# 1 Applicative laws

Identity, composition, homomorphism, interchange.

**Definition : Identity**

```
pure id <*> x
```

**Definition : Composition**

```
pure (.)  <*> u <*> v <*> w
u <*> (v <*> w)
```

**Definition : Homomorphism**

A *homomorphism* is a structure-preserving map between 2 algebraic structures.

**Definition : Interchange**

```
u <*> pure y = pure ($ y) <*> u
```

```haskell
cowFromString :: String
              -> Int
              -> Int
              -> Maybe Cow
cowFromString name' age' weight' =
  case noEmpty name' of
    Nothing -> Nothing
    Just nammy ->
      case noNegative age' of
        Nothing -> Nothing
        Just agey ->
          case noNegative weight' of
            Nothing -> Nothing
            Just weighty ->
              Just (Cow nammy agey weighty)

cowFromString' :: String
               -> Int
               -> Int
               -> Maybe Cow
cowFromString' name' age' weight' =
  Cow <$> noEmpty name' <*> noNegative age' <*> noNegative weight'
```