In semantics and logics of programs, Kleene algebra forms an essential component of Propositional Dynamic Logic (PDL) [10], in which it is mixed with Boolean algebra and modal logic to give a simple yet powerful system for modeling and reasoning about computation at the propositional level.

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

From a practical point of view, many simple program manipulations, such as loop unwinding and basic safety analysis, do not require the full power of PDL, but can be carried out in a purely equational subsystem using the axioms of Kleene algebra. However, tests are an essential ingredient, since they are needed to model conventional programming constructs such as conditional tests and while loops. We define here a variant of Kleene algebra, called *Kleene algebra with tests* (KAT), for reasoning equationally with these constructs. KAT was introduced in [14, 15].

KAT has been applied successfully in a number of low-level verification tasks involving communication protocols, basic safety analysis, concurrency control, and local compiler optimizations [1, 6, 7, 17]. A useful feature of KAT in this regard is its ability to accommodate certain basic equational assumptions regarding the interaction of atomic instructions and tests. This feature makes KAT ideal for reasoning about the correctness of low-level code transformations.

Definition of Kleene Algebra with Tests

A *Kleene algebra with tests* is a two-sorted algebra

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

where $B \subseteq K$ and $\bar{}$ is a unary operator defined only on B, such that

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

The elements of B are called *tests*. We reserve the letters $p, q, r, \ldots$ for arbitrary elements of K and $a, b, c, \ldots$ for tests. In PDL, a test would be written $b?$, but since we are using different symbols for tests we can omit the ?.

The axioms for Kleene algebra were given in an earlier lecture. The axioms for Boolean algebra are just the laws of propositional logic. Expressed in our language with $+$ for join (disjunction, "or", $\vee$), $\cdot$ for meet (conjunction, "and", $\wedge$), and $\bar{}$ for negation (complement, "not", $\neg$), they are

$$\begin{array}{llll}
a + (b + c) = (a + b) + c & a(bc) = (ab)c & a + b = b + a & ab = ba \\
a(b + c) = ab + ac & a + bc = (a + b)(a + c) & a + 0 = a & a1 = a \\
a0 = 0 & a + 1 = 1 & a + a = a & aa = a \\
a\overline{a} = 0 & a + \overline{a} = 1 & \overline{a}\,\overline{b} = \overline{a + b} & \overline{a} + \overline{b} = \overline{ab} \\
\overline{\overline{a}} = a. & & &
\end{array}$$

The definition of a KAT is deceptively simple, but actually carries a lot of information in a concise package. Note the overloading of the operators $+, \cdot, 0, 1$, each of which plays two roles: applied to arbitrary elements of 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.

It follows immediately from the definition that $a \leq 1$ for all $a \in B$, where $\leq$ is the natural order in the Kleene algebra (this is the axiom $a+1=1$ of Boolean algebra). It is tempting to define tests in an arbitrary Kleene algebra to be the set $\{p \in K \mid p \leq 1\}$. This is the approach taken by [5]. This approach makes some sense in algebras of binary relations [19, 21], but it does not work in all Kleene algebras. For example, it rules out the $(\min, +)$ Kleene algebra described in an earlier lecture. In this algebra, $p \leq 1$ for all p, but the idempotence law $pp = p$ fails, so the set $\{p \in K \mid p \leq 1\}$ does not form a Boolean algebra. In our approach, every Kleene algebra extends trivially to a Kleene algebra with tests by taking the two-element Boolean algebra $\mathbb{2} = \{0, 1\}$. Of course, there are more interesting models as well.

However, there is a more serious issue. Even in algebras of binary relations, the approach of [5] forces us to consider all elements $p \leq 1$ as tests, including conditions that in practice would not normally be considered testable. For example, there may be programs p whose input/output relations have no side effects—that is, $p \leq 1$—but the associated test succeeds iff p halts, which in general is undecidable. We intend tests to be viewed as simple predicates that are easily recognizable as such and that are immediately decidable in a given state (and whose complements are therefore also immediately decidable). Having an explicit Boolean subalgebra allows this.

While Programs

We will work with a Pascal-like programming language with sequential composition $p\,;\,q$, a conditional test $\mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q$, and a looping construct $\mathsf{while}\ b\ \mathsf{do}\ p$. Programs built inductively from atomic programs and tests using these constructs are called *while programs*.

