KAT + B!

Niels Bjørn Bugge Grathwohl
University of Copenhagen
bugge@diku.dk

Dexter Kozen Konstantinos Mamouras
Cornell University
{kozen,mamouras}@cs.cornell.edu

Abstract

It is known that certain program transformations require a small amount of mutable state, a feature not explicitly provided by Kleene algebra with tests (KAT). In this paper we show how to axiomatically extend KAT with this extra feature in the form of *mutable tests*. The extension is conservative and is formulated as a general commutative coproduct construction. We give several results on deductive completeness and complexity of the system, as well as some examples of its use.

Categories and Subject Descriptors F.3.3 [*Logics and Meanings of Programs*]: Studies of Program Constructs—Program and recursion schemes

General Terms Theory, Verification, Languages

Keywords Kleene algebra, Kleene algebra with tests, verification

1. Introduction

Kleene algebra with tests (KAT) is a propositional equational system that combines Kleene algebra (KA) with Boolean algebra. It has been shown to be an effective tool for many low-level program analysis and verification tasks involving communication protocols, safety analysis, source-to-source program transformation, concurrency control, and compiler optimization [2, 5, 7–9, 17, 22]. A notable recent success is its adoption as a basis for NetKAT, a foundation for software-defined networks (SDN) [1].

One advantage of KAT is that it allows a clean separation of the theory of the domain of computation from the program restructuring operations. The former typically involves first-order reasoning, whereas the latter is typically propositional. It is often advantageous to separate the two, because the theory of the domain of computation may be highly undecidable. With KAT, one typically isolates the needed properties of the domain as premises in a Horn formula

$$s_1 = t_1 \wedge \cdots \wedge s_n = t_n \to s = t,$$

where the conclusion $s = t$ expresses a more complicated equivalence between (say) an unoptimized or unannotated version of a program and its optimized or annotated version. The premises are verified once and for all using the properties of the domain, and the conclusion is then verified propositionally in KAT under those assumptions.

Certain premises that arise frequently in practice can be incorporated as part of the theory using a technique known as *elimination of hypotheses*, in which Horn formulas with premises of a certain form can be reduced to the equational theory without loss of efficiency [7, 12, 23]. However, there are a few useful ones that cannot. In particular, it is known that there are certain program transformations that cannot be effected in pure KAT, but require extra structure. Two paradigmatic examples are the Böhm–Jacopini theorem [6] (see also [3, 26–28, 30]) and the folklore result that all while programs can be transformed to a program with a single while loop [13, 25].

The Böhm–Jacopini theorem states that every deterministic flowchart can be written as a while program. The construction is normally done at the first-order level and introduces auxiliary variables to remember values across computations. It has been shown that the construction is not possible without some kind of auxiliary structure of this type [3, 15, 24].

Akin to the Böhm–Jacopini theorem, and often erroneously conflated with it, is the folklore theorem that every while program can be written with a single while loop. Like the proof of the Böhm–Jacopini theorem, the proofs of [14, 25], as reported in [13], are normally done at the first-order level and use auxiliary variables. It was a commonly held belief that this result had no purely propositional proof [13], but a partial refutation of this view was given in [17] using a construction that foreshadows the construction of this paper.

One can carry out these constructions in an uninterpreted first-order version of KAT called *schematic KAT* (SKAT) [2, 20], but as SKAT is undecidable in general [18], one would prefer a less radical extension.

In this paper we investigate the minimal amount of structure that suffices to perform these transformations and show how to incorporate it in KAT without sacrificing deductive completeness or decidability. Our main results are:

- We show how to extend KAT with a set of independent *mutable tests*. The construction is done axiomatically with generators and additional equational axioms. We formulate the construction as a general *commutative coproduct* construction that satisfies a certain universality property. The generators are abstract *setters* of the form $b!$ and $\bar{b}!$ and *testers* $b?$ and $\bar{b}?$ for a test symbol b. We can think of these intuitively as operations that set and test the value of a Boolean variable, although we do not introduce any explicit notion of storage or variable assignment.
- We prove a representation theorem (Theorem 2) for the commutative coproduct of an arbitrary KAT K and a KAT of binary relations on a finite set, namely that it is isomorphic to a certain matrix algebra over K.
- As a corollary to the representation theorem, we show that the extension is *conservative*; that is, an arbitrary KAT K can

CSL-LICS 2014, July 14–18, 2014, Vienna, Austria.

http://dx.doi.org/10.1145/2603088.2603095

be augmented with mutable tests without affecting the theory of K. This is captured formally by a general property of the commutative coproduct, namely *injectivity*. It is not known whether the coproduct of KATs is injective in general, but we show that it is injective if at least one of the two cofactors is a finite relational KAT, which is the case in our application.

- We show that the free mutable test algebra on generators b_i, $1 \leq i \leq n$, is isomorphic to the KAT of all binary relations on a set of 2^n states. We also characterize the primitive operations in terms of a tensor product of n copies of a 2-state system. We show that the equational theory of this algebra is PSPACE-complete, thus no easier or harder to decide than KAT.
- We show that the equational theory of an arbitrary KAT K augmented with mutable tests is axiomatically reducible to the theory of K. In particular, the free KAT, augmented with mutable tests, is completely axiomatized by the KAT axioms plus the axioms for mutable tests.
- We show that the equational theory of KAT with mutable tests is EXPSPACE-complete.
- We demonstrate that the program transformations mentioned above, namely the Böhm–Jacopini theorem and the folklore result about while programs, can be carried out in KAT with mutable tests.

Balbiani et al. [4] present a related system DL-PA, a variant of propositional dynamic logic (PDL) with mutable tests only. Their system corresponds most closely to our free mutable test algebra, which is PSPACE-complete. The semantics of DL-PA is restricted to relational models, and they show that model checking and satisfiability are EXPTIME-complete. The added complexity is partly due to the presence of the modal operators in PDL, which are absent in KAT.

