Kleene Algebra with Tests and Commutativity Conditions

Dexter Kozen*

ABSTRACT We give an equational proof, using Kleene algebra with tests and commutativity conditions, of the following classical result: every **while** program can be simulated by a **while** program with at most one **while** loop. The proof illustrates the use of Kleene algebra with extra conditions in program equivalence proofs. We also show, using a construction of Cohen, that the universal Horn theory of *-continuous Kleene algebras is not finitely axiomatizable.

1 Introduction

Kleene algebras are algebraic structures with operators $+$, $\cdot$, *, 0, and 1 satisfying certain axioms. They arise in various guises in many contexts: relational algebra [28, 34], semantics and logics of programs [19, 29], automata and formal language theory [25, 26], and the design and analysis of algorithms [1, 17, 21]. Many authors have contributed to the development of Kleene algebra [2, 3, 4, 5, 6, 9, 12, 18, 19, 20, 22, 24, 26, 30, 31, 32, 33].

In semantics and logics of programs, Kleene algebra forms an essential component of Propositional Dynamic Logic (PDL) [13], in which it is mixed with Boolean algebra and modal logic to give a theoretically appealing and practical system for reasoning about computation at the propositional level.

Syntactically, PDL is a two-sorted logic consisting of *programs* and *propositions*, defined by mutual induction. A test $\varphi?$ can be formed from any proposition φ; intuitively, $\varphi?$ acts as a guard that succeeds with no side effects in states satisfying φ and fails or aborts in states not satisfying φ. Semantically, programs are modeled as binary relations on a set of states, and $\varphi?$ is interpreted as the subset of the identity relation consisting of all pairs (s, s) such that φ is true in state s.

From a practical point of view, many simple program manipulations, such as loop unwinding and basic safety analysis, do not require the full power of PDL, but can be carried out in a purely equational subsystem using the axioms of Kleene algebra. However, tests are an essential ingredi-

---

*Computer Science Department, Cornell University, Ithaca, New York 14853-7501, USA. `kozen@cs.cornell.edu`

ent, since they are needed to model conventional programming constructs such as conditionals and **while** loops. We define in §2 a variant of Kleene algebra, called *Kleene algebra with tests*, for reasoning equationally with these constructs.

Cohen has studied Kleene algebra in the presence of extra Boolean and commutativity conditions. He has given several practical examples of the use of Kleene algebra in program verification, such as lazy caching [10] and concurrency control [11]. He has also shown that Kleene algebra with extra conditions of the form $p = 0$ remains decidable [9], but that *-continuous Kleene algebra in the presence of extra commutativity conditions of the form $pq = qp$, even for atomic p and q, is undecidable [8].

In this paper we give two results, one of a more practical nature and the other theoretical. In §3, we give a complete equational proof of a classical folk theorem [14, 27] which states that every **while** program can be simulated by another **while** program with at most one **while** loop, provided extra Boolean variables are allowed. The approach we take is that of Mirkowska [27], who gives a set of local transformations that allow every **while** program to be transformed systematically to one with at most one **while** loop. For each such transformation, we give a purely equational proof of correctness. This result illustrates the use of Kleene algebra with tests and commutativity conditions in program equivalence proofs.

In §4, we observe that Cohen's construction establishing the undecidability of *-continuous Kleene algebra with added commutativity conditions actually shows that the universal Horn theory of the *-continuous Kleene algebras is not recursively enumerable, therefore not finitely axiomatizable. This resolves an open question of [22].

2 Kleene Algebra

A *Kleene algebra* [22] is an algebraic structure

$$(K,\ +,\ \cdot,\ ^*,\ 0,\ 1)$$

satisfying (1)–(15) below. As usual, we omit the $\cdot$ from expressions, writing pq for $p \cdot q$. The order of precedence of the operators is $^* > \cdot > +$; thus $p + qr^*$ should be parsed as $p + (q(r^*))$. The unary operator $^+$ is defined by $q^+ = qq^*$.

$$\begin{aligned}
p + (q + r) &= (p + q) + r && (1)\\
p + q &= q + p && (2)\\
p + 0 &= p && (3)\\
p + p &= p && (4)\\
p(qr) &= (pq)r && (5)
\end{aligned}$$

$$\begin{array}{rcll} 1p &=& p & (6)\\ p1 &=& p & (7)\\ p(q+r) &=& pq+pr & (8)\\ (p+q)r &=& pr+qr & (9)\\ 0p &=& 0 & (10)\\ p0 &=& 0 & (11)\\ 1+pp^* &=& p^* & (12)\\ 1+p^*p &=& p^* & (13)\\ q+pr\leq r &\rightarrow& p^*q\leq r & (14)\\ q+rp\leq r &\rightarrow& qp^*\leq r & (15) \end{array}$$

where $\leq$ refers to the natural partial order on K:

$$p \leq q \quad \leftrightarrow \quad p+q=q\ .$$

Instead of (14) and (15), we might take the equivalent axioms

$$\begin{array}{rcll} pr\leq r &\rightarrow& p^*r\leq r & (16)\\ rp\leq r &\rightarrow& rp^*\leq r\ . & (17) \end{array}$$

Axioms (12)–(17) say essentially that * behaves like the Kleene star operator of formal language theory or the reflexive transitive closure operator of relational algebra. See [23] for an introduction.

A Kleene algebra is said to be *-*continuous* if it satisfies the infinitary condition

$$pq^*r \quad = \quad \sup_{n\geq 0} pq^n r \tag{18}$$

where

$$\begin{array}{rcl} q^0 &=& 1\\ q^{n+1} &=& qq^n \end{array}$$

and the supremum is with respect to the natural order $\leq$. We can think of (18) as a conjunction of the infinitely many axioms $pq^n r \leq pq^*r$, $n \geq 0$, and the infinitary Horn formula

$$\bigwedge_{n\geq 0} pq^n r \leq s \quad \rightarrow \quad pq^*r \leq s\ .$$

In the presence of the other axioms, the *-continuity condition (18) implies (14)–(17), and is strictly stronger in the sense that there exist Kleene algebras that are not *-continuous [20].

