On Hoare Logic, Kleene Algebra, and Types

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

July 30, 1999

Abstract

We show that propositional Hoare logic is subsumed by the type calculus of typed Kleene algebra augmented with subtypes and typecasting. Assertions are interpreted as typecast operators. Thus Hoare-style reasoning with partial correctness assertions reduces to typechecking in this system.

1 Introduction

In previous work [18, 19], we have shown that Kleene algebra with tests (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. We have also shown that KAT provides a complete deductive system for Hoare-style inference rules involving partial correctness assertions and that PHL is *PSPACE*-complete.

In other recent work [17], we have introduced a simple and natural type system for Kleene algebra (KA) in which objects have types $\mathsf{s} \to \mathsf{t}$. The use of types was motivated by the desire to handle nonsquare matrices, although there are other useful interpretations.

In this note we extend the type system of KA to KAT by adding rules for subtypes and typecasting. Tests are interpreted as typecast operators. We then observe that a Hoare partial correctness assertion $\{b\}\ p\ \{c\}$ can be regarded as a type judgement $p : \mathsf{b} \to \mathsf{c}$. We show that under this encoding, all the inference rules of PHL can be derived in the type calculus. Thus Hoare-style reasoning with partial correctness assertions is essentially a matter of typechecking in this system. Moreover, the typing rules of KAT can be soundly encoded in pure (typeless) KAT.

The interplay of types and assertions, programs and proofs has been observed in many contexts and at many levels, from constructive mathematics and programming language semantics to program analysis and compiler certification. Perhaps the most far-reaching example is the Curry-Howard isomorphism, or propositions-as-types principle, and its ramifications [2, 4, 5, 10, 11, 22, 24]. The present work reveals yet another aspect of this phenomenon.

2 Definitions

2.1 Hoare Logic

Hoare logic is a system for reasoning inductively about well-structured programs. Comprehensive surveys can be found in [1, 8].

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) $\{b\}p\{c\}$, 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 are usually interpreted as binary input/output relations $p^{\mathcal{M}}$ on a domain of computation $\mathcal{M}$, and assertions b are interpreted as subsets $b^{\mathcal{M}}$ of $\mathcal{M}$ [7, 23]. 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 $\{b\}\, p\, \{c\}$ is the same as the meaning of the DL formula $b \to [p]\, c$, where $\to$ is ordinary propositional implication and the modal formula $[p]\, c$ is interpreted in $\mathcal{M}$ 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 for each programming construct. The verification process is inductive on the structure of programs. The traditional Hoare inference rules are:

Assignment rule:

$$\{b\,[x/e]\,\}\; x := e\; \{b\} \tag{1}$$

Composition rule:

$$\frac{\{b\}\; p\; \{c\}, \quad \{c\}\; q\; \{d\}}{\{b\}\; p; q\; \{d\}} \tag{2}$$

Conditional rule:

$$\frac{\{b \wedge c\}\; p\; \{d\}, \quad \{\neg b \wedge c\}\; q\; \{d\}}{\{c\}\; \textbf{if } b \textbf{ then } p \textbf{ else } q\; \{d\}} \tag{3}$$

While rule:

$$\frac{\{b \wedge c\}\, p\, \{c\}}{\{c\}\ \textbf{while}\ b\ \textbf{do}\ p\, \{\neg b \wedge c\}} \tag{4}$$

Weakening rule:

$$\frac{b' \to b, \quad \{b\}\, p\, \{c\}, \quad c \to c'}{\{b'\}\, p\, \{c'\}}. \tag{5}$$

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, while, and weakening rules (2)–(5) and propositional logic. The assignment rule (1) 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 [6, 12]. The axiomatization used here is from [14]. A *Kleene algebra* is an algebraic structure $(K,\ +,\ \cdot,\ ^*,\ 0,\ 1)$ that is an idempotent semiring under $+, \cdot, 0, 1$ and that satisfies

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

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

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

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.

