Kleene Algebra with Tests and the Static Analysis of Programs

Dexter Kozen
Department of Computer Science
Cornell University
Ithaca, New York 14853-7501, USA
kozen@cs.cornell.edu

November 17, 2003

Abstract

We propose a general framework for the static analysis of programs based on Kleene algebra with tests (KAT). We show how KAT can be used to statically verify compliance with safety policies specified by security automata. We prove soundness and completeness over relational interpretations. We illustrate the method on an example involving the correctness of a device driver.

1 Introduction

Kleene algebra with tests (KAT) is an algebraic system for program specification and verification that combines Kleene algebra, or the algebra of regular expressions, with Boolean algebra. One can model basic programming language constructs such as conditionals and while loops, verification conditions, and partial correctness assertions. KAT has been applied successfully in substantial verification tasks involving communication protocols, source-to-source program transformation, concurrency control, compiler optimization, and dataflow analysis [2, 4, 5, 6, 11, 14]. The system is *PSPACE*-complete and deductively complete for partial correctness over relational and trace models [12].

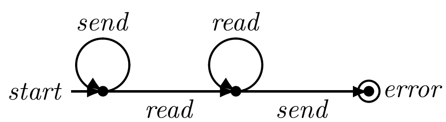
KAT has a rich algebraic theory with many natural and useful models: language-theoretic, relational, trace-based, matrix. Because of its roots in classical algebra and equational logic, KAT provides a mathematically rigorous foundation that subsumes many previous approaches, recasting them in a more classical algebraic framework. Hoare logic and program schematology are two examples of major theories in computer science that can be given a more classical treatment using KAT.

In this paper, we show how KAT provides a general framework for the static analysis of programs. We give a general construction that shows how to use

KAT to statically verify compliance with safety policies specified by security automata. We prove the soundness and completeness of the method over relational interpretations. These results further attest to the versatility of KAT as a general framework for many verification tasks in computer science.

Security automata are a popular mechanism for the specification and enforcement of a large class of safety policies [16]. A security automaton is an ordinary finite-state automaton in which certain states are designated as *error states*. A transition to a new state may occur when a critical operation of a program is executed. Any computation of a program containing a sequence of critical operations that sends the automaton to an error state violates the policy as specified by the automaton.

The following example is from [16]. Suppose we wish to specify that a program may not sent a message out over the network after it has read the disk. This restriction can be specified by the following automaton.



Here the leftmost state indicates that the program has not yet read the disk. This is the initial state. As long as the automaton is in this state, network activity is permitted. Any read of the disk, however, causes a transition to the second state, which indicates that the disk has been read. From this state, further disk reads are allowed, but the program may not send a message out over the net. Any attempt to do so causes a transition to the error state.

The automaton can be used for runtime enforcement of the security policy as well as specification. The program code is instrumented to call the automaton before all critical operations (ones that could change state of the automaton). The automaton aborts the computation if the operation would cause a transition to an error state.

This enforcement mechanism as described in [16] is purely a runtime mechanism. In this paper we describe a general method based on KAT for verifying *statically* compliance with the policy as specified by the security automaton. The method uses the KAT rules to propagate state information throughout the program to all critical operations. If the verification is successful, an independently checkable proof object is produced that can be used to certify that the runtime checks are unnecessary. We prove a soundness theorem (Theorem 4.1) that says that any program verified in this fashion satisfies the policy. There is also a version with a simplified verification condition (Corollary 4.2) for when the program is known to be total. We also prove a completeness theorem (Theorem 5.1) that says that if the propositional abstraction of the program fails to verify, then there is a relational interpretation in which the program is unsafe. Finally, we illustrate the method on an example of [3] involving the verification of a device driver.

2 Preliminary Definitions

2.1 Kleene Algebra

Kleene algebra (KA) is the algebra of regular expressions [9, 7]. The axiomatization used here is from [10]. A *Kleene algebra* is an algebraic structure $(K, +, \cdot, ^*, 0, 1)$ that is an idempotent semiring under $+, \cdot, 0, 1$ such that p^*q is the $\leq$-least solution to $q + px \leq x$ and qp^* is the $\leq$-least solution to $q + xp \leq x$. Here $\leq$ refers to the natural partial order on K: $p \leq q \stackrel{\text{def}}{\Longleftrightarrow} p + q = q$. This is a universal Horn axiomatization.

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

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, also known as the *tropical semiring*, used in shortest path algorithms, and models consisting of convex polyhedra used in computational geometry.

The completeness result of [10] says that all true identities between regular expressions interpreted as regular sets of strings are derivable from the axioms. In other words, the algebra of regular sets of strings over a finite alphabet P is the free Kleene algebra on generators P. The axioms are also complete for the equational theory of relational and trace models.

2.2 Kleene Algebra with Tests

A *Kleene algebra with tests* (KAT) [11] is just a Kleene algebra with an embedded Boolean subalgebra. That is, it is a two-sorted structure $(K, B, +, \cdot, ^*, \bar{\ }, 0, 1)$ such that

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

Elements of B are called *tests* and are denoted $b, c, d, \ldots$. The Boolean complementation operator $\bar{\ }$ is defined only on tests. Arbitrary elements of K are denoted $p, q, r, \ldots$.

The set of terms in the language of KAT over atomic actions P and atomic tests B is denoted $\mathsf{RExp}_{P,B}$.

The **while** program constructs are encoded as in propositional Dynamic Logic [8]:

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

The Hoare partial correctness assertion $\{b\}\,p\,\{c\}$ is expressed in KAT in any one of the following three equivalent forms:

$$
\begin{aligned}
bp &\leq pc \qquad (1)\\
bp &= bpc \qquad (2)\\
bp\overline{c} &= 0. \qquad (3)
\end{aligned}
$$

The proof of the equivalence of these formulas is an elementary exercise in KAT; see [12]. Because of the form (3), all Hoare-style rules of the form

$$
\frac{\{b_1\}\,p_1\,\{c_1\} \quad \ldots \quad \{b_n\}\,p_n\,\{c_n\}}{\{b\}\,p\,\{c\}}
$$

are encoded as Horn formulas of the form

$$
b_1p_1\overline{c}_1 = 0 \to \ldots \to b_np_n\overline{c}_n = 0 \quad \to \quad bp\overline{c} = 0.
$$

Horn formulas all of whose premises are of the form $p = 0$ are called *Hoare formulas*, and KAT is deductively complete for the Hoare theory of relational models [12]; that is, all relationally valid Hoare formulas are theorems of KAT. This theory is also decidable in *PSPACE*.