This paper is organized as follows. In §2 we briefly review KA and KAT and introduce the theory of mutable tests, and prove that the free mutable test algebra on n generators is isomorphic to the KAT of all binary relations on a set of size 2^n. We also introduce the commutative coproduct construction and prove our representation theorem for the commutative coproduct of an arbitrary KAT K and a finite relational KAT. In §3 we prove our main completeness and complexity results. In §4 we apply the theory to give an axiomatic treatment of two applications involving program transformations. In §5 we present conclusions and open problems.

Omitted proofs can be found in the appendix.

2. KAT and Mutable Tests

2.1 KA and KAT

A *Kleene algebra* $(K, +, \cdot, ^*, 0, 1)$ is an idempotent semiring with an iteration operator * satisfying

$$1 + pp^* = p^* \qquad q + pr \leq r \to p^*q \leq r$$
$$1 + p^*p = p^* \qquad q + rp \leq r \to qp^* \leq r$$

where $\leq$ refers to the natural partial order on K. 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, as well as other more unusual interpretations used in shortest path algorithms and computational geometry.

The following are some typical KA identities:

$$(p^*q)^*p^* = (p+q)^* \tag{1}$$
$$p(qp)^* = (pq)^*p \tag{2}$$
$$p^* = (p^n)^*(1 + p + \cdots + p^{n-1}). \tag{3}$$

All KA operations are monotone with respect to $\leq$.

A *Kleene algebra with tests* (KAT) is 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 as a semiring is a subalgebra of K.

The Boolean complementation operator $\bar{}$ 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. 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.

Conventional imperative programming constructs and Hoare partial correctness assertions can be encoded, and propositional Hoare logic is subsumed. The deductive completeness and complexity results for KA and KAT [10, 16, 23] say that the axioms are complete for the equational theory of standard language and relational models and that the equational theory is decidable in PSPACE.

See [17] for a more thorough introduction.

2.2 Mutable Tests

Let $T_n = \{t_1, \ldots, t_n\}$ be a set of primitive test symbols. Consider a set of primitive actions $\{t!, \bar{t}! \mid t \in T_n\}$ (note that $\bar{\bar{t}}! = t!$). We write $t?$ for the test t to emphasize the distinction between $t?$ and $t!$. Let F_n be the free KAT over primitive actions $\{t!, \bar{t}! \mid t \in T_n\}$ and primitive tests T_n modulo the following equations:

(i) $t!t? = t!$

(ii) $t?t! = t?$

(iii) $t!\bar{t}! = \bar{t}!$

(iv) $s!t! = t!s!$, provided $s \neq \bar{t}$.

(v) $s!t? = t?s!$, provided $s \notin \{t, \bar{t}\}$.

Intuitively, axiom (i) says that the action $t!$ makes a subsequent test $t?$ true, (ii) says that if $t?$ is already true, then the action $t!$ is redundant, (iii) says that setting a value overrides a previous such action on the same value, and (iv) and (v) say that actions and tests on different values are independent.

The theory B! refers to the equational consequences of (i)–(v) along with the axioms of KAT on terms over T_n. Two immediate such consequences are

(vi) $t!t! = t!$

(vii) $t!\bar{t}? = 0$.

An *atom* of T_n is a sequence $s_1 s_2 \cdots s_n$, where each s_i is either t_i or $\bar{t}_i$. Atoms are denoted $\alpha, \beta, \ldots$. The set of atoms is denoted At. We write $\alpha \leq t$ if t appears in α. Let $\alpha[t]$ denote the atom α if $\alpha \leq t$ and α with $\bar{t}$ replaced by t if $\alpha \leq \bar{t}$. Each atom $\alpha = s_1 \cdots s_n$ determines a *complete test* $\alpha? = s_1?s_2? \cdots s_n? \in F_n$ and a *complete assignment* $\alpha! = s_1!s_2! \cdots s_n! \in F_n$. The following are elementary consequences of B!:

$$t? = \sum_{\alpha \leq t} \alpha?\alpha! \qquad t! = \sum_{\alpha} \alpha?\alpha[t]! \tag{4}$$

$$\alpha!\alpha? = \alpha! \quad \alpha?\alpha! = \alpha? \quad \alpha!\beta! = \beta! \quad \alpha?\beta? = 0 \text{ if } \alpha \neq \beta. \tag{5}$$

2.3 Mutable Tests and Binary Relations

The following theorem characterizes the free B! algebra F_n. The theorem shows that B! is sound in the sense that the free model does not trivialize to the one-element algebra.

THEOREM 1. *The algebra F_n is isomorphic to the KAT of all binary relations on a set of size 2^n.*

Proof. The set At is of size 2^n. Consider the KAT of binary relations on At. This algebra is isomorphic to $\mathrm{Mat}(\mathsf{At}, \mathbf{2})$, the KAT of $\mathsf{At} \times \mathsf{At}$ matrices over the two-element KAT with the usual Boolean matrix operations. We will construct an isomorphism $h_n : F_n \to \mathrm{Mat}(\mathsf{At}, \mathbf{2})$.

