On Hoare Logic and Kleene Algebra with Tests

Dexter Kozen*
Department of Computer Science
Cornell University
Ithaca, New York 14853-7501, USA
Phone: (607) 255-9209 Fax: (607) 255-4428
kozen@cs.cornell.edu

December 10, 1998

Abstract

We show that Kleene algebra with tests subsumes propositional Hoare logic. Thus the specialized syntax and deductive apparatus of Hoare logic are inessential and can be replaced by ordinary equational reasoning. It follows from the reduction that propositional Hoare logic is in *PSPACE*; we show that it is *PSPACE*-complete. **Keywords: Hoare logic, Kleene algebra, logics of programs, verification, formal methods, combination of logics, universal algebra**

1 Introduction

Hoare logic, introduced by C. A. R. Hoare in 1969 [11], was the first formal system for the specification and verification of well-structured programs. This pioneering work initiated the field of program correctness and inspired dozens of technical articles [6, 2, 7]. For this achievement among others, Hoare received the Turing Award in 1980.

Hoare logic uses a specialized syntax involving *partial correctness assertions* (PCAs) and a deductive apparatus consisting of a system of specialized rules of inference, one for each programming construct. Under certain conditions, these rules are relatively complete [6]; essentially, the propositional fragment of the logic can be used to reduce partial correctness assertions to static assertions about the underlying domain of computation.

In this paper we show that this propositional fragment, which we call *propositional Hoare logic* (PHL), is subsumed by *Kleene algebra with tests*

*Supported by the National Science Foundation under grant CCR-9708915.

(KAT), an equational algebraic system introduced in [15, 16]. The reduction transforms PCAs to ordinary equations and the specialized rules of inference to equational implications (universal Horn formulas). The transformed rules are all derivable in KAT by pure equational reasoning.

A Kleene algebra with tests is defined simply as a Kleene algebra with an embedded Boolean subalgebra. Possible interpretations include the various standard relational and trace-based models used in program semantics, and KAT is complete for the equational theory of these models [17]. Thus for all practical purposes KAT can be used in place of the Hoare rules in program correctness proofs.

The reduction of PHL to KAT in conjunction with results of [17, 4] imply that PHL in *PSPACE*. We show that it is *PSPACE*-complete.

This work shows that the reasoning power represented by propositional Hoare logic is captured in a concise, purely equational system KAT that is complete over various natural classes of interpretations and whose exact complexity is determined.

1.1 Related Work

Equational logic possesses a rich theory and is the subject of numerous papers and texts [20]. Its power and versatility in program specification and verification are widely recognized [18, 9].

In a recent J. ACM article [1], Bloom and Ésik reduce Hoare logic to the equational logic of iteration theories. They do not restrict their attention to **while** programs but attempt to capture all flowchart schemes, requiring some rather unweildy notation for insertion, tupling, and projection. In addition, their entire development is done in the framework of category theory; semantic models consist of morphisms in algebraic theories, a particular kind of category. These models are fairly obscure at first sight, and it is not immediately clear how they relate to the standard models. Indeed, a representation theorem for Boolean algebras in algebraic theories must be proved before the encoding of PCAs can even be stated. The adequacy of the encoding is also much more of an issue, requiring a rather recondite proof running to some ten journal pages. In contrast, the encoding in KAT can be stated quite simply: a partial correctness assertion $\{b\}\,p\,\{c\}$ becomes the equation $bp\bar{c} = 0$. Our adequacy proof is also relatively simple, requiring less than a page (Theorem 3.1).

The encoding of the **while** programming constructs using the regular operators and tests originated with propositional dynamic logic (PDL) [8]. Although strictly less expressive than PDL, KAT has a number of advantages: (i) it isolates the equational part of PDL, allowing program equivalence proofs to be expressed in their natural form; (ii) it conveniently overloads the operators

$+, \cdot, 0, 1$, allowing concise and elegant algebraic proofs; (iii) it is *PSPACE*-complete [4], whereas PDL is *EXPTIME*-complete [8]; (iv) interpretations are not restricted to relational models, but may be any algebraic structure satisfying the axioms; and (v) it admits various general and useful algebraic constructions such as the formation of algebras of matrices over a KAT, which among other things allows a natural encoding of automata.

