Partial vs total functions

```
head :: [a] -> a
head (x : _) = x

headV :: Vec (S n) a -> a
headV (VCons x _) = x
```

partial — total

same implementation (modulo constructor names)

but type is different

⇑ undefined for empty list

⇑ defined for all values of type Vec (S n) a

=

```
data SNat n where
  Zero :: SNat Z
  Succ :: SNat n -> SNat (S n)
  Test :: SNat True
```

↑ Q: why does the type checker complain here and restrict the argument to Nat?

A: after looking at the first constructor (Zero), it concludes that argument has to have kind Nat (only works if "DataKinds" are enabled.
Second def. is ok, because n :: Nat and S n :: Nat
'Test' is not ok, because True :: Bool

Ok, but why is the compiler ok with this:

```
data Expr a where
  Add :: Expr Int -> Expr Int -> Expr Int
  BoolLit :: Expr Bool
```

where Expr has arg Int in first constructor, Bool in second.

A: Because both Int and Bool have the same kind (*), just like Z, n, S n in the SNat example have the same kind.