Equational Verification of Cache Blocking in LU Decomposition using Kleene Algebra with Tests

Adam Barth Dexter Kozen

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501, USA

June 13, 2002

Abstract

In a recent paper of Mateev et al. (2001), a new technique for program analysis called *fractal symbolic analysis* was introduced and applied to verify the correctness of a series of source-level transformations for cache blocking in LU decomposition with partial pivoting. It was argued in that paper that traditional techniques are inadequate because the transformations break definition-use dependencies. We show how the task can be accomplished purely equationally using Kleene algebra with tests.

1 Introduction

Kleene algebra (KA) is the algebra of regular expressions. It was first introduced by Kleene [9] and further developed by Conway [5]. Kleene algebra has appeared in one form or another in relational algebra, semantics and logics of programs, automata and formal language theory, and the design and analysis of algorithms. Many authors have contributed over the years to the development of the algebraic theory; see [11] and references therein.

Kleene algebra with tests (KAT), introduced in [11], combines programs and assertions in a purely equational system. Simply stated, a Kleene algebra with tests is a Kleene algebra with an embedded Boolean subalgebra. KAT strictly subsumes propositional Hoare Logic (PHL), is of no greater complexity than PHL, and is deductively complete over relational models (PHL is not) [14, 4, 12, 15]. KAT is less expressive than propositional Dynamic Logic (PDL) [7], but in the current state of complexity-theoretic knowledge[1] it is also less complex. Moreover, KAT requires nothing beyond classical equational logic, in contrast to PHL or PDL, which depend on a more complicated syntax involving partial correctness assertions or modalities.

KAT has been applied successfully in a number of low-level verification tasks involving communication protocols, basic safety analysis, concurrency control, and local

[1] specifically, unless $PSPACE = EXPTIME$

compiler optimizations [2, 3, 13]. A useful feature of KAT in this regard is its ability to accommodate certain basic equational assumptions regarding the interaction of atomic instructions. This feature makes KAT ideal for reasoning about the correctness of low-level code transformations.

In this paper we report on the use of KAT in a substantial compiler verification task. Mateev et al. [16] have described a series of source-level transformations for automatic cache blocking in LU decomposition with partial pivoting. These transformations are used primarily in large applications to enhance locality of reference. In attempting to verify the correctness of these transformations, Mateev et al. observed that the standard approach involving symbolic dependence analysis is inadequate. The major complication is that, although the transformations are semantically correct, they do not preserve definition-use dependencies. This led them to consider other approaches that exploit knowledge of the semantics of the basic operations. They proposed a new system called *fractal symbolic analysis*, in which programs are repeatedly simplified until symbolic analysis becomes feasible. The semantics is not preserved in the simplification process, but the equality of the simplified programs implies the equality of the original programs.

In this paper we demonstrate that the same verification task studied by Mateev et al. can be adequately handled by KAT in a purely equational way. The semantics of the underlying domain of computation are incorporated only as Boolean axioms, as in Hoare logic. The code transformations themselves are purely schematic. The atomic-level code transformations are instances of a small set of basic schematic rules governing the interaction of atomic programs and tests. These rules play roughly the same role as the assignment rule in Hoare Logic, but are more versatile. All other transformations are instances of theorems of KAT.

2 Kleene Algebra and Kleene Algebra with Tests

Kleene algebra was introduced by S. C. Kleene [9] (see also [5]). We define a *Kleene algebra* (KA) to be a structure $(K, +, \cdot, {}^*, 0, 1)$, where $(K, +, \cdot, 0, 1)$ is an idempotent semiring, $\mathsf{p}^*\mathsf{q}$ is the least solution to $\mathsf{q} + \mathsf{px} \le \mathsf{x}$, and qp^* the least solution to $\mathsf{q} + \mathsf{xp} \le \mathsf{x}$. Here "least" refers to the natural partial order $\mathsf{p} \le \mathsf{q} \leftrightarrow \mathsf{p} + \mathsf{q} = \mathsf{q}$. The operation $+$ gives the supremum with respect to $\le$. This particular axiomatization is from [10].

We normally omit the $\cdot$, writing pq for $\mathsf{p} \cdot \mathsf{q}$. The precedence of the operators is ${}^* > \cdot > +$. Thus $\mathsf{p} + \mathsf{qr}^*$ should be parsed $\mathsf{p} + (\mathsf{q}(\mathsf{r}^*))$.

Typical models include the family of regular sets of strings over a finite alphabet, the family of binary relations on a set, and the family of $n \times n$ matrices over another Kleene algebra.

The following are some elementary theorems of KA.

$$\mathsf{p}^* = 1 + \mathsf{pp}^* = 1 + \mathsf{p}^*\mathsf{p} = \mathsf{p}^*\mathsf{p}^* = \mathsf{p}^{**} \tag{1}$$

$$\mathsf{p}(\mathsf{qp})^* = (\mathsf{pq})^*\mathsf{p} \tag{2}$$

$$\mathsf{p}^*(\mathsf{qp}^*)^* = (\mathsf{p} + \mathsf{q})^* = (\mathsf{p}^*\mathsf{q})^*\mathsf{p}^* \tag{3}$$

$$\mathsf{qp} = 0 \ \rightarrow \ (\mathsf{p} + \mathsf{q})^* = \mathsf{p}^*\mathsf{q}^* \tag{4}$$

$$\mathsf{px} = \mathsf{xq} \ \rightarrow \ \mathsf{p}^*\mathsf{x} = \mathsf{xq}^* \tag{5}$$

The identities (2) and (3) are called the *sliding rule* and the *denesting rule*, respectively. These rules are particularly useful in program equivalence proofs. The property (5) is a kind of bisimulation property. It plays a prominent role in the completeness proof of [10]. We refer the reader to [10] for further definitions and basic results.

A *Kleene algebra with tests* (KAT) [11] is a Kleene algebra with an embedded Boolean subalgebra. More precisely, it is a two-sorted structure $(K,\ B,\ +,\ \cdot,\ ^*,\ ^-,\ 0,\ 1)$, where $^-$ is a unary operator defined only on B, such that $B\ \subseteq\ K$, $(K,\ +,\ \cdot,\ ^*,\ 0,\ 1)$ is a Kleene algebra, and $(B,\ +,\ \cdot,\ ^-,\ 0,\ 1)$ is a Boolean algebra. The elements of B are called *tests*. We reserve the letters $\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}, \ldots$ for arbitrary elements of K and $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ for tests.