All true identities between regular expressions, interpreted as regular sets of strings, are derivable from the axioms of Kleene algebra [22]. In the author's experience, two of the most useful such identities are

$$p^*(qp^*)^* = (p+q)^* \tag{19}$$
$$p(qp)^* = (pq)^*p\ . \tag{20}$$

For example, to derive the more complicated identity $(p^*q)^* = 1+(p+q)^*q$, we could reason equationally as follows:

$$\begin{aligned}(p^*q)^* &= 1+p^*q(p^*q)^* && \text{by (12)}\\ &= 1+p^*(qp^*)^*q && \text{by (20)}\\ &= 1+(p+q)^*q && \text{by (19).}\end{aligned}$$

2.1 Kleene Algebra with Tests

To accommodate tests, we introduce the following variant of Kleene algebra. A *Kleene algebra with tests* is a two-sorted algebra

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

where $B \subseteq K$ and $^-$ is a unary operator defined only on B, such that

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

is a Kleene algebra and

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

is a Boolean algebra. The elements of B are called *tests*. We reserve the letters p, q, r, s, t, u, v for arbitrary elements of K and a, b, c, d, e for tests. In PDL, a test would be written $b?$, but since we are using different symbols for tests we can omit the ?.

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.

It follows 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 $\{b \in K \mid b \leq 1\}$. This is the approach taken by Cohen [9]. This makes sense in algebras of binary relations [28, 34], 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 of the theory of algorithms (see [21]), $b \leq 1$ for all b, but the idempotence law $bb = b$ fails. Thus care must be taken with this approach.

We deliberately forgo this approach in favor of the explicit Boolean subalgebra in order to avoid these difficulties. Even over algebras of binary relations, we would like to admit models with programs whose input/output relations are subsets of the identity (*i.e.*, have no side effects) but whose complements are nevertheless uncomputable. We intend tests b to be viewed as simple predicates that are easily recognizable as such, and that are immediately decidable in a given state (and whose complements are therefore also immediately decidable).

2.2 While Programs

For the results of §3, we work with a PASCAL-like programming language with sequential composition $p\,;q$, a conditional test **if** b **then** p **else** q, and a looping construct **while** b **do** p. Programs built inductively from atomic programs and tests using these constructs are called *while programs*. We take the sequential composition operator to be of lower precedence than the conditional test or **while** loop, parenthesizing with **begin**... **end** where necessary; thus

$$\textbf{while}\ b\ \textbf{do}\ p\ ;q$$

should be parsed as

$$\textbf{begin while}\ b\ \textbf{do}\ p\ \textbf{end}\ ;q$$

and not

$$\textbf{while}\ b\ \textbf{do begin}\ p\ ;q\ \textbf{end}$$

We occasionally omit the **else** clause of a conditional test. This can be considered an abbreviation for a conditional test with the dummy **else** clause 1 (true).

These constructs are modeled in Kleene algebra with tests as follows:

$$\begin{aligned} p\,;q &= pq \\ \textbf{if}\ b\ \textbf{then}\ p\ \textbf{else}\ q &= bp+\overline{b}q \\ \textbf{if}\ b\ \textbf{then}\ p &= bp+\overline{b} \\ \textbf{while}\ b\ \textbf{do}\ p &= (bp)^*\overline{b}\ . \end{aligned}$$

See [23] for further discussion.

2.3 Commutativity Conditions

We will also be reasoning in the presence of *commutativity conditions* of the form $bp = pb$, where p is an arbitrary element of the Kleene algebra and b is a test. The practical significance of these conditions will become apparent in §3. Intuitively, the execution of program p does not affect the value of b. It stands to reason that if p does not affect b, then neither should it affect $\overline{b}$. This is indeed the case:

Lemma 1 *In any Kleene algebra with tests, the following are equivalent:*

(i) $pb = bp$

(ii) $p\overline{b} = \overline{b}p$

(iii) $bp\overline{b} + \overline{b}pb = 0$.

Proof. By symmetry, it suffices to show the equivalence of (i) and (iii). Assuming (i),

$$bp\overline{b} + \overline{b}pb \quad = \quad pb\overline{b} + \overline{b}bp \quad = \quad p0 + 0p \quad = \quad 0 \ .$$

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

$$\begin{array}{lllllllll} pb & = & (b+\overline{b})pb & = & bpb + \overline{b}pb & = & bpb + 0 & = & bpb \\ bp & = & bp(b+\overline{b}) & = & bpb + bp\overline{b} & = & bpb + 0 & = & bpb \ . \end{array}$$

□

Of course, any pair of tests commute, *i.e.*, $bc = cb$; this is an axiom of Boolean algebra.

We conclude this section with a pair of useful results that are fairly evident from an intuitive point of view, but nevertheless require formal justification.

Lemma 2 *In any Kleene algebra with tests, if* $bq = qb$, *then*

$$bq^* \quad = \quad (bq)^*b \quad = \quad q^*b \quad = \quad b(qb)^* \ .$$

Remark Note that it is *not* the case that $bq^* = (bq)^*$: when $b = 0$, the left hand side is 0 and the right hand side is 1.