For the generators, let $h_n(t?)$ and $h_n(t!)$ be $\mathsf{At} \times \mathsf{At}$ matrices with components

$$h_n(t?)_{\alpha\beta} = \begin{cases} 1 & \text{if } \beta = \alpha \leq t \\ 0 & \text{otherwise,} \end{cases} \qquad h_n(t!)_{\alpha\beta} = \begin{cases} 1 & \text{if } \beta = \alpha[t] \\ 0 & \text{otherwise.} \end{cases}$$

One can show without difficulty that the axioms (i)–(v) of B! are satisfied under the interpretation h_n. For example, for (ii),

$$\begin{aligned}(h_n(t?)h_n(t!))_{\alpha\beta} &= \sum_\gamma h_n(t?)_{\alpha\gamma} h_n(t!)_{\gamma\beta} = h_n(t?)_{\alpha\alpha} h_n(t!)_{\alpha\beta} \\ &= \begin{cases} 1 & \text{if } \alpha \leq t \text{ and } \beta = \alpha[t] \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} 1 & \text{if } \alpha \leq t \text{ and } \beta = \alpha \\ 0 & \text{otherwise} \end{cases} \\ &= h_n(t?)_{\alpha\beta}.\end{aligned}$$

Since F_n is the free B! algebra on generators T_n, h_n extends uniquely to a KAT homomorphism $h_n : F_n \to \mathrm{Mat}(\mathsf{At}, \mathbf{2})$. Under this extension, $h_n(\alpha?\beta!)$ is the matrix with 1 in location $\alpha\beta$ and 0 elsewhere. As every matrix in $\mathrm{Mat}(\mathsf{At}, \mathbf{2})$ is a sum of such matrices, h_n is surjective.

We wish also to show that h_n is injective. To do this, we show that every element of F_n is a sum of elements of the form $\alpha?\beta!$. This is true for primitive tests $t?$ and primitive actions $t!$ by (4). The constants 1 and 0 are equivalent to $\sum_\alpha \alpha?\alpha!$ and the empty sum, respectively.

For sums, the conclusion is trivial. For products, we observe using (5) that $\alpha?\beta!\gamma?\delta! = 0$ if $\beta \neq \gamma$ and $\alpha?\beta!\beta?\delta! = \alpha?\delta!$. By distributivity, this allows the product of two sums of elements of the form $\alpha?\beta!$ to be reduced to a sum of the same form. For *, any element of the form e^* where e is a sum of elements of the form $\alpha?\beta!$ is equivalent to $1 + e + e^2 + \cdots + e^m$ for some m, since At is finite.

Now if $A \subseteq \mathsf{At}^2$, then $h_n(\sum_{\alpha\beta\in A} \alpha?\beta!)$ is the matrix with 1 in locations $\alpha\beta \in A$ and 0 elsewhere. Thus if $A, B \subseteq \mathsf{At}^2$ and $h_n(\sum_{\alpha\beta\in A} \alpha?\beta!) = h_n(\sum_{\alpha\beta\in B} \alpha?\beta!)$, then $A = B$, therefore $\sum_{\alpha\beta\in A} \alpha?\beta! = \sum_{\alpha\beta\in B} \alpha?\beta!$. □

There are some interesting facts about F_n that are worth observing, although we will not need them in the sequel. We can express the state set At as the tensor product of n copies of $\mathbf{2}$, one copy for each $t \in T_n$. The structure F_1 is the algebra $\mathrm{Mat}(\mathbf{2}, \mathbf{2})$ of 2×2 matrices. The matrices $h_n(t?)$ and $h_n(t!)$ are Kronecker products

$$h_1(t?) \otimes \bigotimes_{s\neq t} I \qquad\qquad h_1(t!) \otimes \bigotimes_{s\neq t} I$$

where $h_1(t?)$ and $h_1(t!)$ are the matrices

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad\qquad \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

respectively, where the first row and column correspond to index t and the second to index $\bar{t}$, and I is the 2×2 identity matrix. Matrix multiplication in F_n satisfies

$$(A_1 \otimes \cdots \otimes A_n)(B_1 \otimes \cdots \otimes B_n) = A_1B_1 \otimes \cdots \otimes A_nB_n,$$

which can be regarded as an independence condition on the n components. Every matrix in F_n can be expressed as a sum of Kronecker products of 2×2 matrices with exactly one nonzero entry corresponding to expressions of the form $\alpha?\beta!$.

2.4 The Commutative Coproduct

Let K and F be KATs. The *commutative coproduct* of K and F is the coproduct (direct sum) of K and F modulo extra commutativity conditions $\{ps = sp \mid p \in K,\ s \in F\}$ that say that elements of K and F commute multiplicatively. The commutativity conditions model the idea that operations in K and F are independent of each other. We will give an explicit construction below.

The usual coproduct $K \oplus F$ comes equipped with canonical coprojections $i_K : K \to K \oplus F$ and $i_F : F \to K \oplus F$. The coprojections are often called *injections*, although they need not be injective.[1] The coproduct is said to be *injective* if i_K and i_F are injective.

Injectivity is important because it means the extension of an algebra K with extra features F is *conservative* in the sense that it does not introduce any new equations. The coproduct of KATs is not known to be injective in general; however, we shall show that if F is a finite relational KAT, then the coproduct and commutative coproduct are injective.

Our proof relies on an explicit coproduct construction from universal algebra that holds for any variety or quasivariety V (class of algebras defined by universally quantified equations or equational implications) over any signature Σ. We briefly review the construction here.

Let T_K be the set of Σ-terms over K. The identity function $K \to K$ extends uniquely to a canonical homomorphism $T_K \to K$. The *diagram* of K, denoted Δ_K, is the kernel of this homomorphism; this is the set of equations between Σ-terms over K that hold in K. It follows from general considerations of universal algebra that $T_K/\Delta_K \cong K$, where T/E denotes the quotient of T modulo the V-congruence generated by equations E; that is, the smallest Σ-congruence on T containing E and closed under the equations and equational implications defining V.

