1. Introduction

Boolean Algebras can be said to completely characterize any decision problem or question of provability that can be formulated. Trivially by nature of its algebraic foundation, given a basic propositional language such as a Boolean Ring B and a set of provable (or disprovable) statements M which operates as a filter (or ideal), the combination being referred to as a boolean logic, one can show that all interpretations derived from the relations generated by these statements reduce to a binary value; the Boolean Algebra B/M then provides us with the computational process of reducing any statement in B to one of these values and, maybe most importantly, such a reduction is the only algebra that is both consistent and complete [1]. It is therefore of interest for any system inherently based on proof theoretic guarentees i.e. Blockchains and other ledgers, to determine the feasibility of implementing Boolean Algebras in a tractable way. Doing so would allow for a much richer model of the provability to be integrated into the proof system in much the same way Boolean Algebras have provided a computational approach to circuit repair. In theory, a language with such a construct could interpret the most general model of provability and restructure its requirements for determining a valid boolean decision for a given problem.

The most natural example of such distinction is that of Monotonic and Non-Monotonic languages. In a Monotonic system once something is proved, no further application of axioms will ever render it false, so long as the language is consistent. In a Non-Monotonic system however, we have no such guarentee. A natural example of a Monotonic system would be most common modern mathematical constructs such as the algebra generated by the group axioms. In contrast a good example of a Non-Monotonic logic is a financial ledger of monetary transactions. An account A which has balance x at some point cannot prove it has x after it's been spent. In order to prove the validity of any trasaction $t_x : A \to B$ the funds of A must first be verified by ensuring that the funds being transferring have not been double spent. The only way to achieve this is to perform an audit of A's funds over ever recorded transaction within the financial system. A mechanism with access to such records would consistitute what is known as a "full node" in the terminology of blockchain engineers. Indeed, any non-monotonic problem has this inherrent logical requirement when it comes to proving its validity. A mathematical system like the algebra of a group in contrast needs no full nodes to verify it's validity; rather it is enough to provide proof that the correct axiomatic set was applied at every step. Note: An alternative example which may help an average programmer of visualizing the problem of Non-Monotonic logics is to consider the compilation of typesafe languages like Java. To prove the programs syntactic validity, for every object that's instantiated as a given type, all further references to that object must also assume it is of the same type (by type here we technically mean class). This example also more easily reveals the "almost" paradox of the fact that all Non-Monotonic problems can seemingly be "wrapped" into a Monotonic one by means of constructing a type that resembles $\forall p(x)$

For a completely distributed network like Ethereum, determining whether or not a given problem is or is not Monotonic may be worth the computational cost with regards to the resources it would save the network. But besides this, there is also the question of how to best integrate two logical systems (i.e. two blockchains), and when it can even be done while retaining consistency. An algorithmic approach to solving such problems really only reveals it's value when it's constructed in the most general of ways, that being an algebraic system that is either free, or an algebraic system that can have arbitrary algebraic structures embedded into it. It is only by considering these more general systems specifically that we can guarentee arbitrary integration and permanent semantic value in any mechanism.

The pending question is then, what characterizes a distributed system in terms of its algebra? And moreover, what tractable structures can be used to programatically determine consistency across nodes of a network, or even across more disparate constucts like full blockchains? In this paper we will recommend a combinatorial abstraction of token relationships based on algebraic topology (and by token we are now speaking very generally about any set of provable statements, inlcuding potentially an entirely blockchain network). In short, by the construction of Boolean Algebras on a network protocal, a network can be modeled as a simplicial complex, which can be reduced through a set of relations to produce our desired algebraic guarentees over network structure. By the same tool set we can even determine whether or not it's possible to integrate two networks over a given set of relations among its tokens. We will also discuss how the particular implementation of Boolean Algebras can be done in a multitude of ways depending on the circumstances, all yielding the same functional result. Once this level of abstraction has been achieved, all further computational efforts regarding provability or consistency are reduced to basic algebraic topology. Within this paradigm we will also touch upon what the field of algebraic topology has to say about the rich potential structures we can construct in this space using a basic understanding of Homology groups.

2. Constructing A Boolean Algebra Token

The most important data structure that we will make use of throughout this constuction will be that of a mappings between natural numbers. The indirect goal of this construction is to offer a roadmap for constructing this algebraic system in solidity, or other blockchain based languages that naturally make use of ownership of tokens as means of determining truth value. The system we are going to lay out only really relies on the fact that all tokens in question can be reduced to an address, as is the rule for Ethereum.