We occasionally omit the else clause of a conditional test. This can be considered an abbreviation for a conditional test with the dummy else clause skip, which is semantically the same as 1 (true).

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

$$p\,;\,q = pq \qquad \mathsf{if}\ b\ \mathsf{then}\ p = bp + \overline{b}$$
$$\mathsf{while}\ b\ \mathsf{do}\ p = (bp)^*\overline{b} \qquad \mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q = bp + \overline{b}q.$$

These definitions agree with the traditional input-output semantics of while programs in relational models. For example, the meaning of $\mathsf{while}\ b\ \mathsf{do}\ p$ is traditionally described as the least solution x of the equation

$$x = \mathsf{if}\ b\ \mathsf{then}\ p\,;\,x\ \mathsf{else\ skip},$$

or equivalently, the least x such that

$$\mathsf{if}\ b\ \mathsf{then}\ p\,;\,x\ \mathsf{else\ skip} \leq x.$$

Translating to the language of KAT, this just says that $(bp)^*\overline{b}$ is the least x such that $bpx + \overline{b} \leq x$, which is an immediate consequence of the KA axioms.

Conditions

We will often need to reason in the presence of extra assumptions. Recall that KA and KAT are Horn theories, and these extra assumptions take the form of equational premises to the left of the implication symbol in a Horn formula.

There are several common forms of premises that arise in applications. The most common form is a commutativity condition $pq = qp$ or $pb = bp$. A commutativity condition between two atomic programs p and q expresses the fact that the two operations represented by p and q do not interfere with each other and may be done in either order. For example, if p and q are the assignments $x := 3$ and $y := 4$ respectively, then they do not affect each other. A commutativity condition $pb = bp$, where p is a program and b is a test, indicates that the execution of p does not affect the truth value of b. For example, if p is the assignment $x := 3$ and b is the test $y = 4$, then execution of p has no effect on the truth value of b. One can perform the test either before or after the assignment and the result is the same.

Another common assumption comes in the form of a pair of equations $p = pb$ and $bp = b$. The first equation indicates that the execution of p causes b to become true, thus testing b after executing p is redundant. The second equation indicates that if the sole purpose of p is to make b true, then there is no need to execute p if b is already true. For example, if p is the assignment $x := 3$ and b is the test $x = 3$, then these conditions hold: executing p makes b true, and if b is already true, there is no need to execute p.

This is useful for example when we want to eliminate a redundant assignment. We can use the first equation $p = pb$ to generate the condition b in a KAT expression, then use commutativity conditions to move b around until it comes up against another p, then use the second equation $bp = b$ to eliminate the second occurrence of p, then move the b back again and eliminate it by applying $p = pb$ in the opposite direction.

It stands to reason that if p does not affect b, then neither should it affect $\overline{b}$. This is indeed the case. Thus $pb = bp$ should be equivalent to $p\overline{b} = \overline{b}p$. More generally,

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

(i) $bp = pc$

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

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

Proof. Since $\overline{\overline{b}} = b$ for any test b, by symmetry it suffices to show the equivalence of (i) and (iii). Assuming (i), we have

$$bp\overline{c} + \overline{b}pc = pc\overline{c} + \overline{b}bp = p0 + 0p = 0.$$

Conversely, assuming (iii), we have $bp\overline{c} = \overline{b}pc = 0$. Then

$$bp = bp(c + \overline{c}) = bpc + bp\overline{c} = bpc + 0 = bpc + \overline{b}pc = (b + \overline{b})pc = pc. \qquad \square$$

Of course, any pair of tests commute; that is, $bc = cb$. This is an axiom of Boolean algebra.

Models of KAT

Here we introduce the syntax of Kleene algebra with tests (KAT), along with several useful interpretations. Every KA extends trivially to a KAT by taking the Boolean subalgebra to be $2 = \{0, 1\}$, but there are much more interesting ones. We will describe a family of language models consisting of regular sets of *guarded strings*, which play the same role in KAT that the regular sets of strings play in Kleene algebra. We will also discuss relational models. These results are from [15, 18].

KAT Syntax

Let P and B be disjoint finite sets of symbols. Elements of P are called *primitive actions* and elements of B are called *primitive tests*. *Action terms* and *Boolean terms* are defined inductively:

- any primitive action p is an action term
- any primitive test b is a Boolean term
- 0 and 1 are Boolean terms
- if p and q are action terms, then so are $p + q$, pq, and p^* (suitably parenthesized if necessary)
- if b and c are Boolean terms, then so are $b + c$, bc, and $\overline{b}$ (suitably parenthesized if necessary)
- any Boolean term is an action term.

