```
data  Expr
=  Add  Expr  Expr
|  Mul  Expr  Expr
|  Con  Int
```

```
type ESem  b
= (  b → b →  b
  ,  b → b →  b
  ,  Int    →  b  )
```

```
foldE ::    ESem b   →   Expr → b
foldE  (a,m,c)     =  f  where
  f (Add e1 e2)    =  a  (f e1) (f e2)
  f (Mul e1 e2)    =  m (f e1) (f e2)
  f (Con n)        =  c  n
```

```
evalExpr :: Expr → Int
evalExpr = foldE evalSem
```

```
evalSem :: ESem Int
evalSem = (  (+)  , (*)  , id  )
```