6.17 forslag

Grammar Rule	Semantic Rule
exp_1 → ***let*** *dec-list* ***in*** exp_2	$dec\text{-}list.intab = exp_1.symtab$ $dec\text{-}list.locintab = exp_1.symtab$ $dec\text{-}list.outtab =$ $\quad exp_1.symtab + dec\text{-}list.locouttab$ $exp_2.symtab = dec\text{-}list.outtab$
$dec\text{-}list_1$ → $dec\text{-}list_2$, *decl*	$decl_2.intab = dec\text{-}list_1.intab$ $decl.intab = dec\text{-}list_1.intab$ $dec\text{-}list_2.locintab = dec\text{-}list_1.locintab$ $decl.locintab = decl\text{-}list_2.locouttab$ $dec\text{-}list_1.locouttab = decl.locouttab$
dec-list → *decl*	$decl.intab = decl\text{-}list.intab$ $decl.locintab = dec\text{-}list.locintab$ $dec\text{-}list.locouttab = decl.locouttab$
decl → ***id*** = *exp*	$decl.locouttab$ = ... $\quad insert(decl.locintab, ...)$

6.18 forslag

Grammar Rule	Semantic Rule
$exp_1 \rightarrow exp_2 + exp_3$	$exp_1.val =$ **if** $(exp_2.val =$ **error**$)$ **or** $(exp_3.val =$ **error**$)$ **then error** **else** $exp_2.val + exp_3.val$
$exp_1 \rightarrow (exp_2)$	$exp_1.val = exp_2.val$
$exp \rightarrow$ ***id***	$exp.val = lookupVal(exp.symtab, id.name)$
$exp \rightarrow$ ***num***	$exp.val =$ ***num***$.val$
$exp_1 \rightarrow$ ***let*** $dec\text{-}list$ ***in*** exp_2	$exp_1.val =$ **if** $(decl\text{-}list.outtab = errtab)$ **then error** **else** $exp_2.val$

decl → ***id*** = *exp*	*decl.outtab* = **if** *(decl.intab = errtab)* **then** *errtab* **else** **if** *(lookupLevel(* *decl.intab,id.name) =* *decl.nestlevel)* **then** *errtab* **else** *insert(decl.intab,id.name,* *decl.nestlevel, exp.val)*

6.20 forslag

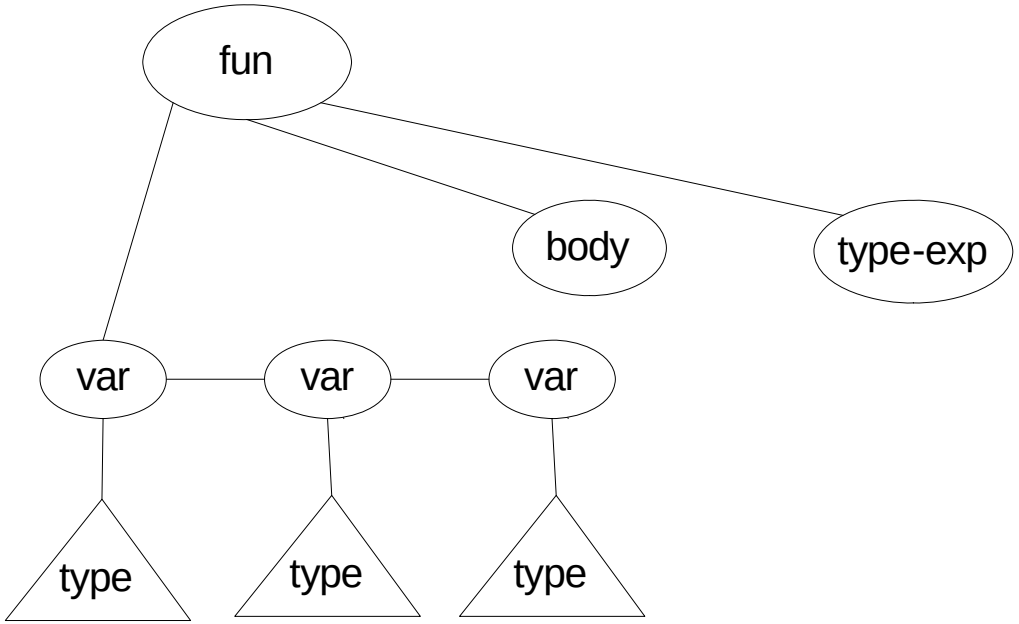
Grammar Rule	Semantic Rule
$S \rightarrow \ exp$	$exp.type$ = **if** $exp.isFloat$ **then** float **else** int $S.val = exp.val$
$exp_1 \rightarrow \ exp_2 + exp_3$	$exp_1.isFloat$ = $exp_2.isFloat$ **or** $exp_3.isFloat$ $exp_1.val$ = **if** $exp_1.isFloat$ **then** $floatAdd($ **if** $not\ exp_2.isFloat$ **then** $FLOAT(exp_2.val)$ **else** $exp_2.val,$ **if** $not\ exp_3.isFloat$ **then** $FLOAT(exp_3.val)$ **else** $exp_3.val$ **else** $intAdd(exp_2.val,\ exp_3.val)$