Expressed as a grammar,

$$p ::= p \in \mathsf{P} \mid p + q \mid pq \mid p^* \mid b$$
$$b ::= a \in \mathsf{B} \mid 0 \mid 1 \mid b + c \mid bc \mid \overline{b}.$$

The set of all terms over P and B is denoted $\mathsf{Exp}\,\mathsf{P},\mathsf{B}$. The set of all Boolean terms over B is denoted $\mathsf{Exp}\,\mathsf{B}$.

An *interpretation* over a KAT K is any homomorphism (function commuting with the distinguished operations and constants) defined on $\mathsf{Exp}\,\mathsf{P},\mathsf{B}$ and taking values in K such that the Boolean terms are mapped to elements of the distinguished Boolean subalgebra.

If K is a KAT and I is an interpretation over K, we write $K, I \vDash \varphi$ if the formula φ holds in K under the interpretation I according to the usual semantics of first-order logic.

We write $\mathsf{KAT} \vDash \varphi$ if the formula φ is a logical consequence the axioms of KAT, that is, if φ holds under all interpretations over Kleene algebras with tests. We write $\mathsf{KAT}^* \vDash \varphi$ if φ holds under all interpretations over star-continuous Kleene algebras with tests. The only formulas we consider are equations or equational implications (universal Horn formulas).

Language Models

Let P and B be disjoint finite sets of symbols. Our language-theoretic models of Kleene algebras with tests are based on the idea of *guarded strings* over P and B. Guarded strings were first studied in [13].

We obtain a guarded string from a string $x \in \mathsf{P}^*$ by inserting *atoms* interstitially among the symbols of x. An *atom* is a Boolean expression representing an atom (minimal nonzero element) of the free Boolean algebra on generators B.

Formally, an *atom* of $\mathsf{B} = \{b_1, \ldots, b_k\}$ is a product of literals $c_1 \cdots c_k$, where each $c_i \in \{b_i, \overline{b}_i\}$. This assumes an arbitrary but fixed order $b_1 < b_2 < \cdots < b_k$ on B. For technical reasons, we require the literals in an atom appearing in a guarded string to occur in this order. There are exactly 2^k atoms, and they are in one-to-one correspondence with the truth assignments to B. We denote atoms of B by $\alpha, \beta, \alpha_0, \ldots$. The set of all atoms of B is denoted At_B, or just At when B is understood. The set At will turn out to be the multiplicative identity of our language-theoretic models.

If $b \in \mathsf{B}$ and α is an atom of B, we write $\alpha \leq b$ if b occurs positively in α and $\alpha \leq \overline{b}$ if b occurs negatively in α. This notation is consistent with the natural order in the free Boolean algebra generated by B. Equivalently, $\alpha \leq b$ iff the truth assignment corresponding to α assigns 1 (true) to b.

Intuitively, the action symbols can be thought of as atomic instructions to be executed and atoms as conditions that must be satisfied at some point in the computation. If $\alpha \leq c_i$, then α asserts that c_i holds (and $\overline{c}_i$ fails) at that point in the computation.

A *guarded string* over P and B is any element of $(\mathsf{At} \cdot \mathsf{P})^* \cdot \mathsf{At}$; that is, any string of the form

$$\alpha_0 p_1 \alpha_1 p_2 \alpha_2 \cdots \alpha_{n-1} p_n \alpha_n, \ n \geq 0,$$

where each $\alpha_i \in \mathsf{At}$ and each $p_i \in \mathsf{P}$. Note that every guarded string x begins and ends with an atom, which we call $\mathsf{first}\, x$ and $\mathsf{last}\, x$, respectively. Thus if x is the guarded string above, then $\mathsf{first}\, x = \alpha_0$ and $\mathsf{last}\, x = \alpha_n$. In the case $n = 0$, x just consists of a single atom α_0, and $\mathsf{first}\, x = \mathsf{last}\, x = \alpha_0$.

The set of all guarded strings over P and B is denoted $\mathsf{GS}_{\mathsf{P},\mathsf{B}}$, or just GS when P and B are understood.