See [10, 11, 12, 15] for a more detailed introduction to KA and KAT.

2.3 Trace and Relational Models

2.3.1 Kripke Frames

For applications in program verification, one usually interprets programs and tests over a KAT consisting of sets of traces or binary relations on a set of states. Both these classes of algebras are defined in terms of *Kripke frames*. A Kripke frame over a set of atomic programs P and a set of atomic tests B is a structure $(K, \mathfrak{m}_K)$, where K is a set of *states*, $\mathfrak{m}_K : P \to 2^{K\times K}$, and $\mathfrak{m}_K : B \to 2^K$. The map $\mathfrak{m}_K$ specifies a canonical interpretation of the atomic actions and tests.

2.3.2 Relation Algebras

The set of all binary relations on a Kripke frame K forms a KAT under the standard binary relation-theoretic interpretation of the KAT operators. The operator $\cdot$ is interpreted as relational composition $\circ$, $+$ as union, 0 and 1 as the empty relation and the identity relation on K, respectively, and * as reflexive transitive closure. The Boolean elements are subsets of the identity relation. This is called the *full relation algebra on K*. One can define a canonical interpretation $[\ ]_K : \mathsf{RExp}_{P,B} \to 2^{K\times K}$ by

$$
\begin{aligned}
[p]_K &\stackrel{\text{def}}{=} \mathfrak{m}_K(p), \quad p \in P\\
[b]_K &\stackrel{\text{def}}{=} \{(u,u) \mid u \in \mathfrak{m}_K(b)\}, \quad b \in B,
\end{aligned}
$$

extended homomorphically. A binary relation is *regular* if it is $[p]_K$ for some $p \in \mathsf{RExp}_{P,B}$. The subalgebra consisting of all regular binary relations on K is denoted Rel_K. We write $\mathsf{REL} \vDash \varphi$ if φ is true in all Rel_K.

2.3.3 Trace Algebras

A *trace* in a Kripke frame K is a sequence $s_0 p_0 s_1 \cdots s_{n-1} p_{n-1} s_n$, where $n \geq 0$, $s_i \in K$, $p_i \in P$, and $(s_i, s_{i+1}) \in \mathfrak{m}_K(p_i)$ for $0 \leq i \leq n-1$. The set of all traces in K is denoted Traces_K. We denote traces by $\sigma, \tau, \ldots$. The first and last states of a trace σ are denoted $\mathbf{first}(\sigma)$ and $\mathbf{last}(\sigma)$, respectively. If $\mathbf{last}(\sigma) = \mathbf{first}(\tau)$, we can fuse σ and τ to get the trace $\sigma\tau$. If $\mathbf{last}(\sigma) \neq \mathbf{first}(\tau)$, then $\sigma\tau$ does not exist. The *label* of the trace $s_0 p_0 s_1 \cdots s_{n-1} p_{n-1} s_n$ is the string $p_0, \ldots, p_{n-1}$.

The powerset of Traces_K forms a KAT in which $+$ is interpreted as set union, $\cdot$ as the operation

$$AB \stackrel{\text{def}}{=} \{\sigma\tau \mid \sigma \in A,\ \tau \in B,\ \mathbf{last}(\sigma) = \mathbf{first}(\tau)\},$$

0 and 1 as $\varnothing$ and K, respectively, and A^* as the union of all finite powers of A. The Boolean elements are the subsets of K, the sets of traces of length 0. This is called the *full trace algebra on K*. A canonical interpretation $[\![\ ]\!]_K$ for KAT expressions over P and B is given by

$$\begin{aligned} [\![p]\!]_K &\stackrel{\text{def}}{=} \{spt \mid (s,t) \in \mathfrak{m}_K(p)\}, \quad p \in P \\ [\![b]\!]_K &\stackrel{\text{def}}{=} \mathfrak{m}_K(b), \quad b \in B, \end{aligned}$$

extended homomorphically. A set of traces is *regular* if it is $[\![p]\!]_K$ for some KAT expression p. The subalgebra of all regular sets of traces of K is denoted Tr_K. We write $\mathsf{TR} \vDash \varphi$ if φ is true in all Tr_K.

If A is a set of traces, define

$$\mathsf{Prefix}(A) \stackrel{\text{def}}{=} \{\sigma \mid \exists\tau\ \sigma\tau \in A\}.$$

Lemma 2.1 *Let p be any program expression and b any test expression. Then* $[\![b]\!]_K \cdot \mathsf{Prefix}([\![p]\!]_K) = \mathsf{Prefix}([\![bp]\!]_K)$.

Proof. For any $\sigma \in \mathsf{Traces}_K$,

$$\begin{aligned} \sigma \in [\![b]\!]_K \cdot \mathsf{Prefix}([\![p]\!]_K) &\Leftrightarrow \sigma \in \mathsf{Prefix}([\![p]\!]_K) \text{ and } \mathbf{first}(\sigma) \in [\![b]\!]_K \\ &\Leftrightarrow \exists\tau\ \sigma\tau \in [\![p]\!]_K \text{ and } \mathbf{first}(\sigma\tau) \in [\![b]\!]_K \\ &\Leftrightarrow \exists\tau\ \sigma\tau \in [\![bp]\!]_K \\ &\Leftrightarrow \sigma \in \mathsf{Prefix}([\![bp]\!]_K). \end{aligned}$$

□

2.3.4 Canonical Homomorphisms

If K, L are KATs with distinguished canonical interpretations $I : \mathsf{RExp}_{P,B} \to K$ and $J : \mathsf{RExp}_{P,B} \to L$, a homomorphism $h : K \to L$ is *canonical* if it commutes with I and J. In particular, a homomorphism involving trace or relation algebras on Kripke frames over P, B is canonical if it commutes with $[\![\ ]\!]_K$ and $[\ ]_K$.

