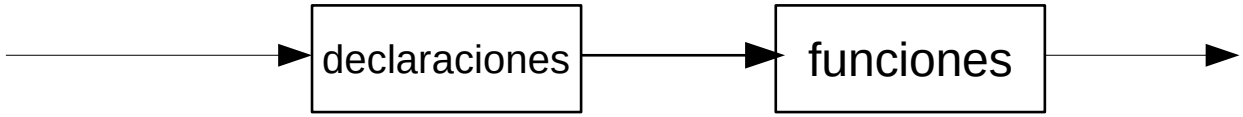
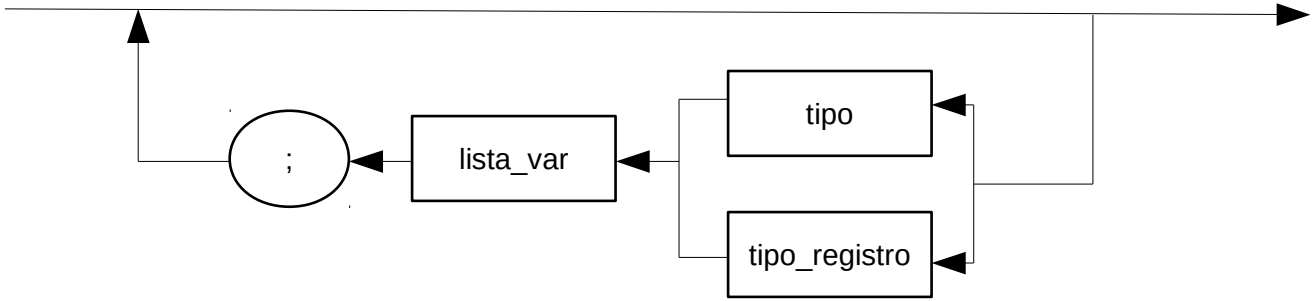
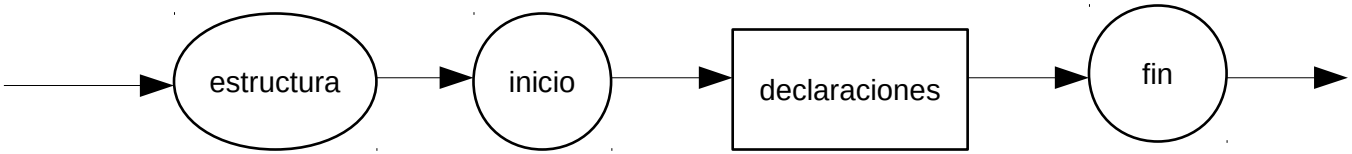
programa → declaraciones funciones



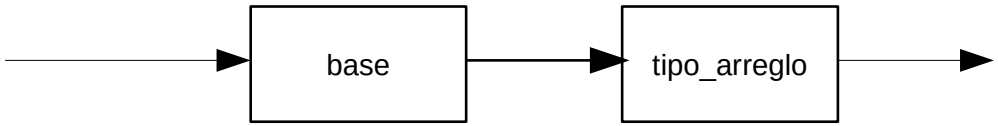
declaraciones → { (tipo | tipo_registro) lista_var ; }



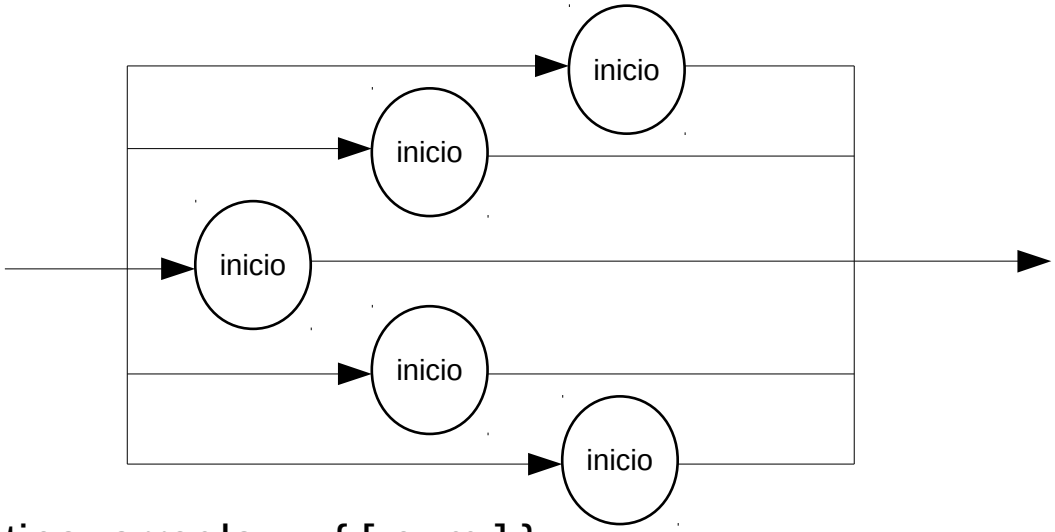
tipo_resgitro → estructura inicio declaraciones fin



