Some Applications of Kleene Algebra

M. Al-Mousa, F. Tchier and F. Toufic
Mathematics department,
King Saud University
malmousa@ksu.edu.sa, ftchier@ksu.edu.sa,
ftoufic@ksu.edu.sa

(Invited Paper)

***Abstract*—We will treat some application by using Kleene algebra and its properties. First, We will find the shortest path which links two given vertices in a directed (finite) graph. Also, we will give an algorithm to solve this problem by using the typed Kleene algebra. Finally, we use Kleene algebra to put any program in while free form.**

***Index Terms*—Kleene algebra, shortest path, graph, While program.**

I. Introduction

KLEENE algebra is an algebraic structure that captures axiomatically the properties of a natural class of structures arising in logic and computer science. It is named after Stephen Cole Kleene $(1909-1994)$, who among his many other achievements invented finite automate and regular expressions, structures of fundamental importance in computer science. Kleene algebra is the algebraic theory of these objects, although it has many other natural and useful interpretations.

Kleene algebras arise in various guises in many contexts; relational algebra [20], [10], [9], semantics, logic of programs [11], [21], automata, formal language theory [15], [14], the design and analysis of algorithms [1], [8], [12], [17]. In this paper, we will use Kleene algebra to solve some fields of applications.

Some Fields of Applications

- Within the field of efficient algorithms it has been applied to path problems on graphs. The shortest path problem consists of finding a path between two vertices such that the sum of the weights of its constituent edges is minimized. Many important algorithms have been used to solve this problem; Dijkstra's algorithm solves the single-pair, single-source, and single-destination shortest path problems [18]. Bellman-Ford algorithm solves the single source problem if edge weights may be negative [17]. Floyd-Warshall algorithm solves all pairs shortest paths [18], [17]. Johnson's algorithm solves all pairs shortest paths, and may be faster than Floyd-Warshall on sparse graphs [18], [17], [3]. Perturbation theory finds (at worst) the locally shortest path.
- More recently, Kleene algebra has been successfully applied to the semantic description of imperative programs. Kleene algebras with tests will give us a mathematical framework for studying and manipulating programs. It provides an effective procedure to decide wether two programs are equal or not and permits us to simulate any `while` program with a program which has at most one `while` loop.

In the following we will present some notions needed for the rest of the paper.

(1) **Definition.**

1. A partial order is a binary relation "$\leq$" over a set S which is reflexive, antisymmetric, and transitive, that is, for all a, b, and $c \in S$, we have
   - $a \leq a$ (reflexivity);
   - if $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetric);
   - if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity).
2. A set with a partial order is called a partially ordered set (also called a poset).
3. Let $(A, \leq)$ and $(B, \leq)$ be two partially ordered sets. A Galois connection between these posets consists of two functions: $F : A \to B$ and $G : B \to A$, such that for all a in A and b in B, we have $F(a) \leq b$ if and only if $a \leq G(b)$.

Now we will give the definition of directed graph [18].

(2) **Definition.** A directed graph $G = (V, B)$ consists of a set V of vertices and a set B of ordered pairs of vertices, called edges. And the adjacency matrix of G is a means of representing which vertices of a graph are adjacent to which other vertices.

II. Kleene algebra

We start by define Kleene algebra.

(3) **Definition.** A Kleene algebra is a structure $\mathcal{K} = (K, +, \cdot, *, 0, 1)$ where the following axioms satisfied

- $(a + b) + c = a + (b + c)$
- $a + a = a$
- $a + b = b + a$
- $a + 0 = a$
- $a.0 = 0.a = 0$
- $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- $a \cdot 1 = 1 \cdot a = a$
- $a \cdot (b + c) = a \cdot b + a \cdot c$
- $(a + b) \cdot c = a \cdot c + b \cdot c$.

$$(K1) \quad 1 + x \cdot x^* \leq x^*$$
$$(K2) \quad 1 + x^* \cdot x \leq x^*$$
$$(K3) \quad b + a \cdot x \leq x \implies a^* \cdot b \leq x$$
$$(K4) \quad b + x \cdot a \leq x \implies b \cdot a^* \leq x.$$

where $a \leq b$ iff $a + b = b$ is called the natural partial order of Kleene algebra. The product $a \cdot b$ will be written as ab.

(4) **Definition.** A Kleene algebra $\mathcal{K}$ is called star-continuous [16] if it satisfies the axiom

$xy^*z = \sup_{n \geq 0} xy^n z$,

for any $x, y, z \in \mathcal{K}$; where $y^0 = 1$, $y^{n+1} = yy^n$ and the supremum is with respect to the natural order.

(5) **Theorem.** Let $\mathcal{K}$ be a Kleene algebra and $c, d \in \mathcal{K}$, then

a) $(cd)^*c = c(dc)^*$,

b) If c is invertible, that is $cc^{-1} = 1$ for some $c^{-1} \in \mathcal{K}$, then $\left(cdc^{-1}\right)^* = cd^*c^{-1}$.