An example of a canonical homomorphism is the map $\mathrm{Ext} : 2^{\mathsf{Traces}_K} \to 2^{K \times K}$ defined by $\mathrm{Ext} : A \mapsto \{(\mathbf{first}(\sigma), \mathbf{last}(\sigma)) \mid \sigma \in A\}$. As shown in [13, §3.6], this is a KAT homomorphism, and $\mathrm{Ext}([\![p]\!]_K) = [p]_K$.

Lemma 2.2 *The Hoare theories of trace algebras and relation algebras coincide.*

Proof. By [13, Lemma 3.1], every trace algebra is isomorphic to a relation algebra, so the Hoare theory of relation algebras is contained in the Hoare theory of trace algebras.

For the other direction, suppose $\mathsf{TR} \vDash r = 0 \to p = q$. (Without loss of generality, we can restrict ourselves to Hoare formulas with a single premise.) Let K be an arbitrary Kripke frame. Let $\mathrm{Ext} : 2^{\mathsf{Traces}_K} \to 2^{K \times K}$ be the canonical homomorphism described above. Note that for $A \subseteq \mathsf{Traces}_K$, $\mathrm{Ext}(A) = \varnothing$ iff $A = \varnothing$. Thus

$$\begin{array}{rlll} [r]_K = \varnothing & \Rightarrow & [\![r]\!]_K = \varnothing & \text{since } [r]_K = \mathrm{Ext}([\![r]\!]_K) = \varnothing \text{ iff } [\![r]\!]_K = \varnothing \\ & \Rightarrow & [\![p]\!]_K = [\![q]\!]_K & \text{since } \mathsf{TR} \vDash r = 0 \to p = q \\ & \Rightarrow & [p]_K = [q]_K & \text{since } \mathrm{Ext}([\![p]\!]_K) = [p]_K. \end{array}$$

Since K was arbitrary, we have $\mathsf{REL} \vDash r = 0 \to p = q$. Thus the Hoare theory of trace algebras is contained in the Hoare theory of relational algebras. $\square$

2.3.5 Induced Subframes

Let $(L, \mathfrak{m}_L)$ be a Kripke frame and let K be a subset of L. The *induced subframe on* K is $(K, \mathfrak{m}_K)$, where

$$\mathfrak{m}_K(b) \stackrel{\text{def}}{=} \mathfrak{m}_L(b) \cap K, \quad b \in B, \tag{4}$$

$$\mathfrak{m}_K(p) \stackrel{\text{def}}{=} \mathfrak{m}_L(p) \cap K^2, \quad p \in P. \tag{5}$$

We say that a binary relation R on L *preserves* K if $t \in K$ whenever $s \in K$ and $(s, t) \in R$.

Lemma 2.3 ([13, Lemma 5.1]) *Let $(K, \mathfrak{m}_K)$ be an induced subframe of $(L, \mathfrak{m}_L)$ such that all atomic actions $\mathfrak{m}_L(p)$ preserve K.*

(a) *The map $A \mapsto A \cap \mathsf{Traces}_K$ for $A \subseteq \mathsf{Traces}_L$ is a canonical* KAT *homomorphism $\mathsf{Tr}_L \to \mathsf{Tr}_K$.*

(b) *The map $A \mapsto A \cap K^2$ for $A \subseteq L^2$ is a canonical* KAT *homomorphism $\mathsf{Rel}_L \to \mathsf{Rel}_K$.*

2.3.6 Coherence

Let K, L be Kripke frames over P, B. A function $f : K \to L$ is said to be *coherent* if

(i) $(s, t) \in \mathfrak{m}_K(p) \Rightarrow (f(s), f(t)) \in \mathfrak{m}_L(p), \quad p \in P$

(ii) $s \in \mathfrak{m}_K(b) \Leftrightarrow f(s) \in \mathfrak{m}_L(b), \quad b \in B.$

Condition (i) implies that f can be extended to traces. Define $f : \mathsf{Traces}_K \to \mathsf{Traces}_L$ by

$$f(s_0p_0s_1 \cdots s_{n-1}p_{n-1}s_n) \stackrel{\mathrm{def}}{=} f(s_0)p_0f(s_1) \cdots f(s_{n-1})p_{n-1}f(s_n).$$

The function f is said to be *onto on traces* if its extension $f : \mathsf{Traces}_K \to \mathsf{Traces}_L$ is onto.

For a coherent function $f : K \to L$ and $A \subseteq \mathsf{Traces}_L$, define

$$f^{-1}(A) \stackrel{\mathrm{def}}{=} \{\sigma \in \mathsf{Traces}_K \mid f(\sigma) \in A\}.$$

Lemma 2.4 ([13, Lemma 5.3]) *If $f : K \to L$ is coherent, then f^{-1} is a* KAT *homomorphism on the full trace algebras of K and L, and its restriction to the regular trace algebra Tr_L is a canonical homomorphism $\mathsf{Tr}_L \to \mathsf{Tr}_K$. If in addition f is onto on traces, then f^{-1} is one-to-one, therefore $f^{-1} : \mathsf{Tr}_L \to \mathsf{Tr}_K$ is a canonical isomorphism.*

2.4 Precomputations

Let $p \in \mathsf{RExp}_{P,B}$ be a KAT expression representing a program. We define by induction an expression $\mathsf{PreComp}(p) \in \mathsf{RExp}_{P,B}$ that describes the set of *precomputations* of p.

$$\begin{aligned}
\mathsf{PreComp}(p) &\stackrel{\mathrm{def}}{=} 1 + p, \quad p \text{ an atomic action}\\
\mathsf{PreComp}(b) &\stackrel{\mathrm{def}}{=} 1, \quad b \text{ a test}\\
\mathsf{PreComp}(p + q) &\stackrel{\mathrm{def}}{=} \mathsf{PreComp}(p) + \mathsf{PreComp}(q)\\
\mathsf{PreComp}(pq) &\stackrel{\mathrm{def}}{=} \mathsf{PreComp}(p) + p \cdot \mathsf{PreComp}(q)\\
\mathsf{PreComp}(p^*) &\stackrel{\mathrm{def}}{=} p^* \cdot \mathsf{PreComp}(p).
\end{aligned}$$

The expression $\mathsf{PreComp}(p)$ is meant to capture the prefixes of computations of the program p. However, the set $[\![\mathsf{PreComp}(p)]\!]_K$ is not the same as $\mathsf{Prefix}([\![p]\!]_K)$, the set of prefixes of traces in $[\![p]\!]_K$ in a Kripke frame K, although the two sets are related. Note that $\mathsf{PreComp}(p)$ depends on the syntactic expression p and not just on its equivalence class modulo the axioms of Kleene algebra.

