A Study of Binomial Numbers using Racket

Ted Szylowiec

Let's say we have a set S with n elements. We know from elementary combinatorics that S has 2^n subsets. Now, if we count all the elements in all these subsets, what is the total?

For example: $S = \{a, b, c\}$. We have six subsets:

$$\begin{gathered} \{\} \quad \{a\} \quad \{b\} \quad \{c\} \\ \{a,b\} \quad \{a,c\} \quad \{b,c\} \\ \{a,b,c\}. \end{gathered}$$

The total number of elements in these subsets is 12. What if we wanted to compute this total for any set S? We can write functions in Racket to study this problem. We will end up discovering something interesting about binomial numbers.

Subsets

We will represent sets by lists. We should keep in mind that lists and sets are not quite the same idea. Order does not matter in a set. The sets $\{a, b, c\}$ and $\{c, b, a\}$ are the same. This is not the case for lists. Order does matter: `'(a b c)` and `'(c b a)` are not the same. This detail is important if we ever want to check the equality of sets.

Begin the project by telling the Racket system to use the Racket language and to load the `rackunit` testing library. Testing is something you should always do when writing programs. Racket makes testing very easy, as you will see.

```
#lang racket
(require rackunit)
```

We need a function that computes all the subsets of a set. Racket has all kinds of functions to do algebra with sets in the `racket/set` library. But we don't need all of that so I'll give you a little function that computes all subsets using recursion and some list kung-fu:

```
(define (subsets set)
  (if (empty? set)
      (list empty)
      (let ((rst (subsets (cdr set))))
        (append rst
                (map (lambda (x)
                       (cons (car set) x))
                     rst)))))
```

This is a very useful function to have in your library. Let's try it.

```
> (subsets '(a b c))
'(() (c) (b) (b c) (a) (a c) (a b) (a b c))
```

```
> (subsets '(a b c d))
'(() (d) (c) (c d) (b) (b d) (b c) (b c d)
     (a) (a d) (a c) (a c d) (a b) (a b d)
     (a b c) (a b c d))
```

Let's add some basic tests for the `subsets` function. The empty set has exactly one subset. A set of one element has exactly two subsets. A set of 10 elements has 1024 subsets. Our `subsets` function needs to pass these tests. The `rackunit` library provides the handy all-purpose `check-equal?` function for writing such tests.

```
(check-equal? (length (subsets '()))  1)
(check-equal? (length (subsets '(a))) 2)
(check-equal? (length
                (subsets
                  '(a b c d e f h i j k)))
              1024)
```

If the tests pass, Racket won't say anything. But if they fail, Racket will complain and give you information as to which test failed and what went wrong.

Generating sets

In the last test above, we had to type in a set with 10 elements. It's tedious to have to specify sets by typing in each element. It's better to write a Racket function to do it. We don't care what these elements are exactly. They could be numbers, or symbols, or strings, or something else. So let's stick with numbers. Using recursion, we can easily write a function that generates sets of whatever length we like.

```
(define (make-set-helper n result)
  (if (< n 1)
      result
      (make-set-helper (- n 1)
                       (cons n result))))
(define (make-set n)
  (make-set-helper n empty))

(check-equal? (make-set 0) '())
(check-equal? (make-set 1) '(1))
```

I included two tests that `make-set` has to pass. You can play with `make-set` to see how it works:

```
> (make-set 5)
'(1 2 3 4 5)

> (make-set 10)
'(1 2 3 4 5 6 7 8 9 10)
```

Lengths of subsets

By mapping `length` onto the list of subsets, we get a list of the sizes of each subset of S. For example, let's work with a set S that has 4 elements.

```
> (define S (make-set 4))
> S
'(1 2 3 4)

> (map length (subsets S))
'(0 1 1 2 1 2 2 3 1 2 2 3 2 3 3 4)
```

Those are the sizes of all the subsets. To get the total number of elements in all subsets, we add these numbers up. That's the total we talked about at the beginning of this paper.

Of course we are not going to add all these numbers by hand. The point is to get the computer to do it. This is a typical task for recursion.

```
(define (sum-list-helper mylist result)
  (if (empty? mylist)
      result
      (sum-list-helper (cdr mylist)
                       (+ (car mylist)
                          result))))
(define (sum-list mylist)
  (sum-list-helper mylist 0))

(check-equal? (sum-list empty) 0)
(check-equal? (sum-list '(1)) 1)
(check-equal? (sum-list '(1 2 3)) 6)
```

Using `sum-list` we easily determine the total for all the subsets of a set of 4 elements:

```
> (sum-list (map length (subsets S)))
32
```

We now have the components we need to write a program that computes the total number of elements in all subsets of any set S of size n. Call this function T. At the beginning of this article we determined that $T(3)$ is 12, so this will be one of the tests `T` must pass.

```
(define (T n)
  (sum-list
    (map length (subsets (make-set n)))))

(check-equal? (T 0) 0)
(check-equal? (T 1) 1)
(check-equal? (T 3) 12)
```

Results

With the help of the T-function Racket program, we get these results:

n	$T(n)$
$2 = 2^1$	$4 = 2^2$
3	12
$4 = 2^2$	$32 = 2^5$
5	80
6	192
7	448
$8 = 2^3$	$1024 = 2^{10}$

What is the pattern here? Notice that when n is a power of two, $T(n)$ is also a power of two. Inspecting these values, we guess that when n is a power of two,

$$T(n) = 2^{n+\log_2 n-1} = n2^{n-1}.$$

We can verify that this formula works for the other values of n in the table.

From basic combinatorics, the number of subsets with k elements is $\binom{n}{k}$. Since each of these subsets has k elements, the number of elements in all the subsets of size k is

$$k\binom{n}{k}.$$

The total number of elements in all subsets of all sizes is therefore

$$0\binom{n}{0} + 1\binom{n}{1} + 2\binom{n}{2} + \cdots + n\binom{n}{n}.$$

Because of our Racket calculations, we believe that this expression is equal to $n2^{n-1}$. With the help of computer programming, we have discovered a beautiful property of binomial numbers:

$$0\binom{n}{0} + 1\binom{n}{1} + 2\binom{n}{2} + \cdots + n\binom{n}{n} = n2^{n-1}.$$

A Combinatorial proof

Computer programming led us to discover a remarkable relationship involving binomial numbers. However, we still have to give a mathematical proof of this relationship. Computer computation can only lead us in the right direction. It is not a proof.

Consider this set with n elements:

$$S = \{a, b, c, d, \ldots\}.$$

How many times does a appear in the subsets of S? First generate all subsets that *do not* have a in them. Those are the subsets of $\{b, c, d, \ldots\}$. There are 2^{n-1} such subsets that *do not* contain a. Then we add a to each one. We have constructed all the subsets that contain a. There are 2^{n-1} of them. Therefore the symbol a occurs 2^{n-1} times in all the subsets of S.

We can do the same for b and c and so on. Each one appears 2^{n-1} times in all the subsets. There are n symbols in all, therefore the total number of symbols in all the subsets of S is $n2^{n-1}$. □