Let $\overline{\mathsf{B}} = \{\overline{b} \mid b \in \mathsf{B}\}$. We denote strings in $(\mathsf{P} \cup \mathsf{B} \cup \overline{\mathsf{B}})^*$, including guarded strings, by the letters $x, y, z, x_1, \ldots$.

The analog of concatenation for guarded strings is *fusion product*, also sometimes called *coalesced product*. It is a *partial* binary operation $\diamond$ on GS defined as follows:

$$x\alpha \diamond \beta y = \begin{cases} x\alpha y, & \text{if } \alpha = \beta, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

That is, if $\mathsf{last}\, x = \mathsf{first}\, y$, then the fusion product $x \diamond y$ exists and is equal to the string obtained by concatenating x and y, but writing the mediating atom $\mathsf{last}\, x = \mathsf{first}\, y$ only once between them; otherwise, $x \diamond y$ is undefined.

For example, if $\mathsf{B} = \{b, c\}$ and $\mathsf{P} = \{p, q\}$, then

$$bcp\overline{b}c \diamond \overline{b}cq\overline{b}\overline{c} = bcp\overline{b}cq\overline{b}\overline{c}.$$

The operation $\diamond$ is associative in the sense that $(x \diamond y) \diamond z$ is defined iff $x \diamond (y \diamond z)$ is defined, and if both are defined, then they are equal.

If $A, B \subseteq \mathsf{GS}$, define

$$A \cdot B \stackrel{\text{def}}{=} \{x \diamond y \mid x \in A, \ y \in B, \ \mathsf{last}\, x = \mathsf{first}\, y\}.$$

We will tend to elide the $\cdot$ and write just AB. Thus AB consists of all existing fusion products of guarded strings in A with guarded strings in B. For example, if $\mathsf{B} = \{b, c\}$, $\mathsf{P} = \{p, q\}$, and

$$A = \{bcp\overline{b}\overline{c}, \overline{b}\overline{c}, bcqb\overline{c}\} \qquad\qquad B = \{\overline{b}\overline{c}pbc, \overline{b}\overline{c}, b\overline{c}qbc\},$$

then

$$AB = \{bcp\overline{b}\overline{c}pbc, bcp\overline{b}\overline{c}, \overline{b}\overline{c}pbc, \overline{b}\overline{c}, bcqb\overline{c}qbc\}.$$

Note that, whereas the operation $\diamond$ on guarded strings is a partial operation, the set-theoretic product $\cdot$ on sets of guarded strings defined in terms of $\diamond$ is a total operation. If there are no existing fusion products of strings from A and B, then $AB = \varnothing$. It is not difficult to show that $\cdot$ is associative, that it distributes over union, and that it has two-sided identity At.

For $A \subseteq \mathsf{GS}$, define inductively

$$A^0 \stackrel{\text{def}}{=} \mathsf{At} \qquad\qquad A^{n+1} \stackrel{\text{def}}{=} AA^n.$$

The asterate operation for sets of guarded strings is defined by

$$A^* \stackrel{\text{def}}{=} \bigcup_{n \geq 0} A^n.$$

Let $^-$ denote set complementation in At. That is, if $A \subseteq \mathsf{At}$, then $\overline{A} = \mathsf{At} - A$.

We now define a language-theoretic model $\mathsf{Reg}\,\mathsf{P},\mathsf{B}$ based on guarded strings. The elements of $\mathsf{Reg}\,\mathsf{P},\mathsf{B}$ will be the regular sets of guarded strings over P and B (although we have not yet defined *regular* in this context). We will also give a standard interpretation $G : \mathsf{Exp}\,\mathsf{P},\mathsf{B} \to \mathsf{Reg}\,\mathsf{P},\mathsf{B}$ analogous to the standard interpretation of regular expressions as regular sets.

Consider the structure

$$(2^{\mathsf{GS}}, 2^{\mathsf{At}}, \cup, \cdot, ^*, ^-, \varnothing, \mathsf{At}),$$

which we denote briefly by 2^{GS}. It is quite straightforward to verify that this is a star-continuous Kleene algebra with tests; that is, it is a model of KAT^*. The Boolean algebra axioms hold for 2^{At} because it is a set-theoretic Boolean algebra.