The reason for the definition of $\mathsf{PreComp}(p)$ as given is to capture the prefixes of nonhalting computations. For example, the program **while true do** p translates to the KAT expression $(1p)^*0$, which is equivalent to 0; thus in any trace algebra, the set of traces represented by this expression is empty. However, $\mathsf{PreComp}((1p)^*0)$ is equivalent to p^*, which represents the set of traces corresponding to precomputations of the nonhalting program **while true do** p. When verifying a program p, we will actually verify the extended program $\mathsf{PreComp}(p)$. This will ensure that even if p does not halt, the safety condition will still be satisfied at all points in the computation.

The definition $\mathsf{PreComp}(b) = 1$ for tests b is intentional. A test b can modify a computation, but only in a position preceding another action. This can be seen in the definition of $\mathsf{PreComp}(pq)$. The test 1 is always a prefix of a test b, because that is the situation before b is executed. This definition ensures that $\mathsf{PreComp}(0^*)$ is equivalent to 1, which is as it should be.

Lemma 2.5 *The following are theorems of* KAT*:*

(i) $1 \leq \mathsf{PreComp}(p)$

(ii) $p \leq \mathsf{PreComp}(p)$

(iii) $\mathsf{PreComp}(\mathsf{PreComp}(p)) = \mathsf{PreComp}(p)$.

Proof. These properties can be proved easily by induction on the structure of p. □

Lemma 2.5(iii) says that the operator PreComp is a closure operator. We also have

Lemma 2.6 *In any Kripke frame K and for any* KAT *expression p,*

(i) $[\![\mathsf{PreComp}(p)]\!]_K$ *is closed under trace prefix; that is,*

$$\mathsf{Prefix}([\![\mathsf{PreComp}(p)]\!]_K) \quad \subseteq \quad [\![\mathsf{PreComp}(p)]\!]_K$$

(ii) $\mathsf{Prefix}([\![p]\!]_K) \subseteq [\![\mathsf{PreComp}(p)]\!]_K$.

Proof. Property (i) follows by induction on the structure of p. Property (ii) is a consequence of (i), Lemma 2.5(ii), and the monotonicity of the Prefix operator. □

3 Security Automata and KAT

Formally, a *security automaton* $M = (Q,\ P_{\mathrm{crit}},\ \delta,\ S,\ E)$ for a program r over atomic actions P and atomic tests B is an ordinary deterministic finite automaton consisting of a finite set of *states* Q, a finite *input alphabet* $P_{\mathrm{crit}} \subseteq P$, a deterministic *transition function* $\delta : Q \times P_{\mathrm{crit}} \to Q$, a *start state* $S \in Q$, and a set of *error states* $E \subseteq Q$. The elements of P_{crit} are called *critical operations*. These are the atomic actions that can cause a transition to another state in M. We denote elements of Q by $U, V, W, \ldots$.

We can extend the automaton to allow noncritical operations by defining $\delta(U,p) \stackrel{\mathrm{def}}{=} U$ for $p \in P - P_{\mathrm{crit}}$.

To capture the information represented by states of a security automaton in KAT, we introduce an atomic test U corresponding to each state $U \in Q$ and premises

$$UV = 0, \quad \text{for } U,V \in Q,\ U \neq V \tag{6}$$

$$Up \leq pV, \quad \text{for } p \in P_{\mathrm{crit}},\ V = \delta(U,p) \tag{7}$$

$$Up \leq pU, \quad \text{for } p \in P - P_{\mathrm{crit}}. \tag{8}$$

Premises (6) assert that it is impossible to be in two states of M simultaneously. State information is propagated forward through the code by means of premises (7) and (8). The former says that for critical operations, the state changes according to the transition function δ of M, and the latter says that for noncritical operations, the state does not change.

Assertions of the form (7) and (8) have useful alternative forms given by (1)–(3). These are various representations in KAT of the Hoare partial correctness assertion $\{U\}\, p\, \{V\}$. Because of the form (3), formulas with premises of this form are Hoare formulas, therefore verifiable in *PSPACE*. The conjunction of all premises (6)–(8) is denoted $\mathcal{A}_M$.

The following lemma is useful in propagating state information in a program.

Lemma 3.1 *The following are theorems of* KAT.

(i) *If $bp \le pc$ and $dp \le pe$, then $(b+d)p \le p(c+e)$.*

(ii) *If $bp \le pc$ and $bq \le qd$, then $b(p+q) \le (p+q)(c+d)$.*

(iii) *Let B be a set of states of M, and let B' be the smallest set of states containing B and closed under p; that is, B' is the smallest set of states such that $B \subseteq B'$, and if $U \in B'$, then $\delta(U,p) \in B'$. Let $b = \sum B$ and $b' = \sum B'$. Then $\mathcal{A}_M \to bp^* \le p^*b'$.*

Now let $(K, \mathfrak{m}_K)$ be a Kripke frame over alphabets P of atomic actions and B of atomic tests. Let $M = (Q, P_{\mathrm{crit}}, \delta, s, E)$ be a security automaton over an alphabet $P_{\mathrm{crit}} \subseteq P$ of critical actions. We can view M as a Kripke frame over atomic actions P and atomic tests Q by defining

$$\mathfrak{m}_M(p) \stackrel{\text{def}}{=} \{(U, \delta(U,p)) \mid U \in Q\}, \quad p \in P_{\mathrm{crit}} \tag{9}$$

$$\mathfrak{m}_M(p) \stackrel{\text{def}}{=} \{(U,U) \mid U \in Q\}, \quad p \in P - P_{\mathrm{crit}} \tag{10}$$

$$\mathfrak{m}_M(U) \stackrel{\text{def}}{=} \{U\}, \quad U \in Q. \tag{11}$$

We also use E to denote the test $\sum_{e\in E} e$.