Terms in the language of Kleene algebra are built from variables $x, y, \ldots$, binary operators $+$ and $\cdot$, unary operator *, and constants 0 and 1. Terms are often called *regular expressions* and are denoted $p, q, \ldots$. Atomic formulas are equations between terms. The expressions $p \le q$ and $q \ge p$ are abbreviations for $p + q = q$.

Kleene algebra is a versatile system with many useful interpretations in semantics, verification, and algorithm design and analysis. 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. A more unusual interpretation is the min,+ algebra used in shortest path algorithms.

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

$$
\begin{aligned}
(p^*q)^*p^* &= (p+q)^* && (12)\\
p(qp)^* &= (pq)^*p && (13)\\
(pq)^* &= 1+p(qp)^*q && (14)\\
p^* &= (pp)^*(1+p). && (15)
\end{aligned}
$$

All the operators are monotone with respect to $\leq$. In other words, if $p \leq q$, then $pr \leq qr$, $rp \leq rq$, $p+r \leq q+r$, and $p^* \leq q^*$ 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 was further developed in [3, 20]. A Kleene algebra with tests is just a Kleene algebra with an embedded Boolean subalgebra. That is, it is a two-sorted structure

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

such that

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

The Boolean complementation operator $^-$ is defined only on B. Elements of B are called *tests*. The letters p, q, r, s denote arbitrary elements of K and a, b, c denote tests. If Σ is an alphabet representing atomic Kleene elements and B is an alphabet representing atomic Boolean elements, then $T_{\Sigma,\mathsf{B}}$ and T_{B} denote the set of all terms and the set of all Boolean terms, respectively.

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 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 [9]:

$$p;q \stackrel{\text{def}}{=} pq \tag{16}$$

$$\textbf{if } b \textbf{ then } p \textbf{ else } q \stackrel{\text{def}}{=} bp + \overline{b}q \tag{17}$$

$$\textbf{while } b \textbf{ do } p \stackrel{\text{def}}{=} (bp)^*\overline{b}. \tag{18}$$

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 $\mathrm{Mat}(\mathcal{K}, n)$ of $n \times n$ matrices over a KAT $\mathcal{K} = (K,\ B)$; the Boolean elements of this structure are the diagonal matrices over 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 [20].

In [18], it was shown that KAT subsumes PHL in the following sense. A partial correctness assertion $\{b\}\ p\ \{c\}$ is encoded as an equation $bp\overline{c} = 0$, or equivalently, $bp = bpc$. If a rule

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

is derivable in PHL, then its translation, the universal Horn formula

$$b_1p_1\overline{c}_1 = 0 \wedge \cdots \wedge b_np_n\overline{c}_n = 0 \quad \rightarrow \quad bp\overline{c} = 0,$$

is a theorem of KAT. More generally, one can show that all relationally valid Horn formulas of the form

$$r_1 = 0 \wedge \cdots \wedge r_n = 0 \quad \rightarrow \quad p = q$$

are theorems of KAT [19].

2.4 Typed Kleene Algebra

Typed Kleene algebra was introduced in [17]. It is motivated primarily by the desire to interpret regular expressions as matrices of various shapes, possibly nonsquare. For example, in the completeness proof of [13, 14], it must be argued that a few essential theorems of Kleene algebra, such as

$$ax \leq xb \quad \rightarrow \quad a^*x \leq xb^*, \tag{19}$$

still hold when the symbols are interpreted as matrices of various sizes and shapes, provided there is no type mismatch. The equational implication (19) holds in $\mathrm{Mat}(\mathcal{K}, n)$, the $n \times n$ matrices over a Kleene algebra $\mathcal{K}$, simply by virtue of the fact that $\mathrm{Mat}(\mathcal{K}, n)$ is a Kleene algebra. However, for the purposes of [13, 14], we need to know that it holds even when a is interpreted as an $m \times m$ matrix, x is interpreted as an $m \times n$ matrix, and b is interpreted as an $n \times n$ matrix for any m and n.