When applied to arbitrary elements of K, the operators $+, \cdot, 0, 1$ refer to nondeterministic choice, composition, fail and skip, respectively. Applied to tests, they take on the additional meaning of Boolean disjunction, conjunction, falsity and truth, respectively. These two usages do not conflict; for example, sequentially testing b and c is the same as testing their conjunction bc.

The encoding of the while program constructs is as in PDL [7]. The conditional test **if** b **then** p **else** q and while loop **while** b **do** p are expressed as $\mathsf{bp} + \overline{\mathsf{b}}\mathsf{q}$ and $(\mathsf{bp})^*\overline{\mathsf{b}}$, respectively.

The propositional fragment of Hoare Logic is subsumed by KAT [12]. The Hoare partial correctness assertion $\{\mathsf{b}\}\ \mathsf{p}\ \{\mathsf{c}\}$ is expressed $\mathsf{bp}\overline{\mathsf{c}} = 0$, or equivalently, $\mathsf{bpc} = \mathsf{bp}$.

The following are some basic theorems of KAT.

$$\mathsf{bq} = \mathsf{qb} \quad \rightarrow \quad \mathsf{bq}^* \ = \ (\mathsf{bq})^*\mathsf{b} \ = \ \mathsf{q}^*\mathsf{b} \ = \ \mathsf{b}(\mathsf{qb})^* \tag{6}$$

$$\mathsf{bq} = \mathsf{bqb} \quad \rightarrow \quad \mathsf{bq}^* \ = \ (\mathsf{bq})^*\mathsf{b} \ = \ \mathsf{bq}^*\mathsf{b} \ = \ \mathsf{b}(\mathsf{qb})^* \tag{7}$$

$$\mathsf{bp} = \mathsf{pc} \quad \leftrightarrow \quad \overline{\mathsf{b}}\mathsf{p} = \mathsf{p}\overline{\mathsf{c}} \quad \leftrightarrow \quad \mathsf{bp}\overline{\mathsf{c}} + \overline{\mathsf{b}}\mathsf{pc} = 0. \tag{8}$$

A proof of (8) was given in [1]. See [11] for further definitions and basic results.

For applications in program verification, the standard interpretation would be a KA of binary relations on a set and the Boolean algebra of subsets of the identity relation.

2.1 Schematic Reasoning

KAT, so far described, is propositional. Programs and tests are interpreted over an abstract set of states, and programs are interpreted as abstract binary relations or traces. In applications, however, we must instantiate these constructs. This involves the introduction of symbols for variables, constants, functions, and relations ranging over a domain of computation. At this level, a state of the computation is typically taken to be a valuation of the program variables over the domain of computation, and state changes are effected by assignment statements $x := e$, where x is an individual program variable and e is a term. This extension is called SKAT (for *schematic* KAT) and was investigated in [1].

To avoid confusion when reasoning at this level, we use the symbol $\equiv$ for equality between programs (equality in KAT), $=$ for equality in the underlying domain, $:=$ for assignment, $\oplus$ for addition in KAT, and $+$ for addition in the underlying domain. (When reasoning purely propositionally, we will continue to use the symbols $=$ and $+$ for equality and addition in KAT.)

Many general properties can already be derived at the schematological level, without specifying a particular interpretation of these new symbols. For example, consider the following identities:

$$x := s\,;\, y := t \quad\equiv\quad y := t[x/s]\,;\, x := s \qquad (y \notin \mathrm{FV}(s)) \tag{9}$$

$$x := s\,;\, y := t \quad\equiv\quad x := s\,;\, y := t[x/s] \qquad (x \notin \mathrm{FV}(s)) \tag{10}$$

$$x := s\,;\, x := t \quad\equiv\quad x := t[x/s] \tag{11}$$

$$\varphi[x/t]\,;\, x := t \quad\equiv\quad x := t\,;\, \varphi \tag{12}$$

where in (9) and (10), x and y are distinct variables and $\mathrm{FV}(s)$ denotes the set of variables occurring in s. Special cases of (9) and (12) are the commutativity conditions

$$x := s\,;\, y := t \quad\equiv\quad y := t\,;\, x := s \qquad (x \notin \mathrm{FV}(t),\ y \notin \mathrm{FV}(s)) \tag{13}$$

$$\varphi\,;\, x := t \quad\equiv\quad x := t\,;\, \varphi \qquad (x \notin \mathrm{FV}(\varphi)) \tag{14}$$

The soundness of these identities over all schematic interpretations was proved in [1]. Another valid identity not considered in [1] is

$$x := x \quad\equiv\quad 1. \tag{15}$$

All of the properties (9)–(15) make sense even in the absence of equality. When equality is present in the language, we may insert any valid formula of the theory of equality. The following property is also valid:

$$s = t\,;\, x := s \quad\equiv\quad s = t\,;\, x := t. \tag{16}$$

It follows from (15) and (16) that

$$x = t\,;\, x := t \quad\equiv\quad x = t. \tag{17}$$

One can also show that (10) is a consequence of (12) and (16).

In traditional Hoare logic, atomic programs are assignments $x := t$ and the only atomic assumption is the *assignment rule*

$$\{\varphi[x/t]\}\, x := t\, \{\varphi\},$$

which would be represented in KAT by either of the two equivalent equations

$$\begin{aligned} \varphi[x/t]\,;\, x := t\,;\, \varphi \quad&\equiv\quad \varphi[x/t]\,;\, x := t \\ \varphi[x/t]\,;\, x := t\,;\, \neg\varphi \quad&\equiv\quad 0. \end{aligned}$$

These equations follow from (12). In fact, (12) is actually equivalent to two applications of the Hoare assignment rule, one for φ and one for its negation. This can be seen by taking b, c, and p to be $\varphi[x/t]$, φ, and $x := t$, respectively, in (8).

To illustrate the use of (9)–(14), consider the equation $\mathsf{pqr} = \mathsf{rqp}$, where p, q, and r are the assignments $y := x$, $y := 2 * y$, and $x := 2 * x$, respectively. This example was used in [16] to illustrate a simple transformation that is sound, yet breaks definition-use dependencies. Here is an equational proof in KAT:

$$\begin{aligned} \mathsf{pqr} \quad&\equiv\quad y := x\,;\, y := 2 * y\,;\, x := 2 * x \\ &\equiv\quad y := 2 * x\,;\, x := 2 * x && \text{by (11)} \\ &\equiv\quad x := 2 * x\,;\, y := x && \text{by (9)} \\ &\equiv\quad x := 2 * x\,;\, y := 2 * y\,;\, y := x && \text{by (11)} \\ &\equiv\quad \mathsf{rqp}. \end{aligned}$$

2.2 Interpreted Reasoning

In specific applications, we are not constrained to reason purely schematically. We may take advantage of the fact that we are reasoning with respect to a particular interpretation or class of interpretations over some underlying domain or class of domains, and deduction is relative to the theory of that class of interpretations. This theory determines the Boolean algebra of the KAT in which we work. A valid assertion φ takes the form of an equation $\varphi = 1$ in KAT, which may be taken as an extra axiom for deductive purposes.