(6) **Example.** (Language-Theoretic Model). Let Σ^* denote the set of finite-length strings over a finite alphabet Σ, including the null string ε. The set Σ^* forms a Kleene algebra under the following constants and operations on subsets of Σ^*

1. $A + B = A \cup B$,
2. $A.B = \{x \bullet z \bullet y \mid x \bullet z \in A,\ z \bullet y \in B\}$,

(where $\bullet$ is concatenation)

3. $0 = \emptyset$,
4. $1 = \{\varepsilon\}$,
5. $A^* = \bigcup_{n \geq 0} A^n =$

$\{x_1 ... x_n \mid n \geq 0 \text{ and } x_i \in A, 1 \leq i \leq n\}$.

Thus the operation $(\cdot)$, applied to two sets of strings A and B, products the set of all strings obtained by concatenating a string from A with a string from B, in the order. Thus A^* is the union of all powers of A, equivalently, A^* consists of all strings obtained by concatenating together any finite collection of string from A in any order. Any subset of the full power set of Σ^* containing $\emptyset$ and $\{\varepsilon\}$ and closed under the operations of $(\cap), (\bullet)$, and $(^*)$ is a Kleene algebra.

A subidentity of a Kleene algebra $(S, \cap, \bullet, *, 0, 1)$ is an element x with $x \leq 1$. We call a Kleene algebra pre-typed if all its subidentities are idempotent, *i.e.*, $x \leq 1 \implies x \bullet x = x$. We call a pretyped Kleene algebra $(S, \cap, \bullet, *, 0, 1)$ typed if its a boolean algebra $(S, \cap, \bullet, 0, 1)$ and the restriction operations distribute through arbitrary meets of subtypes, *i.e.*, if we have for all families $(x_j)_{j \in J}$ of subidentites and all $a \in S$ that

$$(\bigcap_{j \in J} x_j) \bullet a = \bigcap_{j \in J} x_j \bullet a \text{ and } a \bullet (\bigcap_{j \in J} x_j) = \bigcap_{j \in J} a \bullet x_j.$$

In a typed Kleene algebra we can define, for $a \in S$, the domain $\langle a$ and co-domain $a\rangle$ via the Galois connections (y ranges over subidentities only)

$$\langle a \leq y \Longleftrightarrow a \leq y \cdot 1,$$
$$a\rangle \leq y \Longleftrightarrow a \leq 1 \cdot y.$$

Now, we take unusual model that turns out to be useful in the shortest path algorithms in graphs. This algebra is called the tropical algebra, also known as the "min, + algebra". For more details see [3].

(7) **Example.** (The tropical algebra). Let $\mathcal{R} = \mathbf{R}^+ \cup \{\infty\}$ is the set of nonnegative reals with an additional infinite element ∞. This model forms a Kleene algebra under the following constants and operations on subsets of $\mathbf{R}^+ \cup \{\infty\}$:

1. $a + b = \min\{a, b\}$,
2. $a \cdot b = a +_{\mathbf{R}} b$,

(where $+_{\mathbf{R}}$ means the addition in reals)

3. $a^* = 1 = 0_{\mathbf{R}}$,
4. $0 = \infty$,
5. $1 = 0_{\mathbf{R}}$.

The elements 1 and 0 of a Kleene algebra can play the roles of the truth values "true" and "false". Expressions that yield one of these values are therefore also called assertions. The assertion 0 means not only "false", but also "undefined". Negation is defined by $\sim 0 = 1$, and $\sim 1 = 0$. Then for an assertion b and an element c we have

$$b \cdot c = c \cdot b = \begin{cases} c & \text{if } b = 1, \\ 0 & \text{if } b = 0. \end{cases}$$

The conjunction of assertions a, b is their infimum $a \cap b$ or, equivalently, their product $a \cdot b$; their disjunction is their sum $a + b$. We write $a \wedge b$ for $a \cap b$ and $a \vee b$ for $a + b$.

Using this, we can construct a conditional:

$$\text{if } b \text{ then } c \text{ else } d \text{ fi} = b \cdot c \cup \ \sim b \cdot d$$

for assertions b and elements c, d. Note that the conditional is monotonic only in d and e. So, recursions over the condition b need not be well-defined. A property we are going to use in the sequel is

$$\text{if } b \text{ then } d \text{ else if } c \text{ then } d \text{ else } e \text{ fi fi} = \text{if } \ b \vee c \text{ then } d \text{ else } e \text{ fi}$$

For assertions b, c and elements d, e.

A. Matrices Over a Kleene Algebra

Under the natural definitions of the Kleene algebra operators $+, \cdot, ^*, 0$ and 1, the family $\mathcal{M}(n, \mathcal{K})$ of $n \times n$ matrices over a Kleene algebra $\mathcal{K}$ again forms a Kleene algebra. This is a standard result proved for various classes of algebras in [5], [2].

Define $(+)$ and $(\cdot)$ on $\mathcal{M}(n, \mathcal{K})$ to be the usual operations of matrix addition and multiplication, respectively, Z_n the $n \times n$ zero matrix, and I_n the $n \times n$ identity matrix. The partial order $(\leq)$ is defined on $\mathcal{M}(n, \mathcal{K})$ by $A \leq B \Longleftrightarrow A + B = B$.

