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

```
conv : {A B : Set} → List A → List B → List (List (A × B))
conv _ [] = []
conv {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:

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

[?]

References

[1] E. Rivas, M. Jaskelioff, and T. Schrijvers, "From monoids to near-semirings: The essence of MonadPlus and Alternative," in *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming*. ACM, 2015, pp. 196–207, https://www.reddit.com/r/haskell/comments/3dlz6b/from_monoids_to_nearsemirings_the_essence_of/ [Online]. Available: http://www.fceia.unr.edu.ar/~mauro/pubs/FromMonoidstoNearsemirings.pdf