In the application considered in this paper, variables are interpreted as integers or reals, + is interpreted as addition, etc. Valid number-theoretic properties such as $i \leq j \leftrightarrow i < j + 1$ can be introduced as needed.

Here is an example of a property that can be derived at this level involving **for** loops. If i and n are integer variables and p and $i < n$ commute (for example, if p does not assign to i or n), then

$$i \leq n\ ;\ (i < n\ ;\ \mathsf{p}\ ;\ i\text{++})^* \quad \equiv \quad (i < n\ ;\ \mathsf{p}\ ;\ i\text{++})^*\ ;\ i \leq n. \tag{18}$$

Here i++ is an abbreviation for $i := i + 1$. Letting

$$\mathsf{c} \stackrel{\text{def}}{=} i < n \qquad \mathsf{d} \stackrel{\text{def}}{=} i \leq n \qquad \mathsf{q} \stackrel{\text{def}}{=} \mathsf{p}\ ;\ i\text{++},$$

from (12) and number theory we have $\mathsf{c} \leq \mathsf{d}$ and $\mathsf{cq} = \mathsf{qd}$. Once we have established these premises that are particular to the interpretation, we can reason purely propositionally in KAT to obtain

$$\mathsf{c} \leq \mathsf{d}\ \wedge\ \mathsf{cq} = \mathsf{qd} \quad \rightarrow \quad \mathsf{d}(\mathsf{cq})^* = (\mathsf{cq})^*\mathsf{d},$$

the conclusion of which is (18).

2.3 Arrays

For our application, we will need to extend (9)–(17) to handle arrays. Theories of arrays have been considered by several authors, among them [18, 19, 6, 17]. Care must be taken because of the possibility of aliasing.

Semantically, an array variable A is interpreted by a valuation as a map $\mathcal{D} \rightarrow \mathcal{D}$, where $\mathcal{D}$ is the domain of computation (see [8]). An array assignment $A(s) := t$ maps valuation θ to θ', where

$$\begin{aligned}
\theta'(A)(\theta(s)) \quad &= \quad \theta(t) \\
\theta'(A)(a) \quad &= \quad \theta(A)(a),\ a \neq \theta(s) \\
\theta'(x) \quad &= \quad \theta(x),\ x \text{ any array or individual variable not equal to } A.
\end{aligned}$$

Some of the extensions we will need are sound without any restrictions, such as

$$s = t\ ;\ u = v\ ;\ A(s) := u \quad \equiv \quad s = t\ ;\ u = v\ ;\ A(t) := v \tag{19}$$

However, other generalizations which may seem obvious at first glance turn out to be unsound. For example,

$$A(s) := t \quad \equiv \quad A(s) := t\ ;\ A(s) = t$$

is not true in general, even if t is a constant. Over $\mathbb{N}$, if s is $A(2)$ and t is 3, and the assignment is executed in a state in which $A(2) = A(3) = 2$, then the value of t after the assignment is still 3, but the value of $A(s)$ is 2.

Most of the properties we will need are consequences of the following metatheorem, which allows us to transfer properties without arrays to properties with arrays. Define an expression to be *simple* if it contains no array symbol.

Theorem 2.1 *Let V be a finite set of individual variables and let A be an array variable. For each $x \in V$, let i_x be a simple term. Suppose $\mathsf{p} = \mathsf{q}$ is a valid equation such that neither p nor q contains an occurrence of A or an assignment to any variable in i_x, $x \in V$. Then the following is also a valid equation:*

$$\bigwedge_{\substack{x,y\in V\\ x\neq y}} i_x \neq i_y \,;\, \mathsf{p}[x/A(i_x) \mid x \in V] \quad = \quad \bigwedge_{\substack{x,y\in V\\ x\neq y}} i_x \neq i_y \,;\, \mathsf{q}[x/A(i_x) \mid x \in V].$$

Proof. For any expression e, abbreviate $e[x/A(i_x) \mid x \in V]$ by e'. The atomic instructions of p' and q' are all of the form $y := t'$ or $A(i_y) := t'$. Because the i_x are simple and neither p nor q (therefore neither p' nor q') assign to any variable of i_x, none of these atomic instructions can change the value of i_x. Thus the atomic instructions of p' and q' commute with the precondition $\bigwedge_{x,y\in V,\, x\neq y} i_x \neq i_y$. By an inductive argument involving (6), so do all subprograms of p' and q'. Thus the $A(i_x)$, $x \in V$, behave like fixed and distinct individual variables throughout the computation of p' and q'. $\square$

Applying Theorem 2.1 to the axioms (9)–(17) of SKAT gives corresponding axioms that apply to arrays. For example, the following conditions are consequences of Theorem 2.1 applied to (9). These equations hold under the assumptions of Theorem 2.1 and in the presence of the implicit precondition $i_x \neq i_y$. For any expression e, let e_x and e_{xy} abbreviate $e[x/A(i_x)]$ and $e[x/A(i_x), y/A(i_y)]$, respectively.

$$A(i_x) := s_x \,;\, A(i_y) := t_{xy} \quad\equiv\quad A(i_y) := t_y[x/s_x] \,;\, A(i_x) := s_x \tag{20}$$
$$x := s \,;\, A(i_y) := t_y \quad\equiv\quad A(i_y) := t_y[x/s] \,;\, x := s \tag{21}$$
$$A(i_x) := s_x \,;\, y := t_x \quad\equiv\quad y := t[x/s_x] \,;\, A(i_x) := s_x \tag{22}$$

where $y \notin \mathrm{FV}(s)$. Other axioms for arrays obtained similarly are

$$A(i_x) := s_y \,;\, A(i_y) := t_{xy} \quad\equiv\quad A(i_x) := s_y \,;\, A(i_y) := t_y[x/s_y] \tag{23}$$
$$A(i_x) := s_x \,;\, A(i_x) := t_x \quad\equiv\quad A(i_x) := t[x/s_x] \tag{24}$$
$$\varphi[y/t_y] \,;\, A(i_y) := t_y \quad\equiv\quad A(i_y) := t_y \,;\, \varphi_y \tag{25}$$

where $x \notin \mathrm{FV}(s)$ in (23).