(8) **Lemma.** The structure
$(\mathcal{M}(n, \mathcal{K}), +, \cdot, Z_n, I_n)$ is an idempotent semiring.

The definition of E^* for $E \in \mathcal{M}(n, \mathcal{K})$ comes from [14], [5], [7]. We first consider the case $n = 2$. This construction will later be applied inductively.

Let

$$E = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

let $f = a + bd^*c$ and define

$$E^* = \begin{pmatrix} f^* & f^*bd^* \\ d^*cf^* & d^* + d^*cf^*bd^* \end{pmatrix}$$

(9) **Lemma.** The matrix E^* defined above satisfies the Kleene algebra axioms (Def. 3).

(10) **Proposition.** If $\mathcal{K}$ is star-continuous (Def. 4), then so is $\mathcal{M}(n,\mathcal{K})$ for $n \geq 1$.

This Proposition states that the star-continuity passes from the Kleene algebra $\mathcal{K}$ to $\mathcal{M}(n,\mathcal{K})$ and this is exactly the key fact that we use in the following application.

B. All Pair-Shortest Paths Problem

[12]

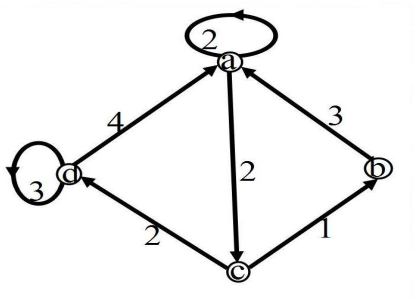
Given a directed graph G which has n vertices $1,...,n$ and each edge is labelled by a positive real number called the weight of the edge. From G, we create a complete graph [17] (still denoted by G) such that the vertex i is connected with j (in this order) by an edge labelled with the same number if i and j are adjacent in G (in this order again), and by an edge labelled by the symbol ∞ otherwise. So, the edges of the new graph G are labelled by an element of $\mathbf{R}^+ \cup \{\infty\}$. In the set $\mathcal{R} = \mathbf{R}^+ \cup \{\infty\}$, we define the operations as in (Ex.7).

The adjacency matrix B of the previous graph G is an element of the Kleene algebra $\mathcal{M}(n,\mathcal{K})$ and the ij^{th} entry of B^k is exactly the number of paths of length k which links i with j and composed by less than or equal to k edges.This result can be used to determine the length of the shortest path between i and j. By the (Prop.10),

$B^* = \sup_{\mathcal{R}} \{B^k \mid k \in N\} = \inf\{B^k \mid k \in N\}$.

So the ij^{th} entry of B^* is exactly the length of the shortest path which links i to j.

(11) **Example.** The adjacency matrix of the following graph is



$$B = \begin{pmatrix} 2 & \infty & 2 & \infty \\ 3 & \infty & \infty & \infty \\ \infty & 1 & \infty & 2 \\ 4 & \infty & \infty & 3 \end{pmatrix} = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}.$$

We have $B_4^* = \begin{pmatrix} 0 & 2 \\ \infty & 0 \end{pmatrix}$,

$F = B_1 + B_2B_4^*B_3 = \begin{pmatrix} 2 & 3 \\ 3 & \infty \end{pmatrix}$ and $F^* = \begin{pmatrix} 0 & 3 \\ 3 & 0 \end{pmatrix}$.

Therefore, with a bit of calculation we find

$$B^* = \begin{pmatrix} F^* & F^*B_2B_4^* \\ B_4^*B_3F^* & B_4^* + B_4^*B_3F^*B_2B_4^* \end{pmatrix} = \begin{pmatrix} 0 & 3 & 2 & 4 \\ 3 & 0 & 5 & 7 \\ 4 & 1 & 0 & 2 \\ 4 & 7 & 6 & 0 \end{pmatrix}.$$

So, we can read on this matrix that the shortest path from the vertex b to the vertex d is of length $(B^*)_{bd} = 7$, it passes through a and after c.

In the following, we give an algorithm to solve the shortest path problem by using the typed Kleene algebra.

C. Shortest Connecting Path

Assume that $(S,\Sigma,\cdot,0,1)$ is a typed Kleene algebra. We define the general operation E by $E(W)(f,g) = g(f(w))$ where

- $w \subseteq S$ is a fixed element of S.
- $f : S \longrightarrow \mathcal{P}(M)$ is a disjunctive abstraction function with some set M of "valuations", where a function f from a Kleene algebra into a lattice is disjunctive, if it distributes through $+$, *i.e.*, satisfies $f(x+y) = f(x) \cup f(y)$,
- $g : \mathcal{P}(M) \longrightarrow \mathcal{P}(M)$ is a selection satisfying the properties

$$(1) \qquad g(K) \subseteq K,$$
$$(2)\ g(K \cup L) = g(g(K) \cup g(L))$$

(weak distributivity), for $K, L \subseteq M$.