2.1. **Boolean Algebras.**

Let T be a token with a mapping $M : a \to o$ where a is an address on the network and $o = \{0, 1\}$. In order to call T a Boolean Algebra it must first have a unary operation $'$ and a binary operation $\cdot$. These operations must satisfy the following axioms:

- $(p \cdot q) \cdot r = p \cdot (q \cdot r)$
- $p \cdot q = q \cdot p$
- $p \cdot q = r \cdot r' \iff p \cdot q = p$

for all addresses p, q, r.

Note that as pointed out by [1] these almost exactly resemble a formulation of the axioms that can be used to define a group. It should also be noted that these axioms are based off of Edward Huntingtons axioms for Boolean algebras, with $\cdot$ being used to construct $+$ later on.

At first glance it may not be obvious at all what the definition of the operations $\cdot,'$ should be. Recalling however that the values they are operating on, are simply addresses. The first key observation is then, we should aim to construct our interpretation of addresses to naturally coincide with our usual interpretation of 'and' and 'negation'. The very first problem to tackle is that of avoiding collisions i.e. if $p \neq q$ we do not want $p \cdot r = q \cdot r$. This means we need $\cdot$ to be an injective mapping, but since our addresses can range over any positive integer value we are essentially limited to pairing functions of the form $N \times N \to N$. Its very important at this point going forward to decide whether or not we would like to represent our token as a finite algebra, or an infinite one. If we decide to limit the overall complexity of our token by making it a finite algebra then we can make use of Stone's theorem to reduce our algebra to the powerset of the Stone space of T. In fact even in the case of infinite T we can use Stone's theorem to reduce any element

$b \in T$ to the set of ultrafilters which contain b. However from the perspective of computability this is not at all obviously tractable in any convenient way. Additionally this does not guarentee us any other nice properties we would want in such general construct (completeness being the main concern). Additionally there is strong motivation for not assuming a finite size m for our algebra. As is remarked in [2] 'The only way to make one theory elastic enough to deal with all finite combinations is to provide it with an infinite supply of things that it may combine'. So for the purpose of making our algebra as general as possible in it's implementation we will only be considering algebras with infinite generating sets, and therefore must make use of infinite pairing functions. The Cantor pairing function as a concrete example provides us with a bijective pairing map, which gives us the additional benefit of immediately having a reversible operation as well. As we will see reversibility will also be a necessary component to the maintence of our algebraic structure.

2.2. **Finding the Right Mapping.** Let $C(a,b) = \frac{1}{2}(a+b)(a+b+1)+b$ be the Cantor pairing function. It is tempting at this point to assign $p \cdot q = C(p,q)$ as we know that this value will be unique to the pair p, q and would make it reasonable for us to interpret the address $C(a,b)$ as the inherrent address for $p \cdot q$. The problem with this naive mapping is that we have no way to define generators or indecomposable tokens as every natural number is reversible in this system. That is the address 4 for example is limited to being equivalent to $C(1,1) = 1 \cdot 1$, but $1 = C(1,0) = 1 \cdot 0$ which commen sense assures us is equal to zero. Hence we cannot make a valid interpretation rule out of this pairing function, and even if we could it's not at all obvious that we benefit in any way practically by forcing our address values to always have reversible interpretations.

The key to achieving irreducible address values is to partition the natural numbers into cosets that will provide us with a semantic guarentee in regards to the reversibility of any operation. For the construction of a boolean algebra we will need 3 such cosets, one for our generators/irreducibles, and one for each of our operations $\cdot, '$. To be more precise, on why these irreducible addresses are so important, we will later need them to define filters over the token (and eventually Boolean algebra) T which will correspond to our systems axioms and guarentee reducibility to a binary value for every proposition generated by them.

In order to construct these cosets we start by mapping all irreducibles to values in $3\mathbb{N}$. For all pairs a, b we map $a \cdot b$ to $3C(a,b)+1$, and for all a we define a' to be $max\{3a-1, 0\}$ if $a \mod 3 = -1$ and $(a-1)/3$ otherwise. This system is significantly more rich in its possibilities than the naive mapping. First of all, observe that all addresses now have a natural reduction rule: the base case being when $a \mod 3 = 0$ a is a generator, $a \mod 3 = 1$ reducing to a pair of addresses and $a \mod 3 = -1$ reducing to an inverse of another address. Furthermore because $\cdot$ is an injective, reversible mapping from all of $N \times N$ to $3N+1$, we have apriori definitions of arbitrary finite joins among our addresses by means of induction, that are also reversible. In short, this mapping scheme allows us to abstract any new tokens derived by our algebraic operations into an address. Hence without ever knowing of the explicit relation among any set of tokens on the network, anyone on the network can be certain that an address that decomposes is logically supposed to. Convsersely it's also possible to deduce the validity of the address $3C(a, b)+1$ from addresses a and b alone, with no explicilty given proof required.