Let r be a program, which is just a term of KAT over P, B. We now define what it means for r to be *safe* with respect to the security policy specified by M.

First we form the *instrumented frame* $K \times M$ over atomic actions P and atomic tests $B \cup Q$ as follows (we assume that $B \cap Q = \varnothing$). The states of $K \times M$ are ordered pairs (u, U), where u is a state of K and U is a state of M. For critical atomic operations $p \in P_{\mathrm{crit}}$, for noncritical atomic operations $q \in P - P_{\mathrm{crit}}$, for atomic tests $b \in B$, and for atomic tests $U \in Q$, define

$$\mathfrak{m}_{K\times M}(p) \stackrel{\text{def}}{=} \{((u,U),(v,V)) \mid (u,v) \in \mathfrak{m}_K(p),\ (U,V) \in \mathfrak{m}_M(p)\} \tag{12}$$

$$\mathfrak{m}_{K\times M}(q) \stackrel{\text{def}}{=} \{((u,U),(v,V)) \mid (u,v) \in \mathfrak{m}_K(q),\ U = V\} \tag{13}$$

$$\mathfrak{m}_{K\times M}(b) \stackrel{\text{def}}{=} \{(u,U) \mid u \in \mathfrak{m}_K(b),\ U \in Q\} \tag{14}$$

$$\mathfrak{m}_{K\times M}(U) \stackrel{\text{def}}{=} \{(u,U) \mid u \in K\}. \tag{15}$$

In other words, a critical transition $p \in P_{\mathrm{crit}}$ causes a transition in both components of the product, and a noncritical transition causes a transition in K but not in M. A state (u, U) of the product satisfies an atomic test in B iff its first component does and satisfies an atomic test in Q iff its second component does.

The instrumented frame $K \times M$ is formal means by which we run a program r in K and the security automaton M simultaneously. The projection of a trace onto the sequence of first components gives a trace in K, and the projection onto the sequence of second components gives a corresponding trace in M. Both components run according to the same sequence of atomic actions, as ensured by (12) and (13).

A trace of $K \times M$ is *erroneous* with respect to M if it contains an error state of M. Thus the non-erroneous traces are represented by the expression $(\overline{E} \cdot \sum P)^*\overline{E}$. Note that this is determined by the second components of the states in the trace.

We say that a program r is *safe* in K with respect to M if

$$[\![S \cdot \mathsf{PreComp}(r)]\!]_{K \times M} \quad \subseteq \quad [\![(\overline{E} \cdot \textstyle\sum P)^*\overline{E}]\!]_{K \times M}. \tag{16}$$

The left-hand side represents the set of traces of the instrumented frame $K \times M$ whose first components give a computation prefix of r in K and whose second components represent a run of the security automaton M starting in its start state. The inclusion (16) states that no such trace is erroneous.

Note that in any Kripke frame L, $[\![(\overline{E} \cdot \sum P)^*\overline{E}]\!]_L$ is closed under Prefix; that is,

$$\mathsf{Prefix}([\![(\overline{E} \cdot \textstyle\sum P)^*\overline{E}]\!]_L) \quad \subseteq \quad [\![(\overline{E} \cdot \textstyle\sum P)^*\overline{E}]\!]_L. \tag{17}$$

Let π_1 and π_2 denote the projections from elements of $K \times M$ to their first and second components, respectively. It follows from (12)–(15) that π_1 is coherent in the sense of §2.3.6, ignoring tests in Q. Similarly, π_2 is coherent, ignoring tests in B. Moreover, π_1 is onto on traces, as shown in the following lemma.

Lemma 3.2 *For every trace $\rho \in \mathsf{Traces}_K$ and every state $U \in Q$, there is a trace $\tau \in \mathsf{Traces}_{K \times M}$ such that $\pi_1(\tau) = \rho$ and $\mathbf{first}(\tau) = (\mathbf{first}(\rho), U)$.*

Proof. For every $(u, v) \in \mathfrak{m}_K(p)$ and $U \in Q$, there is a state $V \in Q$, namely $\delta(U, p)$, such that $((u, U), (v, V)) \in \mathfrak{m}_{K \times M}(p)$. The result follows by induction on the length of ρ. □

Lemma 3.3 *The operations π_1^{-1} and Prefix commute. That is, for all $A \subseteq \mathsf{Traces}_K$, $\pi_1^{-1}(\mathsf{Prefix}(A)) = \mathsf{Prefix}(\pi_1^{-1}(A))$.*

Proof. Let $\sigma \in \mathsf{Traces}_{K \times M}$ and $\rho \in \mathsf{Traces}_K$ such that $\pi_1(\sigma)\rho$ exists. Then $\mathbf{last}(\sigma) = (u, U)$ and $\mathbf{first}(\rho) = u$ for some (u, U). By Lemma 3.2, there is a trace $\tau \in \mathsf{Traces}_{K \times M}$ such that $\mathbf{first}(\tau) = (u, U)$ and $\pi_1(\tau) = \rho$. Thus $\sigma\tau$ exists and $\pi_1(\sigma\tau) = \pi_1(\sigma)\pi_1(\tau) = \pi_1(\sigma)\rho$. From this argument it follows that

$$\begin{aligned}
\sigma \in \pi_1^{-1}(\mathsf{Prefix}(A)) &\Leftrightarrow \pi_1(\sigma) \in \mathsf{Prefix}(A) \\
&\Leftrightarrow \exists\rho\ \pi_1(\sigma)\rho \in A \\
&\Leftrightarrow \exists\tau\ \pi_1(\sigma\tau) \in A \\
&\Leftrightarrow \exists\tau\ \sigma\tau \in \pi_1^{-1}(A) \\
&\Leftrightarrow \sigma \in \mathsf{Prefix}(\pi_1^{-1}(A)).
\end{aligned}$$

Since this holds for arbitrary σ, the result follows. □

A program r over P, B is called *total* in K if $[\![\mathsf{PreComp}(r)]\!]_K \subseteq \mathsf{Prefix}([\![r]\!]_K)$; that is, if every trace of $\mathsf{PreComp}(r)$ in K is a prefix of a trace of r in K.