Motivated by the graph theoretical applications, we now postulate the following conditions about f and g:

1. $\langle c \leq a\rangle \Rightarrow$
   $g(f(a \cdot c)) = g(f(a \cdot c) + f(a \cdot u \cdot c))$,
2. $(a \cdot u)\rangle \leq a\rangle \Rightarrow$
   $g(f(a \cdot c)) = g(f(a \cdot c) + f(a \cdot u \cdot c))$,

with $a, c, u \in S$. These two conditions are used to obtain two termination cases of the algorithm. For more details see [3].

We have the following basic algorithm:

$$F(f,g)(a,b,c) = if \langle c \leq a\rangle \ \vee\ (a \cdot b)\rangle \leq a\rangle$$
$$then\ g(f(a \cdot c))$$
$$else\ g(f(a \cdot c)) + F(f,g)(a + a \cdot b, b, c))fi$$

We define

$$shortestpaths(a,c) = F(id, \min paths)(a,c),$$

with

$$\min paths(a) = \text{let } ml = \min(\bigcup_{x \in a} \|x\|)$$
$$\text{in } \bigcup_{x \in a} \text{ if } \|x\| = ml \text{ then } x \text{ else } \emptyset.$$

Here $path \min length$ select from a set of words the ones with the least number of letters. Therefore, we have the following algorithm for computing the shortest path between a set S and the node y:

$shortestpaths(S,y)$
= {definition in Equ. 1}
$F(id, \min paths)(S,y)$
= {Equ. 1}
if $\langle y \subseteq S\rangle \ \vee \ (S \bowtie R)\rangle \subseteq S\rangle$
then $\min paths(S \bowtie y)$
else $\min paths(S \bowtie y \cup shortestpaths(S \cup S \bowtie R, y))$fi
= {Equ. 1}
if $y \in S\rangle$
then $\min paths(S \bowtie y)$
else if $(S \bowtie R)\rangle \subseteq S\rangle$
then $\min paths(S \bowtie y)$
else $\min paths(S \bowtie y \cup shortestpaths(S \cup S \bowtie R, y))$fi fi
= {$y \notin S\rangle \Rightarrow S \bowtie y = \emptyset$}
if $y \in S\rangle$
then $\min paths(S \bowtie y)$
else if $(S \bowtie R)\rangle \subseteq S\rangle$
then $\min paths(\emptyset)$
else $\min paths(shortestpaths(S \cup S \bowtie R, y))$fi fi
= {definition and idempotence of $\min paths$}
if $y \in S\rangle$
then $\min paths(S \bowtie y)$
else if $(S \bowtie R)\rangle \subseteq S\rangle$
then $\emptyset$
else $shortestpaths(S \cup S \bowtie R, y))$fi fi.

III. Kleene Algebra with Tests

Kleene algebras with tests [6], [19], [13] form a very important class of Kleene algebras. They provide an algebraic framework for manipulating programs. In the context of Kleene algebra with tests, we can show by the mean of equations when two programs are equivalent, i.e. when they perform the same tasks. The language of Kleene algebra with tests is the language of Kleene algebra together with a finite set of new constant symbols called primitive tests and one unary function ($^-$) which will be applied only on Boolean terms.

(12) **Definition.** A Kleene algebra with tests is a two sorted algebra

$$(K, B, +, \cdot, *, 0, 1, ^-) \tag{1}$$

where $^-$ is a unary operator defined only on B, such that

- $B \subseteq K$.
- $(K, +, \cdot, *, 0, 1)$ is a Kleene algebra.
- $(B, +, \cdot, ^-, 0, 1)$ is a Boolean algebra.

The elements of B are called tests. We reserve the letters $p, q, r, ...$ for arbitrary elements of K and $a, b, c, ...$ for tests.

(13) **Remarks.**

1. The sequential composition operator $(\cdot)$ acts as conjunction when applied to tests, and the choice operator $(+)$ acts as disjunction. Intuitively, a test bc succeeds iff both b and c succeed, and $b + c$ succeeds iff either b or c succeeds.

2. It is immediately from the definition that $b \leq 1$ for all $b \in B$. It is tempting to define tests in an arbitrary Kleene algebra to be the set $\{b \in K \mid b \leq 1\}$, and this is the approach taken by [4].

3. The last remark makes sense in algebras of binary relation [20], [10], but in general the set $\{b \in K \mid b \leq 1\}$ may not extend to a Boolean algebra. For example, in the $(min, +)$ Kleene algebra (Ex.7), $b \leq 1$ for all b, but the idempotence law $bb = b$ fails.

4. Every Kleene algebra extends trivially to a Kleene algebra with tests by taking the two-element Boolean algebra $\{0, 1\}$.

5. A Kleene algebra with tests is a Kleene algebra with an embedded Boolean algebra whose elements are called tests. So, the properties of Boolean algebra, like De Morgan's law ($\overline{b + b'} = \overline{b}\ \overline{b'}$ and $\overline{bb'} = \overline{b} + \overline{b'}$), double negation principle ($\overline{\overline{b}} = b$), ... will be used without proofs.