To handle nonsquare matrices and other similar typed applications, we introduced a typing discipline in which regular expressions p have types of the form $\mathsf{s} \to \mathsf{t}$, where s and t are elements of an abstract set Ω. Every expression has a *most general typing* (mgt) under which the expression is well-typed and which refines every other typing for which this is true. For example, the most general typing of the expression ab^*c is $a : \mathsf{u} \to \mathsf{v}$, $b : \mathsf{v} \to \mathsf{v}$, $c : \mathsf{v} \to \mathsf{w}$, where u, v, and w are distinct. Most general typings exist and are unique up to a bijection. These ideas give rise to a theory called *typed Kleene algebra*.

In our principal interpretation, $\Omega = \mathbb{N}$ and the type judgement $p : \mathsf{s} \to \mathsf{t}$ indicates that p is a matrix with row and column dimensions s and t, respectively. There are other useful interpretations as well: sets of traces in a labeled transition system, binary relations with specified domains and ranges, regular sets of guarded strings [16, 20], semiadditive categories [21].

Let Ω be a set and $\omega : \{x, y, \ldots\} \to \Omega^2$. Elements of Ω are denoted $\mathsf{s}, \mathsf{t}, \mathsf{u}, \mathsf{v}, \ldots$ and are called *pretypes*. Elements of Ω^2 are called *types* and are denoted $\mathsf{s} \to \mathsf{t}$. (In [17] we also included a type 2 for Boolean values.)

The map ω is called a *type environment*. If $\omega(x) = \mathsf{s} \to \mathsf{t}$, we write $x : \mathsf{s} \to \mathsf{t}$ and say that x *has type* $\mathsf{s} \to \mathsf{t}$ *under* ω.

We can use the following calculus to derive types for certain expressions from ω. A *type judgement* is an expression

$$p : \mathsf{s} \to \mathsf{t}$$

where p is a regular expression and $\mathsf{s} \to \mathsf{t}$ is a type. Given a type environment ω, types for compound terms and formulas are inferred inductively according to the following rules:

$$\frac{p : \mathsf{s} \to \mathsf{t} \qquad q : \mathsf{s} \to \mathsf{t}}{p + q : \mathsf{s} \to \mathsf{t}} \tag{20}$$

$$\frac{p : \mathsf{s} \to \mathsf{t} \qquad q : \mathsf{t} \to \mathsf{u}}{pq : \mathsf{s} \to \mathsf{u}} \tag{21}$$

$$\frac{p : \mathsf{s} \to \mathsf{s}}{p^* : \mathsf{s} \to \mathsf{s}} \tag{22}$$

$$0 : \mathsf{s} \to \mathsf{t} \tag{23}$$

$$1 : \mathsf{s} \to \mathsf{s} \tag{24}$$

Note that 0 has all types and 1 all square types (types of the form $\mathsf{s} \to \mathsf{s}$ for some $\mathsf{s} \in \Omega$).

Every type environment ω extends uniquely to a minimal set of type judgements closed under these rules. This unique extension is also denoted ω and is called a *typing*.

An expression p is *well-typed* under the typing ω if ω contains a type judgement $p : \mathsf{s} \to \mathsf{t}$. A set of expressions is said to be *well-typed* under ω if every expression in the set is well-typed under ω.

Not all expressions are well-typed under all typings. For example, if $x : \mathsf{s} \to \mathsf{t}$ and $\mathsf{s} \neq \mathsf{t}$, the expression x^* is not well-typed. Moreover, the type of an expression under a typing ω is not unique; for example, if $x : \mathsf{s} \to \mathsf{t}$, then $x0 : \mathsf{s} \to \mathsf{u}$ for all u. However, the type of a variable is unique.

A class of models called *typed Kleene algebras* was defined in [17]; this semantics is reviewed below in Section 4.

In [17] it was shown that a wide class of theorems of untyped Kleene algebra are also theorems of typed Kleene algebra under their most general typings.

3 Encoding Hoare Logic

In [18] we showed that partial correctness assertions can be regarded as equations in the language of KAT and that the usual rules of propositional Hoare logic are theorems of KAT under this encoding.

