**Definition : Catamorphism**

*”Cata”* means ”down” or ”against”, as in ”catacombs”. Catamorphisms are means of deconstructing data. If the spine of a list is the structure of a list, then a fold is what can reduce that structure.

Where a fold allows to break down a list into an arbitrary datatype, a catamorphism is a means of breaking down the structure of any datatype (bool func).

# 1 Fold right

```haskell
foldr f z (x:xs) = f x (foldr f z xs)
--                     ^--------------^
--                     rest of the fold
```

If $f$ doesn’t evaluate its second argument (rest of the fold), no more spine will be forced. For this reason, foldr can be used with lists that are potentially infinite.

The first piece of the spine, the first *cons cell* cannot be undefined.

# 2 Fold left

Because foldl must evaluate its whole spine before it starts evaluating in each cell, it accumulates a pile of unevaluated values as it traverses the spine.

foldl’ (foldl prime) works the same except it is strict, has less negative effect on performance over long lists.

Only beginning to produce values **after reaching the end of the list**. Nearly useless, gotta use foldl’.

**Definition : Tail call**

A tail call is the final result of a function. (foldl)