Information Algebra

by Sven Nilsen
version 001

0 – Motivation

It is hard to reason mathematically about data structures and algorithms that ignores the details of how it is implemented. How do we find which approach is best without writing a single line of code? What can we prove about specific computer programs?

Algebra is a practical and natural notion. We can use algebra to encode some knowledge about the problem we have and transform it into alternative approaches. By comparing different approaches, we can select the one that most probably will give us the greatest benefit. To do this we need to formalize the notion and explain what goes in the thinking process when reading the symbols. The notion should …

- … encourage reusable design when possible.
- … provide an easy overview over the 'kind' of data structure.
- … be programming language agnostic.
- … readable in source comments.
- … tested enough to be proven useful before publishing.
- … be compatible with Category Theory.
- … use modern conventions in programming.

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 is composed with other systems. The sentence “X contains Y that contains Z” is written the following way, reading from left to right:

$$X \circ Y \circ Z$$

Example	**Description**	**Comments**
1	Unity system	Contains all systems.
B	Binary system	Neighbors are bit.
$\mathbb{N}$	Natural number system	Neighbors differ by 1.
$[]$	Linear system	Used for lists and sets such as hash tables or arrays.
f, g	Functions, values	Starts with lower case.
(X, Y)	Tuple	Stores X and Y together.
$(X + Y)$	Union	Stores either X or Y but not both at same time.

2 - Data Structures

A composition of systems gives a data structure. Here is a list of natural numbers, encoded into a binary system:

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

When we are dealing with computer memory, there is always a 'B' at the top. If we have a union of X and Y in a binary system we can transform it into a union of the binary representation of X and Y:

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

This process is called “flattening”.

Fundamental Theorem of Data Structures
$Z = Z \circ 1 = Z \circ (X + \neg X)$
$Z \neq (Z \circ X + Z \circ \neg X)$

The fundamental theorem is what makes the algebra useful, because we can transform one solution into another non-equal solution with different properties:

$$\mathbb{N} \circ (X + Y) \rightarrow \mathbb{N} \circ X + \mathbb{N} \circ Y$$
$$\mathbb{N} \circ (X + Y) \neq \mathbb{N} \circ X + \mathbb{N} \circ Y$$

3 - Addressing Capacity

For addressing capacity we use the following notation:

$|X|$

Example	**Description**	**Comments**
$\|B \circ \mathbb{N}\| = 2^{\|B\|}$	Natural number capacity	The addressing capacity of natural numbers grows exponentially with the number of bits.
$\|B \circ [\,] \circ X\| = k\|B\|\|B \circ X\|$	Linear capacity	The capacity of a linear data structure 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.
$\|(X, Y)\| = \|X\|\|Y\|$	Addressing capacity	The tuple of two systems can address the equal amount of their product.
$\|B^x\| = x$	Binary addressing capacity	The only adressing capacity we get from a binary system without peeking inside is the number of bits.

4 - Equality

A linked list and an array both can do the same job. The linked list needs a pointer to the next element, which require more memory than the array. The array needs a number telling how large the array is. Still, they both grow linearly when expanding the memory with new bits.

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

When the constant is close to each other we can consider the implementations equal, therefore we use just one symbol independently of the list implementation.

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:

$$(H_g, H_m) \rightarrow H$$

H is equal to any function that maps natural numbers to an edge. Now we have two different data structures:

$$\begin{aligned} \mathbb{N} \circ (H_g, H_m) \\ \mathbb{N} \circ H \end{aligned}$$

These two solutions are not equivalent, because in the second case we extend the capacity of H as a whole but in the first 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.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 we must give up the data structure we have to replace it with a new one.

7 - Factoring Lists

We have 3 different particles:

$$P=(e+\mu+\tau)$$

We want to store the experimental result of measuring the mass of the particles.
The first solution we can come up with is storing this as a tuple in a list:

$$[\,]\circ(m,P)$$

The addressing capacity of mass is infinite, but the addressing capacity of P is only 3:

$$|m|=\infty$$
$$|P|=3$$

To save space we decide to factor out the particle information from the list:

$$[\,]\circ(m,P)\rightarrow P\circ[\,]\circ m$$
$$(e+\mu+\tau)\circ[\,]\circ m$$

Why did we not factor out the information about the mass? Because the addressing capacity of the particle information is much lower. If we factored out the mass, it would require many more lists.

Factoring Lists
$[\,]\circ(X,Y)\rightarrow$ $X\circ[\,]\circ Y, if\,\lvert X\rvert<\lvert Y\rvert$ $Y\circ[\,]\circ X, if\,\lvert X\rvert>\lvert Y\rvert$