Halpern and Reif [10] prove *PSPACE*-completeness of strict deterministic PDL (SDPDL), but neither the upper nor the lower bound of our *PSPACE*-completeness result follows from theirs. Not only are PDL semantics restricted to relational models, but the arguments of [10] depend on an additional nonalgebraic restriction: the relations interpreting atomic programs must be single-valued. Without this restriction, even if only **while** programs are allowed, PDL is exponential time hard. In contrast, KAT imposes no such restrictions.

1.2 Plan

In Section 2 we review the definitions of Hoare logic and Kleene algebra with tests. In Section 3 we reduce PHL to KAT and derive the Hoare rules as theorems of KAT. In Section 4 we prove that PHL is *PSPACE*-complete.

2 Preliminary Definitions

2.1 Hoare Logic

Hoare logic is a system for reasoning inductively about well-structured programs. A comprehensive introduction can be found in [7].

A common choice of programming language in Hoare logic is the language of **while** programs. The first-order version of this language contains a simple assignment $x := e$, conditional test **if** b **then** p **else** q, sequential composition $p; q$, and a looping construct **while** b **do** p.

The basic assertion of Hoare logic is the *partial correctness assertion* (PCA)

$$\{b\}\, p\, \{c\}, \tag{1}$$

where b and c are formulas and p is a program. Intuitively, this statement asserts that whenever b holds before the execution of the program p, then if and when p halts, c is guaranteed to hold of the output state. It does not assert that p must halt.

Semantically, programs p in Hoare logic and dynamic logic (DL) are usually interpreted as binary input/output relations $p^{\mathcal{M}}$ on a domain of computation $\mathcal{M}$, and assertions are interpreted as subsets of $\mathcal{M}$ [6, 19]. The definition of the relation $p^{\mathcal{M}}$ is inductive on the structure of p; for example, $(p; q)^{\mathcal{M}} = p^{\mathcal{M}} \circ q^{\mathcal{M}}$, the ordinary relational composition of the relations corresponding to p and q. The meaning of the PCA (1) is the same as the meaning of the DL formula $b \to [p]\,c$, where $\to$ is ordinary propositional implication and the modal construct $[p]\,c$ is interpreted in the model $\mathcal{M}$ as the set of states s such that for all $(s,t) \in p^{\mathcal{M}}$, the output state t satisfies c.

Hoare logic provides a system of specialized rules for deriving valid PCAs, one rule for each programming construct. The verification process is inductive on the structure of programs. The traditional Hoare inference rules are:

Assignment rule:

$$\{b[x/e]\}\; x := e\; \{b\} \tag{2}$$

Composition rule:

$$\frac{\{b\}\; p\; \{c\}, \quad \{c\}\; q\; \{d\}}{\{b\}\; p;q\; \{d\}} \tag{3}$$

Conditional rule:

$$\frac{\{b \wedge c\}\; p\; \{d\}, \quad \{\neg b \wedge c\}\; q\; \{d\}}{\{c\}\; \mathbf{if}\; b\; \mathbf{then}\; p\; \mathbf{else}\; q\; \{d\}} \tag{4}$$

While rule:

$$\frac{\{b \wedge c\}\; p\; \{c\}}{\{c\}\; \mathbf{while}\; b\; \mathbf{do}\; p\; \{\neg b \wedge c\}}. \tag{5}$$

The propositional fragment of Hoare logic (PHL) consists of atomic proposition and program symbols, the usual propositional connectives, **while** program constructs, and PCAs built from these. Atomic programs are interpreted as arbitrary binary relations on a set $\mathcal{M}$ and atomic propositions are interpreted as arbitrary subsets of $\mathcal{M}$. The deduction system consists of the composition, conditional, and while rules (3)–(5). The assignment rule (2) is omitted, since there is no first-order relational structure over which to interpret program variables; in practice, its role is played by PCAs over atomic programs that are postulated as assumptions.

