Information Algebra

by Sven Nilsen
version 000

1 - System

In this algebra, we use the word "system" as a word for anything that encodes information. A system is an abstract object that can be composed with other systems to create concrete objects. The system gives an interface which we can build further abstractions upon. We can chain together systems with composition that read from left to right as "top" in the data structure to "bottom".

Here is an example where a system X contains a system Y that contains a system Z:

$$X \circ Y \circ Z$$

Another way to read this is "Z is encoded into Y which is encoded into X".

There is a system that includes all other systems which we call "unity". It is noted as "1" because it follows the same rules under composition. It can be read as the part where we have not decided or decided not to how to interpret the data further:

$$X = 1 \circ X = X \circ 1 = 1 \circ X \circ 1$$

Example	Description	Comments
1	Unity system	Contains all systems.
B	Binary system	Neighbors are bit.
$\mathbb{N}$	Natural number system	Neighbors differ by 1.
P	Prime number system	
$[\,]$	List system	
H	Lattice in hyperspace	
f, g	Functions	

Notice that functions are written with small capitals because they do not store data directly in memory.

2 - Data Structures

A composition of systems gives a data structure. For example, the natural numbers is a system which we can encode into a binary system and get a data structure:

$$B \circ \mathbb{N}$$

Another one is a list of natural numbers:

$$B \circ [\,] \circ \mathbb{N}$$

We use a fundamental theorem to distinguish data structures. This is used to create definitions of systems without worrying about the specific implementation. The theorem is central to the algebra.

Fundamental Theorem of Data Structures

When we have two sub systems, the semantic information to distinguish between the two systems is stored in one of the upper systems:

$$B \circ (X + Y)$$

Here the semantic information to distinguish between X and Y can be stored in B, but in the following data structure we have no place to put this semantic information:

$$B \circ X + B \circ Y$$

Therefore, we have the fundamental theorem of data structures:

$$B = B \circ 1 = B \circ (X + \neg X) \neq B \circ X + B \circ \neg X$$

There are several ways of implementing the same data structure, but we consider them as equal as long they do not conflict with the fundamental theorem. The fundamental theorem is what makes the algebra useful, because we can transform one solution into another non-equal solution with different properties:

$$B \circ (X + Y) \rightarrow B \circ X + B \circ Y$$
$$B \circ (X + Y) \neq B \circ X + B \circ Y$$

They are not equal in other ways too:

1. The left side stores X and Y together, the right side isolates them.
2. The left side expands differently from the right side in capacity when expanding B.
3. The left side needs to be accessed differently than the right side.

All these differences are equivalent to the fundamental theorem. It is sufficient to say the data structures are not equal to use the algebra to transform solutions.

3 - Capacity

For capacity we use the following notation:

$$|X|$$

The capacity of a binary system is the number of bits in memory, but the capacity of natural numbers stored in binary grows exponentially with the number of bits:

$$|B \circ \mathbb{N}| = 2^{|B|}$$

We are just storing one number that fills the entire memory, but we interpret the capacity of the data differently. Each data structure has its own interpreting of capacity. In most cases it is too complex to determine the exact capacity, but it helps us to make study data structures that are otherwise equal.

Example	**Description**	**Comments**
$\|B \circ \mathbb{N}\| = 2^{\|B\|}$	Natural number capacity	The capacity of storing a natural number grows exponentially with the number of bits.
$\|B \circ [\,] \circ X\| = k\|B\|\|B \circ X\|$	List capacity	The capacity of a list grows linearly with the number of bits.
$\|X\| = \|1 - \neg X\|$		The capacity of X is equal to the capacity of storing anything not X.
$\|X + Y\| = \|X\|\|Y\|$	Adressing capacity	The union of two systems can address the equal amount of their product.
$\|B^x\| = x$	Binary capacity	The only information we get from a binary system without peeking inside is the number of bits.

4 - Equality

The rules of equality is more complex in this algebra compared to numbers. The complexity is due to the fundamental theorem. For example, a linked list and an array both can do the same job and their capacity both grow linearly with the number of bits in computer memory.

$$|B \circ [\,] \circ X| = k|B||B \circ X|$$

When the constant is close to each other we can consider the implementations equal, even though they are not strictly equal. For this reason when we use a list operator we lax the condition but assume that all lists are implemented equally.

One data structure can be transformed to another data structure that is not equal:

$$B \circ [\,] \circ (X + Y) \neq B \circ [\,] \circ X + B \circ [\,] \circ Y$$

We can read the first case as storing objects with different types together, versus the second case where we store the types in different lists. The two cases are not equal, because when we extend memory we can choose which type we want to give the extra memory. With smarter distribution we can describe some data with one type of data structure that would not be possible with another data structure.

Example	**Description**	**Comments**
$X \neq Y \Leftrightarrow X \in \neg Y\,, Y \in \neg X$	Different systems are exclusive	When we say "not X" this includes all systems that are not equal to X.
$X + \neg X = 1$	Law of unity	
$X \circ Y \neq Y \circ X$	Non-commutativity	A data structure where Y is encoded in X is not equal to the data structure where X is encoded in Y.

5 - Binary Lattice in Hyperspace

Assume we have a graph where each node is represented with a sequence of bits. The neighbors of the node differ in bits by one, such that if we have two nodes, we can take an XOR operation between their value and get the bit that is changing:

$$\begin{aligned} a &= 10101101101 \\ b &= 10101001101 \\ xor(a,b) &= 00000100000 \end{aligned}$$

We can divide all edges that this lattice form in hyperspace into exclusive groups, based on their XOR operation. We can count the number of members per group by taking 2 powered to the number of zeroes, which equals the number of groups minus 1.

$$\begin{aligned} |H_g| &= |B_{graph}| \\ |H_m| &= 2^{|H_g|-1} \end{aligned}$$

Now we can use the rule for adressing capacity:

$$\begin{aligned} |B_{graph}| &= n \\ |H_g + H_m| &= |H_g||H_m| \\ |H_g + H_m| &= |H_g| \cdot 2^{|H_g|-1} \\ |H_g + H_m| &= n \cdot 2^{n-1} \end{aligned}$$

Since this includes all groups and their members, we can define a system H for "hypercube" that can address the same amount but does not address group and member separately:

$$H_g + H_m = H$$

For example, H can be a function that maps natural numbers to an edge. Now we have two different data structures:

$$\begin{aligned} & B \circ H \\ B \circ H_g + & B \circ H_m \end{aligned}$$

These two solutions are not equivalent, because in the first case we extend the capacity of H as a whole but in the second case we can choose to extend either for groups or for members.

6 - Prime Number System

Prime numbers can be used to store lists in a single natural number. Every natural number, except 0, can be written as a product of prime numbers, where the exponent allows encoding of information that is not mixed with the other exponents:

$$\mathbb{N} \circ P \circ [\,] \circ X = 2^{x_0} \cdot 3^{x_1} \cdot 5^{x_2} \ldots$$

To read back this information, it requires factorization of the number. It is not efficient neither in computation time or memory usage, but it is useful to prove something amazing about infinite memory. If we had infinite binary memory, we could construct this data structure:

$$B \circ (\mathbb{N} \circ P \circ [\,])^n \circ B$$

For each power of 'n', we could replace the current data structure with a new one that gives us infinitely more memory, but since all natural numbers can be written as a product of primes, we must give up the data structure we have to replace it with a new one. This proves that the choice of a data structure excludes the choice of other data structures stored in same memory. If we wanted to distinguish between the data structures, we would need to store some piece of information in memory telling us how to interpret the data, but there is no memory left!

$$B = B \circ (X + \neg X) \neq B \circ X + B \circ \neg X$$