1 Introduction

Kleene algebras and their extensions represent a powerful tool for analyzing the correctness and equivalence of programs. They are designed to generalize regular expressions, a programmatic tool that can recognize any regular input. In this paper, we will present an axiomatization of Kleene algebras and a proof that they do, in fact, describe the behavior of regular expressions. We will go on to showcase a number of properties of Kleene algebras. In particular, we will find that $n \times n$ matrices over a Kleene algebra are also a Kleene algebra. Finally, we will survey a number of related structures and their additional or missing properties.

3 Kleene Algebras

3.1 Definition of a Kleene Algebra

For the purposes of this paper, we will use the definition of a Kleene algebra given in [1], though others exist. A Kleene algebra consists of a set $K$ with 2 binary operations, $+$ and $\cdot$, a unary operation, $\ast$, and two elements, 0 and 1, that have special properties. Note that we often write $a \cdot b$ as $ab$ for convenience. We define a partial order $\leq$ on $K$ as $a \leq b \iff a + b = b$. For $K$ to be a Kleene algebra, it must satisfy the following axioms:

1. $a + (b + c) = (a + b) + c$
2. $a + b = b + a$
3. $a + a = a$
4. $a + 0 = a$
5. $a(bc) = (ab)c$
6. $1a = a1 = a$
7. $0a = a0 = 0$
8. $(a + b)c = ac + bc$
9. $a(b + c) = ab + ac$
Axioms 1-4 state that + is commutative, associative, idempotent, and has an identity 0. Axioms 5-6 state that · is associative and has an identity 1. Students familiar with ring theory will find axioms 7-9 to be very similar to those relating addition and multiplication in a ring. Axioms 10-13 deal with the unary operator *, and are best illuminated by Example 3.3. We define exponentiation within a Kleene algebra inductively as follows:

\[ a^0 = 1, \quad a^n = a^{n-1}a. \]

3.2 Proof: \( \leq \) is a partial order

For \( \leq \) to be a partial order, it must be reflexive, antisymmetric, and transitive. By axiom 3, \( \leq \) is reflexive. Suppose \( a \leq b \) and \( b \leq a \). Then \( a + b = b \) and \( a + b = a \), so \( a = b \) and \( \leq \) is antisymmetric. Suppose \( a \leq b \) and \( b \leq c \). Then \( a + b = b \) and \( b + c = c \). Then \( b + c = a + b + c = a + c \). Since \( b + c = c \), this implies \( a + c = c \), so \( \leq \) is transitive. This proves that \( \leq \) is a partial order.

3.3 Example: Regular Expressions

Suppose you are writing a computer program that parses text documents, and you want to be able to recognize valid integers. A string of characters representing an integer may begin with a - sign. Then, the first character will be a numeral in the range 1-9. This first numeral might be followed by any finite number of characters in the range 0-9. So how might you teach a computer how to recognize this pattern? One approach would be to write a finite state machine, which would look something like this:

1. If next character is a -, go to state 2. If next character is a 1-9, go to state 3.
2. If next character is a 1-9, go to state 3. Otherwise, go to state 1.
3. If next character is not a 0-9, go to state 1 and record the observed number.

However, this is a cumbersome approach. For proof of that, think about how many more states and cases would be needed to identify decimal numbers. As an alternative, we can adopt a library such as Python’s `re` and write a regular expression that matches against the strings that we are trying to identify. The first building block of a regular expression is literal characters. For example, “a” matches against the letter “a”, and “” matches against any single non-newline character. “ab” matches against an a immediately followed by a b. “a|b” matches against an a or a b. Generally, l is a binary operation that matches against either of its two sides. “a*” matches against 0 or more copies of a concatenated together. So “aaaa”, “a”, and “” would all match, but “aba” would not.

With these operations, we can rewrite our finite state machine above as “(l)(-|12|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*”. Writing down large chains of l operations is cumbersome, so practical systems such typically have shorthands for choosing a single character from a large set. For example, `re` has a special . character that matches against any single non-newline character. `re` also has a number of additional operations, such as +, which matches against 0 or 1 copies of the proceeding regular expression. All of these can be rewritten in terms of concatenation, l, and *. “a?” is a convenient way to write “(la)” - note the empty string before the l. It is a classical result that every regular expression corresponds to a finite state machine. A proof of this can be found in [1].