(14) **Theorem.** The axiom of Kleene algebra with tests is the axiom of Kleene algebra added with:

$$ab = ba \tag{2}$$
$$b + \overline{b} = 1 \tag{3}$$
$$b\overline{b} = 0. \tag{4}$$

(15) **Theorem.** $*$-continuous Kleene algebra with tests is a $*$-continuous Kleene algebra.

(16) **Theorem.** Let a, b be elements of Kleene algebra with tests, then we have

a) $1 + b = 1$
b) $bb = b$
c) $a \leq b \iff ab = a$.

Proof. The equations in a) and b) are follow from the axiom of Kleene algebra with tests. To prove c) suppose

$a \leq b \implies a + b = b$
$\implies aa + ab = ab$
$\implies a = aa \leq ab.$
Since $1 + b = b \implies a + ab = a$
$\implies ab \leq a.$
Conversely, suppose $ab = a \implies a + b = ab + b$
$\implies a + b = (a + 1)b$
$\implies a + b = 1b = b$
$\implies a \leq b.$

The following results will be used when we prove the equivalence of programs under some conditions. These works can be seen in [13].
Let K be a Kleene algebra with tests and B its embedded Boolean algebra.

(17) **Lemma.** For $p \in K$, $b \in B$, the following conditions are equivalent:

a) $bp = pb$.
b) $bp\overline{b} + \overline{b}pb = 0$.
c) $\overline{b}p = p\overline{b}$.

Proof. We start by show the equivalence of $a)$ and $b)$. Assuming $a)$,

$bp\overline{b} + \overline{b}pb$
= {Assumtion}
$pb\overline{b} + \overline{b}bp$
= {Equ.4}
$p0 + 0p = 0.$

Conversely, assuming b), we have $bp\bar{b} = \bar{b}pb = 0$, thus

pb
$=$ {Equ.3}
$(b+\bar{b})pb$
$=$ {Distributivity}
$bpb + \bar{b}pb$
$=$ {$\bar{b}pb = 0$}
$bpb + 0$
$=$ {$bp\bar{b} = 0$}
$bpb + bp\bar{b}$
$=$ {Boolean algebra}
$bp(b+\bar{b})$
$=$ {Equ.3}
bp.

Of course, any pair of tests commute, i.e., $bc = cb$, this is an axiom of Boolean algebra.
Now, we give useful results that are fairly evident from an intuitive point of view, but nevertheless require formal justification.

(18) **Lemma.** Let $p, b \in K$, such that $b^2 = b$ and $bp = pb$, then $bp^* = p^*b = b(pb)^* = (bp)^*b$.

Proof. The first and last equality follow from the (Th.5) with the condition $bp = pb$. Since $pb = pbb = b(pb)$, we have $p^*b = b(pb)^*$ (Th.5 again).

The terms $p+q$ and pq [respectively p^* and $\bar{p}$ (provides that it has a meaning)] are said to be generated by p and q [respectively p].

(19) **Proposition.** If $p \in K$ is generated by elements of K which commute with q then p commutes with q, that is $pq = qp$ is true in K.

Proof. If $p = p_1 + p_2$ then $pq = p_1q + p_2q = qp_1 + qp_2 = qp$. If $p = p_1p_2$ then $pq = (p_1p_2)q = (p_1q)p_2 = q(p_1p_2) = qp$. The case of $(\bar{\ })$ comes from (Lem.17) and the case of $*$ comes from (Th.5).

IV. While Programs

Kleene algebra with tests is a powerful framework to model programs algebraically. It permits us to reason about the equivalence between programs and algebraic equations.

In this section, we are going to model the sequential composition $p;q$, the conditional test `if` b `then` p `else` q and the looping construct `while` b `do` q with terms and operators. So, in the context of Kleene algebra with tests, we use the following definition:

a) $p;q := pq$
b) `if` b `then` p `else` $q := bp + \bar{b}q$
b) `while` b `do` $q := (bp)^*\bar{b}$.

Notice that the clauses `begin`...`end` will be used for parenthesising if necessary. For instance,

`if` b `then` p `else` $q;r$

should be

`begin if` b `then` p `else` q `end` r

but not

`if` b `then` p `else begin` $q;r$ `end`

Then any `while` program can be represented by an expression in the language of Kleene algebra with tests.

The independence of two programs p and q is expressed by a commutativity condition $pq = qp$. Intuitively, the value of the program p is not affected by the execution of the program q and conversely. The results in the first section used the particular case where p or q is a test.

(20) **Definition.** A program of the form p; `while` b `do` q, where p and q are `while` free, is said to be in normal form.

Our goal is to simulate any `while` program with a program in normal form (so contains at most one while loop). Often, we need to preserve the value of a test b across a program p. For, we introduce in a specific place the program $s; bc + \bar{b}\bar{c}$ such that c is a new test, s an atomic program symbol and the condition $cp = pc$ holds. The atomic program s assigns the value of b to some new Boolean variable which is tested in c and $bc + \bar{b}\bar{c}$ insures that the values of b and c are the same just after the execution of s. To reach our aim (Th.24), we need to consider four local transformations [13].