Lemma 3.4 *If r is total in K, then r is total in the instrumented frame $K \times M$.*

Proof. Suppose $[\![\mathsf{PreComp}(r)]\!]_K \subseteq \mathsf{Prefix}([\![r]\!]_K)$. Since $\pi_1 : \mathsf{Traces}_{K\times M} \to \mathsf{Traces}_K$ is coherent in the sense of §2.3.6, ignoring tests in Q, we have

$$\begin{aligned}
[\![\mathsf{PreComp}(r)]\!]_{K\times M} &= \pi_1^{-1}([\![\mathsf{PreComp}(r)]\!]_K) && \text{by Lemma 2.4} \\
&\subseteq \pi_1^{-1}(\mathsf{Prefix}([\![r]\!]_K)) && \text{since } \pi_1^{-1} \text{ is monotone} \\
&= \mathsf{Prefix}(\pi_1^{-1}([\![r]\!]_K)) && \text{by Lemma 3.3} \\
&= \mathsf{Prefix}([\![r]\!]_{K\times M}) && \text{by Lemma 2.4.}
\end{aligned}$$

□

4 Soundness

Theorem 4.1 (Soundness) *Let $\mathcal{D}$ be any set of premises over P, B, and let r be a program over P, B. If*

$$\mathcal{A}_M \quad\to\quad \mathcal{D} \quad\to\quad S \cdot \mathsf{PreComp}(r) \ \leq\ (\overline{E} \cdot \textstyle\sum P)^*\overline{E} \tag{18}$$

is a theorem of KAT*, then r is safe with respect to M in any Kripke frame K whose regular trace algebra Tr_K satisfies $\mathcal{D}$.*

Proof. Let K be a Kripke frame whose regular trace algebra Tr_K satisfies $\mathcal{D}$. It suffices to show that $\mathsf{Tr}_{K\times M}$, the trace algebra of the instrumented frame $K \times M$, satisfies $\mathcal{A}_M$ and $\mathcal{D}$, because then that model must satisfy the right-hand side of (18), which says exactly that (16) holds.

For premises of the form (6), by (11), the only state of M satisfying the test U is the state U, therefore $[\![UV]\!]_M = \varnothing$. Similarly, for premises of the form (7) and (8), (9) and (10) ensure that UpV is the only trace in $[\![Up]\!]_M$, where $V = \delta(U, p)$, therefore $[\![Up]\!]_M = [\![UpV]\!]_M$. Thus Tr_M satisfies all the equations in $\mathcal{A}_M$. Restricting attention to formulas over P, Q (that is, ignoring tests in B), since π_2 is coherent, by Lemma 2.4 we have a canonical homomorphism π_2^{-1} from Tr_M to $\mathsf{Tr}_{K\times M}$, therefore $\mathsf{Tr}_{K\times M}$ satisfies all the equations in $\mathcal{A}_M$ as well.

Similarly, restricting attention to formulas over P, B (that is, ignoring tests in Q), since π_1 is coherent, by Lemma 2.4 we have a canonical homomorphism π_1^{-1} from Tr_K to $\mathsf{Tr}_{K\times M}$, therefore $\mathsf{Tr}_{K\times M}$ satisfies all equations satisfied by Tr_K; in particular, it satisfies $\mathcal{D}$. □

In practice, this result can be simplified when the program is known to be total. This gives a simpler verification condition that is easier to check, since we do not have to deal with the precomputations of the program explicitly.

Corollary 4.2 *Let $\mathcal{D}$ be any set of premises over P, B, and let r be a program over P, B. If*

$$\mathcal{A}_M \quad \to \quad \mathcal{D} \quad \to \quad Sr \ \leq \ (\overline{E} \cdot \sum P)^* \overline{E} \tag{19}$$

is a theorem of KAT*, then r is safe with respect to M in any Kripke frame K whose regular trace algebra Tr_K satisfies $\mathcal{D}$ such that r is total in K.*

Proof. Let K be a Kripke frame whose regular trace algebra Tr_K satisfies $\mathcal{D}$ such that r is total in K. As in the proof of Theorem 4.1, $\mathsf{Tr}_{K\times M}$ satisfies $\mathcal{A}_M$ and $\mathcal{D}$, so by (19) we have

$$[\![Sr]\!]_{K\times M} \quad \subseteq \quad [\![(\overline{E} \cdot \sum P)^* \overline{E}]\!]_{K\times M}. \tag{20}$$

By Lemma 3.4, r is total in the instrumented frame $K \times M$; that is,

$$[\![\mathsf{PreComp}(r)]\!]_{K\times M} \quad \subseteq \quad \mathsf{Prefix}([\![r]\!]_{K\times M}). \tag{21}$$

Then

$$\begin{aligned}
&[\![S \cdot \mathsf{PreComp}(r)]\!]_{K\times M} \\
&\quad = \quad [\![S]\!]_{K\times M} \cdot [\![\mathsf{PreComp}(r)]\!]_{K\times M} \\
&\quad \subseteq \quad [\![S]\!]_{K\times M} \cdot \mathsf{Prefix}([\![r]\!]_{K\times M}) && \text{by (21) and monotonicity of composition} \\
&\quad = \quad \mathsf{Prefix}([\![Sr]\!]_{K\times M}) && \text{by Lemma 2.1} \\
&\quad \subseteq \quad \mathsf{Prefix}([\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_{K\times M}) && \text{by (20) and monotonicity of } \mathsf{Prefix} \\
&\quad \subseteq \quad [\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_{K\times M} && \text{by (17).}
\end{aligned}$$

This inclusion is just (16), which was to be shown. □

5 Completeness

In this section we prove the converse of Theorem 4.1: if the program r fails to verify, then there is a relational interpretation of r in which the program is unsafe. This is a kind of weak completeness theorem. It is weak in the sense that in real applications, we are typically interested in a particular interpretation, and the theorem does not say that that if r fails to verify, then r is unsafe in the interpretation of interest. However, it does show that the method is as powerful as it can be up to the level of expression represented by propositional Horn logic.