The star-continuity condition follows immediately from the definition of * and the distributivity of $\cdot$ over arbitrary union. Since

$$B^* = \bigcup_{n\geq 0} B^n,$$

we have that

$$AB^*C = A(\bigcup_{n\geq 0} B^n)C = \bigcup_{n\geq 0} AB^nC.$$

Both of these expressions denote the set

$$\bigcup_{n\geq 0} \{x \diamond y \diamond z \mid x \in A,\ y \in B^n,\ z \in C,\ \mathsf{last}\, x = \mathsf{first}\, y,\ \mathsf{last}\, y = \mathsf{first}\, z\}.$$

Standard Interpretation

For $p \in \mathsf{P}$ and $b \in \mathsf{B}$, define

$$G(p) \stackrel{\text{def}}{=} \{\alpha p \beta \mid \alpha, \beta \in \mathsf{At}\} \qquad\qquad G(b) \stackrel{\text{def}}{=} \{\alpha \in \mathsf{At} \mid \alpha \leq b\}. \tag{1}$$

The structure $\mathsf{Reg}\,\mathsf{P},\mathsf{B}$ is defined to be the subalgebra of 2^{GS} generated by the elements $G(p)$ for $p \in \mathsf{P}$ and $G(b)$ for $b \in \mathsf{B}$. Elements of $\mathsf{Reg}\,\mathsf{P},\mathsf{B}$ are called *regular sets*.

The map G defined on primitive actions and primitive tests in (1) extends uniquely to a homomorphism $G : \mathsf{Exp}\,\mathsf{P},\mathsf{B} \to \mathsf{Reg}\,\mathsf{P},\mathsf{B}$:

$$G(p+q) \stackrel{\text{def}}{=} G(p) \cup G(q) \qquad G(1) \stackrel{\text{def}}{=} \mathsf{At} \qquad G(\bar{b}) \stackrel{\text{def}}{=} \mathsf{At} \setminus G(b)$$
$$G(pq) \stackrel{\text{def}}{=} G(p)G(q) \qquad G(0) \stackrel{\text{def}}{=} \varnothing \qquad G(p^*) \stackrel{\text{def}}{=} G(p)^*.$$

The map G is called the *standard interpretation* over $\mathsf{Reg}\,\mathsf{P},\mathsf{B}$.

Relational Models

Relational Kleene algebras with tests are interesting because they closely model our intuition about programs. In a relational model, the elements of K are binary relations and $\cdot$ is interpreted as relational composition. Elements of the Boolean subalgebra are subsets of the identity relation.

Formally, a *relational Kleene algebra with tests* on a set X is any structure

$$(K, B, \cup, ;, ^*, ^-, \varnothing, \mathsf{id})$$

such that $(K, \cup, ;, ^*, \varnothing, \mathsf{id})$ is a Kleene algebra of binary relations on X, in which ; is ordinary relational composition, * is reflexive transitive closure, and id is the identity relation on X; and $(B, \cup, ;, ^-, \varnothing, \mathsf{id})$ is a Boolean algebra of subsets of the identity relation id (not necessarily the whole powerset).

All relational Kleene algebras with tests are star-continuous. We write $\mathsf{REL} \vDash \varphi$ if the formula φ holds in all relational Kleene algebras in the usual sense of first-order logic.

KAT and Hoare Logic

In this segment we will show that KAT subsumes propositional Hoare logic (PHL). Thus the specialized syntax and deductive apparatus of Hoare logic are inessential and can be replaced by simple equational reasoning. Moreover, all relationally valid Hoare-style inference rules are derivable in KAT (this is false for PHL). In a future lecture we will show that deciding the relational validity of such rules is PSPACE-complete. These results are from [16, 18].

Hoare logic, introduced by C. A. R. Hoare in 1969 [12], 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 hundreds of technical articles [4,8,9]. For this achievement among others, Hoare received the Turing Award in 1980. Comprehensive introductions to Hoare logic can be found in [2, 3, 9].

Hoare logic uses a specialized syntax involving *partial correctness assertions* (PCAs) of the form $\{b\}p\{c\}$ and a deductive apparatus consisting of a system of specialized rules of inference. Under certain conditions, these rules are relatively complete [8]; essentially, the propositional fragment of the logic can be used to reduce partial correctness assertions to static assertions about the underlying domain of computation.