More general versions hold, but these are sufficient for our purposes, so we leave a more thorough analysis for future work. We will take these conditions as axioms when reasoning in the presence of arrays.

```
do j = 1,N-1
  B1(j):  //swap
    tmp = A(j);
    A(j) = A(j+1);
    A(j+1) = tmp;
  B2(j):  //update
    do i = j+1,N
      A(i) = A(i)/A(j)
```

(a) Original Program

```
do j = 1,N-1
  B1(j):  //swap
    tmp = A(j);
    A(j) = A(j+1);
    A(j+1) = tmp;
do j = 1,N-1
  B2(j):  //update
    do i = j+1,N
      A(i) = A(i)/A(j);
```

(b) Transformed Program

Figure 1: Loop Distribution Example from [16]

3 Loop Distribution—A Simplified Example

The transformation shown in Figure 1 is from [16]. This is a simplified version of LU factorization with partial pivoting used to illustrate various aspects of their technique. As they describe it:

> The source program of Figure 1(a) traverses an array A; at the j^{th} iteration, it swaps elements $A(j)$ and $A(j+1)$, and updates all the elements from $A(j+1)$ through $A(N)$ using the new value in $A(j)$. This is a much simplified version of LU factorization with partial pivoting in which entire rows of a matrix are swapped and entire submatrices are updated at each step . . .
>
> Loop distribution transforms this program into the one shown in Figure 1(b). In this program, all the swaps are done first, and then all the updates are done together. This transformation is useful because the second loop nest is perfectly nested and can be tiled to get good locality of reference. Are these programs equal?
>
> Dependence analysis requires that there not be a dependence from an instance $B2(j_2)$ to an instance $B1(j_1)$ where $j_1 > j_2$. Unfortunately, this condition is violated: instance $B2(j_0)$ writes to location $A(j_0+1)$, and instance $B1(j_0+1)$ reads from it. Symbolic analysis of these programs on the other hand is too difficult. [16]

The remainder of this article is devoted to giving a formal, purely equational proof of equivalence using KAT.

Let $\mathsf{swap}(j)$ denote the three-line subprogram labeled $B1(j)$ and let $\mathsf{update}(j)$ denote the two-line subprogram labeled $B2(j)$ in Figure 1. Let $\mathsf{u}(i,j)$ denote the array assignment $A(i) := A(i)/A(j)$. Expressed in the language of KAT,

$$\begin{aligned}\mathsf{swap}(j) \quad &\equiv \quad tmp := A(j)\ ;\ A(j) := A(j+1)\ ;\ A(j+1) := tmp\\ \mathsf{update}(j) \quad &\equiv \quad i := j+1\ ;\ (i \le N\ ;\ \mathsf{u}(i,j)\ ;\ i\text{++})^*\ ;\ i > N,\end{aligned}$$

and the programs of Figure 1(a) and (b) are

$$j := 1\,;\ (j < N\,;\ \mathsf{swap}(j)\,;\ \mathsf{update}(j)\,;\ j\text{++})^*\,;\ j \geq N \tag{26}$$

$$\begin{array}{l} j := 1\,;\ (j < N\,;\ \mathsf{swap}(j)\,;\ j\text{++})^*\,;\ j \geq N\,; \\ j := 1\,;\ (j < N\,;\ \mathsf{update}(j)\,;\ j\text{++})^*\,;\ j \geq N, \end{array} \tag{27}$$

respectively. We wish to show that (26) and (27) are equivalent.

Lemma 3.1 *Let a, b, t be distinct variables. Let $f(x)$ be a term with a variable x but no occurrence of a, b, or t. The following two schemes are equivalent:*

$$a := f(a)\,;\ b := f(b)\,;\ t := a\,;\ a := b\,;\ b := t\,;\ t := \bot \tag{28}$$

$$t := a\,;\ a := b\,;\ b := t\,;\ t := \bot\,;\ a := f(a)\,;\ b := f(b). \tag{29}$$

Proof. Starting from (28), we can move $b := f(b)$ right past the next two assignments using (13) and (9), then annihilate it using (11) to obtain

$$a := f(a)\,;\ t := a\,;\ a := f(b)\,;\ b := t\,;\ t := \bot.$$

Similarly, we can move $a := f(a)$ right past the next assignment using (9), then annihilate it using (11) to obtain

$$t := f(a)\,;\ a := f(b)\,;\ b := t\,;\ t := \bot.$$

Applying (11) in the right-to-left direction to $t := f(a)$, we obtain

$$t := a\,;\ t := f(t)\,;\ a := f(b)\,;\ b := t\,;\ t := \bot.$$

Now we can move $t := f(t)$ right past the next two assignments using (13) and (9), then annihilate it using (11) to obtain

$$t := a\,;\ a := f(b)\,;\ b := f(t)\,;\ t := \bot. \tag{30}$$

Starting from (29), we can apply (13) three times to move $t := \bot$ all the way to the right and exchange $b := t$ and $a := f(a)$ to obtain

$$t := a\,;\ a := b\,;\ a := f(a)\,;\ b := t\,;\ b := f(b)\,;\ t := \bot.$$

Now two independent applications of (11) yield (30). ☐

Lemma 3.2 *Let j, k, N be distinct variables. The following programs are equivalent:*

$$j < k < N\,;\ \mathsf{update}(j)\,;\ \mathsf{swap}(k) \tag{31}$$

$$j < k < N\,;\ \mathsf{swap}(k)\,;\ \mathsf{update}(j). \tag{32}$$

Proof. Let q abbreviate the program $\mathsf{u}(i,j)\,;\,i{+}{+}$ and let w abbreviate the program $i := j+1$. First we show that under the precondition $j<k<N$, we can decompose $\mathsf{update}(j)$ as follows:

$$\begin{aligned}
& j<k<N\,;\,\mathsf{update}(j) \\
&\quad\equiv\quad j<k<N\,;\,\mathsf{w}\,;\,(i<k\,;\,\mathsf{q})^*\,; \\
&\qquad\qquad i=k\,;\,\mathsf{u}(k,j)\,;\,\mathsf{u}(k+1,j)\,; \\
&\qquad\qquad i:=k+2\,;\,(i\le N\,;\,\mathsf{q})^*\,;\,i>N.
\end{aligned} \tag{33}$$

