6.20 forslag

Grammar Rule	Semantic Rule

exp_1 → 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 → 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

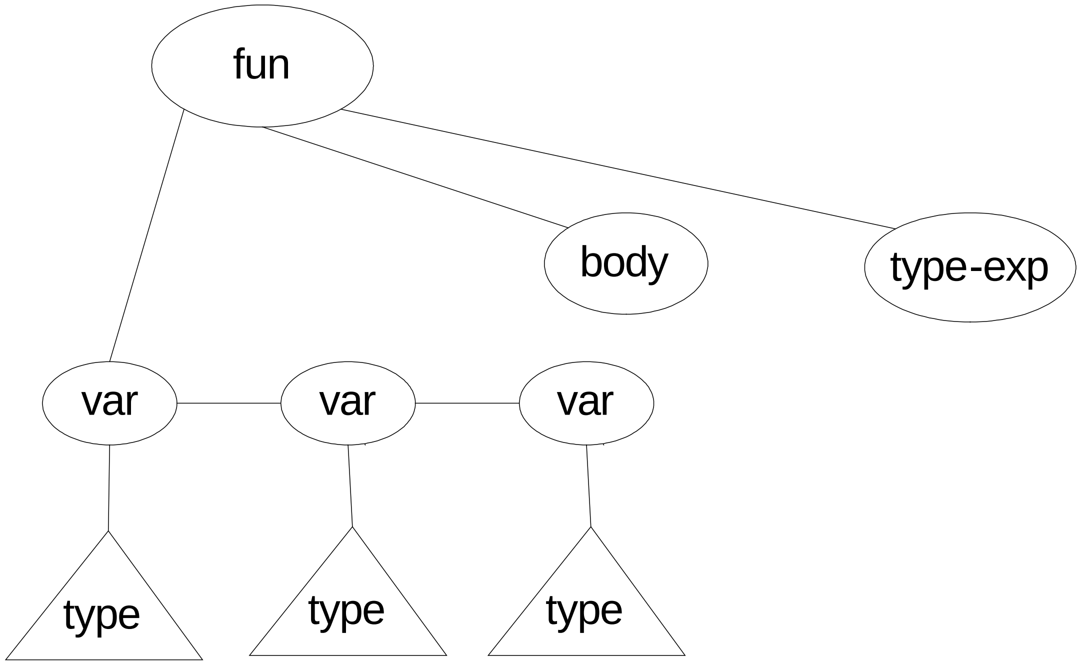
exp_1 → (exp_2)	exp_1.val = exp_2.val

exp → **num**	exp.isFloat = false
	exp.val = **num**.val

exp → **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-decls$_1$ → *var-decls*$_2$;*var-decl*	*var-decls* $_1$*.types =* *var-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.