Now let $T_{K,F}$ denote the set of mixed Σ-terms over the disjoint union of the carriers of K and F. The coproduct is

$$K \oplus F = T_{K,F}/(\Delta_K \cup \Delta_F).$$

The canonical injection $i_K : K \to K \oplus F$ is obtained from the identity embedding $T_K \to T_{K,F}$ reduced modulo Δ_K on the left and $\Delta_K \cup \Delta_F$ on the right; the map is well-defined on Δ_K-classes since Δ_K refines $\Delta_K \cup \Delta_F$. This construction satisfies the usual universality property for coproducts, namely that for any pair of homomorphisms $k : K \to H$ and $f : F \to H$, there is a unique homomorphism $\langle k, f\rangle : K \oplus F \to H$ such that $k = \langle k, f\rangle \circ i_K$ and $f = \langle k, f\rangle \circ i_F$.

Now let K and F be KATs, and let D be the set of commutativity conditions

$$D = \{ps = sp \mid p \in K,\ s \in F\}$$

on $K \oplus F$. (This is actually an abuse of notation; it would be more accurate to say

$$D = \{i_K(p)i_F(s) = i_F(s)i_K(p) \mid p \in K,\ s \in F\}.)$$

[1] For example, $\mathbb{Z}_m \oplus \mathbb{Z}_n \cong \mathbb{Z}_{\gcd(m,n)}$ in the category of commutative rings.

The *commutative coproduct* is the quotient $(K \oplus F)/D$. Composed with the canonical map $[\cdot] : K \oplus F \to (K \oplus F)/D$, i_K and i_F inject K and F, respectively, into $(K \oplus F)/D$. The following universality property is satisfied:

LEMMA 1. *For any pair of homomorphisms $k : K \to H$ and $f : F \to H$ such that*

$$\forall p \in K \; \forall s \in F \; k(p)f(s) = f(s)k(p), \tag{6}$$

there is a unique universal arrow $\langle k, f\rangle_D : (K \oplus F)/D \to H$ such that $k = \langle k, f\rangle_D \circ [\cdot] \circ i_K$ and $f = \langle k, f\rangle_D \circ [\cdot] \circ i_F$.

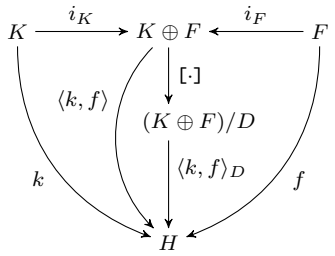

Figure 1. Universality property of the commutative coproduct

Proof. Property (6) implies that D refines the kernel of $\langle k, f\rangle : K \oplus F \to H$, therefore $\langle k, f\rangle$ factors uniquely as $\langle k, f\rangle_D \circ [\cdot]$, as shown in Fig. 1. □

Our main results depend on the following key lemma.

LEMMA 2. *Let K and F be KATs. If F is finite, then every element of $(K \oplus F)/D$ can be expressed as a finite sum $\sum_{s\in F} p_s s$, where $p_s \in K$.*

Remark. The lemma is not true in general without the assumption of finiteness. For example, it can be shown that the commutative coproduct of two copies of the free KA on one generator does not satisfy the lemma.

Proof. The lemma is certainly true of individual elements of K and F. We show that the property is preserved under the KAT operations. The cases of $+$ and $\cdot$ are quite easy, using commutativity and distributivity.

The only difficult case is that of *. We wish to show that $(\sum_{s\in F} p_s s)^*$ is equivalent to a finite sum of the form $\sum_{t\in F} q_t t$. Let $\Sigma = \{a_s \mid s \in F\}$ be a finite alphabet with one letter for each element of F, and let Reg_Σ be the free KA on generators Σ. Consider the following homomorphisms generated by the indicated actions on Σ:

$$\begin{array}{lll} f : \Sigma^* \to F & g : \mathrm{Reg}_\Sigma \to K \oplus F & h : \mathrm{Reg}_\Sigma \to K \\ f(a_s) = s & g(a_s) = p_s s & h(a_s) = p_s. \end{array}$$

For each $t \in F$, the set $f^{-1}(t) = \{x \in \Sigma^* \mid f(x) = t\}$ is a regular set, as it is the set accepted by the deterministic finite automaton with states F, start state 1, accept state t, and transitions $\delta(s, a) = s \cdot f(a)$. It is easily shown by induction that for all $x \in \Sigma^*$, $\delta(s, x) = s \cdot f(x)$. Thus the automaton accepts x exactly when $t = \delta(1, x) = f(x)$, that is, when $x \in f^{-1}(t)$.

Let A be the $F \times F$ transition matrix of this automaton: $A_{st} = \sum_{sr=t} a_r$. Then $(A^*)_{st}$ represents the set of strings x such that $s \cdot f(x) = t$. Moreover,

$$(\sum_{s\in F} a_s)^* = \sum_{t\in F} (A^*)_{1t} \tag{7}$$

since every string is accepted at some state t.

Let M be the $F \times F$ diagonal matrix with diagonal elements $M_{ss} = s$ and off-diagonal elements $M_{st} = 0$ for $s \neq t$. The homomorphisms g and h lift to $F \times F$ matrices over Reg_Σ with

$$g(A)_{st} = \sum_{sr=t} p_r r \qquad h(A)_{st} = \sum_{sr=t} p_r.$$

Then for any $s, t \in F$,

$$\begin{aligned} (M \cdot g(A))_{st} &= \sum_r M_{sr} g(A)_{rt} = M_{ss} g(A)_{st} = s \sum_{sr=t} p_r r \\ &= \sum_{sr=t} p_r sr = \sum_{sr=t} p_r t = h(A)_{st} M_{tt} \\ &= \sum_r h(A)_{sr} M_{rt} = (h(A) \cdot M)_{st}. \end{aligned}$$