While we now have reasonable mappings to assign semantic meaning to address values, we still need to do some work to make these mappings work as our Boolean operators. Specifically we still need to guarentee associativity, and commutativity and other equalities given by our axioms from before. However, the bulk of this can be handled by a straightforward relational structure between addresses, now that we can assign a singular address to any logical construct we want by way of our $\cdot,'$ mappings. Hence if we want to guarentee the equalities in our axioms, we are able to simply assert these relations and use them to find representatives of each coset upon interpretation of an address.

2.3. **Handling Relations Among Addresses.** Our relations will be handled by a separate mapping altogether from our mapping M that produces a boolean value. Instead we construct a mapping $R : a \to \bar{a}$ which maps an address to a representative of its coset. It should be noted that it may be theoretically neater, but not at all practical, to add a fourth coset in the construction of our addressess with an additional relational operator $\equiv$ as follows: $a \equiv b = 4C(a, b) + 2$ with the other mappings assigned accordingly to work in $\mathbb{N} \mod 4\mathbb{N}$. Under M we would have $M(4C(a, b)+2) = 1$ if and only if a and b are related. While this method might be satisfying in its theoretical neatness by encoding absolutely everything we need to complete our algebraic structure into the addresses themselves, this does not practically provide us with a natural way to find a good representative of each coset. It should be noted that we could do the same thing to also represent another binary operator $+$, but so far is not necessary by the fact that it's natural definition can be reduced to the two operators we have already constructed.

Constructing R is straightforward if done inductively. First, every address is related to itself upon instantiation. From there we are able to define the lowest order representative which we will generally use to find unique representatives of each coset through a function $Lor(a)$. We then will be able to say that $a \equiv b$ in T if $Lor(a) = Lor(b)$. $Lor(a)$ will be defined as the terminal relation to a in R i.e. the result of calling $a = R[R[a]]$ until we reach an address that is related to itself, $R[a] = a$. The obvious issue with the recursive definition of Lor is the presence of loops inside of R which will never allow this process to terminate. We must therefore construct R in a way that never produces loops. Again this can be done inductively. For R which is assumed to already be loop free, we can add any relation $a \equiv b$ we want by the following logic: $R[a] = b$ if $Lor(a) \neq Lor(b)$. This guarentees that no loop is formed by our new relation hence all values of Lor are still computable. It would also work just as well to make the assignment $R[a] = Lor(b)$ if we wanted to optimize lookup time for values of Lor, whereas when $R[a] = b$ we are instead optimizing for the time it takes to produce the relation in the first place. A combination strategy may also be used, as without the presence of loops Lor will always remain well defined as R has been constructed to be a set of directed tree structures which will always produce terminal values. This method of representing relations is in fact equivalent to [6] with the focus on this paper being the combinatorial representation of simplicial complexes as a delta complex, a problem that coincides exactly with our use case.

As we remarked before, this relational definition can be used to do most of the work in maintaining our axioms. Commutativity is the most straightforward case: upon derivation of an address $p \cdot q$ we can simply force the addition of the relation $p \cdot q \equiv q \cdot p$. Upon interpretation we would then have that $Lor(p \cdot q) = Lor(q \cdot p)$. As for associativity, we can do a similar thing with more effort. For any three addresses p, q, r we assert that $(p \cdot q) \cdot r \equiv p \cdot (q \cdot r)$. This assertion is also done similarly every time $\cdot$ is called but only on addresses a, b such that $a \mod 3 = 1$ i.e. when a is proven to be the product $x \cdot y$ for some addresses x, y. Inductively this then gives us the associativity we want all without every having to actively decompose the token a into its factors. As for the last axiom, we again assert the relation $p \cdot q = p$ upon derivation of p' noted by the fact that $p \mod 3 = -1$.

