I once crashed Pilosa by sending the query `Intersect()`, because this query expects two or more bitmap set arguments. What should it do in this case? What does an an intersection of zero sets (an "empty intersection") even mean?

1 Preliminaries

Let's compare with other operations for some insight. Addition is written as $a + b$, but there is a more general notation:

$$\sum_{i=1}^{n} a_i = a_1 + a_2 + ... + a_n. \tag{1}$$

Even more generally, instead of specifying n indices for a_i, we can use set notation:

$$\sum_{a \in A} a \tag{2}$$

which means the sum of all elements of the set A, and is equivalent to the previous notation, as long as $A = \{a_i, i \in \{1, 2, ..., n\}\}$. Other operations (multiplication, set union, and set intersection) have similar notation:

$$\prod_{a \in A} a = \prod_{i=1}^{n} a_i \qquad = a_1 \times a_2 \times ... \times a_n \tag{3}$$

$$\bigcup_{a \in A} a = \bigcup_{i=1}^{n} a_i \qquad = a_1 \cup a_2 \cup ... \cup a_n \tag{4}$$

$$\bigcap_{a \in A} a = \bigcap_{i=1}^{n} a_i \qquad = a_1 \cap a_2 \cap ... \cap a_n \tag{5}$$

It should be clear what all of these mean when $n \geq 2$. When $n = 1$, or $A = \{v\}$, it's sort of a degenerate case, but it still makes sense:

- $\left(\sum_{a \in \{v\}} a \right) = v$ - a sum of one number is that number.
- $\left(\prod_{a \in \{v\}} a \right) = v$ - a product of one number is that number.
- $\left(\bigcup_{a \in \{v\}} a \right) = \{v\}$ - a union of one set is that set.
- $\left(\bigcap_{a \in \{v\}} a \right) = \{v\}$ - an intersection of one set is that set.

What about when $n = 0$, or $A = \{\}$? This is where things get contentious - not everyone can agree on the meaning, even for summation. I will concede that these operations can be defined in whatever way you like. But there is only one definition that is sensible and consistent:

- $\left(\sum_{a \in \{\}} a \right) = 0$ - a sum of zero numbers, or empty sum, is 0.
- $\left(\prod_{a \in \{\}} a \right) = 1$ - an empty product is 1.
- $\left(\bigcup_{a \in \{\}} a \right) = \{\}$ - an empty union is the empty set.
- $\left(\bigcap_{a \in \{\}} a \right) = \phi$ - an empty intersection is the *universal set*.

Questions. Why do any of these mean anything? How can multiplying zero things result in a non-zero answer? The empty sum being zero kinda makes sense, why wouldn't the empty product be zero as well? Similarly why wouldn't the empty intersection be the same as the empty union? What the crap is a universal set?

2 Empty arithmetic

Let's start with the empty sum. The operation works conceptually by starting from a base value, and adding each element of the set to that, in turn. The base value is 0, since it is the additive identity element.

Another perspective is the consistency argument. Here is an example:

$$\left(\sum_{a \in \{3,5\}} a \right) = 3 + 5 = 8 \tag{6}$$

$$\left(\sum_{a \in \{3\}} a \right) = \left(\sum_{a \in \{3,5\}} a \right) - 5 = (3 + 5) - 5 = 3 \tag{7}$$

$$\left(\sum_{a \in \{\}} a \right) = \left(\sum_{a \in \{3\}} a \right) - 3 = (3) - 3 = 0 \tag{8}$$

That is, start with the set $\{3, 5\}$, and sum all its elements to get 8. Then subtract 5 from the result to get 3, and notice that the same result can be found by removing 5 from the set, leaving you with the sum over $\{3\}$. Now, apply the same concept again, but starting with the set $\{3\}$. The empty sum can only have one value that makes things consistent: 0. The operation doesn't *need* to be defined

when the input is an empty set. But if there is exactly one consistent value for that case, why not use it?

