Write-protected.

Save anyway?

Essays on Programming, Mathematics, and Attempting to Understand the World

Mitchell Vitez

With thanks to E. A. Chavis, a true friend and shining wit. Your feedback made this better.

This is (largely) a work of non-fiction. Despite that, some names, characters, places, and incidents are either the product of the author's imagination, or are used ficticiously. Any resemblance to actual persons—living, dead, or in superposition—is entirely coincidental.

The number of mistakes contained here is not zero, for which I take full responsibility. The Haskell bits are probably closest to correct, because at least they were typechecked. Read the rest at your own risk.

Cover design by Mitchell Vitez. Photoshop on canvas, 2020.

Compliments and cheery witticisms may be directed to `mitchell@vitez.me`. Likewise for well-reasoned disagreement. Scathing and unreasonable critiques may be directed to `nobody@example.com`

Published via `leanpub.com`, whose services I heartily recommend

The author may be most easily located via `vitez.me`

Write-protected. Save anyway?

Essays on Programming, Mathematics, and Attempting to Understand the World

Mitchell Vitez

Preface

I have long been the kind of person who figured I'd probably write a book…someday. I sure read enough of them. How hard can it be?

Well, it turns out to be quite the endeavor just to write a bunch of words.

I began writing in earnest on January 1, 2019. For the first fifty days of that year, I wrote an essay every day, with many of them breaking the 1000 word mark. Back then, I felt that I was a little tired of being read-only, and thought it was time to write. I'm not quite so tired anymore.

This book is structured as a collection of essays, with fairly wide-ranging topics. Many of the essays are about programmingish or mathy subjects, especially functional programming and Haskell. This is because Haskell's maybe the only topic I'm anywhere close to having even a smidge of expertise about. If you're not into that kind of thing, maybe it'll spark some intrigue? But of course, feel free to breeze through. I won't be able to stop you.

The essays with topics straying farther from that core tend to be weaker, admittedly, but I had tons of fun thinking about the ideas involved. There's even a chapter dedicated to "Weird Thoughts", which probably contains the most bad arguments per capita, but also some of the most intriguing ideas. I have not gone through and updated the arguments since the time I first made them, so a few may be outdated.

I tend to write about technical (or at least vaguely technical-ish) topics in a very informal, hand-wavy style. I use the word "things" a lot, when I just need some handy

abstraction for an arbitrary referent. The writing is not anywhere close to a textbook's veracity, but instead is much more conversational. I of course encourage the reader to do their own research, especially about anything unfamilar or that I seem especially hand-wavy about. Forming the correct conclusion is, in all cases, left as an exercise for the reader.

People sometimes say a great book exists to fill a niche. This book doesn't really fill a strong niche, so maybe it will never be a great book. I'm okay with that. As long as I give you something to think about, and you enjoy yourself, I think we'll get along just fine.

Contents

Computer Science

Infinite Recursive Permutation Generators

Consider the following ten lines of code:

```
def gen_perms(s, n=1):
    if n < 1:
        yield ''
    for prefix in gen_perms(s, n-1):
        for c in s:
            yield prefix + c
    gen_perms(s, n+1)

for perm in gen_perms('abcdefghijklmnopqrstuvwxyz'):
    print(perm)
```

In a nutshell, this prints out every possible permutation of letters of the alphabet, in "spreadsheet column name" order (a, b, c…, aa, ab, ac…, aaa, aab, aac…). Every dictionary word. Every great speech. Entire (unpunctuated) works of Shakespeare.

For now, we'll ignore a few basic improvements. This could easily be generalized to work with more than just strings. The runtime isn't too great either since we're doing a bunch of recalculation. Memoization and choosing better data types would fix these, but let's focus on the aesthetic qualities of this code, and on how it actually works.

Starting with the easy bit. Assume we have an infinite permutation generator called `gen_perms` that takes a string and gives back all the possible permutations of that string based on the order the characters were passed in. In that case, this loop will run through those infinite permutations and print out each one:

```
for perm in gen_perms('abcdefghijklmnopqrstuvwxyz'):
    print(perm)
```

Let's get into the real meat of it though. `gen_perms` takes a string `s`, and a parameter `n`. You can think of `n` as the length of the permutations currently being generated.

```
def gen_perms(s, n=1):
```

The body of the function has three parts. First of all, if `n` is less than one, we want to give back an empty string. This is our recursive base case, and should be pretty straightforward.

```
    if n < 1:
        yield ''
```

Glossing over the tricky middle bit for a second, here's the third part of the function body. Just like we need a base case for termination, if we want infinite generation we need an ever-growing case. Here, we just call `gen_perms` with the next largest permutation length so it can compute all of those for us. This will keep growing as `n` reaches ever higher towards infinity.

```
    gen_perms(s, n+1)
```

Here's the fun stuff. First, we inductively assume that `gen_perms` works for `n-1`. That should generate all the permutations of length $n-1$ for us, and we'll iterate over each of them as `prefix`. Because a prefix is of length $n-1$, and we want to generate a permutation of length n, all we need to do is add a single character to `prefix`. These single characters have an obvious source: each character in `s`, in order.

Eventually, with all the possible prefixes having all the possible characters in `s` appended to them, we manage to come up with all the possible permutations of length n.

```
    for prefix in gen_perms(s, n-1):
        for c in s:
            yield prefix + c
```

This code manages to combine several fairly tricky topics—recursion, infinity, permutations, generators—into one short sample, which I think makes it a super fun example to work with. For bonus points, try writing the memoized version, or a version that works with `set`s instead of `str`s!