```haskell
-- 1

{-

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

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

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

-}

-- 2

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

cf :: Expr a -> Expr a
cf v@(Var _) = v
cf v@(IVal _) = v
cf (Add l r) =
    case (cf l, cf r) of
        (IVal ll, IVal rr) -> IVal (ll+rr)
        (lx,rx) -> Add lx rx

--3

{-

and [] = True                 -- 1
and (x:xs) = x && and xs      -- 2

and xs = foldr (&&) True xs

foldr f z [] = z                            -- 3
foldr f z (x:xs) = f x (foldr f z xs)    -- 4

1)
xs = []

L = and [] =|1
  = True

P = foldr (&&) True [] =|3
  = True

L = P

2)
IH: and as = foldr (&&) True as
xs = (a:as)

L = and (a:as) =|2
  = a && and as =|IH
  = a && foldr (&&) True as =|zavorky na zdurazneni priorit
  = a && (foldr (&&) True as) =|infix->prefix
  = (&&) a (foldr (&&) True as)

P = foldr (&&) True (a:as) =|4
  = (&&) a (foldr (&&) True as)

L = P

Q.E.D.
```

```
-}
```