Proof. We prove the three inequalities

$$bq^* \quad \leq \quad (bq)^*b \quad \leq \quad q^*b \quad \leq \quad bq^* \ ;$$

the equivalence of $b(qb)^*$ with these expressions follows from (20). For the first inequality, it suffices by axiom (15) to show that $b + (bq)^*bq \leq (bq)^*b$. By Boolean algebra and the commutativity assumption, we have $bq = bbq = bqb$, therefore

$$b + (bq)^*bq \quad = \quad b + (bq)^*bqb \quad = \quad (1 + (bq)^*bq)b \quad = \quad (bq)^*b \ .$$

The second inequality follows from $b \leq 1$ and the monotonicity of the Kleene algebra operators.

For the last inequality, it suffices by (14) to show $b + qbq^* \leq bq^*$:

$$b + qbq^* \quad = \quad b + bqq^* \quad = \quad b(1 + qq^*) \quad = \quad bq^* \ .$$

Note that in this last argument, we did not use the fact that b was a test.

□

Theorem 3 *In any Kleene algebra, if p is generated by a set of elements all of which commute with q, then p commutes with q.*

Proof. Let p be an expression in the language of Kleene algebra, and assume that all atomic subexpressions of p commute with q. The proof is by induction on the structure of p. The basis and all inductive cases except for programs of the form r^* are straightforward. For the inductive case $p = r^*$, we have by the induction hypothesis that $qr = rq$, and we need to argue that $qr^* = r^*q$. The inequality in one direction is given by the argument in the last paragraph in the proof of Lemma 2, which uses (14), and in the other direction by a symmetric argument using (15). □

3 A Folk Theorem

In this section we give an equational proof, using Kleene algebra with tests and commutativity conditions, of a classical result: *every* **while** *program can be simulated by a* **while** *program with at most one* **while** *loop, provided extra Boolean variables are allowed.* This theorem is the subject of a treatise on folk theorems by Harel [14], who notes that it is commonly but erroneously attributed to Böhm and Jacopini [7], and argues with some justification that it was known to Kleene. The version as stated here is originally due to Mirkowska [27], who gives a set of local transformations that allow every **while** program to be transformed systematically to one with at most one **while** loop. We consider a similar set of local transformations and give a purely equational proof of correctness for each. This result illustrates the use of Kleene algebra with tests and commutativity conditions in program equivalence proofs.

It seems to be a commonly held belief that this result has no purely schematic (*i.e.*, propositional, uninterpreted) proof [14]. The proofs of [15, 27], as reported in [14], use extra variables to remember certain values at certain points in the program, either program counter values or the values of tests. Since having to remember these values seems unavoidable, one might infer that the only recourse is to introduce extra variables, along with an explicit assignment mechanism for assigning values to them. Thus, as the argument goes, proofs of this theorem cannot be purely propositional.

We do not agree completely with this conclusion. The only purpose of these extra variables is to *preserve values across computations.* In our treatment, we only need to preserve the values of certain tests b over certain computations p. We can handle this equationally by introducing a new test c, which we can assume is set to the value of b in some precomputation, and postulating a commutativity condition of the form $cp = pc$, which says intuitively that the value of c is not affected by the execution of p. No explicit assignment mechanism is necessary; we just assume that c already has the correct value.

3.1 An Example

To illustrate this technique, consider the simple program

$$\begin{array}{l}\textbf{if } b \textbf{ then begin } p\,;\, q \textbf{ end}\\ \quad\;\textbf{else}\;\; \textbf{begin } p\,;\, r \textbf{ end}\end{array} \tag{21}$$

If the value of b were preserved by p, then we could rewrite this program more simply as

$$p\ ;\ \textbf{if } b \textbf{ then } q \textbf{ else } r \tag{22}$$

Formally, the assumption that the value of b is preserved by p takes the form of the commutativity condition $bp = pb$. By Lemma 1, we also have $\overline{b}p = p\overline{b}$. Expressed in the language of Kleene algebra, the equivalence of (21) and (22) becomes the equation

$$bpq + \overline{b}pr \quad = \quad p(bq + \overline{b}r)\ .$$

This identity can be established by simple equational reasoning:

$$\begin{array}{rcll} p(bq + \overline{b}r) & = & pbq + p\overline{b}r & \text{by (8)}\\ & = & bpq + \overline{b}pr & \text{by the commutativity assumptions.}\end{array}$$

But what if b is not preserved by p? This situation seems to call for a Boolean variable to remember the value of b across p, and an assignment mechanism to set the value of the variable. However, we do not need to take such a drastic step. We can stay within the realm of uninterpreted equational logic by introducing a new atomic test c and commutativity condition $pc = cp$, intuitively modeling the idea that c tests a variable that is not modified by p. We make the program (22) test c instead of b. We then preface both programs with the guard $bc + \overline{b}\overline{c}$ (in the language of **while** programs, **if** b **then** c **else** $\overline{c}$) which asserts that initially b and c have the same value. Intuitively, *we are assuming that c has already been set to the value of b in some (omitted) precomputation.* We can even include an atomic program s and pretend that s performs this precomputation if we like, although this is not really necessary: if the two programs are already equivalent without the s in front, then they are certainly equivalent with it.