3. Properties of Natural Algebras

The construction of T in section 2 led us to what one might call a practical implementation of an infinite boolean algebra. It is practical in the sense that all elements in the algebra generated by anything in $3\mathbb{N}$ are reduced to an element of O by a finite number of interpretations. This algebra also has much richer properties and exhibits behavior similar to what one might think of as a natural language rather than an inherently computational one. Completely coincidentally, these properties actually come about because the algebra itself is embedded in the naturals $\mathbb{N}$ while having at least $\mathbb{N}$ generators coming from the coset $3\mathbb{N}$. We will refer to algebras that can be embedded in $\mathbb{N}$ with at least $\mathbb{N}$ generators as Natural Algebras.

3.1. **Properties Guarenteed by Algebraic Theory.** There is an immense theoretical freedom in deducing properties of a Natural Algebra once we have at least one concrete representation available to us. In fact in many cases proving that a Natural Algebra has a certain property can seem much easier than finding a robust way of implementing it. We will discuss later a general method for concretely representing any additional structure we desire, instead in this section focusing on what we know by virtue of the pure algebra.

Lemma: 3.1. *Halmos*[2] *Any Boolean Algebra that satisfies the countable chain condition is complete.*

Corrallary: 3.1. *Natural Algebras are complete.*

Proof. There is a countable number of elements, hence our algebra satisfies the countable chain condition, and is therefore complete. □

Theorem: 3.1. *For any Natural Algebra T not every token in T dominates atoms in T.*

Proof. A Boolean Algebra T is isomorphic to a field of all subsets of some set X if and only if T is complete and atomic [2]. Suppose that T is natural and therefore has the cardinality of $\mathbb{N}$, but also $\mathbb{N}$ generators. If T is infinite then the minimum cardinality of X would be $\mathbb{N}$ as all cardinalities less than $\mathbb{N}$ will have finite powersets, hence X has a powerset at least of size $2^{\mathbb{N}}$ and therefore cannot be isomorphic to T. But we know T is complete by our corrallary. Therefore it must be that T is not atomic, and therefore has elements which do not dominate any atoms. □

This can be recast to our construction from section 2 as follows. Essentially T has at least some tokens that are inherrently not parseable, at least to the degree of not being interpretable to O by a finite number of steps (we would need atoms to terminate any such computation of value). This is the first of the 'natural language' like properties, where one is always able to freely use terms/symbols/words without always being able to reduce them to a fixed meaning. Anyone that is skeptical of this fact about natural language can start with [3] as a good resource for how natural language operates from a psychoanalytic perspective. Psychoanalysis may at first seem like an absolutely bizarre domain to derive mathematical intuition from with any rigour. However thus far there really is no agreed upon structural characteristics of natural language in any domain at all known to academia. It is in this authors opinion that the work of stripping natural language down to its fundamental mechanisms has been thus far most generally described by the psychoanalytic discipline, specifically in the Lacanian school of thought. While this school of thought cannot be assumed to definitively prove anything about the structure of natural language, it is a surprisingly consistent and general description for not being the work of a mathematician. It should at the very least give the strong impression that meaning derived out of an everyday sentence is not in every case reducible to some combination of simpler components and instead that the very basis of natural language is to arbitrarily construct symbols to represent that which has no fundamental meaning. A curious reader interested in exploring this alternate perspective on the topic of language is encouraged to do so as [3] especially dives into many of the most pecurliar ways in which we can interpret meaning in our symbols for reality. This is however not at all necessary for carrying forward in this paper but those that takes the time to understand the logic at work in [3] may find a very satisfying analogue of what is described as 'the big Other' in the next theorem.

Lemma: 3.2. *Halmos*[2] *if E_1 and E_2 are free subsets of B_1 and B_2 respectively and are of the same cardinality and there are homomorphisms $h_1 : E_1 \to B_1$ $h_2 : E_2 \to B_2$ then B_1 and B_2 are isomorphic.*

Note that thus far it has only been assumed that the generating set of Natural Algebra must be at least the size of $\mathbb{N}$, not that they are equal. We will now show why they are not.

Theorem: 3.2. *The generating set of a Natural Algebra T has cardinality strictly greater than T.*

Proof. Recall by the countable chain condition being satisfied T must be complete, hence we know it cannot be isomorphic to the finite-cofinite algebra freely generated over $\mathbb{Z}$ [2]. But then by the above lemma, the generating set for our algebra T cannot be of cardinality $\mathbb{N}$. Clearly no cardinality less than $\mathbb{N}$ could generate T so in fact the genrating set of T must be of strictly greater cardinality than T itself. □

The intuition behind this result can understandably be hard to grasp at first. After all in every usual case of a direct construction of an algebra, or any consistent language really, the meaning we derive out of any constructed formulae should always reduce to formulae using only our generating elements. It is not that this fundamental rule changes for natural algebras, but by their very nature no matter what semantic structure a natural algebra is equipped with, there is always additional structure outside of its concrete representation will further describe it's algebra. This may make natural algebras sound limited in their ability to represent structure. But in fact it is just the opposite, and in the next section we will show in spite of this limitation, any new semantic structure we desire can be incoporated into our natural algebra. Moreover the next two results suggest we cannot have it any other way, i.e. any algebra that is general enough to include all semantics must reduce to a natural algebra.

