There's a particular function on lists that I'm a little obsessed with:

```
co : ∀ {a} {A B : Set a} → List A → List B → List (List (A × B))
co _ [] = []
co {_} {A} {B} xs (yh :: ys) = foldr f [] xs
  where
  g : A → B → (List (List (A × B)) → List (List (A × B))) → List (List (A × B)) → List (List (A × B))
  g x y a (z :: zs) = ((x , y) :: z) :: a zs
  g x y a [] = [(x , y)] :: a []
  f : A → List (List (A × B)) → List (List (A × B))
  f x zs = [ x , yh ] :: foldr (g x) id ys zs
```

It's an implementation of discrete convolution on lists. Previously I discussed it in relation to search patterns: it corresponds (somewhat) to breadth-first search (rather than depth-first).

Here though, I want to talk about its more traditional interpretation: the multiplication of two polynomials. Indeed, if you write out your polynomial backwards:

$$\begin{aligned} & 2x^2 && +x && - && 4 && (1)\\ = & 2x^2 && +1x^1 && +- && 4x^0\{\text{With explicit powers of } x\} && (2)\\ = & -4x^0 && +1x^1 && + && 2x^2\{\text{Reversed}\} && (3) \end{aligned}$$

– Reflection – Counterexamples