2.2 Kleene Algebra

Kleene algebra (KA) is the algebra of regular expressions [13, 5]. The axiomatization used here is from [14]. A *Kleene algebra* is an algebraic structure $(\mathcal{K},\ +,\ \cdot,\ ^*,\ 0,\ 1)$ that is an idempotent semiring under $+, \cdot, 0, 1$ satisfying

$$
\begin{aligned}
1 + pp^* &= p^* & (6)\\
1 + p^*p &= p^* & (7)\\
q + pr \le r &\to p^*q \le r & (8)\\
q + rp \le r &\to qp^* \le r & (9)
\end{aligned}
$$

where $\le$ refers to the natural partial order on $\mathcal{K}$:

$$
p \le q \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad p + q = q \; .
$$

The operation $+$ gives the supremum with respect to the natural order $\le$. Instead of (8) and (9), we might take the equivalent axioms

$$
\begin{aligned}
pr \le r &\to p^*r \le r & (10)\\
rp \le r &\to rp^* \le r \; . & (11)
\end{aligned}
$$

These axioms say essentially that * behaves like the Kleene asterate operator of formal language theory or the reflexive transitive closure operator of relational algebra.

Kleene algebra is a versatile system with many useful interpretations. Standard models include the family of regular sets over a finite alphabet; the family of binary relations on a set; and the family of $n \times n$ matrices over another Kleene algebra. Other more unusual interpretations include the min,+ algebra used in shortest path algorithms and models consisting of convex polyhedra used in computational geometry [12].

The following are some typical identities that hold in all Kleene algebras:

$$
\begin{aligned}
(p^*q)^*p^* &= (p+q)^* & (12)\\
p(qp)^* &= (pq)^*p & (13)\\
p^* &= (pp)^*(1+p) \; . & (14)
\end{aligned}
$$

All the operators are monotone with respect to $\le$. In other words, if $p \le q$, then $pr \le qr$, $rp \le rq$, $p + r \le q + r$, and $p^* \le q^*$ for any r.

The completeness result of [14] says that all true identities between regular expressions interpreted as regular sets of strings are derivable from the axioms of Kleene algebra. In other words, the algebra of regular sets of strings over the finite alphabet Σ is the free Kleene algebra on generators Σ. The axioms are also complete over relational models.

See [14] for a more thorough introduction.

2.3 Kleene Algebra with Tests

Kleene algebras with tests (KAT) were introduced in [15, 16] and their theory further developed in [17, 4]. A Kleene algebra with tests is just a Kleene algebra with an embedded Boolean subalgebra. That is, it is a two-sorted structure

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

such that

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

The Boolean complementation operator $^-$ is defined only on $\mathcal{B}$. Elements of $\mathcal{B}$ are called *tests*. The letters p, q, r, s denote arbitrary elements of $\mathcal{K}$ and a, b, c denote tests.

This deceptively simple definition actually carries a lot of information in a concise package. The operators $+, \cdot, 0, 1$ each play two roles: applied to arbitrary elements of $\mathcal{K}$, they refer to nondeterministic choice, composition, fail, and skip, respectively; and 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, sequential testing of b and c is the same as testing their conjunction—and their coexistence admits considerable economy of expression.

The encoding of the **while** program constructs is as in PDL [8]:

$$p; q \stackrel{\text{def}}{=} pq \tag{15}$$

$$\textbf{if } b \textbf{ then } p \textbf{ else } q \stackrel{\text{def}}{=} bp + \overline{b}q \tag{16}$$

$$\textbf{while } b \textbf{ do } p \stackrel{\text{def}}{=} (bp)^*\overline{b}. \tag{17}$$

For applications in program verification, the standard interpretation would be a Kleene algebra of binary relations on a set and the Boolean algebra of subsets of the identity relation. One could also consider trace models, in which the Kleene elements are sets of traces (sequences of states) and the Boolean elements are sets of states (traces of length 0). As with KA, one can form the algebra $\mathrm{Mat}(\mathcal{K}, \mathcal{B}, n)$ of $n \times n$ matrices over a KAT $(\mathcal{K}, \mathcal{B})$; the Boolean elements of this structure are the diagonal matrices over $\mathcal{B}$. There is also a language-theoretic model that plays the same role in KAT that the regular sets of strings over a finite alphabet play in KA, namely the family of regular sets of *guarded strings* over a finite alphabet Σ with guards from a set B. This is the free KAT on generators Σ, B; that is, the equational theory of this structure is exactly the set of all equational consequences of the KAT axioms. Moreover, KAT is complete for the equational theory of relational models [17].