To see this, first note that

$$\begin{aligned}
& \mathsf{update}(j) \\
&\quad\equiv\quad \mathsf{w}\,;\,(i\le N\,;\,\mathsf{q})^*\,;\,i>N \\
&\quad\equiv\quad \mathsf{w}\,;\,(i\le N\,;\,(i<k\oplus i\ge k)\,;\,\mathsf{q})^*\,;\,i>N \\
&\quad\equiv\quad \mathsf{w}\,;\,((i\le N\,;\,i<k\,;\,\mathsf{q})\oplus(i\le N\,;\,i\ge k\,;\,\mathsf{q}))^*\,;\,i>N \\
&\quad\equiv\quad \mathsf{w}\,;\,(i\le N\,;\,i<k\,;\,\mathsf{q})^*\,;\,(i\le N\,;\,i\ge k\,;\,\mathsf{q})^*\,;\,i>N.
\end{aligned} \tag{34}$$

The last step follows from (4). The precondition of (4) is

$$i\le N\,;\,i\ge k\,;\,\mathsf{q}\,;\,i\le N\,;\,i<k\,;\,\mathsf{q}\quad\equiv\quad 0. \tag{35}$$

To see (35), note that $i\ge k$ and $\mathsf{u}(i,j)$ commute by (14) (amended by Theorem 2.1 to handle arrays), and $i<k$ and $i\le N$ commute by Boolean algebra. It thus suffices to show $i\ge k\,;\,i{+}{+}\,;\,i<k\equiv 0$. But this is immediate from (12) and number theory.

By Boolean algebra and (12), we have

$$j<k<N\,;\,\mathsf{w}\quad\equiv\quad j<k<N\,;\,\mathsf{w}\,;\,i\le k<N,$$

thus by (34),

$$\begin{aligned}
& j<k<N\,;\,\mathsf{update}(j) \\
&\quad\equiv\quad j<k<N\,;\,\mathsf{w}\,;\,i\le k<N\,; \\
&\qquad\qquad (i\le N\,;\,i<k\,;\,\mathsf{q})^*\,;\,(i\le N\,;\,i\ge k\,;\,\mathsf{q})^*\,; \\
&\qquad\qquad i>N.
\end{aligned} \tag{36}$$

Since $k<N$ and $i<k$ imply $i\le N$, by (6) we have

$$i\le k<N\,;\,(i\le N\,;\,i<k\,;\,\mathsf{q})^*\quad\equiv\quad i\le k<N\,;\,(i<k\,;\,\mathsf{q})^*.$$

By (6) and (18), this is equivalent to

$$i\le k<N\,;\,(i<k\,;\,\mathsf{q})^*\,;\,i\le k<N. \tag{37}$$

We also have

$$\begin{aligned}
& i<k<N\,;\,(i\le N\,;\,i\ge k\,;\,\mathsf{q})^*\,;\,i>N \\
&\quad\equiv\quad i<k<N\,;\,i\le N\,;\,i\ge k\,;\,\mathsf{q}\,;\,(i\le N\,;\,i\ge k\,;\,\mathsf{q})^*\,;\,i>N \\
&\qquad\quad \oplus\, i<k<N\,;\,i>N \\
&\quad\equiv\quad 0,
\end{aligned}$$

therefore by (6),

$$\begin{aligned} & i \le k < N \,;\, (i \le N \,;\, i \ge k \,;\, \mathsf{q})^* \,;\, i > N \\ & \quad \equiv \quad i = k < N \,;\, (i \le N \,;\, i \ge k \,;\, \mathsf{q})^* \,;\, i > N \\ & \quad \equiv \quad i = k < N \,;\, (i \le N \,;\, \mathsf{q})^* \,;\, i > N. \end{aligned} \tag{38}$$

Combining (36), (37), and (38), we have

$$\begin{aligned} & j < k < N \,;\, \mathsf{update}(j) \\ & \quad \equiv \quad j < k < N \,;\, \mathsf{w} \,; \\ & \qquad\quad i \le k < N \,;\, (i < k \,;\, \mathsf{q})^* \,; \\ & \qquad\quad i = k < N \,;\, (i \le N \,;\, \mathsf{q})^* \,;\, i > N. \end{aligned} \tag{39}$$

Let $\mathsf{s} = i \le N \,;\, \mathsf{q}$, the body of the last loop in (39). Unrolling this loop twice, the last line of (39) becomes

$$\begin{aligned} & i = k < N \,;\, \mathsf{s}^* \,;\, i > N \\ & \quad \equiv \quad i = k < N \,;\, i > N \;\oplus\; i = k < N \,;\, \mathsf{s} \,;\, i > N \;\oplus\; i = k < N \,;\, \mathsf{s} \,;\, \mathsf{s} \,;\, \mathsf{s}^* \,;\, i > N \end{aligned}$$

and the first two terms vanish. Also, an elementary argument using (12), (16), and number theory yields

$$i = k < N \,;\, \mathsf{s} \,;\, \mathsf{s} \quad \equiv \quad i = k < N \,;\, \mathsf{u}(k,j) \,;\, \mathsf{u}(k+1,j) \,;\, i := k+2.$$

Combining these observations with (39) gives

$$\begin{aligned} & j < k < N \,;\, \mathsf{update}(j) \\ & \quad \equiv \quad j < k < N \,;\, \mathsf{w} \,; \\ & \qquad\quad i \le k < N \,;\, (i < k \,;\, \mathsf{q})^* \,; \\ & \qquad\quad i = k < N \,;\, \mathsf{u}(k,j) \,;\, \mathsf{u}(k+1,j) \,;\, i := k+2 \,;\, \mathsf{s}^* \,;\, i > N. \end{aligned} \tag{40}$$

Thus the program (31) is equivalent to

$$\begin{aligned} & j < k < N \,; \\ & \mathsf{w} \,;\, i \le k < N \,;\, (i < k \,;\, \mathsf{q})^* \,; \\ & i = k < N \,;\, \mathsf{u}(k,j) \,;\, \mathsf{u}(k+1,j) \,; \\ & i := k+2 \,;\, (i \le N \,;\, \mathsf{q})^* \,;\, i > N \,; \\ & \mathsf{swap}(k), \end{aligned}$$

and $\mathsf{swap}(k)$ commutes with each of the three lines above it by (13), Lemma 3.1, and (13), respectively (amended by Theorem 2.1 to handle arrays). The final result is (32). □