Of course, the completeness theorem can be made arbitrarily strong subject to this restriction by choosing a sufficiently large set of premises $\mathcal{D}$. The larger $\mathcal{D}$, the fewer models are available as counterexamples. Normally the style of reasoning in KAT is to use the specific properties of the interpretation of interest only to establish the validity of the premises $\mathcal{D}$, thereafter reasoning purely propositionally under those assumptions. This result fits well with that view in that it characterizes the strength of the deduction system when it is used in this way.

For completeness, we must also make the added assumption that the premises in $\mathcal{D}$ are of the form $p = 0$ and that $\mathcal{D}$ is finite. This allows us to exploit the completeness result for Hoare formulas over relational and trace models. This is not a strong restriction, since most premises that arise in practice, including all Hoare-style partial correctness assertions, are of this form.

Theorem 5.1 (Completeness) *Let $\mathcal{D}$ be any finite set of premises of the form $p = 0$ over P, B, and let r be a program over P, B. If r is safe with respect to M in all Kripke frames K whose trace algebra Tr_K satisfies $\mathcal{D}$, then* (18) *is a theorem of* KAT.

Proof. We prove the contrapositive. Suppose (18) is not a theorem of KAT. Since KAT is complete for the Hoare theory of relation algebras [12, Theorem 4.1], and since this theory coincides with the Hoare theory of trace algebras (Lemma 2.2), there exists a trace algebra Tr_K in which (18) is not satisfied. Here K is a Kripke frame over atomic actions P and atomic tests $B \cup Q$, where as in the previous section we are assuming $B \cap Q = \varnothing$. Then Tr_K satisfies $\mathcal{A}_M$ and $\mathcal{D}$, but there exists a trace σ of K such that

$$\sigma \quad \in \quad [\![S \cdot \mathsf{PreComp}(r)]\!]_K \;-\; [\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_K.$$

Let

$$g(u) \quad \stackrel{\text{def}}{=} \quad \begin{cases} U, & \text{if } u \in [\![U]\!]_K,\ U \in Q \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

If $g(u)$ exists, then it is unique, since Tr_K satisfies the premises (6). (One could guarantee the existence of $g(u)$ by including the premise $\sum Q = 1$ in $\mathcal{A}_M$; but this is not necessary for the result.) Let K' be the induced subframe of K on the set of states $\{u \in K \mid g(u) \text{ exists}\}$. Since Tr_K satisfies (7) and (8), the states of K' are closed under the actions of P; that is, if $g(u)$ exists and $(u, v) \in \mathfrak{m}_K(p)$, then $g(v)$ exists. Since $\sigma \in [\![S \cdot \mathsf{PreComp}(r)]\!]_K$, we have $\mathbf{first}(\sigma) \in [\![S]\!]_K$, therefore $g(\mathbf{first}(\sigma)) = S$ and $\sigma \in \mathsf{Traces}_{K'}$. By Lemma 2.3(a),

$$\begin{aligned} [\![S \cdot \mathsf{PreComp}(r)]\!]_K \cap \mathsf{Traces}_{K'} \quad &= \quad [\![S \cdot \mathsf{PreComp}(r)]\!]_{K'} \\ [\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_K \cap \mathsf{Traces}_{K'} \quad &= \quad [\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_{K'}, \end{aligned}$$

therefore

$$\sigma \quad \in \quad [\![S \cdot \mathsf{PreComp}(r)]\!]_{K'} \;-\; [\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_{K'}.$$

Now let K'' be the Kripke frame over P, B obtained from K' by ignoring the interpretations of the tests Q. That is, $\mathfrak{m}_{K''}(p) = \mathfrak{m}_{K'}(p)$ and $\mathfrak{m}_{K''}(b) = \mathfrak{m}_{K'}(b)$ for $p \in P$ and $b \in B$, and $\mathfrak{m}_{K''}(U)$ is undefined for $U \in Q$. Let $K'' \times M$ be the instrumented frame. Let $f : K' \to K'' \times M$ be the map $f(u) = (u, g(u))$. It follows easily from the definitions (12)–(15) that the map f is coherent in the sense of §2.3.6. By Lemma 2.4, for any q, $[\![q]\!]_{K'} = f^{-1}([\![q]\!]_{K''\times M})$, therefore

$$\begin{aligned}
\sigma \quad &\in \quad [\![S \cdot \mathsf{PreComp}(r)]\!]_{K'} \; - \; [\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_{K'} \\
&= \quad f^{-1}([\![S \cdot \mathsf{PreComp}(r)]\!]_{K''\times M}) \; - \; f^{-1}([\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_{K''\times M}) \\
&= \quad f^{-1}([\![S \cdot \mathsf{PreComp}(r)]\!]_{K''\times M} \; - \; [\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_{K''\times M}),
\end{aligned}$$

or in other words,

$$f(\sigma) \quad \in \quad [\![S \cdot \mathsf{PreComp}(r)]\!]_{K''\times M} \; - \; [\![(\overline{E} \cdot \textstyle\sum P)^* \overline{E}]\!]_{K''\times M}.$$

Thus r is not safe in K'' with respect to M.

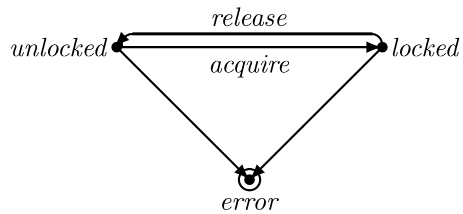
It remains to argue that $\mathsf{Tr}_{K''}$ satisfies $\mathcal{D}$. But $\mathsf{Tr}_{K''}$ satisfies any equation over P, B satisfied by Tr_K, since the map $[\![p]\!]_K \mapsto [\![p]\!]_{K'}$ is a homomorphism $\mathsf{Tr}_K \to \mathsf{Tr}_{K'}$ by Lemma 2.3(a), and $[\![p]\!]_{K'} = [\![p]\!]_{K''}$ for any p over P, B by definition. □

6 An Example

The following example is from [3]. It is a code fragment from a device driver consisting of a loop that alternately acquires and releases a lock on a resource.

```
do {
   KeAcquireSpinLock();
   nPacketsOld = nPackets;
   if (request) {
      request = request->Next;
      KeReleaseSpinLock();
      nPackets++;
   }
} while (nPackets != nPacketsOld);
KeReleaseSpinLock();
```

If the driver currently holds the lock and attempts to reacquire it, or if the driver does not currently hold the lock and attempts to release it, the driver will hang. Hence a precondition for safe execution of any acquire (respectively, release) operation is that the driver does not (respectively, does) currently hold the lock. The correct behavior is specified by the following automaton.