3 KAT and Hoare Logic

In this section we encode Hoare logic in KAT and derive the Hoare composition, conditional, and **while** rules as theorems of KAT.

The PCA $\{b\}\ p\ \{c\}$ is encoded in KAT by the equation

$$bp\overline{c} \;=\; 0. \tag{18}$$

Intuitively, this says that the program p with preguard b and postguard $\overline{c}$ has no halting execution. An equivalent formulation is

$$bp \;=\; bpc, \tag{19}$$

which says intuitively that testing c after executing bp is always redundant.

The equivalence of (18) and (19) can be argued easily in KAT. Assuming (18),

$$\begin{aligned} bp &= bp(c+\overline{c}) && \text{by the axiom } a1 = a \text{ and Boolean algebra} \\ &= bpc + bp\overline{c} && \text{by distributivity} \\ &= bpc && \text{by (18) and the axiom } a + 0 = a. \end{aligned}$$

Conversely, assuming (19),

$$\begin{aligned} bp\overline{c} &= bpc\overline{c} && \text{by (19)} \\ &= bp0 && \text{by associativity and Boolean algebra} \\ &= 0 && \text{by the axiom } a0 = 0. \end{aligned}$$

The equation (19) is equivalent to the inequality $bp \leq bpc$, since the reverse inequality is a theorem of KAT; it follows immediately from the axiom $c \leq 1$ of Boolean algebra and monotonicity of multiplication.

Using (15)–(17) and (19), the Hoare rules (3)–(5) take the following form:

Composition rule:

$$bp = bpc \wedge cq = cqd \quad\rightarrow\quad bpq = bpqd \tag{20}$$

Conditional rule:

$$bcp = bcpd \wedge \overline{b}cq = \overline{b}cqd \quad\rightarrow\quad c(bp + \overline{b}q) = c(bp + \overline{b}q)d \tag{21}$$

While rule:

$$bcp = bcpc \quad\rightarrow\quad c(bp)^*\overline{b} = c(bp)^*\overline{b}\,\overline{b}c \tag{22}$$

These implications are to be interpreted as universal Horn formulas; that is, the variables are implicitly universally quantified. To establish the adequacy of the translation, we show that (20)–(22) encoding the Hoare rules (3)–(5) are theorems of KAT.

Theorem 3.1 *The universal Horn formulas* (20)–(22) *are theorems of* KAT.

Proof. First we derive (20). By the law of congruence, we have

$$\begin{aligned} bp = bpc &\ \rightarrow\ bpq = bpcq, \\ cq = cqd &\ \rightarrow\ bpcq = bpcqd, \end{aligned}$$

thus

$$bp = bpc \wedge cq = cqd \ \rightarrow\ bpq = bpcqd. \tag{23}$$

The inequalities $bpcqd \leq bpqd$ and $bpqd \leq bpq$ are theorems of KAT, following immediately from the inequalities $c \leq 1$ and $d \leq 1$ and monotonicity of multiplication. Combining these inequalities with (23) gives

$$bp = bpc \wedge cq = cqd \ \rightarrow\ bpq = bpqd,$$

which is just (20).

The rule (21) is immediate from the multiplicative commutativity of tests and the distributive laws.

