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".

Example	Description	Comments
B	Binary system	
$\mathbb{N}$	Natural number system	
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. We use a fundamental theorem to distinguish data structures. This is used to create definitions of systems without worrying about the specific implementation.

Fundamental Theorem of Data Structures
Two data structures are equal only when they expands equally in capacity when the top system increases in capacity.

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.

3 - 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.

Example	**Description**	**Comments**
$X \circ 1 = X$	Identity system	"1" does not mean the number 1, but the set of all possible systems.
$X \neq Y \Leftrightarrow X \in \neg Y\,,\, Y \in \neg X$	Different systems are exclusive	When we say "not X" this includes all systems that are not equal to X.
$X + Y$	Union	
$X + \neg X = 1$	Law of unity	
$X \circ X = X$		