In this section we extend the type calculus of KA as described in Section 2.4 to account for tests. We augment the system with subtypes, intersection types, and rules for subtyping and typecasting. Tests are interpreted as typecast operators. We then show how PHL is subsumed by this type calculus. To complete the triangle, we show how to encode the rules of the type calculus as valid universal Horn formulas in the language of pure (typeless) KAT. These two encodings compose to give the encoding of [18].

Let $\mathcal{K} = (K, B)$ be an arbitrary KAT. We impose a type structure on $\mathcal{K}$ as follows. Take the Boolean algebra B as the set of pretypes; we write $b \in B$ in a different font b when using it as a pretype. As in Section 2.4, a type judgement is an expression of the form $p : \mathsf{b} \to \mathsf{c}$.

We regard the natural order $\leq$ on B as a subtype order and the conjunction operation on B as a type intersection operator on pretypes. We postulate the following subtyping rule:

$$\frac{\mathsf{b}' \leq \mathsf{b}, \quad p : \mathsf{b} \to \mathsf{c}, \quad \mathsf{c} \leq \mathsf{c}'}{p : \mathsf{b}' \to \mathsf{c}'}. \tag{25}$$

(We will reconcile this view with the flat type structure of typed KA in Section 4 below.)

We regard a test $b \in B$ as a *typecast* or *coercion* operator that takes an object of type c and casts it down to an object of type bc. This is reflected in the following typing rule:

$$b : \mathsf{c} \to \mathsf{bc} \tag{26}$$

Note that in the presence of the subtype rule (25), this subsumes the rules $1 : \mathsf{s} \to \mathsf{s}$ and $0 : \mathsf{s} \to \mathsf{t}$ of typed KA.

The rule (26) represents an idealized form of the behavior of typecast operators and runtime type checks in modern programming languages. For example, consider the following Java fragment:

```
class High {}
class Low extends High {}
...
void fun(High y) {
   Low x = null;
   try {
      x = (Low)y;
   } catch (ClassCastException e) {}
}
...
fun(new Low());
```

The typecast operator `(Low)` is applied to an object whose runtime type `Low` is a proper subtype of its type `High` as determined by the static type environment. If the cast is unsuccessful, then a `ClassCastException` is thrown. But successful or not, the type of the expression `(Low)y` is `Low`, and after the cast it is type-correct to assign the object to a variable of that type.

The type calculus of KAT consists of the rules (20)–(24) of typed KA, the subtype rule (25), and the typecast rule (26).

To encode PHL in the type calculus of KAT, we encode the PCA $\{b\}\ p\ \{c\}$ by the type judgement

$$p : \mathsf{b} \to \mathsf{c}. \tag{27}$$

Using (16)–(18) and (27), we obtain the following translations of the Hoare rules (2)–(5):

Composition rule:

$$\frac{p : \mathsf{b} \to \mathsf{c}, \quad q : \mathsf{c} \to \mathsf{d}}{pq : \mathsf{b} \to \mathsf{d}} \tag{28}$$

Conditional rule:

$$\frac{p : \mathsf{bc} \to \mathsf{d}, \quad q : \overline{\mathsf{b}}\mathsf{c} \to \mathsf{d}}{bp + \overline{b}q : \mathsf{c} \to \mathsf{d}} \tag{29}$$

While rule:

$$\frac{p : \mathsf{bc} \to \mathsf{c}}{(bp)^*\overline{b} : \mathsf{c} \to \overline{\mathsf{b}}\mathsf{c}} \tag{30}$$

Weakening rule:

$$\frac{\mathsf{b}' \leq \mathsf{b}, \quad p : \mathsf{b} \to \mathsf{c}, \quad \mathsf{c} \leq \mathsf{c}'}{p : \mathsf{b}' \to \mathsf{c}'} \tag{31}$$

We now show that these rules can be derived in the type calculus of KAT.

Theorem 3.1 *The rules* (28)–(31) *are derived rules of the type calculus* (20)–(26).

Proof. The rule (28) is just the composition rule (21) and the rule (31) is just the subtype rule (25), so in these two cases there is nothing to prove.

