```haskell
-- 1

{-

LET True = \ x y. x
LET False = \ x y. y

LET equ = \a b . a b (b False True)

equ True True =
(\a b . a b (b False True)) True True =
(\b . True b (b False True)) True =
True True (True False True) =
(\ x y. x) (True False True) =
(\ y. True) (True False True) =
True

-}

-- 2

data Expr a
    = Var a
    | IVal Int
    | Add (Expr a) (Expr a)
    deriving (Show,Eq)

subst :: Eq a => Expr a -> a -> Int -> Expr a
subst o@(IVal _) _ _ = o
subst o@(Var v)  w i =
    if v==w then IVal i else o
subst (Add l r) w i =
    Add ll rr
    where
        ll = subst l w i
        rr = subst r w i

-- 3

{-

sum [] = 0                    -- 1
sum (x:xs) = x + sum xs       -- 2

    [] ++ ys = ys             -- 3
(x:xs) ++ ys = x:(xs ++ ys)   -- 4

sum (xs ++ ys) = sum xs + sum ys

1)
xs = []
L = sum ([] ++ ys) =|3
  = sum (ys) =|zbytečné závorky
  = sum ys
P = sum [] + sum ys =|1
  = 0 + sum ys =|0jeNulovýPrvekProSčítání
  = sum ys
L = P

2)
IH: sum (as ++ ys) = sum as + sum ys
xs = (a:as)
L = sum ((a:as) ++ ys) =|4
  = sum (a:(as ++ ys)) =|2
  = a + sum (as ++ ys) =|IH
  = a + sum as + sum ys
P = sum (a:as) + sum ys =|2
  = a + sum as + sum ys
L = P

Q.E.D.

-}

-- EOF
```