The propositional fragment of Hoare logic, called *propositional Hoare logic* (PHL), is subsumed by KAT. 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. More generally, all Hoare-style inference rules of the form

$$\frac{\{b_1\}p_1\{c_1\} \qquad \dots \qquad \{b_n\}p_n\{c_n\}}{\{b\}p\{c\}} \tag{2}$$

that are valid over relational models are derivable in KAT; this is trivially false for PHL.

Encoding of While Programs and Partial Correctness Assertions

The encoding of the while programming constructs using the regular operators and tests originated with propositional Dynamic Logic (PDL) [10]. KAT is strictly less expressive than PDL, but is simpler and purely equational, and is only PSPACE-complete, whereas PDL is EXPTIME-complete. In addition, PDL interpretations are restricted to relational models.

Halpern and Reif [11] prove PSPACE-completeness of strict deterministic PDL, but neither the upper nor the lower bound of the KAT PSPACE-completeness result follows from theirs. Not only are PDL semantics restricted to relational models, but the arguments of [11] 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 EXPTIME-hard. In contrast, KAT imposes no such restrictions.

Hoare Logic

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 $\mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q$, sequential composition $p\,;\,q$, and a looping construct $\mathsf{while}\ b\ \mathsf{do}\ p$.

The encoding of the while program constructs in KAT is as in PDL [10]:

$$p\,;\,q \stackrel{\text{def}}{=} pq \tag{3}$$

$$\mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q \stackrel{\text{def}}{=} bp + \overline{b}q \tag{4}$$

$$\mathsf{while}\ b\ \mathsf{do}\ p \stackrel{\text{def}}{=} (bp)^*\overline{b}. \tag{5}$$

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

$$\{b\}p\{c\}, \tag{6}$$

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.

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.

Semantically, programs p in Hoare logic and dynamic logic (DL) are usually interpreted as binary input/output relations p^M on a domain of computation M, and assertions are interpreted as subsets of M [8,20]. The definition of the relation p^M is inductive on the structure of p; for example, $(p\,;\,q)^M = p^M\,;\,q^M$, the ordinary relational composition of the relations corresponding to p and q. The meaning of the PCA (6) is the same as the meaning of the DL formula $b \Rightarrow [p]c$, where $\Rightarrow$ is ordinary propositional implication and the modal construct $[p]c$ is interpreted in the model M as the set of states s such that for all $(s,t) \in p^M$, 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 traditional Hoare inference rules are:

Assignment rule

$$\{b\{e/x\}\}\,x := e\{b\} \tag{7}$$

Composition rule

$$\frac{\{b\}p\{c\} \qquad \{c\}q\{d\}}{\{b\}p\,;\,q\{d\}} \tag{8}$$

Conditional rule

$$\frac{\{b \wedge c\}p\{d\} \qquad \{\neg b \wedge c\}q\{d\}}{\{c\}\,\mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q\{d\}} \tag{9}$$

While rule

$$\frac{\{b \wedge c\}p\{c\}}{\{c\}\,\mathsf{while}\ b\ \mathsf{do}\ p\{\neg b \wedge c\}} \tag{10}$$

Weakening rule

$$\frac{b' \Rightarrow b \qquad \{b\}p\{c\} \qquad c \Rightarrow c'}{\{b'\}p\{c'\}} \tag{11}$$

Cook [8] showed that these rules are complete relative to first-order number theory when interpreted over the structure of arithmetic $\mathbb{N}$.

Propositional 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 binary relations on a set M and atomic propositions are interpreted as subsets of M. The deduction system of PHL consists of the composition, conditional, while, and weakening rules (8)–(11) and propositional logic. The assignment rule (7) 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.

In PHL, we are concerned with the problem of determining the validity of rules of the form

$$\frac{\{b_1\}p_1\{c_1\} \qquad \dots \qquad \{b_n\}p_n\{c_n\}}{\{b\}p\{c\}} \tag{12}$$

over relational interpretations. The premises $\{b_i\}p_i\{c_i\}$ take the place of the assignment rule (7) and are an essential part of the formulation.

Encoding Hoare Logic in KAT

The propositional Hoare rules can be derived as theorems of KAT. The PCA $\{b\}p\{c\}$ is encoded in KAT in any one of the following four equivalent ways:

Lemma 2. *The following are equivalent:*

(i) $bp\bar{c} = 0$

(ii) $bp = bpc$

(iii) $bp \leq bpc$

(iv) $bp \leq pc$

The proof is easy, and we leave it as an exercise. Intuitively, the formulation (i) says that the program p with preguard b and postguard $\bar{c}$ has no halting execution, and the formulation (ii) says that testing c after executing bp is always redundant.