Finally, for (22), by trivial simplifications it suffices to show

$$cbp \leq cbpc \ \rightarrow\ c(bp)^* \leq c(bp)^*c.$$

Assume

$$cbp \ \leq\ cbpc \tag{24}$$

By (9) we need only show

$$c + c(bp)^*cbp \ \leq\ c(bp)^*c.$$

But

$$\begin{aligned} c + c(bp)^*cbp &\leq c + c(bp)^*cbpc && \text{by (24) and monotonicity} \\ &\leq c1c + c(bp)^*cbpc && \text{by Boolean algebra} \\ &\leq c(1 + (bp)^*cbp)c && \text{by distributivity} \\ &\leq c(1 + (bp)^*bp)c && \text{by monotonicity} \\ &\leq c(bp)^*c && \text{by (7).} \end{aligned}$$

□

4 Complexity of PHL

We formulate the decision problem for propositional Hoare logic as follows: Given a finite set of propositional PCAs $\varphi_1, \ldots, \varphi_m$ and a propositional PCA ψ, is ψ a logical consequence of $\varphi_1, \ldots, \varphi_m$?

The assumptions $\varphi_1, \ldots, \varphi_m$ play the role of the assignment rule (2) and are an essential part of the formulation. We might restrict $\varphi_1, \ldots, \varphi_m$ to allow only atomic programs in order to emphasize this; this turns out not to affect the complexity of the decision problem.

Theorem 4.1 *The decision problem for propositional Hoare logic is PSPACE-complete.*

Proof. The reduction of Section 3 using the form (18) transforms the decision problem for PHL to the problem of the universal validity of Horn formulas

$$p_1 = 0 \wedge \cdots \wedge p_m = 0 \quad \to \quad q = 0$$

of KAT. By a result of [17] generalizing a similar result of Cohen [3] for Kleene algebra without tests, a Horn formula all of whose premises are of the form $p = 0$ can be transformed to a single equation without premises whose validity is equivalent to the validity of the original Horn formula (this is not true in general for premises not of the form $p = 0$). The equational theory of KAT is decidable in *PSPACE* [4], thus the decision problem for PHL is in *PSPACE*.

We now show that the problem is *PSPACE*-hard. This holds even if the premises $\varphi_1, \ldots, \varphi_m$ are restricted to refer only to atomic programs, and even if they are restricted to refer only to a single atomic program p. We give a direct encoding of the computation of a polynomial space-bounded one-tape deterministic Turing machine in an instance of the decision problem for PHL. The approach is similar to [10], using the premises $\varphi_1, \ldots, \varphi_m$ to circumvent the determinacy assumption.

Consider the computation of a polynomially-space-bounded one-tape deterministic Turing machine M on some input x of length n. Let N be a polynomial bound on the amount of space used by M on input x. Let Q be the set of states of M, let Γ be its tape alphabet, let s be its start state, and let t be its unique halt state. We use polynomially many atomic propositional symbols with the following intuitive meanings:

$T_{i,a}$	"the i^{th} tape cell currently contains symbol a,"	$0 \le i \le N, \quad a \in \Gamma$
H_i	"the tape head is currently scanning the ith tape cell,"	$0 \le i \le N$
S_q	"the machine is currently in state q,"	$q \in Q$.

Let p be an atomic program. Intuitively, p represents the action of one step of M. We will devise a set of assumptions $\varphi_1, \ldots, \varphi_m$ that will say that p

faithfully models the action of M. The PCA ψ will say that if started in state s on input x, the program

while the current state is not t **do** p

fails. The PCA ψ will be a logical consequence of $\varphi_1, \ldots, \varphi_m$ iff M does not halt on input x.

The start configuration of M on x consists of a left endmarker $\vdash$ written on tape cell 0, the input $x = a_1 \cdots a_n$ written on cells 1 through n, and the remainder of the tape filled with the blank symbol $\sqcup$ out to the N^{th} cell. The machine starts in state s scanning the left endmarker. This situation is captured by the propositional formula

$$\text{START} \stackrel{\text{def}}{=} T_{0,\vdash} \wedge \bigwedge_{1 \le i \le n} T_{i,a_i} \wedge \bigwedge_{n+1 \le i \le N} T_{i,\sqcup} \wedge S_s \wedge H_0.$$

We will need a formula to ensure that M is in at most one state, that it is scanning at most one tape cell, and that there is at most one symbol written on each tape cell:

$$\text{FORMAT} \stackrel{\text{def}}{=} \bigwedge_{0 \le i \le N} \bigwedge_{a \ne b} \neg(T_{i,a} \wedge T_{i,b}) \wedge \bigwedge_{p \ne q} \neg(S_p \wedge S_q) \wedge \bigwedge_{0 \le i < j \le N} \neg(H_i \wedge H_j).$$

We include the PCA

$$\{\text{FORMAT}\}\ p\ \{\text{FORMAT}\}$$

as one of the assumptions φ_i to ensure that FORMAT is an invariant of p, thus is maintained throughout the simulation of M.