For (29), we have

$$\dfrac{\dfrac{b : \mathsf{c} \to \mathsf{bc}^{\,(a)} \qquad p : \mathsf{bc} \to \mathsf{d}^{\,(b)}}{bp : \mathsf{c} \to \mathsf{d}}\,(e) \qquad \dfrac{\overline{b} : \mathsf{c} \to \overline{\mathsf{b}}\mathsf{c}^{\,(c)} \qquad q : \overline{\mathsf{b}}\mathsf{c} \to \mathsf{d}^{\,(d)}}{\overline{b}q : \mathsf{c} \to \mathsf{d}}\,(f)}{bp + \overline{b}q : \mathsf{c} \to \mathsf{d}}\,(g)$$

Here (a) and (c) are instances of the typecast rule (26), (b) and (d) are the premises, (e) and (f) are applications of the composition rule (21), and (g) is an application of the sum rule (20).

For (30), we have

$$\dfrac{\dfrac{\dfrac{b : \mathsf{c} \to \mathsf{bc}^{\,(a)} \qquad p : \mathsf{bc} \to \mathsf{c}^{\,(b)}}{bp : \mathsf{c} \to \mathsf{c}}\,(c)}{(bp)^* : \mathsf{c} \to \mathsf{c}}\,(d) \qquad \overline{b} : \mathsf{c} \to \overline{\mathsf{b}}\mathsf{c}^{\,(e)}}{(bp)^*\overline{b} : \mathsf{c} \to \overline{\mathsf{b}}\mathsf{c}}\,(f)$$

Here (a) and (e) are instances of the typecast rule (26), (b) is the premise, (c) and (f) are applications of the composition rule (21), and (d) is an application of the iteration rule (22). □

Now we encode the rules of the type calculus as universal Horn formulas in the language of pure (typeless) KAT; thus the type calculus is redundant. Interpret the type judgement $p : \mathsf{b} \to \mathsf{c}$ as one of the two equivalent equations

$$bp\overline{c} = 0 \quad \text{or} \quad bp = bpc. \tag{32}$$

The typing rule

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

becomes the universal Horn formula

$$b_1 p_1 \overline{c}_1 = 0 \wedge \cdots \wedge b_n p_n \overline{c}_n = 0 \quad \to \quad bp\overline{c} = 0. \tag{33}$$

Theorem 3.2 *The typing rules* (20)–(26)*, encoded as Horn formulas according to* (32) *and* (33)*, are all theorems of* KAT*. In other words, the type calculus of* KAT *is sound under the interpretation* (32).

Proof. Translating the rules (20)–(26) according to (32) and (33), we obtain

$$\begin{aligned}
bp\overline{c} = 0 \wedge bq\overline{c} = 0 \quad &\rightarrow \quad b(p+q)\overline{c} = 0 && (34)\\
bp = bpc \wedge cq = cqd \quad &\rightarrow \quad bpq = bpqd && (35)\\
bp = bpb \quad &\rightarrow \quad bp^* = bp^*b && (36)\\
& \quad\;\; b0\overline{c} = 0 && (37)\\
& \quad\;\; b1\overline{b} = 0 && (38)\\
b' \leq b \wedge bp\overline{c} = 0 \wedge c \leq c' \quad &\rightarrow \quad b'p\overline{c}' = 0 && (39)\\
& \quad\;\; cb\overline{bc} = 0, && (40)
\end{aligned}$$

respectively. These are all easy exercises in KAT. The most difficult is (36), which we argue explicitly. The inequality $bp^*b \leq bp^*$ holds by monotonicity of multiplication, thus it suffices to show

$$bp \leq bpb \quad \rightarrow \quad bp^* \leq bp^*b.$$

By (9), it suffices to show

$$bp \leq bpb \quad \rightarrow \quad b + bp^*bp \leq bp^*b.$$

But if $bp \leq bpb$, then $bp \leq pb$ by monotonicity, therefore

$$b + bp^*bp \quad \leq \quad bb + bp^*pb \quad = \quad b(1+p^*p)b \quad = \quad bp^*b.$$

□

4 Embedding Typed KA in KAT

Although based on the type discipline of typed KA [17] as described in Section 2.4, the type discipline of KAT as described in Section 3 looks quite different. The latter assumes a Boolean algebra structure on pretypes, whereas the former is flat. In this section we show that the two type disciplines are compatible by showing that every typed KA has a natural embedding in a KAT such that the type structure is preserved. The embedding is an extension of a natural embedding of a typed KA in a typeless KA described in [17].