A. While within conditional

One considers the program P_1

`if` b `then begin` p_1; `while` b_1 `do` q_1 `end`
`else begin` p_2; `while` b_2 `do` q_2 `end`

We try to find a program equivalent to P_1 with at most one while loop together with the programs p_1, p_2, q_1, q_2 and the tests b, b_1, b_2. We need to preserve the value of b across p_1, p_2, q_1 and q_2, then we introduce an atomic program s and a test c which commutes with these programs. To do this, we first give the following theorem.

(21) **Theorem.** In the language of Kleene algebra with tests, the statement

$$(bc + \bar{b}\bar{c})(bp_1(b_1q_1)^*\bar{b_1} + \bar{b}p_2(b_2q_2)^*\bar{b_2}) \tag{5}$$

is equal to

$$(bc + \bar{b}\bar{c})(cp_1 + \bar{c}p_2)[(cb_1 + \bar{c}b_2)(cq_1 + \bar{c}q_2)]^*\overline{cb_1 + \bar{c}b_2}. \tag{6}$$

Proof.
$(bc + \bar{b}\bar{c})(cp_1 + \bar{c}p_2)[(cb_1 + \bar{c}b_2)(cq_1 + \bar{c}q_2)]^*\overline{cb_1 + \bar{c}b_2}$
$=$ {De Morgan's lows}
$(bc+\bar{b}\bar{c})(cp_1+\bar{c}p_2)[(cb_1+\bar{c}b_2)(cq_1+\bar{c}q_2)]^*(\bar{c}+\bar{b_1})(c+\bar{b_2})$
$=$ {Equ.4}
$(bc + \bar{b}\bar{c})(cp_1 + \bar{c}p_2)[(cb_1 + \bar{c}b_2)(cq_1 + \bar{c}q_2)]^*(c\bar{b_1} + \bar{c}\bar{b_2} + \bar{b_1}\bar{b_2})$
$=$ {$r = (cb_1 + \bar{c}b_2)(cq_1 + \bar{c}q_2) = cb_1q_1 + \bar{c}b_2q_2$}
$(bc + \bar{b}\bar{c})(cp_1 + \bar{c}p_2)r^*(c\bar{b_1} + \bar{c}\bar{b_2} + \bar{b_1}\bar{b_2})$
$=$ {$\overline{b_1b_2}(c\bar{b_1} + \bar{c}\bar{b_2}) = c\bar{b_1}\bar{b_2} + \bar{c}\bar{b_1}\bar{b_2} = \bar{b_1}\bar{b_2} \Longrightarrow \bar{b_1}\bar{b_2} \leq c\bar{b_1} + \bar{c}\bar{b_2}$}
$(bc + \bar{b}\bar{c})(cp_1 + \bar{c}p_2)r^*(c\bar{b_1} + \bar{c}\bar{b_2})$
$=$ {Distributivity}

$bcp_1r^*c\bar{b_1} + bcp_1r^*\bar{c}\bar{b_2} + \bar{b}\bar{c}p_2r^*c\bar{b_1} + \bar{b}\bar{c}p_2r^*\bar{c}\bar{b_2}$

Since r is generated by programs which commute with c and $\bar{c}$ (Lem.17), r and r^* commute with c and $\bar{c}$ (Prop.19). So, the middle terms vanish.
For the first term, we have

$$\begin{aligned} bcp_1(cb_1q_1 + \bar{c}b_2q_2)^*c\bar{b_1} &= bp_1c(cb_1q_1c + \bar{c}b_2q_2c)^*\bar{b_1} \\ &= bp_1c(cb_1q_1)^*\bar{b_1} \\ &= bp_1c(b_1q_1)^*\bar{b_1} \end{aligned}$$

and the last term is

$$\begin{aligned} \bar{b}\bar{c}p_2(cb_1q_1 + \bar{c}b_2q_2)^*\bar{c}\bar{b_2} &= \bar{b}p_2\bar{c}(cb_1q_1\bar{c} + \bar{c}b_2q_2\bar{c})^*\bar{b_2} \\ &= \bar{b}p_2\bar{c}(\bar{c}b_2q_2)^*\bar{b_2} \\ &= \bar{b}p_2\bar{c}(b_2q_2)^*\bar{b_2} \end{aligned}$$

The development of the expression (5) is exactly the sum of the first and last terms.

Then, by use the last theorem we get that the program P_1'

$s; bc + \bar{b}\bar{c}$
`if` b `then begin` p_1; `while` b_1 `do` q_1 `end`
`else begin` p_2; `while` b_2 `do` q_2 `end`

is equivalent to the program P_2

$s; bc + \bar{b}\bar{c};$
`if` c `then begin` p_1; `else` p_2;
`while` $cb_1 + \bar{c}b_2$ `do`
`if` c `then` q_1 `else` q_2

P_1 and P_1' are equivalent since the subprogram $s; bc + \bar{b}\bar{c}$ does not affect any values of the subprograms in the P_1-like part of P_1'. Without loss of generality, we can ignore the precomputation s since if these two programs are equivalent without s, then they are equivalent with it.

