import BOOL, INT, REAL, K-BASIC

eval : Exp → Val
result : State → Val

Var, Bool, Int, Real < Exp
Bool, Int, Real < Val

not : Exp → Exp ![notBool : Bool → Bool]
± : Exp × Exp → Exp ![±Int : Int × Int → Int, ±Real : Real × Real → Real]
≤ : Exp × Exp → Exp ![≤Int : Int × Int → Bool, ≤Real : Real × Real → Bool]
skip : → Exp [unit : → Val]

if-then-else : Exp × Exp × Exp → Exp ![ifThenElse : Exp × Exp × Exp → Exp]
fun : VarList × Exp → Exp

 closure : VarList × Exp × VarLocSet → Val

let, letrec : VarList × ExpList × Exp → Exp

cons : Exp × Exp → Exp ![cons : Exp × Exp → Exp]

while : Exp × Exp → Exp

Figure 8: K definition of a simple functional language