Lemma 3.3 *Let j, k, N be distinct variables. The following programs are equivalent:*

$$j < k \,;\, j < N \,;\, \mathsf{update}(j) \,;\, j\text{++} \,;\, k < N \,;\, \mathsf{swap}(k) \,;\, k\text{++} \tag{41}$$

$$j < k \,;\, k < N \,;\, \mathsf{swap}(k) \,;\, k\text{++} \,;\, j < N \,;\, \mathsf{update}(j) \,;\, j\text{++}. \tag{42}$$

Proof. By (13) and (14), these programs are equivalent to

$$j < k\ ;\ k < N\ ;\ j < N\ ;\ \mathsf{update}(j)\ ;\ \mathsf{swap}(k)\ ;\ k{+}{+}\ ;\ j{+}{+}$$

$$j < k\ ;\ k < N\ ;\ j < N\ ;\ \mathsf{swap}(k)\ ;\ \mathsf{update}(j)\ ;\ k{+}{+}\ ;\ j{+}{+},$$

respectively. The equivalence of these two programs follows immediately from Lemma 3.2. $\square$

Lemma 3.4 *Let* p, q *be program symbols and* $\mathsf{e}, \mathsf{c}, \mathsf{d}$ *test symbols. Under the assumptions*

$$\begin{aligned} \mathsf{epq} &= \mathsf{pqe} && (43)\\ \mathsf{dp} &= \mathsf{pd} && (44)\\ \mathsf{cq} &= \mathsf{qc} && (45)\\ \mathsf{ec} &= \mathsf{ed} && (46)\\ \overline{\mathsf{c}}\mathsf{p} &= 0 && (47)\\ \overline{\mathsf{d}}\mathsf{q} &= 0, && (48) \end{aligned}$$

the following equation holds:

$$\mathsf{e(pq)}^*\overline{\mathsf{c}}\overline{\mathsf{d}} \quad = \quad \mathsf{e(pq)}^*(\mathsf{p}^* + \mathsf{q}^*)\overline{\mathsf{c}}\overline{\mathsf{d}}. \tag{49}$$

Proof. It follows from (46) and Boolean algebra that $\mathsf{e}\overline{\mathsf{c}} = \mathsf{e}\overline{\mathsf{d}}$. Then

$$\begin{aligned} \mathsf{e(pq)^*q^*}\overline{\mathsf{c}} &= \mathsf{(pq)^*e(1+qq^*)}\overline{\mathsf{c}} && \text{by (43) and (6)}\\ &= \mathsf{(pq)^*(e}\overline{\mathsf{c}} + \mathsf{eqq^*}\overline{\mathsf{c}}) &&\\ &= \mathsf{(pq)^*(e}\overline{\mathsf{c}} + \mathsf{e}\overline{\mathsf{c}}\mathsf{qq^*}) && \text{by (45), using (6) and (8)}\\ &= \mathsf{(pq)^*(e}\overline{\mathsf{c}} + \mathsf{e}\overline{\mathsf{d}}\mathsf{qq^*}) &&\\ &= \mathsf{(pq)^*e}\overline{\mathsf{c}} && \text{by (48)}\\ &= \mathsf{e(pq)^*}\overline{\mathsf{c}} && \text{by (43) and (6).} \end{aligned}$$

A symmetric argument using (44) and (47) shows that $\mathsf{e(pq)^*p^*}\overline{\mathsf{d}} = \mathsf{e(pq)^*}\overline{\mathsf{d}}$. The equation (49) follows immediately from these two equations. $\square$

Lemma 3.5 *Let* p, q *be program symbols and* a, b *test symbols. Under the assumptions*

$$\begin{aligned} \mathsf{bp} &= \mathsf{pa} && (50)\\ \mathsf{aq} &= \mathsf{qb} && (51)\\ \mathsf{ap} &= \mathsf{apa} && (52)\\ \mathsf{apq} &= \mathsf{aqp}, && (53) \end{aligned}$$

the following equations hold:

$$\begin{aligned} \mathsf{ap^*q} &= \mathsf{aqp^*} && (54)\\ \mathsf{a(qp)^*(p^* + q^*)} &= \mathsf{ap^*q^*} && (55)\\ \mathsf{b(pq)^*(p^* + q^*)} &= \mathsf{bp^*q^*}. && (56) \end{aligned}$$

Proof. We first show that (54) follows from (52) and (53). For the direction $\leq$, using (53) we have

$$\mathsf{aq} + \mathsf{apaqp}^* \;\leq\; \mathsf{aq} + \mathsf{apqp}^* \;=\; \mathsf{aq} + \mathsf{aqpp}^* \;=\; \mathsf{aqp}^*,$$

therefore $(\mathsf{ap})^*\mathsf{aq} \leq \mathsf{aqp}^*$ by an axiom of Kleene algebra. Then by (52), (7), and (2),

$$\mathsf{ap}^*\mathsf{q} \;=\; \mathsf{a}(\mathsf{pa})^*\mathsf{q} \;=\; (\mathsf{ap})^*\mathsf{aq} \;\leq \mathsf{aqp}^*.$$

For the reverse inequality, by (7) and (53),

$$\mathsf{aq} + \mathsf{ap}^*\mathsf{qp} \;=\; \mathsf{aq} + \mathsf{ap}^*\mathsf{aqp} \;=\; \mathsf{aq} + \mathsf{ap}^*\mathsf{apq} \;=\; \mathsf{aq} + \mathsf{ap}^*\mathsf{pq} \;=\; \mathsf{ap}^*\mathsf{q},$$

therefore $\mathsf{aqp}^* \leq \mathsf{ap}^*\mathsf{q}$ by an axiom of Kleene algebra.

It follows from (50) and (51) using (6) that $\mathsf{aqp} = \mathsf{qpa}$ and $\mathsf{a}(\mathsf{qp})^* = (\mathsf{qp})^*\mathsf{a}$. Also, by (52) and (7), $\mathsf{ap}^* = \mathsf{ap}^*\mathsf{a}$.