We can now give a purely equational proof of the equivalence of the two programs

$$\begin{array}{l} bc + \overline{b}\overline{c}\ ;\\ \textbf{if } b \textbf{ then begin } p\,;\, q \textbf{ end}\\ \quad\;\textbf{else}\;\; \textbf{begin } p\,;\, r \textbf{ end}\end{array} \tag{23}$$

and

$$\begin{array}{l} bc + \overline{b}\overline{c}\ ;\\ p\,;\, \textbf{if } c \textbf{ then } q \textbf{ else } r\end{array} \tag{24}$$

using the axioms of Kleene algebra with tests and the commutativity conditions $pc = cp$ and $p\overline{c} = \overline{c}p$. Expressed in the language of Kleene algebra, the equivalence of (23) and (24) becomes

$$(bc + \overline{b}\overline{c})(bpq + \overline{b}pr) \quad = \quad (bc + \overline{b}\overline{c})p(cq + \overline{c}r) \ . \tag{25}$$

Using the distributive laws (8) and (9) and the laws of Boolean algebra, we can simplify the left hand side of (25) as follows:

$$\begin{aligned}(bc + \overline{b}\overline{c})(bpq + \overline{b}pr) \quad &= \quad bcbpq + \overline{b}\overline{c}bpq + bc\overline{b}pr + \overline{b}\overline{c}\overline{b}pr \\ &= \quad bcpq + \overline{b}\overline{c}pr \ .\end{aligned}$$

The right hand side of (25) simplifies in a similar fashion to the same expression:

$$\begin{aligned}(bc + \overline{b}\overline{c})p(cq + \overline{c}r) \quad &= \quad bcpcq + \overline{b}\overline{c}pcq + bcp\overline{c}r + \overline{b}\overline{c}p\overline{c}r \\ &= \quad bccpq + \overline{b}\overline{c}cpq + bc\overline{c}pr + \overline{b}\overline{c}\overline{c}pr \\ &= \quad bcpq + \overline{b}\overline{c}pr \ .\end{aligned}$$

Here the commutativity assumptions are used in the second step.

We can even do away with the guard $bc + \overline{b}\overline{c}$ in this argument by the following consideration. If we assume that c is assigned the value of b in the precomputation in both programs, then we might as well test c instead of b in (23) as well as (24). But then we don't need the guard at all, since the two programs are already equivalent without it by the original two-line proof given at the beginning of this section with b replaced by c.

3.2 Normal Form

A program is in *normal form* if it is of the form

$$p\,;\,\textbf{while}\ b\ \textbf{do}\ q \tag{26}$$

where p and q are **while**-free. The subprogram p is called the *precomputation* of the normal form.

We show that every program can be transformed to a program in normal form. This is done inductively on the structure of the program. Each programming construct accounts for one case in the inductive proof, and we consider each case separately. For each case, we give a transformation that moves an inner **while** loop to the outside and an equational proof of its correctness.

3.3 Conditional

We first show how to move two programs in normal form in the **then** and **else** clauses of a conditional, respectively, outside the conditional. Consider the program

$$\begin{array}{l} \textbf{if } b \textbf{ then begin } p_1 \textbf{ ; while } c_1 \textbf{ do } q_1 \textbf{ end} \\ \quad\quad \textbf{else begin } p_2 \textbf{ ; while } c_2 \textbf{ do } q_2 \textbf{ end} \end{array} \tag{27}$$

We can assume without loss of generality that b commutes with p_1, p_2, q_1, and q_2. If not, we could introduce a new test whose value would be set to the value of b in the precomputation, then use it in place of b in the conditional test, as described in §3.1.

Under these assumptions, we show that (27) is equivalent to

$$\begin{array}{l} \textbf{if } b \textbf{ then } p_1 \textbf{ else } p_2 \textbf{ ;} \\ \textbf{while } bc_1 + \overline{b}c_2 \textbf{ do} \\ \quad \textbf{if } b \textbf{ then } q_1 \textbf{ else } q_2 \end{array} \tag{28}$$

Note that if the two programs in the **then** and **else** clauses of (27) are in normal form, then (28) is in normal form.

Written in the language of Kleene algebra, (27) becomes

$$bp_1(c_1q_1)^*\overline{c}_1 + \overline{b}p_2(c_2q_2)^*\overline{c}_2 \tag{29}$$

and (28) becomes

$$(bp_1 + \overline{b}p_2)((bc_1 + \overline{b}c_2)(bq_1 + \overline{b}q_2))^*\overline{bc_1 + \overline{b}c_2}\ . \tag{30}$$

The subexpression $\overline{bc_1 + \overline{b}c_2}$ of (30) is equivalent by propositional reasoning to $b\overline{c}_1 + \overline{b}\overline{c}_2$. Here we have used the familiar propositional equivalence

$$bc_1 + \overline{b}c_2 \quad = \quad (\overline{b} + c_1)(b + c_2)$$

and a De Morgan law. The starred expression in (30) can be simplified using distributivity and Boolean algebra:

$$\begin{aligned} (bc_1 + \overline{b}c_2)(bq_1 + \overline{b}q_2) \quad &= \quad bc_1bq_1 + bc_1\overline{b}q_2 + \overline{b}c_2bq_1 + \overline{b}c_2\overline{b}q_2 \\ &= \quad bc_1q_1 + \overline{b}c_2q_2\ . \end{aligned}$$

Substituting these simplified expressions in the original expression (30), we obtain

$$(bp_1 + \overline{b}p_2)(bc_1q_1 + \overline{b}c_2q_2)^*(b\overline{c}_1 + \overline{b}\overline{c}_2)\ . \tag{31}$$

Using distributivity, this can be broken up into the sum of four terms:

$$\begin{aligned} & bp_1(bc_1q_1 + \overline{b}c_2q_2)^*b\overline{c}_1 && (32) \\ + \quad & bp_1(bc_1q_1 + \overline{b}c_2q_2)^*\overline{b}\overline{c}_2 && (33) \\ + \quad & \overline{b}p_2(bc_1q_1 + \overline{b}c_2q_2)^*b\overline{c}_1 && (34) \\ + \quad & \overline{b}p_2(bc_1q_1 + \overline{b}c_2q_2)^*\overline{b}\overline{c}_2\ . && (35) \end{aligned}$$

Under the commutativity assumptions, Lemma 2 implies that (33) and (34) reduce to 0; and for the remaining two terms (32) and (35),

$$\begin{aligned} bp_1(bc_1q_1 + \overline{b}c_2q_2)^*b\overline{c}_1 &= bp_1(bbc_1q_1 + b\overline{b}c_2q_2)^*\overline{c}_1 \\ &= bp_1(c_1q_1)^*\overline{c}_1 \\ \overline{b}p_2(bc_1q_1 + \overline{b}c_2q_2)^*\overline{b}\overline{c}_2 &= \overline{b}p_2(\overline{b}bc_1q_1 + \overline{b}\overline{b}c_2q_2)^*\overline{c}_2 \\ &= \overline{b}p_2(c_2q_2)^*\overline{c}_2 \ . \end{aligned}$$

The sum of these two terms is (29).