First we review the semantics of typed KA from [17]. Briefly, a *typed Kleene algebra* is structure in which

- each element has a unique type of the form $\mathsf{s} \rightarrow \mathsf{t}$;
- there is a collection of polymorphic typed operators $+$, $\cdot$, *, 0, 1 and binary relation $=$ whose application is governed by the typing rules;
- all well-typed instances of the Kleene algebra axioms hold.

Formally, a *typed Kleene algebra* is a structure

$$\mathcal{K} \quad = \quad (K,\ \Omega,\ \omega,\ +,\ \cdot,\ ^*,\ 0,\ 1,\ =)$$

where K and Ω are sets and $\omega : K \to \Omega^2$; we write $\omega(p) = \mathsf{s} \to \mathsf{t}$. Elements of K are denoted $p, q, r, \ldots$.

For $\mathsf{s}, \mathsf{t} \in \Omega$, define

$$K_{\mathsf{st}} \quad = \quad \{p \in K \mid \omega(p) = \mathsf{s} \to \mathsf{t}\}.$$

The operators $+,\ \cdot,\ ^*,\ 0,\ 1$ and relation $=$ have the following polymorphic types:

$$\begin{array}{rcl}
+ & : & \Lambda\mathsf{s},\mathsf{t} \in \Omega.(\mathsf{s} \to \mathsf{t}) \times (\mathsf{s} \to \mathsf{t}) \to (\mathsf{s} \to \mathsf{t}) \\
\cdot & : & \Lambda\mathsf{s},\mathsf{t},\mathsf{u} \in \Omega.(\mathsf{s} \to \mathsf{t}) \times (\mathsf{t} \to \mathsf{u}) \to (\mathsf{s} \to \mathsf{u}) \\
^* & : & \Lambda\mathsf{s} \in \Omega.(\mathsf{s} \to \mathsf{s}) \to (\mathsf{s} \to \mathsf{s}) \\
0 & : & \Lambda\mathsf{s},\mathsf{t} \in \Omega.(\mathsf{s} \to \mathsf{t}) \\
1 & : & \Lambda\mathsf{s} \in \Omega.(\mathsf{s} \to \mathsf{s}) \\
= & : & \Lambda\mathsf{s},\mathsf{t} \in \Omega.(\mathsf{s} \to \mathsf{t}) \times (\mathsf{s} \to \mathsf{t}) \to 2.
\end{array}$$

This means for example that $+$ consists of a family of functions $+_{\mathsf{st}} : K_{\mathsf{st}}^2 \to K_{\mathsf{st}}$, one for each choice of $\mathsf{s}, \mathsf{t} \in \Omega$. The operator $+_{\mathsf{st}}$ can only be applied to arguments of type $\mathsf{s} \to \mathsf{t}$ and produces a sum of type $\mathsf{s} \to \mathsf{t}$. The polymorphic constant 0 represents a family of elements 0_{st}, one for each choice of $\mathsf{s}, \mathsf{t} \in \Omega$. The polymorphic constant 1 represents a family of square elements 1_{ss}, $\mathsf{s} \in \Omega$.

To be a typed Kleene algebra, $\mathcal{K}$ must also satisfy all well-typed instances of the Kleene algebra axioms. For example, the multiplicative associativity property $p(qr) = (pq)r$ must hold whenever the expression $p(qr) = (pq)r$ is well typed; that is, whenever $p : \mathsf{s} \to \mathsf{t}$, $q : \mathsf{t} \to \mathsf{u}$, and $r : \mathsf{u} \to \mathsf{v}$ for some $\mathsf{s}, \mathsf{t}, \mathsf{u}, \mathsf{v} \in \Omega$.