[2] provides a good formalism of regular expressions, which will be largely replicated here. Consider a word to be a possibly-empty sequence of inputs. Each input comes from a set A, referred to as the alphabet. An event is a set of words. The empty set is denoted 0, and the set containing only the empty word is denoted 1. The operation l is defined as \( A \cup B = A \cup B \), the set union operation. Concatenation is defined as \( AB = \{ ab | a \in A, b \in B \} \). When working with strings, the elementwise operation is string concatenation. The * operator is defined as \( A^0 \cup A^1 \cup A^2 \cup \cdots \) where exponentiation uses the same inductive definition as in section 3.1.
3.4 Proof: Regular Expressions are a Kleene Algebra

Using the formalism above, we can prove that regular expressions are a Kleene algebra. $K$ is the set of sets of events. The operation $\|$ is the Kleene $+$, concatenation is $\cdot$, and $*$ is *. Axioms 1-4 follow readily from the fact that $\|$ is set union:

\[
A + (B + C) = (A \cup B) \cup C = (A \cup B) + C \\
A + B = A \cup B = B \cup A = B + A \\
A + A = A \cup A = A \\
A + 0 = A \cup 0 = A.
\]

For axiom 5, we have:

\[
(AB)C = \{abc \mid a \in A, b \in B, c \in C\} = A\{bc \mid b \in B, c \in C\} = A(BC).
\]

Let $e$ be the empty word. For any word $a$, $ae = ea = a$. Then we can prove that axiom 6 holds:

\[
1A = \{ca \mid a \in A\} = \{a \mid a \in A\} = A \\
A1 = \{ae \mid a \in A\} = \{a \mid a \in A\} = A.
\]

For axiom 6, we note that $0A = \{za \mid a \in A, z \in \emptyset\}$, but $0 = \emptyset$ so $0A = \emptyset$. A symmetric argument holds for $A \cdot 0$, so axiom 7 is satisfied as well. The following chain of equations demonstrates that axiom 8 is satisfied

\[
AB + BC = \{xc \mid x \in A, c \in C\} \cup \{xc \mid x \in B, c \in C\} = \{xc \mid x \in A \cup B, c \in C\} = (A \cup B)C = (A + B)C.
\]

A symmetrical argument demonstrates that axiom 9 is satisfied. For axiom 10, we begin with

\[
1 + AA^* = (A^0) + (A(A^0 + A^1 + A^2 + \cdots)).
\]

Using axiom 9, we can distribute the $A$ over the $A^*$:

\[
(A^0) + (A(A^0 + A^1 + A^2 + \cdots)) = (A^0) \cup (A^1 + A^2 + A^3 + \cdots) = A^*
\]

Since $1 + AA^* = A^*$ and $\leq$ is reflexive, $1 + AA^* \leq A^*$. The demonstration that axiom 11 is satisfied is largely similar, but requires the use of the other distributivity axiom, axiom 8.

For axiom 12, suppose $AX \leq X$. By the definition of $\leq$, this implies that $AX + X = X$. This implies that $AX \cup X = X$, which is true exactly when $AX \subseteq X$. That is, $ax \in X$ for all $a \in A, x \in X$. Suppose $A^{n-1}X \subseteq X$. Let $z \in A^nX$. This implies that $z = a_n\cdot a_{n-1}a_{n-2}a_{n-3}\ldots a_1x$ for some $a_i \in A, x \in X$. Since $A^{n-1}X \subseteq X$, $a_n\cdot a_{n-1}a_{n-2}a_{n-3}\ldots a_1x \in X$. Since $ax \in X$ for all $a \in A, x \in X$ and $a_n \in A, z \in X$. So $A^nX \subseteq X$. By induction, this holds for all $A^n$. Then

\[
A^*X = (A^0 + A^1 + A^2 + A^3 + \cdots)X = (X + AX + A^2X + A^3X + \cdots) = (X + X + X \cdots + X) = X.
\]

The proof of axiom 13 is essentially symmetrical to the above. So $K$, with $\|$, $\cdot$, $\ast$, $0$, and $1$, satisfies all of the axioms of a Kleene algebra.

3.5 Example: Algebras of Binary Relations

Given a set $S$, any subset of $S \times S$ is a binary relation. We can build a Kleene algebra with a set of binary relations. The $\|$ operation is set union. $\cdot$ is relational composition, defined as

\[
A \circ B = \{(a, b) \mid \exists y(x, y) \in A, (y, z) \in B\}.
\]

The Kleene $\ast$ star operator is defined as the reflexive transitive closure. In other words, $A^\ast$ is the smallest relation containing $A$ with both the reflexive and transitive properties. As for the two special elements of a Kleene algebra, 0 is the empty relation and 1 is the identity relation.

4 Properties of Kleene Algebras

4.1 Elementary Properties

In any Kleene algebra:

1. $1 \leq a^\ast$  
2. $a \leq a^\ast$
(3): \( a \leq b \implies ac \leq bc \)
(4): \( a \leq b \implies ca \leq cb \)
(5): \( a \leq b \implies a + c \leq b + c \)
(6): \( a \leq b \implies a^* \leq b^* \)
(7): \( 1 + a + a^*a^* = a^* \)
(8): \( a^{**} = a^* \)
(9): \( 0^* = 1 \)
(10): \( 1 + aa^* = a^* \)
(11): \( 1 + a^*a = a^* \)
(12): \( b + ax \leq x \implies a^*b \leq x \)
(13): \( b + xa \leq x \implies ba^* \leq x \)
(14): \( ax = xb \implies a^*x = xb^* \)
(15): \( (cd)^*c = c(d(c))^* \)
(16): \( (a + b)^* = a^*(ba^*)^* \)

Notice that properties (10) and (11) are stronger versions of axioms (10) and (11).