Since s, t were arbitrary, $M \cdot g(A) = h(A) \cdot M$. By the bisimulation rule of KA [16, Proposition 4],

$$M \cdot g(A^*) = M \cdot g(A)^* = h(A)^* \cdot M = h(A^*) \cdot M,$$

thus for all $s, t \in F$,

$$\begin{aligned} s g(A^*)_{st} = M_{ss} g(A^*)_{st} &= \sum_{r\in F} M_{sr} g(A^*)_{rt} = (M \cdot g(A^*))_{st} \\ &= (h(A^*) \cdot M)_{st} = \sum_{r\in F} h(A^*)_{sr} M_{rt} \\ &= h(A^*)_{st} M_{tt} = h(A^*)_{st} t. \end{aligned}$$

In particular, setting $s = 1$ and summing over $t \in F$,

$$\sum_{t\in F} g(A^*)_{1t} = \sum_{t\in F} h(A^*)_{1t} t. \tag{8}$$

Using (7) and (8),

$$\begin{aligned} (\sum_{s\in F} p_s s)^* = (\sum_{s\in F} g(a_s))^* = g((\sum_{s\in F} a_s)^*) = g(\sum_{t\in F} (A^*)_{1t}) \\ = \sum_{t\in F} g(A^*)_{1t} = \sum_{t\in F} h(A^*)_{1t} t. \end{aligned}$$

Setting $q_t = h(A^*)_{1t}$, we have expressed $(\sum_{s\in F} p_s s)^*$ in the desired form. □

THEOREM 2. *If K is a KAT and F is the KAT of all binary relations on a finite set S, then $(K \oplus F)/D \cong \mathrm{Mat}(S, K)$.*

Proof. For $p \in K$, let $k(p) \in \mathrm{Mat}(S, K)$ be the $S \times S$ diagonal matrix with p on the main diagonal and 0 elsewhere. For $s \in F$, let $f(s)$ be the standard representation of the binary relation s as an $S \times S$ Boolean matrix. The maps $k : K \to \mathrm{Mat}(S, K)$ and $f : F \to \mathrm{Mat}(S, K)$ are injective KAT homomorphisms and embed K and F isomorphically in $\mathrm{Mat}(S, K)$. The image of F under f is $\mathrm{Mat}(S, \mathbf{2})$, a subalgebra of $\mathrm{Mat}(S, K)$. By the universality property for coproducts, we have that

$$\langle k, f\rangle : K \oplus F \to \mathrm{Mat}(S, K)$$

and k and f factor as $k = \langle k, f\rangle \circ i_K$ and $f = \langle k, f\rangle \circ i_F$.

Moreover, because $k(p)$ is a diagonal matrix for $p \in K$ and $f(s)$ is a Boolean matrix for $s \in F$, the commutativity conditions D are satisfied in the sense that $k(p)f(s) = f(s)k(p)$, thus Lemma 1 applies and we have a KAT homomorphism

$$\langle k, f\rangle_D : (K \oplus F)/D \to \mathrm{Mat}(S, K).$$

That this homomorphism is an isomorphism follows from Lemma 2 by an argument similar to that of Theorem 1. For $\alpha, \beta \in S$, let $n_{\alpha\beta} \in F$ such that $h(n_{\alpha\beta})_{\alpha\beta} = 1$ and all other entries are 0. Then

for all $s \in F$,

$$n_{\alpha\alpha} s n_{\beta\beta} = \begin{cases} n_{\alpha\beta} & \text{if } h(s)_{\alpha\beta} = 1 \\ 0 & \text{if } h(s)_{\alpha\beta} = 0 \end{cases} \qquad \sum_{\alpha} n_{\alpha\alpha} = 1.$$

We have

$$h(\sum_s p_s s)_{\alpha\beta} = \sum_s p_s h(s)_{\alpha\beta} = \sum_{h(s)_{\alpha\beta}=1} p_s \tag{9}$$

$$\begin{aligned} \sum_s p_s s &= \sum_s p_s (\sum_\alpha n_{\alpha\alpha}) s (\sum_\beta n_{\beta\beta}) \\ &= \sum_{\alpha,\beta} \sum_s p_s n_{\alpha\alpha} s n_{\beta\beta} = \sum_{\alpha,\beta} \sum_{h(s)_{\alpha\beta}=1} p_s n_{\alpha\beta} \end{aligned} \tag{10}$$

If $h(\sum_s p_s s) = h(\sum_s q_s s)$, then for all $\alpha, \beta \in S$, $h(\sum_s p_s s)_{\alpha\beta} = h(\sum_s q_s s)_{\alpha\beta}$. By (9) and (10),

$$\sum_s p_s s = \sum_{\alpha,\beta} \sum_{h(s)_{\alpha\beta}=1} p_s n_{\alpha\beta} = \sum_{\alpha,\beta} \sum_{h(s)_{\alpha\beta}=1} q_s n_{\alpha\beta} = \sum_s q_s s.$$

The construction is illustrated in Fig. 2. □

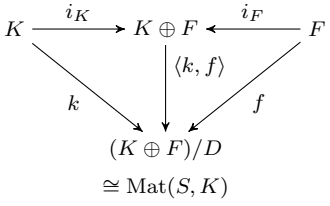

Figure 2. Matrix representation of the commutative coproduct

COROLLARY 1. *If K is a KAT and F is any KAT of binary relations on a finite set S, then $(K \oplus F)/D$ is isomorphic to a subalgebra of* $\mathrm{Mat}(S, K)$.

Proof. Compose an embedding of F into the KAT of all binary relations on S with the map f of Theorem 2. □

The following corollary says that the extension of an arbitrary KAT with mutable tests is conservative.

