Peter's Sum-Choose Theorem

John Berberian, Jr. Peter Tessier

December 27, 2023

Contents

Acknowledgements

Ben Furukawa, for suggesting an inductive approach.

1 Introduction

We intend to prove:

$$\binom{a+b}{b+1} = \sum_{i_0=1}^{a} \sum_{i_2=1}^{i_0} \sum_{i_2=1}^{i_3} \cdots \sum_{i_b=1}^{i_{b-1}} 1 \tag{1}$$

2 Examples

To help the reader understand equation 1, we provide some examples:

$$\begin{aligned}
\text{(i) } a, b = a, 0 : \binom{a+0}{0+1} &= a = \sum_{i_0=1}^{a} 1 \\
\text{(ii) } a, b = 3, 1 : \binom{3+1}{1+1} = \binom{4}{2} &= 6 = 1 + 2 + 3 = 1 + (1+1) + (1+1+1) \\
&= \sum_{i_0=1}^{3} i_0 = \sum_{i_0=1}^{3} \sum_{i_1=1}^{i_0} 1 \\
\text{(iii) } a, b = 3, 2 : \binom{3+2}{2+1} = \binom{5}{3} &= 10 = 1 + 3 + 6 = 1 + (1+2) + (1+2+3) \\
&= \sum_{i_0=1}^{3} \sum_{i_1=0}^{i_0} i_1 = \sum_{i_0=1}^{3} \sum_{i_1=0}^{i_0} \sum_{i_2=1}^{i_1} 1
\end{aligned}$$

3 Lemma

$$\binom{a+1}{n+1} = \sum_{i=1}^{a} \binom{i}{n} \mid n, a \in \mathbb{N} \tag{2}$$

We prove by induction.

3.1 Base Case

We begin by proving this for the base case $a = 1$.

$$\binom{2}{n+1} = \binom{1}{n} \tag{3}$$

For $n = 1$, both sides are equal to 1. For $n > 1$, both sides are equal to 0. So equation 3 is true.

3.2 Inductive Step

We now seek to show the inductive step. First we show the following two equations hold:

$$\binom{a}{n+1} = \frac{a!}{(a-n-1)!(n+1)!} = \frac{a-n}{n+1}\frac{a!}{(a-n)!n!} = \frac{a-n}{n+1}\binom{a}{n} \tag{4}$$

$$\begin{aligned}\binom{a+1}{n+1} &= \frac{(a+1)!}{(n+1)!(a-n)!} = \frac{a+1}{n+1}\cdot\frac{a!}{n!(a-n)!} = \frac{a+1}{n+1}\binom{a}{n} \\ &= \frac{n+1+a-n}{n+1}\binom{a}{n} = \left(1+\frac{a-n}{n+1}\right)\binom{a}{n}\end{aligned} \tag{5}$$

We then prove that if equation 2 holds for any $a-1 \in \mathbb{N}$, it must hold for a:

$$\begin{aligned}\binom{a}{n+1} &= \sum_{i=1}^{a-1}\binom{i}{n} \\ = \frac{a-n}{n+1}\binom{a}{n} &= \sum_{i=1}^{a-1}\binom{i}{n} \text{ (by Eq. (4))} \\ \implies \binom{a}{n} + \frac{a-n}{n+1}\binom{a}{n} &= \sum_{i=1}^{a-1}\binom{i}{n} + \binom{a}{n} \\ = \left(1+\frac{a-n}{n+1}\right)\binom{a}{n} &= \sum_{i=1}^{a}\binom{i}{n} \\ = \binom{a+1}{n+1} &= \sum_{i=1}^{a}\binom{i}{n} \text{ (by Eq. (5))}\end{aligned}$$

So, because the base case and inductive step hold, equation 2 is true for any $a \in \mathbb{N}$ by induction.

4 Proof

The right-hand side of equation 1 can be rewritten as a recursive function:

$$f(a,b) = \begin{cases} a, & \text{if } b = 0 \\ \sum_{i=1}^{a} f(i, b-1), & \text{otherwise} \end{cases}$$

We easily show this by induction.

The base case of $b = 0$ is shown to have $f(a,b)$ equal to the right side of equation (1) via example (i).

For $b > 0$, we assume:

$$f(a, b-1) = \sum_{i_0=1}^{a} \sum_{i_1=1}^{i_0} \sum_{i_2=1}^{i_1} \cdots \sum_{i_{b-1}=1}^{i_{b-2}} 1 \tag{6}$$

So,

$$\begin{aligned} f(a,b) &= \sum_{i=1}^{a} f(i, b-1) \\ &= \sum_{i=1}^{a} \sum_{i_0=1}^{i} \sum_{i_1=1}^{i_0} \sum_{i_2=1}^{i_1} \cdots \sum_{i_{b-1}=1}^{i_{b-2}} 1 \text{ (by Eq. (6))} \\ &= \sum_{i_0=1}^{a} \sum_{i_1=1}^{i_0} \sum_{i_2=1}^{i_1} \cdots \sum_{i_b=1}^{i_{b-1}} 1 \text{ (relabelling indices)} \end{aligned}$$

Having shown that $f(a, b)$ indeed equals the right hand side of equation 1, we now observe that if $f(i, b-1)$ can be expressed as $\binom{i}{n}$ for some $n \in \mathbb{N}$, then by equation 2,

$$f(a,b) = \sum_{i=1}^{a} f(i, b-1) = \sum_{i=1}^{a} \binom{i}{n} = \binom{a+1}{n+1}$$

Trivially,

$$f(a, 0) = a = \binom{a}{1}$$

Hence,

$$f(a, 1) = \binom{a+1}{1+1}$$

And more generally,

$$f(a, b) = \binom{a+b}{b+1}$$

$\square$

Appendix A: Numerical Verification Script

```python
#!/usr/bin/env python3
import math

def crunch(this, a, b):
    if b == 0:
        return a
    return sum((this(this, i, b-1) for i in range(1, a+1)))

def crunch_single_slow(this, a, b, c):
    if b == 0:
        return a if a == c else 0
    return sum((this(this, i, b-1, c) for i in range(1, a+1)))

def memoize_recursive(func):
    cache = dict()
    def memoized(this, *args):
        if args in cache:
            return cache[args]
        result = func(this, *args)
        cache[args] = result
        return result
    return memoized

def recursive_wrapper(func):
    def wrapper(*args):
        return func(func, *args)
    return wrapper

def check_generic(a, b, testfunc):
    hypothesis = math.comb(a+b, b+1)
    ourresult = testfunc(a, b)
    return (ourresult == hypothesis, hypothesis, ourresult)

crunch_memo = recursive_wrapper(memoize_recursive(crunch))
crunch_single = recursive_wrapper(memoize_recursive(crunch_single_slow))

def check(a, b):
    return check_generic(a, b, crunch_memo)
```