We now show how to construct a KAT $\mathcal{M}$ from a given typed KA $\mathcal{K}$ with pretypes Ω. Let B be the smallest Boolean subalgebra on 2^Ω containing all singleton sets. Thus B consists of the finite and cofinite subsets of Ω. (If Ω is finite, then B is just 2^Ω.) We denote elements of B by $b, c, \ldots$. Consider the $\Omega \times \Omega$ matrices P of finite support with $P_{\mathsf{st}} \in K_{\mathsf{st}}$. As argued in [17], this is a typeless 1-free KA, and $\mathcal{K}$ embeds homomorphically into it under the map $p \mapsto P$ such that $P_{\mathsf{st}} = p$, where $\omega(p) = \mathsf{s} \to \mathsf{t}$, and $P_{\mathsf{uv}} = 0_{\mathsf{uv}}$ elsewhere. (The word "embedding" is used here in the sense of *typed embedding* [17]; although all 0_{st} are mapped to the zero matrix, no pair of distinct elements of the same type are collapsed.) Now include the Boolean algebra B in the form of diagonal matrices; the matrix corresponding to $b \in B$ has as its s^{th} diagonal element either 1_{ss} or 0_{ss} according as $\mathsf{s} \in b$ or $\mathsf{s} \notin b$, respectively. Closing under the KA operations, the resulting matrices are no longer necessarily of finite support, but all rows and columns are still of finite support, so that multiplication is defined. Moreover, the diagonal elements are almost all 1 or almost all 0, thus each matrix decomposes into a block diagonal matrix of two blocks, one a finite square matrix and the other a square zero or identity matrix, thus * is defined. This gives a KAT $\mathcal{M}$ whose Boolean elements are the diagonal matrices over 0,1 corresponding to elements of B.

In $\mathcal{M}$, the interpretation (32) says that $P : \mathsf{b} \to \mathsf{c}$ iff

$$s \in b \text{ and } P_{\mathsf{st}} \neq 0 \quad \Rightarrow \quad t \in c; \tag{41}$$

in other words, the $b \times \overline{c}$ submatrix of P is the zero matrix.

It is instructive to understand the significance of the typing rules (20)–(26) in light of (41). For example, the rule (22) says that if $P : \mathsf{b} \to \mathsf{b}$, then $P^* : \mathsf{b} \to \mathsf{b}$. The matrix P can be decomposed (after permuting the rows and columns) into quadrants

$$P \quad = \quad \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

where A, B, C, and D are the $b\times b$, $b\times\overline{b}$, $\overline{b}\times b$, and $\overline{b}\times\overline{b}$ submatrices of P, respectively. The type judgement $P : \mathsf{b} \to \mathsf{b}$ says that $B = 0$. But by the definition of * for matrices,

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^* \quad = \quad \begin{bmatrix} (A+BD^*C)^* & (A+BD^*C)^*BD^* \\ (D+CA^*B)^*CA^* & (D+CA^*B)^* \end{bmatrix},$$

and $(A+BD^*C)^*BD^* = 0$, therefore P^* satisfies the same property.

Similarly, $I : \mathsf{b} \to \mathsf{b}$ for any b, where I is the identity matrix, since if $s \in b$ and $I_{\mathsf{st}} \neq 0$, then $s = t$, therefore $t \in b$; thus (41) holds.

Finally, the typecast rule (26) says that for any Boolean element b, if B is the diagonal matrix corresponding to b, then $B : \mathsf{c} \to \mathsf{bc}$. By (41), if $s \in c$ and $B_{\mathsf{st}} \neq 0$, then we should have $t \in bc$. But $B_{\mathsf{st}} \neq 0$ iff $s = t$ and $s \in b$, therefore $t \in bc$.

The following theorem describes the relationship between the typing disciplines of $\mathcal{K}$ and $\mathcal{M}$.