The start state is the unlocked state.

Using the propositional abbreviations

kA	`KeAcquireSpinLock()`
kR	`KeReleaseSpinLock()`
n	`nPacketsOld = nPackets`
u	`request = request->Next`
m	`nPackets++`
R	`request`
B	`nPackets == nPacketsOld`

and the KAT encoding of the control structures

$$\begin{aligned}\texttt{do } p \texttt{ while } C &= p;(C;p)^*;\overline{C}\\ \texttt{if } C \texttt{ then } p &= C;p+\overline{C},\end{aligned}$$

the program above becomes

$$kA;n;(R;u;kR;m+\overline{R});(\overline{B};kA;n;(R;u;kR;m+\overline{R}))^*;B;kR \qquad (22)$$

Let A be a new test representing the assertion that the driver is in the locked state. The precondition for safe execution of kA (acquire) is $\overline{A}$ (unlocked), and the precondition for safe execution of kR (release) is A (locked). In addition, the resource is initially unlocked, so $\overline{A}$ is a global precondition.

To specify the correctness conditions, we can annotate the program above by inserting before every occurrence of a critical operation kA or kR its precondition for safe execution. This gives the annotated program

$$\overline{A};kA;n;(R;u;A;kR;m+\overline{R});(\overline{B};\overline{A};kA;n;(R;u;A;kR;m+\overline{R}))^*;B;A;kR \qquad (23)$$

The program is otherwise identical to (22).

To argue that the program is safe, we wish to show that the unannotated program (22) and the annotated program (23) are equivalent. This would say that the precondition for safe execution of any critical operation is guaranteed to hold immediately before that operation, so that transition to the error state is impossible. These preconditions become true in the first place by executing other operations at other points in the program. For example, execution of a kA (acquire) operation acquires the lock, therefore causes A to become true immediately afterward. This assumption takes the form of a premise $kA = kA;A$. Other operations such as n or u that do not affect the truth value of

A commute with it, giving rise to commutativity conditions $A; n = n; A$ and $A; u = u; A$. For this particular task, we postulate the following premises.

$$\begin{aligned} kA &= kA; A && (24) \\ kR &= kR; \overline{A} && (25) \\ B; m &= B; m; \overline{B} && (26) \\ n &= n; B && (27) \\ A; n &= n; A && (28) \\ A; u &= u; A && (29) \\ A; m &= m; A && (30) \\ B; u &= u; B && (31) \\ B; kR &= kR; B && (32) \end{aligned}$$

These conditions have the following meanings. Condition (24) says that acquiring the lock acquires it. Condition (25) says that releasing the lock releases it. Condition (26) says that if two integer variables are equal and we increment one, then they are no longer equal. Condition (27) says that assigning the value of one variable to another makes them equal. Conditions (28)–(32) are commutativity conditions. They say that the execution of the specified atomic action does not affect the truth of the specified test. All these premises are self-evident and follow from basic properties of the domain of computation.

Let $\mathcal{E}$ be the conjunction of (24)–(32). These are our premises. To show that the program is safe, we wish to prove in KAT that the Horn formula

$$\mathcal{E} \quad \to \quad \overline{A}; p \leq \overline{A}; q \tag{33}$$

holds, where p and q are the unannotated and annotated programs (22) and (23), respectively. The prefix $\overline{A}$ in both programs is the global precondition that states that initially the driver does not have the lock. This annotation is necessary, because the first critical operation performed by the driver is an acquire operation kA.

The formula (33) has been formally verified using the KAT-ML theorem prover [1].

7 Future Work

The structure of security automata can be generalized to handle other types of static analysis. For example, let L be an upper semilattice such that all ascending chains are finite. This is called the *ascending chain condition* (ACC). Elements are called *types* or *abstract values*. Associated with each atomic action p is a strict, partial, finitely additive map $f_p : L \to L$ called its *transfer function*. The propagation of type information can be coded in KAT by premises $Xp \leq pf_p(X)$. The ACC needed for *. It seems clear Java bytecode verification can be handled in this way. We leave this question for future investigation.

Acknowledgements

Thanks to Kamal Aboul-Hosn and Fred B. Schneider for valuable comments. 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 author 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] Kamal Aboul-Hosn and Dexter Kozen. KAT-ML: An interactive theorem prover for Kleene algebra with tests. In Boris Konev and Renate Schmidt, editors, *Proc. 4th Int. Workshop on the Implementation of Logics (WIL'03)*, pages 2–12. University of Manchester, September 2003.

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

[3] Thomas Ball and Sriram K. Rajamani. The SLAM project: Debugging system software via static analysis. In *Proc. Conf. Principles of Programming Languages (POPL'02)*, pages 1–3. ACM, January 2002.

[4] Adam Barth and Dexter Kozen. Equational verification of cache blocking in LU decomposition using Kleene algebra with tests. Technical Report 2002-1865, Computer Science Department, Cornell University, June 2002.

[5] Ernie Cohen. Lazy caching in Kleene algebra. `http://citeseer.nj.nec.com/22581.html`.

[6] Ernie Cohen. Hypotheses in Kleene algebra. Technical Report TM-ARH-023814, Bellcore, 1993. `http://citeseer.nj.nec.com/1688.html`.

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

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

[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. Some results in dynamic model theory. Technical Report 2002-1882, Computer Science Department, Cornell University, October 2002. Science of Computer Programming, to appear.

[14] 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.

[15] Dexter Kozen and Jerzy Tiuryn. Substructural logic and partial correctness. *Trans. Computational Logic*, 4(3):355–378, July 2003.

[16] Fred B. Schneider. Enforceable security policies. *ACM Trans. Information and System Security*, 3(1):30–50, February 2000.