3.4 Nested Loops

We next consider the case that is perhaps the most interesting: denesting two nested **while** loops. This construction is particularly remarkable in that no commutativity conditions (thus no extra variables) are needed; compare the corresponding transformations of [15, 27], as reported in [14], which do use extra variables.

We show that the program

```
while b do begin
             p ;
             while c do q
           end
```
(36)

is equivalent to the program

```
if b then begin
           p ;
           while b + c do
             if c then q else p
         end
```
(37)

This construction transforms a pair of nested **while** loops to a single **while** loop inside a conditional test. No commutativity conditions are assumed in the proof.

After this transformation, the **while** loop can be taken outside the conditional using the transformation of §3.3 (this part does require a commutativity condition). A dummy normal form such as 1**;** **while** 0 **do** 1 can be supplied for the missing **else** clause. Note that if the program inside the **begin...end** block of (36) is in normal form, then the resulting program will be in normal form.

Not surprisingly, the key property used in the proof is the denesting property (19), which equates a regular expression of *-depth two with another of *-depth one.

Translating to the language of Kleene algebra, (36) becomes

$$(bp(cq)^*\overline{c})^*\overline{b} \tag{38}$$

and (37) becomes

$$bp((b+c)(cq+\overline{c}p))^*\overline{b+c}+\overline{b}\ . \tag{39}$$

The $\overline{b}$ in (39) is for the nonexistent **else** clause of the outermost conditional of (37). Unwinding the outer loop in (38) using (12) and distributing $\overline{b}$ over the resulting sum, we obtain

$$\overline{b}+bp(cq)^*\overline{c}(bp(cq)^*\overline{c})^*\overline{b}\ .$$

Removing $\overline{b}$ and bp from this expression and (39), it suffices to show

$$(cq)^*\overline{c}(bp(cq)^*\overline{c})^*\overline{b} \quad = \quad ((b+c)(cq+\overline{c}p))^*\overline{b+c}\ .$$

Using (20) on the left hand side and propositional reasoning on the right, this simplifies to

$$(cq)^*(\overline{c}bp(cq)^*)^*\overline{c}\overline{b} \quad = \quad ((b+c)(cq+\overline{c}p))^*\overline{c}\overline{b}\ .$$

Removing the $\overline{c}\overline{b}$ on both sides, this further simplifies to

$$(cq)^*(\overline{c}bp(cq)^*)^* \quad = \quad ((b+c)(cq+\overline{c}p))^*\ . \tag{40}$$

Now here is the key step at which the loop is denested. Applying (19) to the left hand side of (40), we obtain

$$(cq+\overline{c}bp)^* \quad = \quad ((b+c)(cq+\overline{c}p))^*\ ,$$

so it suffices to show the equivalence of the subexpressions

$$cq+\overline{c}bp \quad = \quad (b+c)(cq+\overline{c}p)\ .$$

The right hand side is easily transformed to the left using the basic laws of Kleene and Boolean algebra:

$$\begin{aligned}(b+c)(cq+\overline{c}p) &= bcq+b\overline{c}p+ccq+c\overline{c}p\\ &= bcq+cq+\overline{c}bp\\ &= (b+1)cq+\overline{c}bp\\ &= cq+\overline{c}bp\ .\end{aligned}$$

3.5 Eliminating Postcomputations

We wish to show that a program occurring after a **while** loop can be absorbed into the **while** loop. Consider a program of the form

while b **do** $p\,;q$ (41)

By introducing a new test if necessary, we can assume without loss of generality that b commutes with q. (Intuitively, the value of the new test will have to be set implicitly both in the precomputation and at the end of p. Formally, we would establish the equivalence of the two programs

$$(bc+\overline{b}\overline{c})\,;\,\textbf{while}\ b\ \textbf{do begin}\ p\,;\,(bc+\overline{b}\overline{c})\ \textbf{end}$$

$$(bc+\overline{b}\overline{c})\,;\,\textbf{while}\ c\ \textbf{do begin}\ p\,;\,(bc+\overline{b}\overline{c})\ \textbf{end}$$

We leave this as an exercise.) Under this assumption, we show that the program (41) is equivalent to the program

```
if b̄ then q
    else  while b do begin
                        p ;
                        if b̄ then q
                      end
```
(42)

Note that if p and q are **while**-free, then (42) consists of a program in normal form inside a conditional, which can be transformed to normal form using the construction of §3.3.