If that's not convincing, think about this from a programming perpsective: If you had a function that accepted a list of integers, and returned the sum of all of them, what would you want it to do when the list is empty? Crash? Raise an exception? Return some ugly sentinel value, maybe NaN? No, those would be terrible design in almost any context. The special case would be handled externally, almost always by using zero instead. The function itself should just return zero, and this is how real languages work:

```
sum([])  # python
0

np.sum([])  # numpy
0.0

sum([])  # octave
0

val l = List()  // scala
l.fold(0)(_ + _)
0
```

Similar arguments hold for the product. The multiplicative identity is 1; using that as the base value is sensible and uniquely consistent. In addition to being self-consistent, the product and sum operations should be consistent *with each other*. The logarithm of a product is a sum, and $\log(1) = 0$, so the base value of 0 for sum implies that the base value for product should be 1.

The `prod` built-in function is less common, but when it's there, this is how it works:

```
np.prod([])  # numpy
1.0

prod([])  # octave
1
```

There is a common point of confusion here: "how can a product with zero be anything besides zero?" Zero *is not being multiplied*. An empty product is a multiplication of N numerical values, where $N = 0$. None of those values can be equal to zero, because there aren't any.

3 Empty set operations

With clear notation and motivation, we can move on to set operations. What should an empty union be? Think about the base value, and consistency, and it should be clear that the correct answer is the empty set. If the union operation proceeds

by unioning sets sequentially with a base value, then the base value must be $\{\}$, because $\{\} \cup A = A$. The empty set is the identity element of the union operation. Thus:

$$\left(\bigcup_{A \in \{\}} A \right) = \{\} \tag{9}$$

This is also consistent with the empty sum, when looking at the cardinality of intersections of mutually exclusive sets.

Now things get tricky. DeMorgan's Law is a classic rule of logic:

$$\neg A \cap \neg B = \neg (A \cup B) \tag{10}$$

We can invert it by replacing $A \to \neg A$, $B \to \neg B$:

$$A \cap B = \neg (\neg A \cup \neg B) \tag{11}$$

This gives a formula for computing an intersection, by using unions. This gives us a way to define an empty intersection in terms of an empty union, which will be consistent by definition. First, the relationship must be extended to work for a variable number of inputs (like Pilosa's `Intersect`). To do so, we use set notation in the limits of the operators. Let's say S is the set of sets that are provided as input to the intersect operation, then we can write:

$$\bigcap_{A \in S} A = \neg \bigcup_{A \in S} \neg A \tag{12}$$

As a sanity check, let's look at the one-input case:

$$\bigcap_{A \in \{V\}} A = \neg \bigcup_{A \in \{V\}} \neg A \tag{13}$$

$$= \neg \neg V \tag{14}$$

$$= V \tag{15}$$

That makes sense. Now, the empty intersection $S = \{\}$, or `Intersect()` in Pilosa:

$$\bigcap_{A \in \{\}} A = \neg \bigcup_{A \in \{\}} \neg A \tag{16}$$

The expression on the right can be confusing. It is a union of this thing, $\neg A$, what does that mean exactly? It doesn't matter, because the union is empty - it is a union of zero sets, no matter how those sets appear to be written. We have already seen that an empty union is defined as the empty set. So:

$$\bigcap_{A \in \{\}} A = \neg \{\} \tag{17}$$

$$= \xi \tag{18}$$

But what does it mean to invert the empty set? In general, this is a meaningless operation. But when the context is specifically a finite universe of possible sets, we can assign meaning to it. For example, in a context where the only values allowed are those in the universal set $\xi = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, we can talk about the negation of specific sets:

$$\neg\{1, 2, 3, 4\} = \{5, 6, 7, 8, 9, 10, 11, 12\} \tag{19}$$

$$\neg\{4, 6, 9, 11, 12\} = \{1, 2, 3, 5, 7, 8, 10\} \tag{20}$$

It should be obvious now how to negate the empty set:

$$\neg\{\} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} = \xi \tag{21}$$

$$\tag{22}$$

ξ is simply the set of all set elements that are allowed in the context. Pilosa works with sets of integers in the range $[0, 2^{64} - 1]$, so Pilosa's ξ is simply the set of all 64-bit integers, and the idea of negating the empty set is perfectly sound, in theory.

Practically, returning a bitmap with almost every bit set is another matter, which is why Pilosa currently responds to an empty `Intersect` with an error.