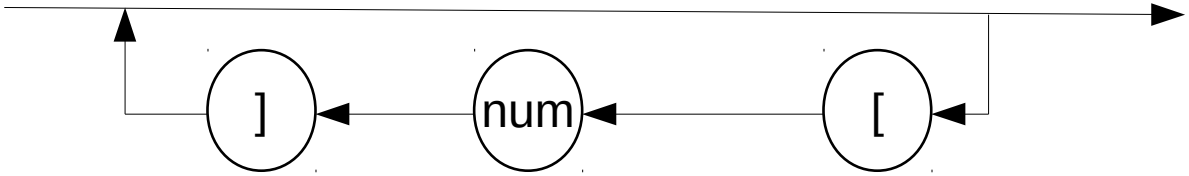
tipo → base tipo_arreglo



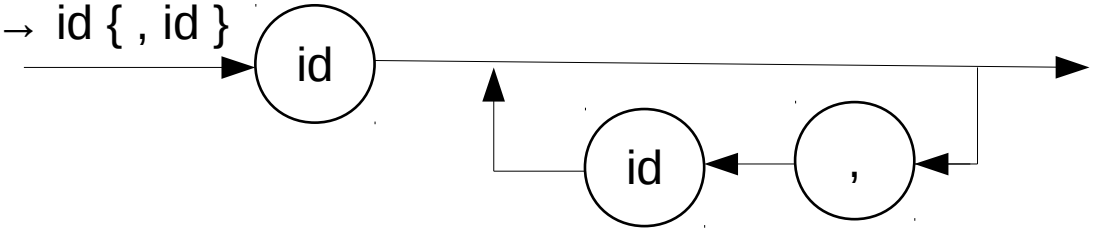
base → ent | real | dreal | car | sin



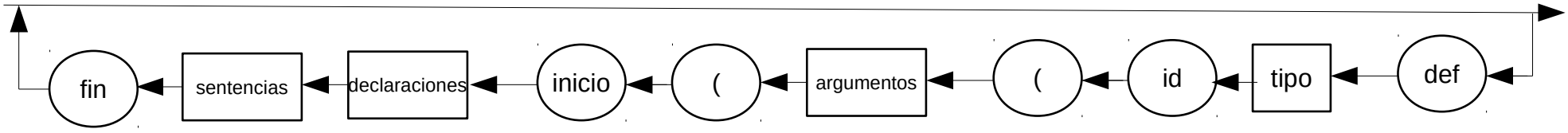
tipo_arreglo → { [num] }



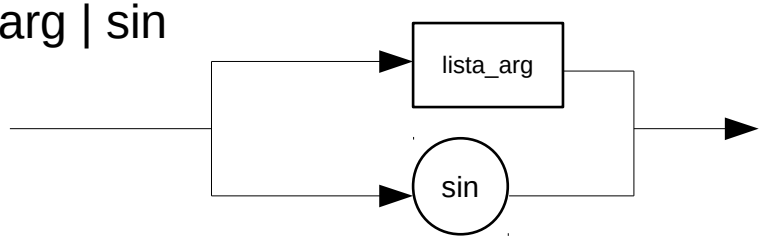
lista_var → id { , id }



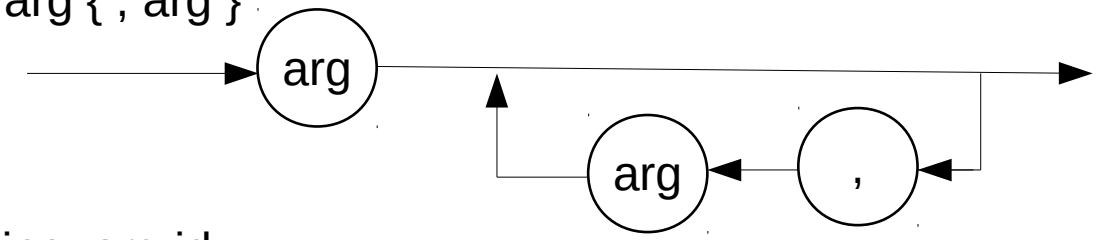
funciones → { def tipo id (argumentos) inicio declaraciones sentencias fin }