Therefore, the programs P_1 and P_2 are equivalent in the context of Kleene algebra with tests. Moreover, if the subprograms in the `then` and `else` of P_1 are in normal form then P_2 is in normal form.

B. Nested loops

One considers the program P_3

`while` b `do begin`
p; `while` c `do` q `end`

(22) **Theorem.** In the language of Kleene algebra with tests, the statement

$$(bp(cq)^*\bar{c})^*\bar{b}$$

is equal to

$$bp[(b+c)(cq+\bar{c}p)]^*\overline{b+c} + \bar{b}.$$

Proof.
$(bp(cq)^*\bar{c})^*\bar{b}$
$=$ {Use (k_1) with $x = bp(cq)^*\bar{c}$ }
$[1 + bp(cq)^*\bar{c}(bp(cq)^*\bar{c})^*]\bar{b}$
$=$ {Distributivity}
$\bar{b} + bp(cq)^*\bar{c}(bp(cq)^*\bar{c})^*\bar{b}$
$=$ {Lem.18}
$\bar{b} + bp(cq)^*(\bar{c}bp(cq)^*)^*\bar{c}\bar{b}.$
On the other hand
$bp[(b+c)(cq+\bar{c}p)]^*\overline{b+c} + \bar{b}$
$=$ {Distributivity}
$bp[bcq + b\bar{c}p + cq]^*\overline{b+c} + \bar{b}$
$=$ {De Morgan's lows}
$bp[bcq + b\bar{c}p + cq]^*\bar{b}\bar{c} + \bar{b}$
$=$ {Commutivity and Th.16}
$\bar{b} + bp(cq + b\bar{c}p)^*\bar{b}\bar{c}$
$=$ {Th.5}
$\bar{b} + bp(cq)^*(\bar{c}bp(cq)^*)^*\bar{c}\bar{b}.$

Then, by use the last theorem we get that the program P_3 is equivalent to the program P_4

`if` b `then begin`
p; `while` $b + c$ `do`
`if` c `then` q `else` p
`end`

After a transformation of P_3 into P_4, we apply the transformation (A) with 1; `while` 0 `do` 1 as subprogram in the missing else clause of P_4. Moreover, if p and q are while free, the resulting program is in normal form.

C. Removing postcomputation

We consider the program P_5

`while` b `do` $p; q$

Firstly, we need that the test b commutes with q so we introduce the program $s; bc + \bar{b}\bar{c}$ with the condition $cq = qc$. The program P_5 can be written as

$s; bc + \bar{b}\bar{c}$; `while` b `do begin` $p; s; bc + \bar{b}\bar{c}$ `end` q

and we claim that this is equivalent to

$s; bc + \bar{b}\bar{c}$; `while` c `do begin` $p; s; bc + \bar{b}\bar{c}$ `end` q

One can ignore the first occurrence of s and q when proving the equivalence. We have

$$(bc + \bar{b}\bar{c})(bps(bc + \bar{b}\bar{c}))^*\bar{b} = ((bc + \bar{b}\bar{c})bps)^*(bc + \bar{b}\bar{c})\bar{b} = (bcps)^*\bar{b}\bar{c}$$

and,

$$(bc + \bar{b}\bar{c})(cps(bc + \bar{b}\bar{c}))^*\bar{c} = ((bc + \bar{b}\bar{c})cps)^*(bc + \bar{b}\bar{c})\bar{c} = (bcps)^*\bar{b}\bar{c}$$

Thus, P_5 is equivalent to the following program P_5'

r; `while` c `do` $p'; q$
where $r = s; bc + \bar{b}\bar{c}$, $p' = p; s; bc + \bar{b}\bar{c}$ and with the condition $cq = qc$.

(23) **Theorem.** In the language of Kleene algebra with tests, the statement

$$(cp')^*\bar{c}q \tag{7}$$

is equal to

$$\bar{c}q + c(cp'(\bar{c}q + c))^*\bar{c} \tag{8}$$

Proof. By (k_1), this last expression becomes

$$\begin{aligned}
&\bar{c}q + c(1 + cp'(\bar{c}q + c)(cp'(\bar{c}q + c))^*)\bar{c} \\
&= \bar{c}q + cp'(\bar{c}q + c)(cp'(\bar{c}q + c))^*\bar{c} \\
&= \bar{c}q + cp'((\bar{c}q + c)cp')^*(\bar{c}q + c)\bar{c} \\
&= \bar{c}q + cp'(cp')^*\bar{c}q \\
&= (1 + cp'(cp')^*)\bar{c}q \\
&= (cp')^*\bar{c}q.
\end{aligned}$$

By (k_1), this last expression becomes.

Now, by use the last theorem we get that P_5' is equivalent to the program P_6

r; `if` $\bar{c}$ `then` q
  `else while` c `do begin`
    p'; `if` $\bar{c}$ `then` q
  `end`

Still with the same reason as before, one can ignore r.

Thus the programs P_5 and P_6 are equivalent. We apply the transformation (A) to P_6 and if p and q were `while` free, the resulting program is in normal form.