Written in the language of Kleene algebra, (41) becomes

$$(bp)^*\overline{b}q \tag{43}$$

and (42) becomes

$$\overline{b}q + b(bp(\overline{b}q+b))^*\overline{b}\ . \tag{44}$$

Unwinding one iteration of (43) using (12) and distributing $\overline{b}q$ over the resulting sum, we obtain

$$\overline{b}q + bp(bp)^*\overline{b}q\ .$$

By distributivity, (44) is equivalent to

$$\overline{b}q + b(bp\overline{b}q + bpb)^*\overline{b}\ .$$

Eliminating the term $\overline{b}q$ from both sides, it suffices to prove

$$bp(bp)^*\overline{b}q \quad = \quad b(bp\overline{b}q+bpb)^*\overline{b}\ . \tag{45}$$

At this point we seem to have reached an impasse, since b does not necessarily commute with p, so Lemma 2 does not apply. The trick here is to use the denesting rule (19) in the wrong direction. Starting with the right hand side,

$$\begin{aligned} & b(bp\overline{b}q+bpb)^*\overline{b} \\ & \quad = \quad b(bpb)^*(bp\overline{b}q(bpb)^*)^*\overline{b} && \text{by (19)} \end{aligned}$$

$$\begin{aligned}
&= (bbp)^*b(bp\overline{b}q(1+bpb(bpb)^*))^*\overline{b} && \text{by (20) and (12)}\\
&= (bp)^*b(bp\overline{b}q + bp\overline{b}qbpb(bpb)^*)^*\overline{b} && \\
&= (bp)^*b(bp\overline{b}q)^*\overline{b} && \text{since } \overline{b}qb = \overline{b}bq = 0\\
&= (bp)^*b(1 + bp\overline{b}q(1+bp\overline{b}q(bp\overline{b}q)^*))\overline{b} && \text{by (12)}\\
&= (bp)^*(b\overline{b} + bp\overline{b}q\overline{b} + bp\overline{b}qbp\overline{b}q(bp\overline{b}q)^*\overline{b}) && \\
&= (bp)^*bp\overline{b}q\overline{b} && \text{since } b\overline{b} = \overline{b}qb = 0\\
&= bp(bp)^*\overline{b}q && \text{by (20).}
\end{aligned}$$

3.6 Composition

The composition of two programs in normal form

$$\begin{array}{l} p_1\ ;\\ \textbf{while}\ b_1\ \textbf{do}\ q_1\ ;\\ p_2\ ;\\ \textbf{while}\ b_2\ \textbf{do}\ q_2 \end{array} \tag{46}$$

can be transformed to a single program in normal form. We have actually already done all the work needed to handle this case. First, we use the result of §3.5 to absorb the **while**-free program p_2 into the first **while** loop. We can also ignore p_1, since it can be absorbed into the precomputation of the resulting normal form when we are done. It therefore suffices to show how to transform a program

$$\begin{array}{l} \textbf{while}\ b\ \textbf{do}\ p\ ;\\ \textbf{while}\ c\ \textbf{do}\ q \end{array} \tag{47}$$

to normal form, where p and q are **while**-free.

As argued previously, we can assume without loss of generality that the test b commutes with the program q by introducing a new test if necessary and assuming that its value is set in the precomputation and at the end of p. Since b also commutes with c by Boolean algebra, by Theorem 3 we have that b commutes with the entire second **while** loop. This allows us to use the transformation of §3.5, absorbing the second **while** loop into the first. The resulting program looks like

$$\begin{array}{l} \textbf{if}\ \overline{b}\ \textbf{then while}\ c\ \textbf{do}\ q\\ \quad\textbf{else}\ \ \textbf{while}\ b\ \textbf{do begin}\\ \qquad\qquad\qquad p\ ;\\ \qquad\qquad\qquad \textbf{if}\ \overline{b}\ \textbf{then while}\ c\ \textbf{do}\ q\\ \qquad\qquad\quad \textbf{end} \end{array} \tag{48}$$

At this point we can apply the transformation of §3.3 to the subprogram

$$\textbf{if}\ \overline{b}\ \textbf{then while}\ c\ \textbf{do}\ q$$

using a dummy normal form for the omitted **else** clause, giving two nested loops in the **else** clause of (48); then the transformation of §3.4 to the **else** clause of (48); finally, the transformation of §3.3 to the entire resulting program, yielding a program in normal form.

The transformations of §§3.3–3.6 give a systematic method for moving **while** loops outside of any other programming construct. By applying these transformations inductively from the innermost loops outward, we can transform any program into a program in normal form.

None of these arguments needed explicit Boolean variables or any assignment mechanism. Where did they go? Of course they would be there in a real implementation, but they do not play a role in the proofs because they are hidden in "without loss of generality..." assumptions. The point is that it is not significant exactly how a Boolean value is preserved across a computation, but rather that it *can be* preserved; and for the purposes of formal verification, this fact is completely captured by a commutativity assumption. Thus we are justified in our claim that we have given a purely equational proof of this result.

4 Undecidability

Cohen [8] has shown that *-continuous Kleene algebra with extra commutativity conditions of the form $pq = qp$ is undecidable. We reproduce his construction below.

Theorem 4 (Cohen) *It is undecidable whether a given identity holds in all* **-continuous Kleene algebras satisfying a given finite set of identities of the form* $pq = qp$.

Proof. We encode Post's Correspondence Problem (PCP) (see [16]). Let I be an instance of PCP consisting of k pairs of strings $x_i, y_i \in \{p,q\}^+$, $1 \le i \le k$, where p and q are atomic symbols. For $\alpha \in \{1,\ldots,k\}^*$, define x_α inductively by

$$\begin{aligned} x_\epsilon &= \epsilon \\ x_{\alpha i} &= x_\alpha x_i \ , \end{aligned}$$

and define y_α similarly. A *solution* to the instance I of PCP is a string $\alpha \in \{1,\ldots,k\}^+$ such that $x_\alpha = y_\alpha$.