Suppose the transition function of M says that when scanning a cell containing symbol a in state p, M prints the symbol b on that cell, moves right, and enters state q. We capture this constraint by the family of PCAs

$$\{T_{i,a} \wedge H_i \wedge S_p\}\ p\ \{T_{i,b} \wedge H_{i+1} \wedge S_q\}, \quad 0 \le i \le N-1.$$

All these PCAs are included for each possible transition of the machine; there are only polynomially many in all.

We must also ensure that the symbols on tape cells not currently being scanned do not change; this is accomplished by the family of PCAs

$$\{T_{i,a} \wedge \neg H_i\}\ p\ \{T_{i,a}\}, \quad 0 \le i \le N,\ a \in \Gamma.$$

These are the assumptions $\varphi_1, \ldots, \varphi_m$ in our instance of the decision problem. It is apparent that under any interpretation of p satisfying these PCAs, successive executions of p starting from any state satisfying START $\wedge$ FORMAT move only to states whose values for the atomic propositions S_q, $T_{i,a}$, and H_i

model valid configurations of M, and the values change in such a way as to model the computation of M. Thus there is a reachable state satisfying S_t iff M halts on x.

We take as our conclusion ψ the PCA

$$\psi \stackrel{\text{def}}{=} \{\text{START} \wedge \text{FORMAT}\}\ \textbf{while}\ \neg S_t\ \textbf{do}\ p\ \{\text{FALSE}\},$$

which says intuitively that when started in the start configuration, repeatedly executing p will never cause M to enter state t. The PCA ψ is therefore a logical consequence of $\varphi_1, \ldots, \varphi_m$ iff M does not halt on x. $\square$

References

[1] Stephen L. Bloom and Zoltán Ésik. Floyd–Hoare logic in iteration theories. *J. Assoc. Comput. Mach.*, 38(4):887–934, October 1991.

[2] E. M. Clarke, S. M. German, and J. Y. Halpern. Effective axiomatizations of Hoare logics. *J. Assoc. Comput. Mach.*, 30:612–636, 1983.

[3] Ernie Cohen. Hypotheses in Kleene algebra. Available as ftp://ftp.bellcore.com/pub/ernie/research/homepage.html, April 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, U.K., 1971.

[6] S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7(1):70–90, February 1978.

[7] Patrick Cousot. Methods and logics for proving programs. In J. van Leeuwen, editor, *Handbood of Theoretical Computer Science*, volume B, pages 841–993. Elsevier, Amsterdam, 1990.

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

[9] Joseph A. Goguen and Grant Malcolm. *Algebraic Semantics of Imperative Programs.* Foundations of Computing. MIT Press, Cambridge, MA, 1996.

[10] J. Y. Halpern and J. H. Reif. The propositional dynamic logic of deterministic, well-structured programs. *Theor. Comput. Sci.*, 27:127–165, 1983.

[11] C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. Assoc. Comput. Mach.*, 12:576–80, 1969.

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

[13] S. 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.

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

[15] Dexter Kozen. Kleene algebra with tests and commutativity conditions. In T. Margaria and B. Steffen, editors, *Proc. Second Int. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 14–33, Passau, Germany, March 1996. Springer-Verlag.

[16] Dexter Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, pages 427–443, May 1997.

[17] 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.

[18] Michael J. O'Donnell. *Equational Logic as a Programming Language.* MIT Press, Cambridge, MA, 1985.

[19] V. R. Pratt. A practical decision method for propositional dynamic logic. In *Proc. 10th Symp. Theory of Comput.*, pages 326–337. ACM, 1978.

[20] Walter Taylor. Equational logic. *Houston J. Math.*, pages iii+83, 1979. survey.