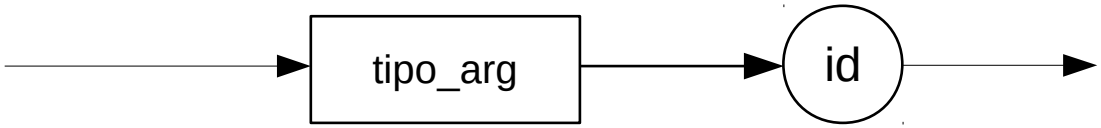
argumentos → lista_arg | sin



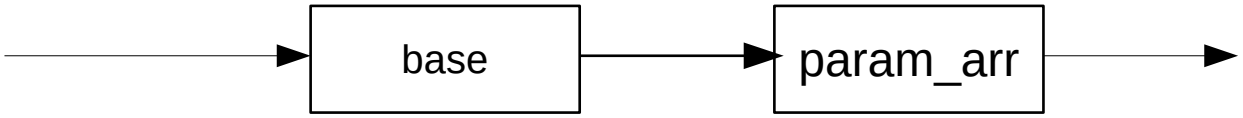
lista_arg → arg { , arg }



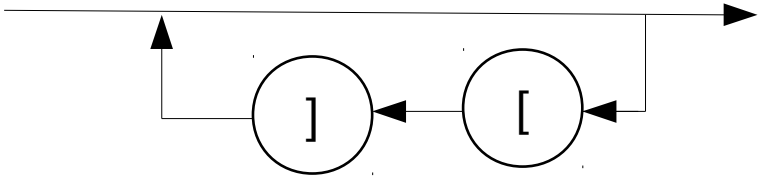
arg → tipo_arg id



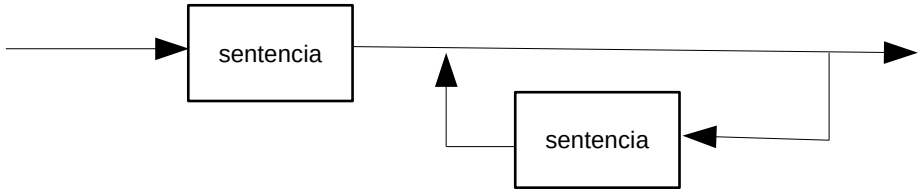
tipo_arg →base param_arr



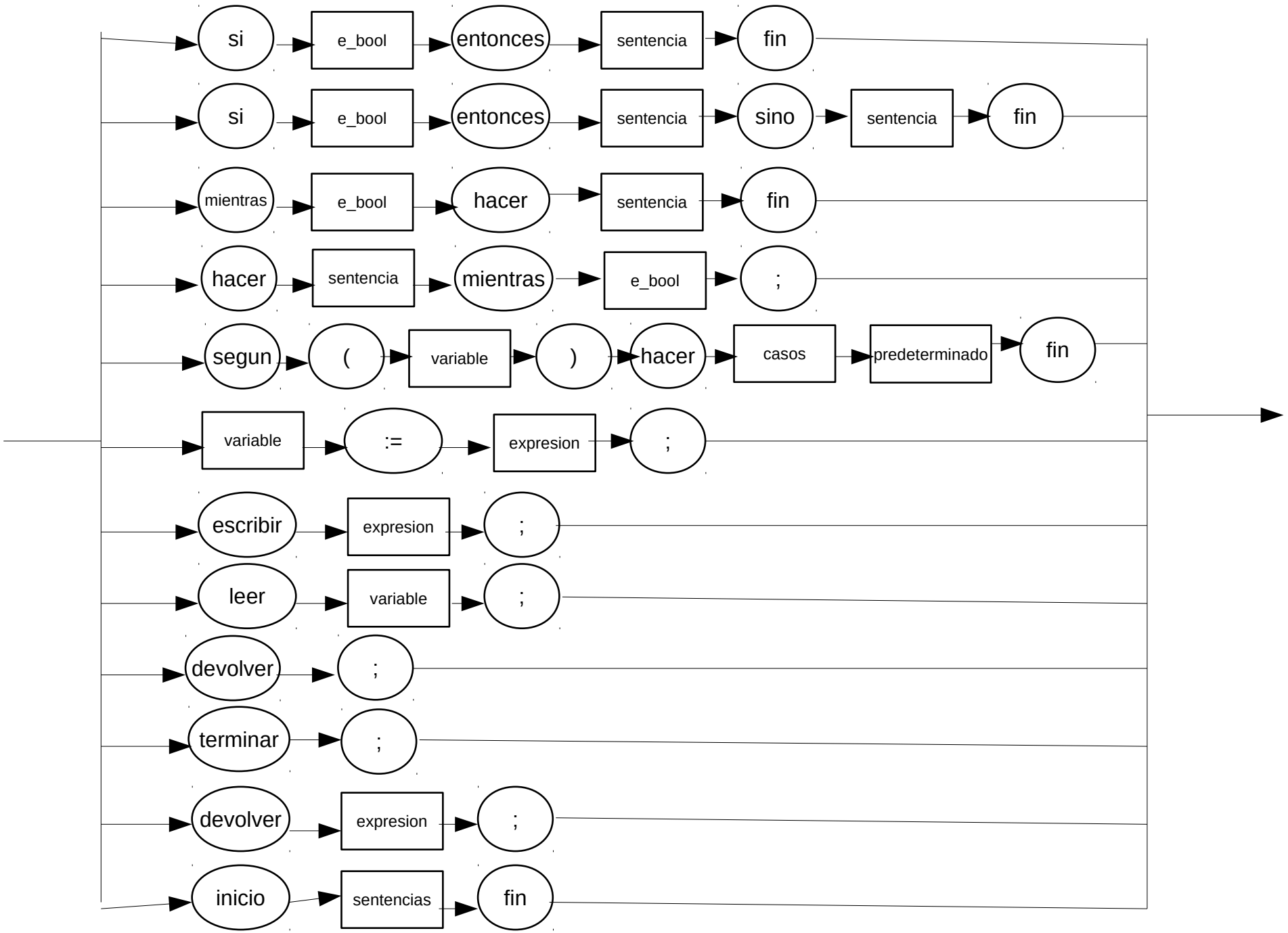
param_arr→ { [ ] }



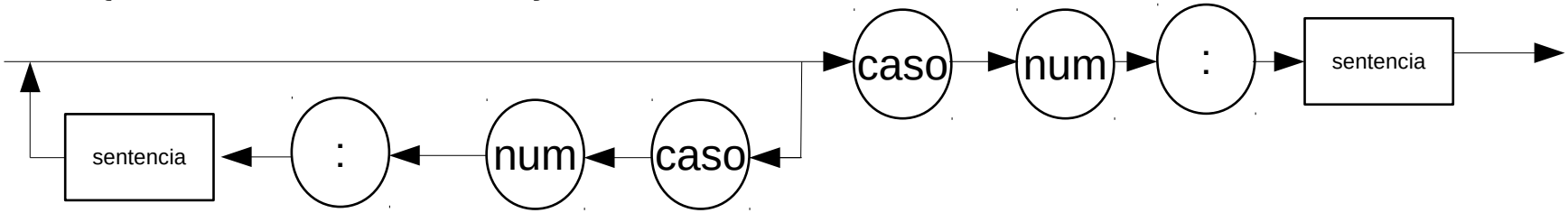
sentencias → sentencia { sentencia }



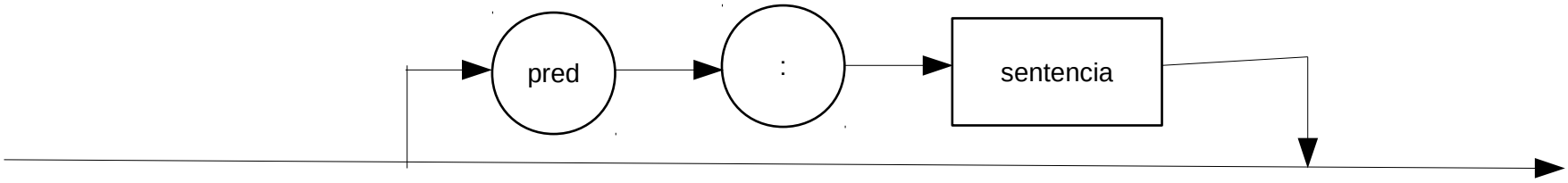
sentencia → si e_bool entonces sentencia fin
| si e_bool entonces sentencia sino sentencia fin
| mientras e_bool hacer sentencia fin
| hacer sentencia mientras e_bool ;
| segun (variable) hacer casos predeterminado fin
| variable := expresion ;
| escribir expresion ;
| leer variable ;
| devolver ;
| devolver expresion ;
| terminar ;
| inicio sentencias fin


si
e_bool
entonces
sentencia
fin
si
e_bool
entonces
sentencia
sino
sentencia
fin
mientras
e_bool
hacer
sentencia
fin
hacer
sentencia
mientras
e_bool
;
segun
(
variable
)
hacer
casos
predeterminado
fin
variable
:=
expresion
;
escribir
expresion
;
leer
variable
;
devolver
;
terminar
;
devolver
expresion
;
inicio
sentencias
fin