Theorem 4.1 *Let $\mathcal{K}$ be an arbitrary typed* KA *with pretypes Ω, and let $\mathcal{M}$ be the* KAT *constructed from it as described above. Let $p \in \mathcal{K}$ such that $p : \mathsf{s} \to \mathsf{t}$, and let P be its image in $\mathcal{M}$. Then either*

(i) $p = 0_{\mathsf{st}}$, *in which case $P : \mathsf{b} \to \mathsf{c}$ for all $\mathsf{b}, \mathsf{c} \in 2^\Omega$; or*

(ii) $p \neq 0_{\mathsf{st}}$, *in which case $P : \{\mathsf{s}\} \to \{\mathsf{t}\}$ but not $P : \{\mathsf{s}\} \to \{\mathsf{u}\}$ for any $\mathsf{u} \neq \mathsf{t}$ (thus by* (25)*, not $P : \{\mathsf{s}\} \to \varnothing$), and $P : \{\mathsf{u}\} \to \varnothing$ for all $\mathsf{u} \neq \mathsf{s}$.*

Proof. (i) If $p = 0_{\mathsf{st}}$, then P is the zero matrix, thus by (41), $P : \mathsf{b} \to \mathsf{c}$ for all $\mathsf{b}, \mathsf{c} \in 2^\Omega$.

(ii) If $p \neq 0_{\mathsf{st}}$ and $p : \mathsf{s} \to \mathsf{t}$, then $P_{\mathsf{st}} = p \neq 0$ and $P_{\mathsf{uv}} = 0$ elsewhere. This says that the s^{th} row of P is nonzero in column t and zero elsewhere, therefore by (41) $P : \{\mathsf{s}\} \to \{\mathsf{t}\}$ but not $P : \{\mathsf{s}\} \to \{\mathsf{t}\}$ for any $\mathsf{u} \neq \mathsf{t}$. All other rows are zero, therefore by (41), $P : \{\mathsf{u}\} \to \varnothing$ for $\mathsf{u} \neq \mathsf{s}$. ☐

Acknowledgements

I am indebted to Robert Constable, Neal Glew and Greg Morrisett for valuable ideas and comments. This work was supported by the National Science Foundation under grant CCR-9708915.

References

[1] K. R. Apt. Ten years of Hoare's logic: a survey—part 1. *ACM Trans. Prog. Lang. Syst.*, 3:431–483, 1981.

[2] J. L. Bates and R. L. Constable. Proofs as programs. *ACM Trans. Program. Lang. Syst.*, 7(1):53–71, 1985.

[3] 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.

[4] Robert L. Constable. Themes in the development of programming logics circa 1963–1987. *Ann. Rev. Comput. Sci.*, 3:147–165, 1988.

[5] Robert L. Constable. Types in mathematics, logic, and programming. In S. R. Buss, editor, *Handbood of Proof Theory*, chapter X, pages 683–786. Elsevier, Amsterdam, 1998.

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

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

[8] 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.

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

[10] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In *Proc. 2nd Scand. Logic Symp.*, pages 63–69. Springer-Verlag, 1971.

[11] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[12] 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.

[13] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *Proc. 6th Symp. Logic in Comput. Sci.*, pages 214–225, Amsterdam, July 1991. IEEE.

[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. Typed Kleene algebra. Technical Report 98-1669, Computer Science Department, Cornell University, March 1998.

[18] Dexter Kozen. On Hoare logic and Kleene algebra with tests. In *Proc. Conf. Logic in Computer Science (LICS'99)*, pages 167–172. IEEE, July 1999.

[19] Dexter Kozen. On Hoare logic and Kleene algebra with tests. *Trans. Computational Logic*, 1999. Submitted.

[20] 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.

[21] Ernest Manes. *Predicate transformer semantics*. Cambridge University Press, 1992.

[22] P. Martin-Löf. Constructive mathematics and computer programming. In *6th Int. Congr. Logic, Methodology, and Philosophy of Science*, pages 153–175. North-Holland, 1982.

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

[24] J. C. Reynolds. Towards a theory of type structure. In *Proc. Colloque sur la Programmation*, volume 19 of *Lect. Notes in Comput. Sci.*, pages 408–423. Springer-Verlag, 1974.