 vtable)

compileDec(exps, vtable, fail) = case exps of	
[]	[]
	t1 = newvar() t2 = newvar() code1 = compileExp e vtable t1 code2 = compileTuple ees vtable mem code1 @ [SW (t1, mem, 0), mem := mem + 4] @ code2

5 Fejl og mangler

Vores compiler kompilerer alle test-filerne korrekt. Der er dog et par af error-filerne som ikke genererer fejl. Det gælder error14.cat, error17.cat, error18.cat og error19.cat. Fejlende skyldes manglende tjek i Type.sml. Vi mangler f.eks. at teste let deklarations-typerne ordenligt. Hvis der deklareres to variable med samme navn bliver det ikke dedekteret. Med lidt mere arbejde i typecheckeren ville vores løsning fungere fuldstændig efter hensigten.

6 Tests

Vi har ikke kunnet finde nogen mangler i de udleverede tests

7 Bilag 1: Produktioner for Exp

```
Exp :     NUM              { Cat.Num $1 }
        | TRUE             { Cat.True $1 }
        | FALSE            { Cat.False $1 }
        | NULL COLON ID    { Cat.Null (#1 $3, $1) }
        | ID               { Cat.Var $1 }
        | LPAR Exp COMMA Exps RPAR COLON ID
                           { Cat.MkTuple ( $2 :: $4, #1 $7, $1 ) }
        | LPAR Exp RPAR COLON ID
                           { Cat.MkTuple ([$2], #1 $5, $1) }
        | Exp PLUS Exp     { Cat.Plus ($1, $3, $2) }
        | Exp MINUS Exp    { Cat.Minus ($1, $3, $2) }
        | Exp EQUAL Exp    { Cat.Equal ($1, $3, $2) }
        | Exp LESSTHAN Exp
                           { Cat.Less ($1, $3, $2) }
        | NOT Exp          { Cat.Not ($2, $1) }
        | Exp AND Exp      { Cat.And ($1, $3, $2) }
        | Exp OR Exp       { Cat.Or ($1, $3, $2) }
        | IF Exp THEN Exp ELSE Exp
                           { Cat.If ($2, $4, $6, $1) }
        | LET Dec IN Exp
                           { Cat.Let ($2, $4, $1) }
        | CASE Exp OF Match END
                           { Cat.Case ($2, $4, $1) }
        | ID Exp %prec WRITE
                           { Cat.Apply (#1 $1, $2, #2 $1) }
        | READ             { Cat.Read $1 }
        | WRITE Exp        { Cat.Write ($2, $1) }
        | LPAR Exp RPAR    { $2 }
;
```