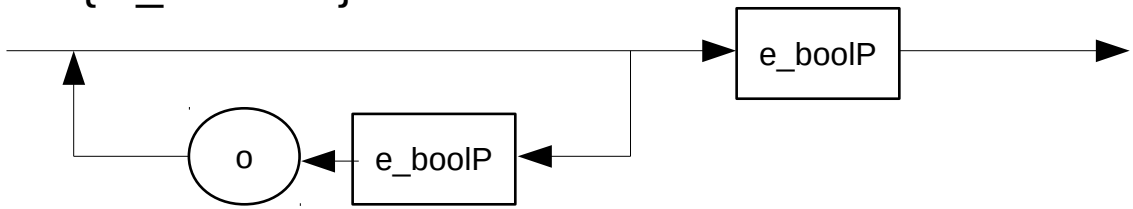
casos → { caso num: sentencia } caso num: sentencia



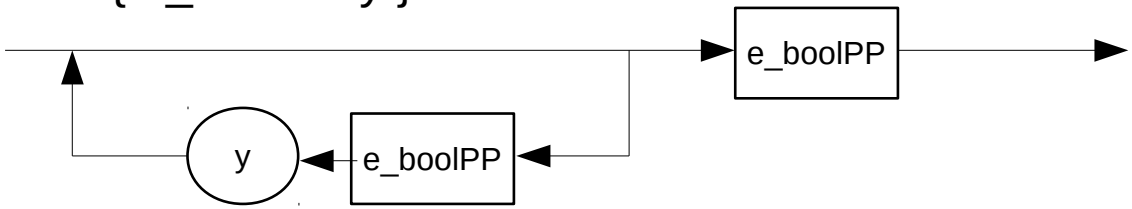
predeterminado → pred : sentencia



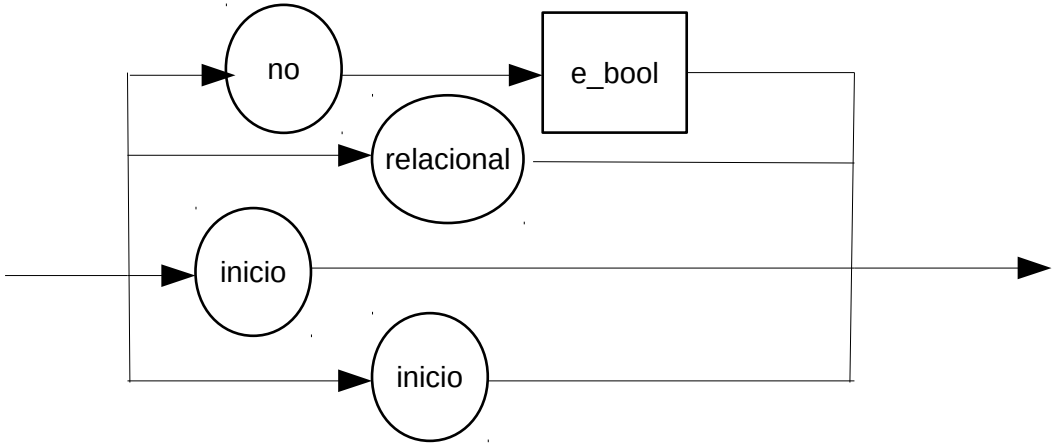
e_bool → { e_boolP o } eboolP