Theorem: 3.3. *All Boolean Algebras can be completely embedded in a Natural Algebra.*

Proof. This is simply a consequence of the fact that any Boolean Algebra B can be completely embedded into one B' generated by a set of cardinality $\mathbb{N}$ [5]. By the definition of a Natural Algebra, we know that each of the generators of B' can be mapped into it's generating set by a monomorphism. Natural Algebras are themselves complete hence we have natural definitions for all suprema formed by any subset of B'. □

This result presents the following obvious correlarry.

Corrallary: 3.2. *There is a unique natural algebra, up to isomorphism.*

Previously we connected our results with some not so mainstream resources for understanding natural language structure. However it is at this point we can definitively review this new knowledge in relation to the more mainstream understanding of language structure first characterized by Noam Chomsky. Our three theorems agree with Chomsky's early work, but more so his difficulties, on the structure of grammars for natural languages. In particular in [4] he discusses the difficultly of modeling the full behavior of natural languages while retaining the consistency of their interpretability. He admits that even the very general linguistic rules he proposed to characterize language, which do render a seemingly large number of sentences in the language reducible to a fixed meaning, is still only assumed, and not proven, to completely describe the behavior of natural language. From our results we can see Chomskys only error was choosing to assume that natural language can be described by a finite set of base symbols with semantic value being defined by the parsing of strings via transformation rules and eventually reducing to this base. However, without even considering Natural Algebras we can deduce the following:

- A boolean algebra defined by a maximal filter is the only decision logic that is consistent and complete [1] (both of which seem self evidently to be present in the interpretation of a natural language).
- Any interpretation of a natural language sentence, even by means of Chomskys proposed grammars, must reduce to a series of consistent binary decisions (for Chomsky and most other language theorists this is modeled by a parse tree over the sentence in question).
- Any boolean algebra generated freely by a finite set will be finite by Stones representation theorem.

Chomsky himself shows that english is not reducible to a finite-state grammar [4], hence the algebra used to interpret natural sentences is not finite; if it was algebraically finite it's grammar could be characterized by a state machine where the elements of the algebra are the machine states. It's generating set must therefore also be infinite. But whatever the size is of the algebra itself it will, by our corrallary, reduce to a natural algebra which must have not just infinite generators, but a generating set of cardinality strictly greater than $\mathbb{N}$. In his introductory paragraphs Chomsky even remarks that it is unclear whether anything resembling natural language can be effectively described with the assumption of a finite generating set (in his case a finite alphabet with finite grammar). Impressively in spite of making the assumptions that he did, Chomsky was still able to deduce something similar to this impossibility. He summarizes in his concluding paragraph 'we picture a (sic. Natural) language as having a small, possibly finite kernal of basic sentences with phrase structure' where in his characterization, the base sentences are the only elements guarenteed to reduce to a consistent meaning. And in a sense he was spot on as we know now that in order to allow for the consistent interpretations of all decision logics, the only such languages that do so must be constructed from a set larger than itself.

3.2. **Concrete Representations of Suprema and Other Objects.** The fact that Natural Algebras are complete at first glance seems a bit counterintuitive when considering the concrete case of our construction from section 2. After all, while suprema generated by a finite amount of elements is easily constructible by applications of the operator $\cdot$, it is not at all obvious what a consistent representation of any countable suprema would entail. However, the main difficulty here is not actually in the representation of of any infinite suprema, but instead in it's direct construction by applications of $\cdot$. If for the moment we choose to forgoe a constructible definition of infinite suprema, we can indeed define a consistent way to represent these objects.