$exp_1 \rightarrow exp_2 \ / \ exp_3$	$exp_1.val =$ **if** (**not** $exp_2.isFloat$ **and** **not** $exp_3.isFloat$) **then** $exp_2.val$ **div** $exp_3.val$ **else** $exp_2.val$ / $exp_3.val$

$exp_1 \rightarrow (exp_2)$	$exp_1.val = exp_2.val$
$exp \rightarrow$ ***num***	$exp.isFloat = false$ $exp.val =$ **if** $exp.etype = int$ **then** ***num***$.val$ **else** FLOAT(***num***$.val$)
$exp \rightarrow$ ***num.num***	$exp.isFloat = true$ $exp.val =$ ***num***$.val$

6.21 forslag

a)



b)

Grammar Rule	Semantic Rule
fun-decls → **fun id (***var-decls***):** *type-exp* **;** *body*	*fun-decls.type = makeTypeNode (fun, var-decls.types, type-exp.type)*
$var\text{-}decls_1$ → $var\text{-}decls_2$;*var-decl*	$var\text{-}decls_1.types =$ $var\text{-}decls_2.types$ + *var-decl.type*
var-decls → *var-decl*	*var-decls.types = var-decl.type*
exp → **id(***exps***)**	**if** *isFunType(lookup(**id**.name))* **and** *exps.types = parameterTypesOf(**id**.name)* **then** *exp.type = lookup(**id**.name)* **else** *type-error*
$exps_1$ → $exps_2$ **,** *exp*	$exps_1.types = exps_2.types$ + *exp.type*
exps → *exp*	*exps.types = exp.type*

Forutsetter at

- *var-decls.types* defineres som en liste av de typer, som de enkelte var-decl bidrar med;
- *exps.types* defineres som listen av typene til de enkelte exp i listen av exp;
- funksjonen *parameterTypesOf* gir tilsvarende listen av de typer som finnes i TypeNode for funksjonen.

6.22 forslag

a)
Hvis parseren skal skille mellom cast uttrykk og aritmetiske uttrykk, da må parseren, når den behandler A, kunne slå opp i symboltabellen og finne ut om A er erklært som et typenavn eller som et variabelnavn. For å kunne gjøre dette må parseren innsette navn (sammen med oplysning om det er et typenavn eller et variablenavn) når det erklæres. Det er ikke nødvendig med mer typesjekking. Parseren konstruerer en node i syntakstreet som representerer typeuttrykket (A). Basert på dette tre kan parseren avgjøre om det følgende minustegn er ’unary’ eller ’binary’.

b)
Hvis skanneren skal skille mellom forskjellig bruk av A, da må skanneren kunne slå A opp i symboltabellen. Dette krever også at parseren setter inn navne på variable og typer når de erklæres, da skanneren ikke ser nok av input til å bestemme om et navn er et typenavn eller et variabelnavn. Hvis skanneren slår opp A og finner ut at det er et typenavn, da kan den returnere et tilsvarende token: i stedet for en ID kan den returnere en TypeID token. Parseren trenger da ikke å foreta noe oppslag. Basert på den token den får fra skanneren kan den generere det riktige syntakstre.