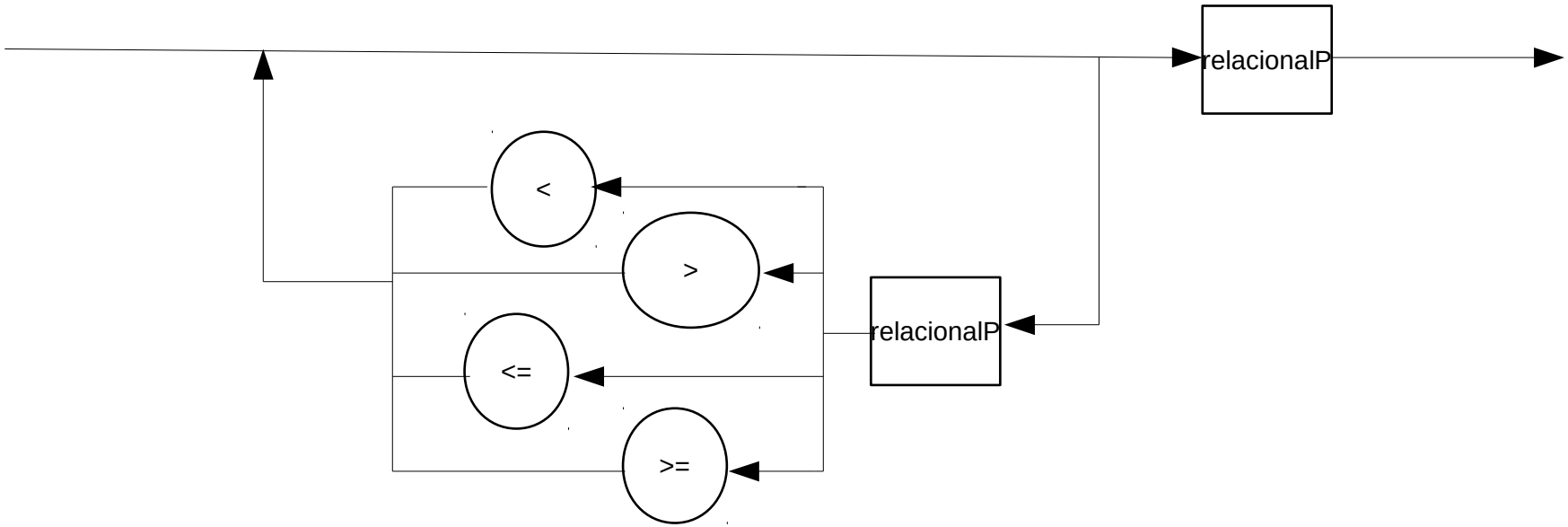
e_boolP → { e_boolPP y } eboolPP



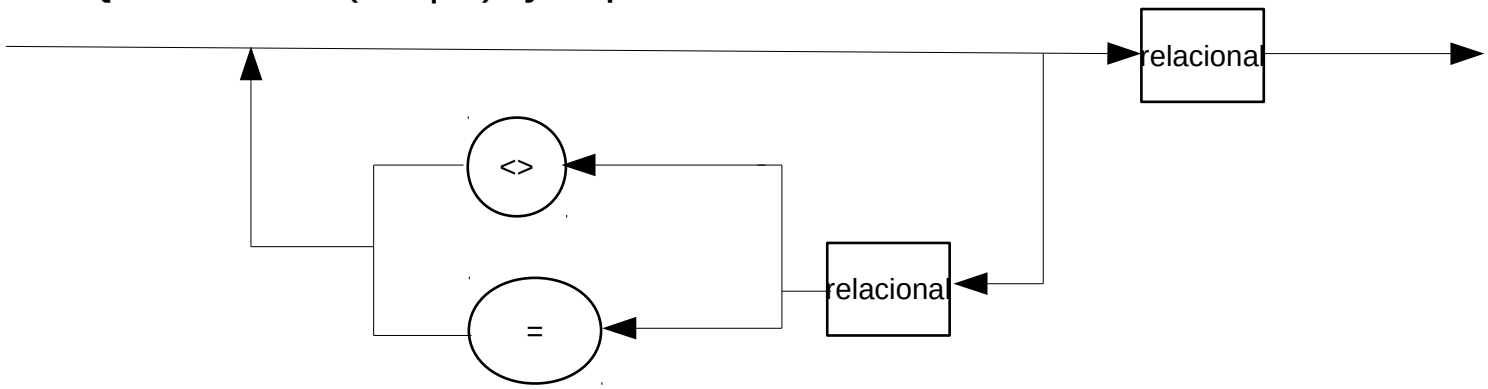
e_boolPP → no ebool | relacional | verdadero | falso



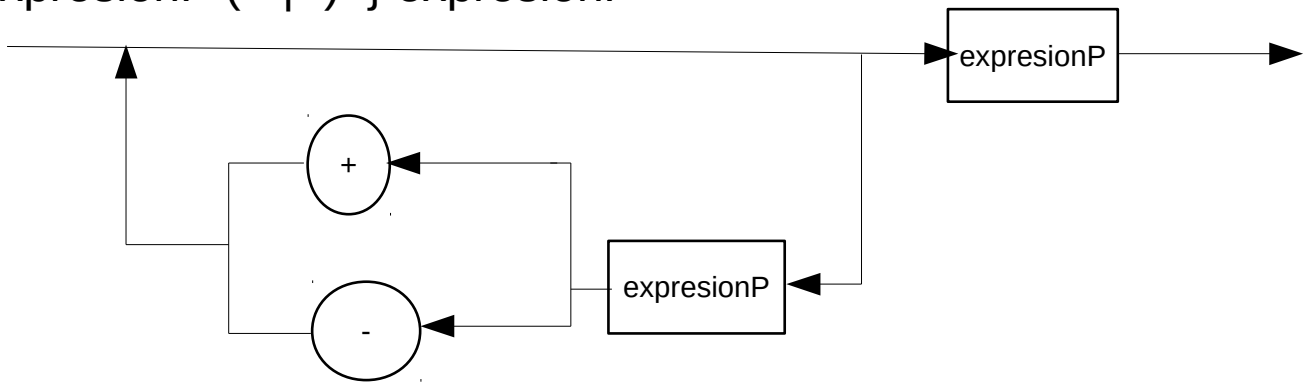
relacional → { relacionalP (< | > | <= | >=) } relacionalP