D. Composition of whiles

We consider the program P_7

p_1 `while` b_1 `do` q_1;
p_2 `while` b_2 `do` q_2

where p_1, p_2, q_1 and q_2 are `while` free.

The subprogram p_2 can be considered as a post computation of the first `while` subprogram. So by the previous transformation, P_7 is equivalent to the program P_7'

p_1'; `while` b_1' `do` q_1';
  `while` b_2 `do` q_2

Again, the subprogram `while` b_2 `do` q_2 is a postcomputation of the first while program. So, P_7' is equivalent to

p_1';r_1; `if` $\bar{c}$ `then while` b_2 `do` q_2
  `else while` c `do begin`
    q_1'; `if` $\bar{c}$ `then while` b_2 `do` q_2
  `end`

for some `while` free program r_1 and some test c.

Next, we use the transformation (A) on the program `if` $\bar{c}$ `then while` b_2 `do` q inside the `while` (with 1; `while` 0 do 1 in the missing `else` clause) and we should have a nested `while` loop in the `else` clause. The program P_7 will be equivalent to

p_1';r_1; `if` $\bar{c}$ `then while` b_2 `do` q_2
  `else while` c `do begin`
    q_1';r_2; `while` d `do` q_2'
  `end`

for some `while` free program r_2, q_2' and some test d.

After, we apply the transformation (B) to unwind the nested loops in the `else` clause. Thus P_7 will be equivalent to an instance of the program P_1 up to some precomputation. Then, we conclude with the transformation (A) that P_7 is equivalent to a program in normal form.

By a finite applications of the transformations (A, B, C and D), we have proved the following theorem:

(24) **Theorem.** In the context of Kleene algebra with tests, any `while` program is equivalent to a `while` program in normal form augmented with finitely many subprograms of the form $s; bc + \bar{b}\bar{c}$ and under some commutativity condition.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass., 1990.
[2] Backhouse, R.: Closure Algorithms and the star-Height Problem of Regular Languages. PhD thesis, Imperial College, London, U. K., 1975.
[3] Brunn, T., Moller, B. and Russling, M.: Layered Graph Traversals and Hamiltonian Path Problems - An Algebraic Approach. In: Jeuring, J. (ed.)MPC 1998. LNCS, vol.1422, pp.96-121. Springer, Heidelberg (1998).
[4] Cohen, E.: Hypotheses in Kleene algebra. Technical Report TM-ARH-023814, Bellcore, 1993.
[5] Conway, J.: Regular Algebra and finite Machines. Chapman and Hall, London, 1971.
[6] Desharnais, J., Mller, B. and Struth, G.: Kleene algebra with domain. TechnicalReport 2003-07, Universität Augsburg, Institut für Informatik, 2003.
[7] Eilenberg, S.: Automata, Languages, and Machines, volume A. Academic Press, New York, 1974.
[8] Iwano, K. and Steiglitz, K.: A semiring on convex polygons and zero-sum cycle problems. SIAM J. Comput., 19(5): 883-901, 1990.
[9] K. C. Ng. and Tarski, A.: Relation algebras with transitive closure, abstract 742-02-09. Notices Amer. Math. Soc., 24: A 29-A 30, 1977.
[10] K. C. Ng.: Relations with Transitive Closure. PhD thesis, University of California, Berkeley, 1984.
[11] Kozen, D.: On induction vs. $*-$continuity. In Kozen, editor, Proc. Workshop on Logic of Programs, volume 131 of Lecture Notes in Computer Science, Page 167-176, New York,1981. Springer-Verlag.
[12] Kozen, D.: The Design and Analysis of Algorithms. Springer-Verlag, New York, 1991.
[13] Kozen, D.: Kleene algebra with tests. Transactions on Programming Languages and Systems 19 (1997), no. 3, 427443.
[14] Kuich, W. and Salomaa, A.: Semirings, Automata, and Languages. Springer-Verlag, Berlin, 1986.
[15] Kuich, W.: The Kleene and Parikn theorem in complete semirings. In T. Ottmann, editor, Proc. 14th Colloq: Automata, Language, and Programming, volume 267 of Lecture Notes in Computer Science, page 212-225, New York, 1987. EATCS, Springer-Verlag.
[16] Mclver, A.: Mathematical foundations of Kleene Algebra. African Institute for Mathematical Sciences (AIMS). Stellenboch University, South Africa, (22 May 2008).
[17] Mehlhorn, K.: Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1984.
[18] Schmidt, G. and Ströhlein, T.: Relations and graphs EATCS Monographs on Theoretical Computer Science. Springer - Verlag, Berlin, Heidelberg 1993.
[19] Struth, G.: Calculating Church-Rosser proofs in Kleene algebra. In: H.C.M. deSwart (ed.). Relational Methods in Computer Science, 6th International Conference. Lecture Notes in Computer Science 2561. Springer 2002, 276-290.
[20] Tarski, A.: On the calculus of relations. J. Symb. Log. 1941.
[21] V. R. Pratt.: Dynamic algebras as a well-behaved fragment of relation algebras. In D. Pigozzi, editor, Proc. Conf. on Algebra and Computer Science. Springer-Verlag, June 1988.