COROLLARY 2. *If K is a KAT and F is any KAT of binary relations on a finite set S, then the commutative coproduct $(K \oplus F)/D$ is injective.*

Proof. The maps $k = \langle k, f \rangle \circ i_K : K \to \mathrm{Mat}(S, K)$ and $f = \langle k, f \rangle \circ i_F : F \to \mathrm{Mat}(S, K)$ are injective. By Theorem 2, $(K \oplus F)/D \cong \mathrm{Mat}(S, K)$, and k and f compose with this isomorphism to give the canonical injections from K and F, respectively, to $(K \oplus F)/D$. □

3. Completeness and Complexity

In §2, we showed that an arbitrary KAT K can be conservatively extended with a small amount of state in the form of a finite set of mutable tests and their corresponding mutation actions. As shown in Theorem 2, the resulting algebra is isomorphic to $\mathrm{Mat}(\mathsf{At}, K)$, where At is the set of atoms of the free Boolean algebra generated by the mutable tests.

In this section we prove three results. First, the KAT axioms along with the axioms B! for mutable tests and the commutativity conditions D are complete for the equational theory of $(K \oplus F_n)/D$ relative to the equational theory of K. This is quite a strong result in the sense that it holds for an arbitrary KAT K, regardless of its nature. In particular, for the special case in which K is the free KAT on some set of generators, the model $(K \oplus F_n)/D$ is the free KAT with mutable tests T_n. Most of the work for this result has already been done in §2.

The second result is that the equational theory B! is complete for PSPACE. This complexity class is characterized by alternating polynomial-time Turing machines; see [21].

The third result is that the equational theory of a free KAT augmented with mutable tests is complete for EXPSPACE, deterministic exponential space. This result is quite surprising, as both KAT and B! separately are complete for PSPACE, yet their combination is exponentially more complex in the worst case.

3.1 Completeness

Let K be an arbitrary KAT. Let KAT+B! denote the deductive system consisting of the axioms of KAT, the axioms for mutable tests B!, and the commutativity conditions D over a language of KAT terms with primitive action and test symbols interpreted in K as well as a set of mutable tests T_n. Let Δ_K be the diagram of K.

THEOREM 3. *The axioms KAT+B!+Δ_K are complete for the equational theory of $(K \oplus F_n)/D$. In other words, the axioms KAT+B! are complete for the equational theory of $(K \oplus F_n)/D$ relative to the equational theory of K.*

Proof. Let e_1 and e_2 be expressions denoting elements of $(K \oplus F_n)/D$. By Theorem 1 and Lemma 2, we have

$$\text{KAT+B!+}\Delta_K \vdash e_1 = \sum_{\alpha,\beta \in \mathsf{At}} p_{\alpha\beta} \alpha?\beta!$$

$$\text{KAT+B!+}\Delta_K \vdash e_2 = \sum_{\alpha,\beta \in \mathsf{At}} q_{\alpha\beta} \alpha?\beta!.$$

If $(K \oplus F_n)/D \vDash e_1 = e_2$, we have under the canonical interpretation $\langle k, i \rangle$ that the matrices $\langle k, i \rangle(e_1)$ and $\langle k, i \rangle(e_2)$ are equal, thus for all $\alpha, \beta \in \mathsf{At}$,

$$p_{\alpha\beta} = \langle k, i \rangle(e_1)_{\alpha\beta} = \langle k, i \rangle(e_2)_{\alpha\beta} = q_{\alpha\beta},$$

and conversely. □

COROLLARY 3. *The axioms KAT+B! are complete for the equational theory of $(K \oplus F_n)/D$, where K is the free KAT on some set of generators.*

3.2 Complexity

THEOREM 4. *The equational theory B! is PSPACE-complete.*

Remark. We note that neither the upper nor the lower bound follows from previous results. The upper bound does not follow from results on elimination of hypotheses [7, 12, 23], as axioms (i) and (ii) can be eliminated by these results, but not the others.

Proof. We first show that the problem of deciding $\alpha?\beta! \leq e$, where $\alpha, \beta \in \mathsf{At}$, is in PSPACE. We give an alternating polynomial-time algorithm that operates inductively on the structure of e.

To decide $\alpha?\beta! \leq t?$ or $\alpha?\beta! \leq t!$, using (4) we can ask whether $\alpha = \beta \leq t$ or $\beta = \alpha[t]$, respectively.

For addition, we have $\alpha?\beta! \leq e_1 + e_2$ iff $\alpha?\beta! \leq e_1$ or $\alpha?\beta! \leq e_2$. We nondeterministically choose one of these alternatives and check it recursively.

For multiplication, we have $\alpha?\beta! \leq e_1 e_2$ iff there exists γ such that $\alpha?\gamma! \leq e_1$ and $\gamma?\beta! \leq e_2$. We guess γ nondeterministically using existential branching and check both conditions recursively using universal branching.

Finally, to check $\alpha?\beta! \leq e^*$, by Theorem 1 it suffices to check that $\alpha?\beta! \leq e^k$ for some $0 \leq k < 2^n$. We guess k nondeterministically using existential branching. To check $\alpha?\beta! \leq e^k$, we

guess γ nondeterministically using existential branching, and for each such γ, we check recursively using universal branching that $\alpha?\gamma! \leq e^{\lfloor k/2 \rfloor}$ and $\gamma?\beta! \leq e^{\lceil k/2 \rceil}$.

To decide the equational theory in PSPACE, we note that $e_1 \leq e_2$ if for all $\alpha, \beta \in \mathsf{At}$, if $\alpha?\beta! \leq e_1$, then $\alpha?\beta! \leq e_2$. The α and β can be chosen universally and the implication $\alpha?\beta! \leq e_1 \Rightarrow \alpha?\beta! \leq e_2$ checked in PSPACE.