Let $\{p',q'\}$ be a disjoint copy of $\{p,q\}$, and let $z' \in \{p',q'\}^*$ denote the image of the string $z \in \{p,q\}^*$ under the homomorphism $p \mapsto p'$, $q \mapsto q'$. Consider the commutativity conditions

$$uv' = v'u \ , \quad u,v \in \{p,q\} \ . \tag{49}$$

Let s and t be the expressions

$$\begin{aligned} s &= (x_1y_1' + x_2y_2' + \cdots + x_ky_k')^+ \\ t &= (pp' + qq')^*((p+q)^+ + (p'+q')^+ + (pq' + qp')(p+q+p'+q')^*) \ . \end{aligned}$$

Intuitively, modulo the commutativity conditions (49), the regular expression s represents the set of all $x_\alpha y_\alpha'$, and the regular expression t denotes the set of all non-solutions to I.

We claim that the inequality $s \leq t$ is a logical consequence of the identities (49) and the axioms of *-continuous Kleene algebra if and only if the instance I of PCP has no solution. In other words, the universal Horn formula

$$pp' = p'p \quad \wedge \quad pq' = q'p \quad \wedge \quad qp' = p'q \quad \wedge \quad qq' = q'q \quad \to \quad s \leq t$$

holds in all *-continuous Kleene algebras iff I has no solution.

Suppose I has no solution. For $\alpha \in \{1, \ldots, k\}^+$, let $\alpha = \alpha_1\alpha_2\cdots\alpha_n$ where $n \geq 1$ and each $\alpha_i \in \{1, \ldots, k\}$, $1 \leq i \leq n$. Let z be the longest common prefix of x_α and y_α. By the commutativity conditions, $x_\alpha y_\alpha$ is equivalent to a string of the form $zz'pq'w$ or $zz'qp'w$ for w an arbitrary string, or $zz'w$ for w a nonnull string of all primed or all unprimed symbols. There are no other possibilities, since α is not a solution to I. By the commutativity conditions and Kleene algebra, all such strings can be shown to be less than or equal to t. By [21, pp. 221, 246], s is the supremum of all these elements, therefore $s \leq t$.

Conversely, if I has a solution $\alpha = \alpha_1\alpha_2\cdots\alpha_n \in \{1, \ldots, k\}^n$, $n \geq 1$, say $x_\alpha = y_\alpha = z$, we claim that $s \leq t$ is not a logical consequence of (49) and the axioms of *-continuous Kleene algebra. It suffices to construct a *-continuous Kleene algebra satisfying (49) in which $s \not\leq t$. Consider the Kleene algebra of binary relations on the set of strings $\{p,q\}^* \cup \{p', q'\}^*$, where the operators have their standard relation-theoretic interpretations. We interpret the symbols p, q, p', q' as follows:

$$\begin{aligned} u &= \{(x, xu) \mid x \in \{p,q\}^*\} \cup \{(u'x', x') \mid x \in \{p,q\}^*\} \ , \quad u \in \{p,q\} \\ u' &= \{(x', x'u') \mid x \in \{p,q\}^*\} \cup \{(ux, x) \mid x \in \{p,q\}^*\} \ , \quad u \in \{p,q\} \ . \end{aligned}$$

Let $e = \{(\epsilon, \epsilon)\}$. It is straightforward to verify that the equations (49) hold in this model, and that $epp' = eqq' = e$. It follows that $e(pp' + qq')^* = e$ and $ezz'e = e$. Since $zz' = x_\alpha y_\alpha' \leq s$, we have $e \leq ese$, therefore $ese \neq 0$.

Now it also follows that $epq' = eqp' = 0$ and $e(p+q)^+e = e(p'+q')^+e = 0$, therefore

$$\begin{aligned} ete &= e((p+q)^+ + (p'+q')^+ + (pq'+qp')(p+q+p'+q')^*)e \\ &= e(p+q)^+e + e(p'+q')^+e + e(pq'+qp')(p+q+p'+q')^*e \\ &= 0 \ . \end{aligned}$$

Since $ese \neq 0$ and $ete = 0$, we cannot have $s \leq t$. □

Let $H(\mathrm{KA})$ (respectively, $H(\mathrm{KA}^*)$) denote the universal Horn theory of the Kleene algebras (respectively, the *-continuous Kleene algebras). Cohen's proof establishes more than just the undecidability of $H(\mathrm{KA}^*)$: it actually shows that $H(\mathrm{KA}^*)$ is not recursively enumerable, therefore not finitely axiomatizable. This is because his proof gives a many-one reduction of PCP to the complement of $H(\mathrm{KA}^*)$; *i.e.*, the given instance of PCP is satisfiable iff the resulting Horn formula is *not* valid. Since PCP is r.e.-complete, its complement is not r.e., therefore neither is $H(\mathrm{KA}^*)$. This answers an open question of [22], which asked whether the axioms of Kleene algebra were complete for $H(\mathrm{KA}^*)$; in other words, do $H(\mathrm{KA})$ and $H(\mathrm{KA}^*)$ coincide? The answer is no: the former is recursively enumerable (it is a universal Horn theory), whereas the latter is not.

5 Related Results and Open Problems

Using [21, pp. 221, 246], it can be shown that the equational theory of *-continuous Kleene algebras with tests is complete for relational models, and also admits a free language-theoretic model consisting of sets of "guarded strings". Using this result and a technique based on [22], it can be shown that the equational theories of Kleene algebras with tests and *-continuous Kleene algebras with tests coincide. This result is the analog of [22] for the case of Kleene algebras with tests.

Although Theorem 4 shows that *-continuous Kleene algebra with general commutativity conditions is undecidable, the only commutativity conditions needed in the proof of §3 are of the form $bq = qb$, where b is a test. Lemma 1 shows that these conditions are equivalent to conditions of the form $p = 0$. Cohen [9] shows that Kleene algebra with conditions $p = 0$ reduces efficiently to Kleene algebra without conditions. A construction similar to Cohen's can be used to show that Kleene algebra with tests and conditions $p = 0$ reduces efficiently to Kleene algebra with tests alone, and similarly for *-continuous Kleene algebra with tests.