Lemma: 3.3. *Natural Algebras can have at most a countable number of suprema*

Proof. The amount of suprema in a natural algebra T must correspond to the number of countable subsets of T (technically to the number of infinite subsets of T but these subsets are themselves must be countable by our restrictions on T). There must then be at most countably many subsets for T itself to be of cardinality $\mathbb{N}$. $\square$

This lemma guarentees for us that we always have the 'space' in our Natural Algebras to provide all suprema with representations. This fact at the very least reveals the most naive way of concretely representing infinite suprema; that is to just axiomatically choose to represent them within our generating coset and assume the initial relations necessary to form their algebra. In fact we can even show this method produces a much more general result about the representation of other algebraic objects. Recall our theorem from the previous section which tells us that any booelan algebra can be completely embedded into a natural algebra. Ignoring completeness over infinite elements, we still at the very least have a computational method of handling any boolean algebras with finite interpretation and finite construction rules, and at most countable elements. As we would with anything in our generating set, we implement these rules by way of pairing functions being used to define operations, and relations being used to satisfy equalities involved in the boolean axioms or any others for that matter. In the case of our suprema, we know this is possible by the fact that there must be countably many of them, and adding them to our original algebra would simply come down to implementing the subalgebra generated by them (or at least what has not already been specified). We can rephrase this in the following way so that we don't have to directly worry about describing how we partition our generating set to achieve this.

Theorem: 3.4. *given a Natural Algebra T with k cosets, we are always permitted to add any number m cosets so long as we can define their behavior through pairing functions and relational mappings and produce a consistent and finite construction rule.*

This shows us that just as was the case for our suprema, we always have the room in our algebra to accomadate new semantic rules. Additionally just like any of our semantic cosets prior, we can always assume these new cosets are closed under any new, or old, binary or unary operations where we need to, by consistently offsetting after performing the operation. And again in exactly the same way as we have done, if these new operations need to adopt new interpretation rules to remain consistent this can always be accomodated. Assuming this, the algebra including our suprema could then be built in the following manner. We first assume two new semantic cosets, X for our suprema, and Y for representing set inclusion via mappings (we will see why this becomes necessary later). The required mappings become evident with a simple example regarding suprema. Abstracting elements our natural algebra to $\mathbb{N}$, we can for example assume a value $x \in X$ that represents the value $\wedge 2\mathbb{N}$. While we could make use of our operations to compute values like $x \cdot a$ for any a in $\mathbb{N}$, we do not, by our current mappings, have any natural reduction of x into other elements such as $\wedge 4\mathbb{N} \cdot \wedge 4\mathbb{N} + 2$ where we know this should be equal to x. In order to obtain such a guarentee, we need a new binary operator $-$ which we can construct out of whatever pairing function has already been assumed, in our case C. We functionally desire $x - 4\mathbb{N}$ to be mapped to whatever address represents $4\mathbb{N} + 2$. The interpretation rule of $x - a$ represented by the address $C(x, a)$ in (offset to) coset X will then be $x_a = \wedge(\bar{x} \setminus \bar{a})$ where $\wedge\bar{x} = x$ and $\wedge\bar{a} = a$, with the relation $R[x_a \cdot a] = x$ always being enforced. In the case when $a = x - b$ for some finite element b (i.e. $b \notin X$,), which can always be determined by the reversibility of C, we then also enforce the relation $R[x_a] = b$. For example if $x = \mathbb{N}, b = \wedge\{1, 2, 3, 4\}$ and $a = \mathbb{N} + 4 = x - b$ then $x - a \equiv \wedge\{1, 2, 3, 4\}$ which is a finite element already well defined in our algebra. Our semantic coset Y represents the case where a is one of the elements in $\bar{x}$ where $C(x, a)$ in Y maps to 1 under our interpretation map M if this is the case. We then use this to determine whether or not other standard equalities associated with suprema need to be enforced as relations such as (i) $x \cdot a = x$ if $a \in \bar{x}$ and (ii) $x - a = x$ if

$a \notin \bar{x}$. Basically, the elements of the coset Y can be said to represent the statement $a \in \bar{x}$ as a token in our algebra. And this statement is precisely what we would need to ever concretely make determinations about how a and x relate to one another. However because these two additional rules require interpretation through our master mapping M, we cannot rely on enforcing them at the time of constructing $x - a$. Instead, we rely on the fact that upon interpretation of $x - a$ and $x \cdot a$, we can always reduce to x when our rules are satisfied in Y, or reduce by our other means when they are not.

References

[1] Paul Halmos, *"Algebraic Logic"*
[2] Paul Halmos, *"Lectures on Boolean Algebras"*
[3] Slavoj Zizek, *"The Sublime Object of Ideology"*
[4] Naom Chomsky,*"Three Models for Language"*
[5] Saul A. Kripke, *"An Extension of a Theorem of Gaifman-Hales-Solovay"*
[6] Anton, Renzuollo *"Generating Minimal Boundary Maps"*