Using the encoding of while programs (3)–(5) and Lemma 2(ii), the Hoare rules (8)–(11) take the following form:

Composition rule	$bp = bpc \wedge cq = cqd \;\Rightarrow\; bpq = bpqd$	(13)
Conditional rule	$bcp = bcpd \wedge \bar{b}cq = \bar{b}cqd \;\Rightarrow\; c(bp + \bar{b}q) = c(bp + \bar{b}q)d$	(14)
While rule	$bcp = bcpc \;\Rightarrow\; c(bp)^*\bar{b} = c(bp)^*\bar{b}\,\bar{b}c$	(15)
Weakening rule	$b' \leq b \wedge bp = bpc \wedge c \leq c' \;\Rightarrow\; b'p = b'pc'.$	(16)

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 the KAT encodings (13)–(16) of the Hoare rules (8)–(11) are theorems of KAT.

Theorem 3. *The universal Horn formulas (13)–(16) are theorems of* KAT.

Proof. First we derive the composition rule (13). Assuming the premises

$$bp = bpc \tag{17}$$
$$cq = cqd, \tag{18}$$

we have

$$\begin{aligned} bpq &= bpcq && \text{by (17)} \\ &= bpcqd && \text{by (18)} \\ &= bpqd && \text{by (17).} \end{aligned}$$

Thus the implication (13) holds.

For the conditional rule (14), assume the premises

$$bcp = bcpd \tag{19}$$
$$\bar{b}cq = \bar{b}cqd. \tag{20}$$

Then

$$\begin{aligned} c(bp + \bar{b}q) &= cbp + c\bar{b}q && \text{by distributivity} \\ &= bcp + \bar{b}cq && \text{by commutativity of tests} \\ &= bcpd + \bar{b}cqd && \text{by (19) and (20)} \\ &= cbpd + c\bar{b}qd && \text{by commutativity of tests} \\ &= c(bp + \bar{b}q)d && \text{by distributivity.} \end{aligned}$$

For the while rule (15), by trivial simplifications it suffices to show

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

Assume

$$cbp \leq cbpc. \tag{21}$$

By an axiom of KA, we need only show

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

But

$$\begin{aligned} c + c(bp)^*cbp &\leq c + c(bp)^*cbpc && \text{by (21) and monotonicity} \\ &\leq c1c + c(bp)^*cbpc && \text{by Boolean algebra} \\ &\leq c(1 + (bp)^*cbp)c && \text{by distributivity} \\ &\leq c(1 + (bp)^*bp)c && \text{by monotonicity} \\ &\leq c(bp)^*c && \text{by unwinding.} \end{aligned}$$

Finally, for the weakening rule (16), we can rewrite the rule as

$$b' \leq b \wedge bp\bar{c} = 0 \wedge \bar{c}' \leq \bar{c} \;\Rightarrow\; b'p\bar{c}' = 0,$$

which follows immediately from the monotonicity of multiplication. □

References

[1] Allegra Angus and Dexter Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, Computer Science Department, Cornell University, July 2001.

[2] Krzysztof R Apt. Ten years of Hoare's logic: A survey–Part I. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 3(4):431–483, 1981.

[3] Krzysztof R. Apt. Ten years of Hoare's logic: A survey–Part II: Nondeterminism. *Theoretical Computer Science*, 28(1):83–109, 1983.

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

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

[6] Ernie Cohen. Lazy caching in Kleene algebra, 1994. `http://citeseer.nj.nec.com/22581.html`.

[7] Ernie Cohen. Using Kleene algebra to reason about concurrency control. Technical report, Telcordia, Morristown, N.J., 1994.

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

[9] 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.

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

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

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

[13] Donald M. Kaplan. Regular expressions and the equivalence of programs. *J. Comput. Syst. Sci.*, 3:361–386, 1969.

[14] 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.

[15] Dexter Kozen. Kleene algebra with tests. *ACM Trans. Programming Languages and Systems (TOPLAS'97)*, 19(3):427–443, May 1997.

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

[17] 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 (CL'00)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 568–582, London, July 2000. Springer-Verlag.

[18] 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.

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

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

[21] A. Tarski. On the calculus of relations. *J. Symbolic Logic*, 6(3):73–89, 1941.