99 questions/Solutions/14

From HaskellWiki

< 99 questions | Solutions

(*) Duplicate the elements of a list.

```
dupli [] = []
dupli (x:xs) = x:x:dupli xs
```

or, using list comprehension syntax:

```
dupli list = concat [[x,x] | x <- list]
```

or, using the list monad:

```
dupli xs = xs >>= (\x -> [x,x])
```

or, using the list instance of
`Applicative`
:

```
dupli = (<**> [id,id])
```

or, using
`concatMap`
:

```
dupli = concatMap (\x -> [x,x])
```

also using
`concatMap`
:

```
dupli = concatMap (replicate 2)
```

or, using
`foldl`
:

```
dupli = foldl (\acc x -> acc ++ [x,x]) []
```

or, using
`foldr`
:

```
dupli = foldr (\ x xs -> x : x : xs) []
```

or, using silliness:

```
dupli = foldr (\x -> (x:) . (x:)) []
```

or, even sillier:

```
dupli = foldr ((.) <$> (:) <*> (:)) []
```

and here is the proof that
`((.) <$> (:) <*> (:)) = (\y z -> y:y:z)`
:

```haskell
(.) <$> (:) <*> (:) =
((.) <$> (:)) <*> (:) = -- (<$>) is infixl 4, (<*>) is infixl 4
((.) . (:)) <*> (:) = -- (<$>) == (.) for functions
(\x -> (.) (x:)) <*> (:) = -- definition of (.)
\y -> (\x -> (.) (x:)) y (y:) = -- definition of (<*>) for functions
\y -> ((.) (y:)) (y:) = -- beta reduction (applying y to (\x -> (.) (x:)))
\y -> (y:) . (y:) = -- changing (.) to its prefix form
\y -> (\z -> (y:) ((y:) z)) = -- definition of (.)
\y z -> y:y:z -- making it look nicer
```