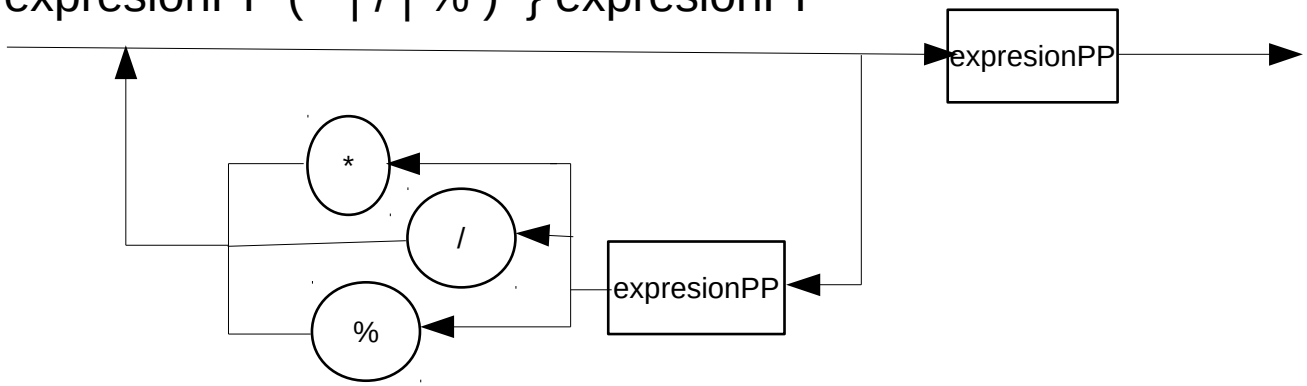
relacionalP → { relacional (<> | =) } expresion



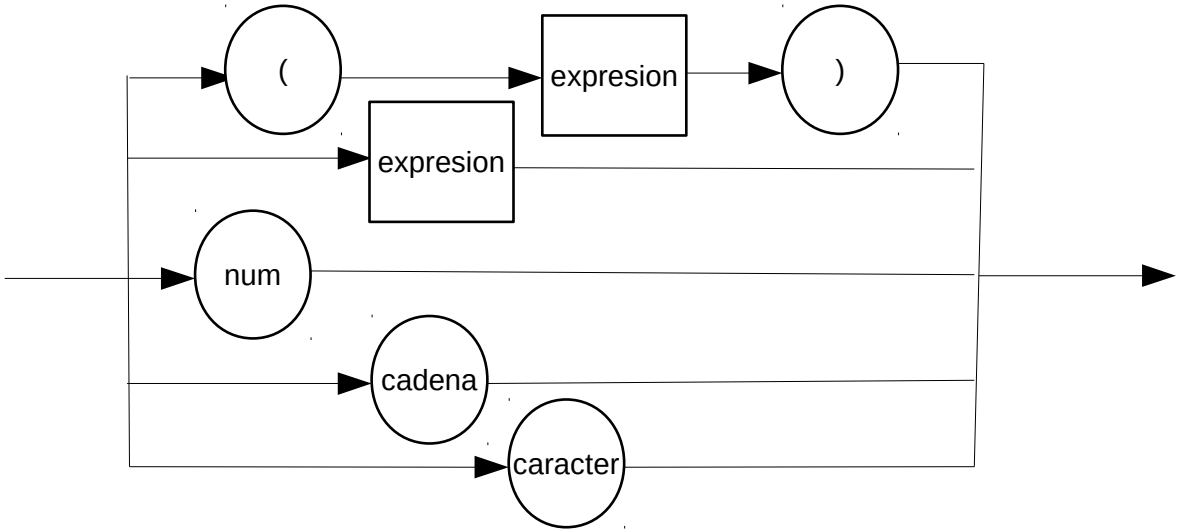
expresion → { expresionP (+ | -) } expresionP



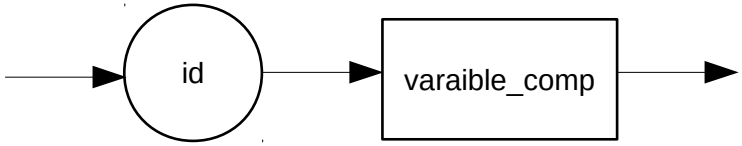
expresionP → { expresionPP (* | / | %) } expresionPP