4.2 Selected Proofs

(3): Assume \( a \leq b \). Then \( a + b = b \). By distributivity,
\[
ac + bc = (a + b)c.
\]
By our assumption,
\[
(a + b)c = bc.
\]
Since \( ac + bc = bc \), by the definition of \( \leq \), \( ac \leq bc \).

(9): Using axiom 10 with 0 for \( a \) and then applying the definition of \( \leq \), we get, \( 1 + 0 \cdot 0^* + 0 = 0^* \). By axiom 7 and axiom 4, this simplifies to \( 1 = 0^* \).

4.3 Matrices over Kleene Algebras

The family \( M(n, K) \) of \( n \times n \) matrices over a Kleene algebra is, with certain choices of operations, a Kleene algebra. The Kleene algebra + is defined as matrix addition, \( \cdot \) is defined as matrix multiplication. Defining \( * \), however, takes significantly more effort. First, we consider the \( n = 2 \) case. Let
\[
E = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.
\]
Then
\[
E^* = \begin{bmatrix} (a + bd^*c)^* & (a + bd^*c)^*bd^* \\ d^*c(a + bd^*c)^* & d^* + d^*c(a + bd^*c)^*bd^* \end{bmatrix}.
\]
To understand where this construction comes from, consider a finite state machine of the kind described in Example 3.3. It has two states, \( s \) and \( t \). While in \( s \), seeing a \( b \) will cause the machine to remain in \( s \) and seeing an \( a \) will cause the machine to transition to \( t \). In \( t \), seeing a \( c \) will cause a transition to \( s \), while a \( d \) will cause the machine to remain in \( t \). Assume that the machine will never see a \( c \) or \( d \) while in \( s \) and never see an \( a \) or \( b \) while in \( t \). The columns in the matrix correspond to the starting state, and the rows correspond to the ending state. For instance, entry \((1, 1)\) in the matrix above corresponds to all input strings that, assuming the machine starts in \( s \), will result in it ending in \( s \). We can generalize this construction to \( n > 2 \) by making \( a, b, c, d \) into submatrices where \( a \) and \( d \) are square. A proof that \( M(n, K) \) is a Kleene algebra can be found in [1].

5 Extensions and Related Structures

5.1 Definition: Right- and Left- handed Kleene algebras

A right-handed Kleene algebra is an algebraic structure for which all of the Kleene algebra axioms hold except (13). A left-handed Kleene algebra satisfies all the axioms except (12). An algebraic structure is a Kleene algebra if and only if it is both a left-handed and right-handed Kleene algebra.
5.2 Definition: *-continuous Kleene algebras
A Kleene algebra is *-continuous if it satisfies the *-continuity condition:
\[ ab^* c = \sum_{n} ab^n c. \]

5.3 Proposition
The *-continuity condition implies axioms 10-13.

5.4 Examples
All of the examples in section 3 are *-continuous.

5.5 Proposition
All finite Kleene algebras are *-continuous.

5.6 Proposition: There is a non *-continuous Kleene algebra
We use a construction given in [4]. Let \( K \) be a set containing every ordered pair of natural numbers and two additional elements, \( x \) and \( y \). Give \( K \) an ordering \( \leq \) such that \( x \) is the minimal element, \( y \) is the maximal element, and all the ordered pairs are sorted by lexicographic order. Define + such that \( a + b = a \) if \( a \geq b \) and \( b \) otherwise. Define \( \cdot \) as 
\[ xa = ax = x \]
\[ ya = ay = y \text{ for } a \neq x \]
\[ (a, b) \cdot (c, d) = (a + c, b + d). \]
Notice that \( xy = x \). Then, define \( * \) as 
\[ x^* = (0, 0)^* = (0, 0) \]
\[ a \neq x, a \neq (0, 0) \implies a^* = y. \]
We leave the verification that this structure is a Kleene algebra as an exercise for the reader. Consider the *-continuity condition. \( (0, 1)^* = y \), by definition. However,
\[ \sum_{n} (0, 1)^n = \sum_{n} (0, n) = (1, 0) \neq y. \]
So \( K \) is not *-continuous.

5.7 Definition: Kleene Algebra with Tests
A Kleene algebra with tests is a Kleene algebra \( K \) which has a subset \( B \) such that \( B \) is a Boolean algebra with + as the meet operation and \( \cdot \) as the join operation. In particular, this implies that there is a unary complement operator \( ' \) that can be applied to elements of \( B \). This allows for the construction of conditional statements. For example, the statement if \( a \) then \( b \) else \( c \) can be encoded as
\[ ab + a'c. \]

References