The following interesting questions present themselves:

1. The equational theory of Kleene algebras with tests can be shown decidable by a simple reduction to PDL. What is its complexity? It is at most deterministic exponential time (since PDL is) and at least *PSPACE*-hard (since the equational theory of Kleene algebras is). We conjecture that it is *PSPACE*-complete.

2. What is the complexity of $H(\mathrm{KA}^*)$? It is not r.e., but how high does it go?

3. By the results of §4, there must exist a universal Horn sentence that is true in all *-continuous Kleene algebras but violated in some Kleene algebra. Is there a natural example of such a sentence?

Acknowledgements

Ernie Cohen, David Gries, David Harel, Vaughan Pratt, and Fred B. Schneider provided valuable comments. I am indebted to Ernie Cohen for his kind permission to include his previously unpublished Theorem 4.

6 References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1975.

[2] S. Anderaa. On the algebra of regular expressions. *Appl. Math.*, pages 1–18, January 1965.

[3] K. V. Archangelsky. A new finite complete solvable quasiequational calculus for algebra of regular languages. Manuscript, Kiev State University, 1992.

[4] Roland Carl Backhouse. *Closure Algorithms and the Star-Height Problem of Regular Languages.* PhD thesis, Imperial College, 1975.

[5] Stephen L. Bloom and Zoltán Ésik. Equational axioms for regular sets. Technical Report 9101, Stevens Institute of Technology, May 1991.

[6] Maurice Boffa. Une remarque sur les systèmes complets d'identités rationnelles. *Informatique théoretique et Applications/Theoretical Informatics and Applications*, 24(4):419–423, 1990.

[7] C. Böhm and G. Jacopini. Flow diagrams, Turing machines, and languages with only two formation rules. *Comm. Assoc. Comput. Mach.*, 9(5):366–371, May 1966.

[8] Ernie Cohen, February 1994. Personal communication.

[9] Ernie Cohen. Hypotheses in Kleene algebra. `ftp://ftp.bellcore.com/pub/ernie/research/homepage.html`, April 1994.

[10] Ernie Cohen. Lazy caching. `ftp://ftp.bellcore.com/pub/ernie/research/homepage.html`, 1994.

[11] Ernie Cohen. Using Kleene algebra to reason about concurrency control. `ftp://ftp.bellcore.com/pub/ernie/research/homepage.html`, 1994.

[12] John Horton Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.

[13] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.

[14] David Harel. On folk theorems. *Comm. Assoc. Comput. Mach.*, 23(7):379–389, July 1980.

[15] K. Hirose and M. Oya. General theory of flowcharts. *Comment. Math. Univ. St. Pauli*, 21(2):55–71, 1972.

[16] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[17] Kazuo Iwano and Kenneth Steiglitz. A semiring on convex polygons and zero-sum cycle problems. *SIAM J. Comput.*, 19(5):883–901, 1990.

[18] Stephen C. Kleene. Representation of events in nerve nets and finite automata. In Shannon and McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[19] Dexter Kozen. On induction vs. *-continuity. In Kozen, editor, *Proc. Workshop on Logic of Programs*, volume 131 of *Lect. Notes in Comput. Sci.*, pages 167–176. Springer, 1981.

[20] Dexter Kozen. On Kleene algebras and closed semirings. In Rovan, editor, *Proc. Math. Found. Comput. Sci.*, volume 452 of *Lect. Notes in Comput. Sci.*, pages 26–47. Springer, 1990.

[21] Dexter Kozen. *The Design and Analysis of Algorithms.* Springer, 1991.

[22] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Infor. and Comput.*, 110(2):366–390, May 1994.

[23] Dexter Kozen and Jerzy Tiuryn. Logics of programs. In van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 789–840. North Holland, 1990.

[24] Daniel Krob. A complete system of B-rational identities. *Theoretical Computer Science*, 89(2):207–343, October 1991.

[25] Werner Kuich. The Kleene and Parikh theorem in complete semirings. In Ottmann, editor, *Proc. 14th Colloq. Automata, Languages, and Programming*, volume 267 of *Lect. Notes in Comput. Sci.*, pages 212–225. EATCS, Springer, 1987.

[26] Werner Kuich and Arto Salomaa. *Semirings, Automata, and Languages.* Springer, 1986.

[27] G. Mirkowska. *Algorithmic Logic and its Applications.* PhD thesis, University of Warsaw, 1972. In Polish.

[28] K. C. Ng. *Relation Algebras with Transitive Closure.* PhD thesis, University of California, Berkeley, 1984.

[29] Vaughan Pratt. Dynamic algebras as a well-behaved fragment of relation algebras. In D. Pigozzi, editor, *Proc. Conf. Algebra and Computer Science*, volume 425 of *Lect. Notes in Comput. Sci.*, pages 77–110. Springer, June 1988.

[30] Vaughan Pratt. Action logic and pure induction. In J. van Eijck, editor, *Proc. Logics in AI: European Workshop JELIA '90*, volume 478 of *Lect. Notes in Comput. Sci.*, pages 97–120. Springer, September 1990.

[31] V. N. Redko. On defining relations for the algebra of regular events. *Ukrain. Mat. Z.*, 16:120–126, 1964. In Russian.

[32] Jacques Sakarovitch. Kleene's Theorem revisited: a formal path from Kleene to Chomsky. In A. Kelemenova and J. Keleman, editors, *Trends, Techniques, and Problems in Theoretical Computer Science*, volume 281 of *Lect. Notes in Computer Science*, pages 39–50. Springer, 1987.

[33] Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. Assoc. Comput. Mach.*, 13(1):158–169, January 1966.

[34] A. Tarski. On the calculus of relations. *J. Symb. Logic*, 6(3):65–106, 1941.