For the direction $\leq$ of (55), we must show

$$\mathsf{a}(\mathsf{qp})^*\mathsf{p}^* \quad\leq\quad \mathsf{ap}^*\mathsf{q}^* \tag{57}$$

$$\mathsf{a}(\mathsf{qp})^*\mathsf{q}^* \quad\leq\quad \mathsf{ap}^*\mathsf{q}^*. \tag{58}$$

For (57), by (53), (52), and (54), we have

$$\begin{aligned}\mathsf{ap}^* + \mathsf{qpap}^*\mathsf{q}^* \quad&=\quad \mathsf{ap}^* + \mathsf{aqpp}^*\mathsf{q}^* \;=\; \mathsf{ap}^* + \mathsf{apqp}^*\mathsf{q}^* \\ &=\quad \mathsf{ap}^* + \mathsf{apaqp}^*\mathsf{q}^* \;=\; \mathsf{ap}^* + \mathsf{apap}^*\mathsf{qq}^* \;\leq\; \mathsf{ap}^*\mathsf{q}^*.\end{aligned}$$

By (6) and an axiom of Kleene algebra,

$$\mathsf{a}(\mathsf{qp})^*\mathsf{p}^* \;=\; (\mathsf{qp})^*\mathsf{ap}^* \;\leq\; \mathsf{ap}^*\mathsf{q}^*.$$

This is (57). Equation (58) follows, since

$$\mathsf{a}(\mathsf{qp})^*\mathsf{q}^* \;\leq\; \mathsf{a}(\mathsf{qp})^*\mathsf{p}^*\mathsf{q}^* \;\leq\; \mathsf{ap}^*\mathsf{q}^*\mathsf{q}^* \;\leq\; \mathsf{ap}^*\mathsf{q}^*.$$

For the direction $\geq$ of (55), by an axiom of Kleene algebra it suffices to show

$$\mathsf{ap}^* + \mathsf{a}(\mathsf{qp})^*(\mathsf{p}^* + \mathsf{q}^*)\mathsf{q} \quad\leq\quad \mathsf{a}(\mathsf{qp})^*(\mathsf{p}^* + \mathsf{q}^*).$$

This follows from the four inequalities

$$\begin{aligned}\mathsf{ap}^* \quad&\leq\quad \mathsf{a}(\mathsf{qp})^*(\mathsf{p}^* + \mathsf{q}^*) \\ \mathsf{a}(\mathsf{qp})^*\mathsf{q} \quad&\leq\quad \mathsf{a}(\mathsf{qp})^*(\mathsf{p}^* + \mathsf{q}^*) \\ \mathsf{a}(\mathsf{qp})^*\mathsf{pp}^*\mathsf{q} \quad&\leq\quad \mathsf{a}(\mathsf{qp})^*(\mathsf{p}^* + \mathsf{q}^*) \\ \mathsf{a}(\mathsf{qp})^*\mathsf{q}^*\mathsf{q} \quad&\leq\quad \mathsf{a}(\mathsf{qp})^*(\mathsf{p}^* + \mathsf{q}^*),\end{aligned}$$

of which all but the third are obvious. For the third, we use (6), (52), (53), and (54):

$$\begin{aligned}\mathsf{a}(\mathsf{qp})^*\mathsf{pp}^*\mathsf{q} \quad&=\quad (\mathsf{qp})^*\mathsf{app}^*\mathsf{q} \;=\; (\mathsf{qp})^*\mathsf{apap}^*\mathsf{q} \;=\; (\mathsf{qp})^*\mathsf{apaqp}^* \\ &=\quad (\mathsf{qp})^*\mathsf{apqp}^* \;=\; (\mathsf{qp})^*\mathsf{aqpp}^* \;=\; \mathsf{a}(\mathsf{qp})^*\mathsf{qpp}^* \;\leq\; \mathsf{a}(\mathsf{qp})^*\mathsf{p}^*.\end{aligned}$$

Finally, for (56), we have by (2), (6), (54), and (55) that

$$\begin{aligned}\mathsf{aq(pq)^*(p^* + q^*)} &= \mathsf{a(qp)^*q(p^* + q^*)} \;=\; \mathsf{(qp)^*aq(p^* + q^*)} \\ &= \mathsf{(qp)^*a(p^* + q^*)q} \;=\; \mathsf{a(qp)^*(p^* + q^*)q} \;=\; \mathsf{ap^*q^*q},\end{aligned}$$

thus by (50),

$$\mathsf{bpq(pq)^*(p^* + q^*)} \;=\; \mathsf{paq(pq)^*(p^* + q^*)} \;=\; \mathsf{pap^*qq^*} \;=\; \mathsf{bpp^*qq^*}.$$

It follows that

$$\begin{aligned}\mathsf{b(pq)^*(p^* + q^*)} &= \mathsf{bp^* + bq^* + bpq(pq)^*(p^* + q^*)} \\ &= \mathsf{bp^* + bq^* + bpp^*qq^*} \\ &= \mathsf{bp^*q^*}.\end{aligned}$$

□

We are now ready to prove our main theorem.

Theorem 3.6 *The following two programs, with $k := \bot$ implicitly appended, are equivalent:*

$$j := 1\,;\; (j < N\,;\; \mathsf{swap}(j)\,;\; \mathsf{update}(j)\,;\; j{+}{+})^*\,;\; j \geq N \tag{59}$$

$$\begin{aligned}&k := 1\,;\; (k < N\,;\; \mathsf{swap}(k)\,;\; k{+}{+})^*\,;\; k \geq N\,; \\ &j := 1\,;\; (j < N\,;\; \mathsf{update}(j)\,;\; j{+}{+})^*\,;\; j \geq N.\end{aligned} \tag{60}$$

Proof. Under the abbreviations

$$\begin{array}{lcllcl} \mathsf{a} & \stackrel{\text{def}}{=} & j < k & \mathsf{p} & \stackrel{\text{def}}{=} & k < N\,;\; \mathsf{swap}(k)\,;\; k{+}{+} \\ \mathsf{b} & \stackrel{\text{def}}{=} & j \leq k & \mathsf{q} & \stackrel{\text{def}}{=} & j < N\,;\; \mathsf{update}(j)\,;\; j{+}{+} \\ \mathsf{c} & \stackrel{\text{def}}{=} & k < N & \mathsf{r} & \stackrel{\text{def}}{=} & k := 1 \\ \mathsf{d} & \stackrel{\text{def}}{=} & j < N & \mathsf{s} & \stackrel{\text{def}}{=} & j := 1, \\ \mathsf{e} & \stackrel{\text{def}}{=} & j = k & & & \end{array}$$

all the premises (43)–(48) and (50)–(53) of Lemmas 3.4 and 3.5 hold. These facts are all immediate except (53), which is Lemma 3.3. Program (59) is

$$\mathsf{s}\,;\; (\mathsf{d}\,;\; \mathsf{swap}(j)\,;\; \mathsf{update}(j)\,;\; j{+}{+})^*\,;\; \overline{\mathsf{d}}.$$

