Det følgende er den interessante del av grammatikken for et språk, hvor funksjoner kan ha maksimalt én parameter. Parameteroverføring er 'by-reference'. Språket har også klasser, men grammatikken for disse er ikke viktig her.

```
decls → decls ; decl | decl
decl → var-decl | function-decl
var-decl → type id = expression
function-decl → type id ( parameter? ) body
type → int | bool | void
parameter → type id
call → id ( expression )
expression → id | reference-expr.id | arith-expr | bool-expr
```

Ord i *kursiv* er ikke-terminaler, ord og tegn i **fet** skrift er terminal-symboler. ***Id*** representerer et navn.

Et *expression* kan enten være en ***id*** (som identifiserer en variabel), et 'remote expression' (et *reference-expr* etterfulgt av '.' og en ***id***) eller mer komplekse uttrykk med aritmetiske eller boolske operatorer. Detaljene for *reference-expr*, *arith-expr* og *bool-expr* er ikke viktige.

Den enkle regelen i dette språket er at da parameteroverføring er ’by-reference’, så må en aktuell parameter være enten variabel (dvs en ***id***), eller et ’remote expression’, som også viser til en variabel (men som en del av et objekt). Typen til den aktuelle parameteren skal være samme type som den formelle, men dette skal *ikke* uttrykkes i attributgrammatikken. En funksjon uten parameter må kalles uten aktuell parameter.

Fyll ut de tomme felter (markert med *) i følgende attributtgrammatikk slik at attributtet `ok` for `call` er `true` hvis kallet er gjort ifølge denne regelen (bortsett fra at typene stemmer overens), ellers `false`.

Du kan anta at det finnes semantiske regler som legger navn inn i symboltabellen. Du kan også anta at at `lookup(`***`id`***`.`*`name`*`).has_parameter` gir verdien `true` eller `false` for en funksjon (med navnet ***`id`***`.`*`name`*) avhengig av om funksjonen har en parameter eller ikke. Det er ikke behov for å sjekke om funksjonsnavnet (***`id`***) i en `call`-setning faktisk er deklarert.

Grammar Rule	Semantic Rule
function-decl → *type* **id ( )** *body*	
function-decl → *type* **id (** *parameter* **)** *body*	
call → **id** *()*	
call → **id** *(expression)*	
expression → **id**	
expression → *reference-expression***.id**	
expression → *arith-expr*	
expression → *bool-exp*	