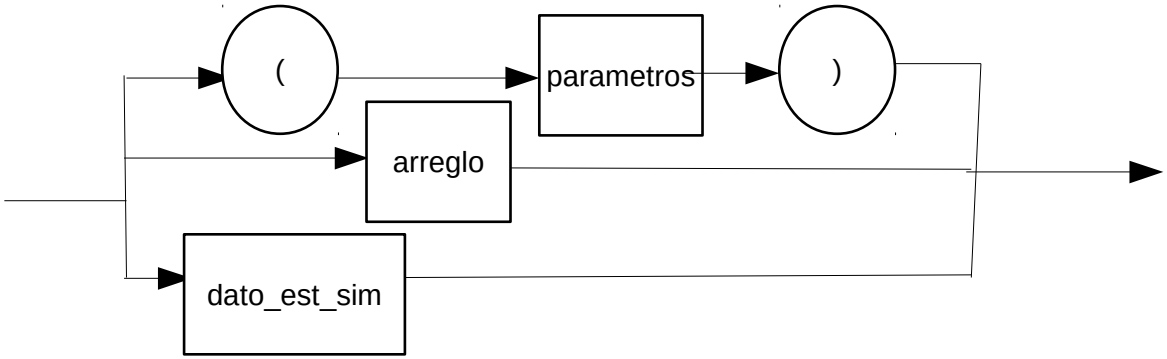
expresionPP → (expresion) | variable | num | cadena | caracter



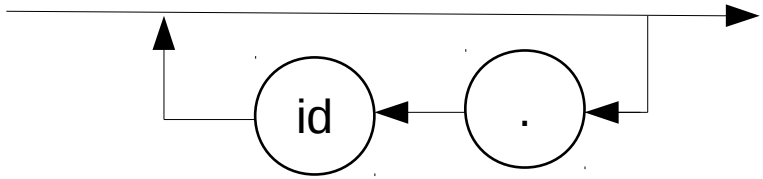
variable → id variable_comp



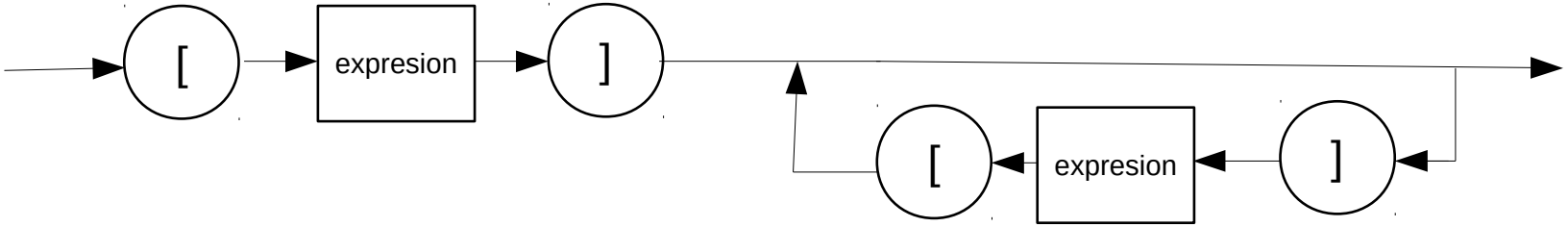
variable_comp → dato_est_sim | arreglo | (parametros)



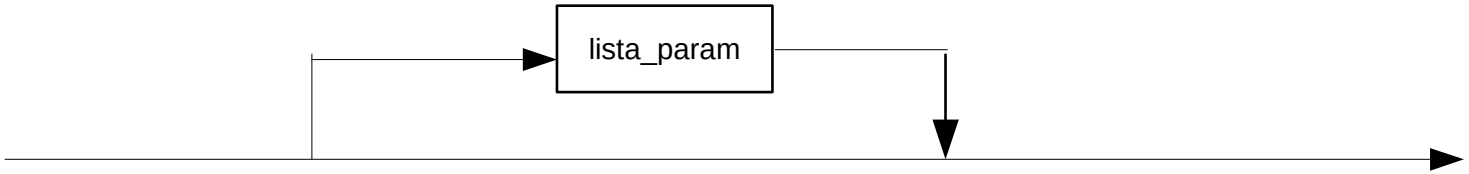
dato_est_sim → { . id }



arreglo → [expresion] { [expresion] }



parametros → lista_param



arreglo → [expresion] { , expresion }