Because $k := \bot$ is implicitly appended, by [1, Lemma 4.5], this is equivalent to

$$\mathsf{rs}\,;\; (\mathsf{d}\,;\; \mathsf{swap}(j)\,;\; \mathsf{update}(j)\,;\; k := j + 1\,;\; j{+}{+})^*\,;\; \overline{\mathsf{d}}.$$

By two applications of (12), the assignments $k := 1\,;\; j := 1$ establish the property $j = k$; thus $\mathsf{rs} = \mathsf{rse}$. Since e commutes with $\mathsf{d}\,;\; \mathsf{swap}(j)\,;\; \mathsf{update}(j)$ and with $k := j+1\,;\; j{+}{+}$, by (6) we obtain

$$\begin{aligned}&\mathsf{rse}\,;\; (\mathsf{de}\,;\; \mathsf{swap}(j)\,;\; \mathsf{update}(j)\,;\; \mathsf{e}\,;\; k := j + 1\,;\; j{+}{+})^*\,;\; \overline{\mathsf{d}}\mathsf{e} \\ \equiv\;\; &\mathsf{rse}\,;\; (\mathsf{cde}\,;\; \mathsf{swap}(j)\,;\; \mathsf{update}(j)\,;\; \mathsf{e}\,;\; k := j + 1\,;\; j{+}{+})^*\,;\; \overline{\mathsf{c}}\overline{\mathsf{d}}\mathsf{e}.\end{aligned}$$

By two applications of (16), this is equivalent to

$$\mathsf{rse}\ ;\ (\mathsf{cde}\ ;\ \mathsf{swap}(k)\ ;\ \mathsf{update}(j)\ ;\ \mathsf{e}\ ;\ k\text{++}\ ;\ j\text{++})^*\ ;\ \overline{\mathsf{c}}\overline{\mathsf{d}}\mathsf{e}.$$

Removing e again by (6), then using commutativity, this is equivalent to

$$\begin{aligned} & \mathsf{rs}\ ;\ (\mathsf{c}\ ;\ \mathsf{swap}(k)\ ;\ k\text{++}\ ;\ \mathsf{d}\ ;\ \mathsf{update}(j)\ ;\ j\text{++})^*\ ;\ \overline{\mathsf{c}}\overline{\mathsf{d}} \\ \equiv\ & \mathsf{rs}(\mathsf{pq})^*\overline{\mathsf{c}}\overline{\mathsf{d}}. \end{aligned}$$

Applying Lemmas 3.4 and 3.5 and some obvious commutativity conditions, we obtain

$$\begin{aligned} \mathsf{rs}(\mathsf{pq})^*\overline{\mathsf{c}}\overline{\mathsf{d}} &= \mathsf{rse}(\mathsf{pq})^*\overline{\mathsf{c}}\overline{\mathsf{d}}\ =\ \mathsf{rse}(\mathsf{pq})^*(\mathsf{p}^* + \mathsf{q}^*)\overline{\mathsf{c}}\overline{\mathsf{d}} \\ &= \mathsf{rseb}(\mathsf{pq})^*(\mathsf{p}^* + \mathsf{q}^*)\overline{\mathsf{c}}\overline{\mathsf{d}}\ =\ \mathsf{rsebp}^*\mathsf{q}^*\overline{\mathsf{c}}\overline{\mathsf{d}}\ =\ \mathsf{rsp}^*\mathsf{q}^*\overline{\mathsf{c}}\overline{\mathsf{d}} \\ &= \mathsf{rp}^*\overline{\mathsf{c}}\mathsf{sq}^*\overline{\mathsf{d}}, \end{aligned}$$

which is (60). □

Acknowledgements

This work was supported in part by NSF grant CCR-0105586 and by ONR Grant N00014-01-1-0968. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the US Government.

References

[1] Allegra Angus and Dexter Kozen. Kleene algebra with tests and program schematology. Technical Report 2001-1844, Computer Science Department, Cornell University, July 2001.

[2] Ernie Cohen. Lazy caching. Unpublished, 1994.

[3] Ernie Cohen. Using Kleene algebra to reason about concurrency control. Unpublished, 1994.

[4] Ernie Cohen, Dexter Kozen, and Frederick Smith. The complexity of Kleene algebra with tests. Technical Report 96-1598, Computer Science Department, Cornell University, July 1996.

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

[6] P. Downey and R. Sethi. Assignment commands with array references. *J. Assoc. Comput. Mach.*, 25(4):652–666, October 1978.

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

[8] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, Cambridge, MA, 2000.

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

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

[11] Dexter Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.

[12] Dexter Kozen. On Hoare logic and Kleene algebra with tests. *Trans. Computational Logic*, 1(1):60–76, July 2000.

[13] Dexter Kozen and Maria-Cristina Patron. Certification of compiler optimizations using Kleene algebra with tests. In John Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luis Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proc. 1st Int. Conf. Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 568–582, London, July 2000. Springer-Verlag.

[14] Dexter Kozen and Frederick Smith. Kleene algebra with tests: Completeness and decidability. In D. van Dalen and M. Bezem, editors, *Proc. 10th Int. Workshop Computer Science Logic (CSL'96)*, volume 1258 of *Lecture Notes in Computer Science*, pages 244–259, Utrecht, The Netherlands, September 1996. Springer-Verlag.

[15] Dexter Kozen and Jerzy Tiuryn. On the completeness of propositional Hoare logic. In J. Desharnais, editor, *Proc. 5th Int. Seminar Relational Methods in Computer Science (RelMiCS 2000)*, pages 195–202, January 2000.

[16] Nikolay Mateev, Vijay Menon, and Keshav Pingali. Fractal symbolic analysis. In *Proc. 15th Int. Conf. on Supercomputing*, pages 38–49. ACM, ACM Press, 2001.

[17] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *Trans. Programming Languages and Systems*, 1(2):245–257, 1979.

[18] Aaron Stump, Clark W. Barrett, David L. Dill, and Jeremy Levitt. A decision procedure for an extensional theory of arrays. In *16th Symp. Logic in Comput. Sci.*, pages 29–37. IEEE, June 2001.

[19] N. Suzuki and D. Jefferson. Verification decidability of presburger array programs. In *Proc. Conf. Theor. Comput. Sci.* University of Waterloo, 1977.