To show PSPACE-hardness, we encode the membership problem for deterministic linear-bounded automata, a well known PSPACE-complete problem. Let M be a deterministic linear-bounded automaton with states Q and tape alphabet Γ. Let $x = x_1 \cdots x_n$ be an input string of length n over M's input alphabet. For $a \in \Gamma$, $q \in Q$, and $0 \leq i \leq n+1$, introduce mutable tests P_i^a and Q_i^q with the following intuitive meanings:

$$P_i^a = \text{the symbol currently occupying tape cell } i \text{ is } a,$$
$$Q_i^q = M \text{ is currently in state } q \text{ scanning tape cell } i.$$

The operation of the machine is governed by a transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{+1, -1\}$. Intuitively, the transition $\delta(p, a) = (q, b, d)$ means, "When in state p scanning symbol a, print b on that cell, move the head in direction d, and enter state q." For each such transition, consider the expressions

$$P_i^a ? Q_i^p ? \bar{P}_i^a ! \bar{Q}_i^p ! P_i^b ! Q_{i+d}^q ! \tag{11}$$

for all i. The part $P_i^a ? Q_i^p ?$ tests whether the machine is currently scanning a on cell i in state p. If so, $\bar{P}_i^a ! \bar{Q}_i^p ! P_i^b ! Q_{i+d}^q !$ effects the transition to the new configuration as dictated by the transition function δ. The truth values of variables not mentioned do not change.

Assume that the input is delimited by left and right endmarkers $\vdash$ and $\dashv$, that M starts in its start state s scanning the left endmarker $\vdash$, that M never overwrites the endmarkers, and that before accepting, M erases its tape by writing a blank symbol $\sqcup$ on all tape cells except for the endmarkers, moves its head all the way to the left, and enters state t. The start and accept configurations are atoms

$$\text{start} = Q_0^s \, P_0^{\vdash} \, P_1^{x_1} \, P_2^{x_2} \, \cdots P_n^{x_n} \, P_{n+1}^{\dashv} \, U$$
$$\text{accept} = Q_0^t \, P_0^{\vdash} \, P_1^{\sqcup} P_2^{\sqcup} \cdots P_n^{\sqcup} \, P_{n+1}^{\dashv} \, V$$

where U and V are the negations of the remaining variables. Let e be the sum of all expressions (11). Then M accepts x if and only if $\text{start?accept!} \leq e^*$. □

Let K be the free KAT on some set of generators. As shown in Corollary 3, the equational theory of $(K \oplus F_n)/D$ is completely axiomatized by KAT+B!.

THEOREM 5. *The set of equational consequences of KAT+B! (that is, the equational theory of a free KAT augmented with mutable tests) is EXPSPACE-complete.*

Proof. Let K be the free KAT on generators Σ and B. The atomic tests B are ordinary KAT tests and are not mutable. The set of equational consequences of KAT+B! coincides with the equational theory of the structure $(K \oplus F_n)/D$ (Corollary 3). This structure is isomorphic to the matrix algebra $\mathrm{Mat}(\mathsf{At}, K)$ (Theorem 2), where At is the set of 2^n atoms generated by the mutable tests T_n. Every element of $\mathrm{Mat}(\mathsf{At}, K)$ is an $\mathsf{At} \times \mathsf{At}$ matrix, each entry of which is a regular set of guarded strings over Σ, B. Regular sets of guarded strings are recognized by nondeterministic automata on guarded strings [19]. Such an automaton is a tuple $M = (Q, \Delta, \mathsf{start}, \mathsf{final})$, where Q is the set of states, $\mathsf{start} \subseteq Q$ are the start states, $\mathsf{final} \subseteq Q$ is the set of final or accepting states, and $\Delta \subseteq Q \times (\Sigma \cup \mathcal{B}) \times Q$ is the transition relation, where $\mathcal{B}$ is the set of composite tests built from the atomic tests B. A transition of the form (s, p, t) with $p \in \Sigma$ is called an action transition, and one of the form (s, b, t) with $b \in \mathcal{B}$ is called a test transition. In particular, a test transition of the form $(s, 1, t)$ is called an ε-transition. We refer the reader to [19] for a definition of how these automata compute on guarded strings. Let $\mathcal{L}(s, t)$ be the set of the guarded strings x so that there is some computation on x starting from state s that ends in state t. The automaton M recognizes the language of guarded strings $\bigcup_{s \in \mathsf{start}} \bigcup_{t \in \mathsf{final}} \mathcal{L}(s, t)$. We extend this automaton model so that it recognizes matrices of regular sets of guarded strings. A *matrix automaton* is a tuple

$$M = (Q \times \mathsf{At}, \Delta, \mathsf{start}, \mathsf{final}),$$

where $\mathsf{start}, \mathsf{final} : \mathsf{At} \to \wp Q$ and $\Delta \subseteq (Q \times \mathsf{At}) \times (\Sigma \cup \mathcal{B}) \times (Q \times \mathsf{At})$. We write $\wp Q$ to denote the powerset of Q. The automaton recognizes the $\mathsf{At} \times \mathsf{At}$ matrix L, each entry of which is a regular language of guarded strings:

$$L(\alpha, \beta) = \bigcup_{s \in \mathsf{start}(\alpha)} \bigcup_{t \in \mathsf{final}(\beta)} \mathcal{L}(\langle s, \alpha \rangle, \langle t, \beta \rangle).$$

We will describe now a construction similar to Kleene's theorem. Given a KAT+B! expression e over Σ, B, T_n we will give a matrix automaton that recognizes the matrix of languages denoted by e under its standard interpretation in the structure $\mathrm{Mat}(\mathsf{At}, K)$. For all base cases $p, b, t?, t!$ we define the set $Q = \{s_1, s_2\}$, the start states $\mathsf{start}(\alpha) = \{s_1\}$, and the accepting states $\mathsf{final}(\alpha) = \{s_2\}$, for every $\alpha \in \mathsf{At}$. We give the set Δ of transitions separately for each of these base cases:

- Case: action letter p in Σ. For every atom α, we put a transition $\langle s_1, \alpha \rangle \xrightarrow{p} \langle s_2, \alpha \rangle$.
- Case: arbitrary test b in $\mathcal{B}$. For every atom α, we have a transition $\langle s_1, \alpha \rangle \xrightarrow{b} \langle s_2, \alpha \rangle$.
- Case: mutable test $t?$. We put the transitions $\langle s_1, \alpha \rangle \xrightarrow{1} \langle s_2, \alpha \rangle$ for every $\alpha \leq t$.
- Case: primitive action $t!$. The automaton has the transitions $\langle s_1, \alpha \rangle \xrightarrow{1} \langle s_2, \alpha[t] \rangle$ for every α. Recall that $\alpha[t]$ is the modification of α so that t holds.

The remaining base cases are for 1 and 0. We define the corresponding automata as follows:

- For the case of 1, we have the trivial automaton with $Q = \{s\}$, $\mathsf{start}(\alpha) = \{s\}$, $\mathsf{final}(\alpha) = \{s\}$, and $\Delta = \emptyset$.
- The automaton for 0 is defined as $Q = \{s\}$, $\mathsf{start}(\alpha) = \{s\}$, $\mathsf{final}(\alpha) = \emptyset$, and $\Delta = \emptyset$.

Suppose that $M_1 = (Q_1 \times \mathsf{At}, \Delta_1, \mathsf{start}_1, \mathsf{final}_1)$ and $M_2 = (Q_2 \times \mathsf{At}, \Delta_2, \mathsf{start}_2, \mathsf{final}_2)$ are the matrix automata for the expressions e_1 and e_2 respectively. W.l.o.g. the sets Q_1 and Q_2 are disjoint.

- For the expression $e_1 + e_2$ we define the automaton $M = (Q \times \mathsf{At}, \Delta, \mathsf{start}, \mathsf{final})$ by $Q = Q_1 \cup Q_2$,
$$\mathsf{start}(\alpha) = \mathsf{start}_1(\alpha) \cup \mathsf{start}_2(\alpha)$$
$$\mathsf{final}(\alpha) = \mathsf{final}_1(\alpha) \cup \mathsf{final}_2(\alpha)$$
and $\Delta = \Delta_1 \cup \Delta_2$.
- For the expression $e_1 \cdot e_2$ define $M = (Q \times \mathsf{At}, \Delta, \mathsf{start}, \mathsf{final})$ by $Q = Q_1 \cup Q_2$, $\mathsf{start}(\alpha) = \mathsf{start}_1(\alpha)$, $\mathsf{final}(\alpha) = \mathsf{final}_2(\alpha)$, and $\Delta = \Delta_1 \cup \Delta_2 \cup \Delta'$, where
$$\Delta' = \{\langle s, \alpha \rangle \xrightarrow{1} \langle t, \alpha \rangle \mid s \in \mathsf{final}_1(\alpha),\ t \in \mathsf{start}_2(\alpha)\}.$$

Now, suppose that $M = (Q \times \mathsf{At}, \Delta, \mathsf{start}, \mathsf{final})$ is the matrix automaton for the expression e. The automaton for $e \cdot e^*$ results from M by adding ε-transitions from the final states back to the start states: $\langle s, \alpha \rangle \xrightarrow{1} \langle t, \alpha \rangle$, where $s \in \mathsf{final}(\alpha)$, $t \in \mathsf{start}(\alpha)$, and $\alpha \in \mathsf{At}$. Finally, the automaton for $e^* = 1 + e \cdot e^*$ can be obtained using the constructions for 1 and $+$ that we have already described.

Consider now two KAT+B! expressions e_1, e_2 and the problem of checking whether they denote the same matrix in the structure $\mathrm{Mat}(\mathsf{At}, K)$. We can construct effectively the corresponding matrix automata M_1 and M_2, as described in the previous paragraph.

We can have an explicit representation of these automata, since exponential space suffices for this. Let L_1 and L_2 be the matrices of languages accepted by M_1 and M_2 respectively. For every pair of atoms α, β we have to check whether $L_1(\alpha, \beta) = L_2(\alpha, \beta)$. This problem amounts to checking the equivalence of automata on guarded strings, which can be done in space polynomial in the size of the automata [19]. It follows that we can decide whether $e_1 = e_2$ in exponential space.

For the corresponding lower bound, we encode the membership problem for exponential-space bounded Turing machines. Given such a machine M and an input x of length n, we use n mutable tests to construct an integer counter that can count up to $2^n - 1$, as illustrated in Fig. 3. We use the counter as a "yardstick" to

$\bar{t}_0!; \bar{t}_1!; \cdots ; \bar{t}_{n-1}!;$
while $\bar{t}_0? + \bar{t}_1? + \cdots + \bar{t}_{n-1}?$ {
  if $\bar{t}_0?$ then $t_0!$;
  else if $\bar{t}_1?$ then $\bar{t}_0!; t_1!$;
  else if $\bar{t}_2?$ then $\bar{t}_0!; \bar{t}_1!; t_2!$;
  else ...
  else if $\bar{t}_{n-1}?$ then $\bar{t}_0!; \bar{t}_1!; \cdots ; \bar{t}_{n-2}!; t_{n-1}!$;
  else skip;
}

Figure 3. A counter

construct an expression e simulating a nondeterministic automaton that accepts all strings that are not valid computation histories of M on input x. The automaton decides nondeterministically where to look for an incorrect move of M. It remembers a few symbols of the input string, then starts the counter. With each iteration of the counter, it skips over an input symbol (not shown in Fig. 3). In this way it can compare symbols a distance 2^n apart to check whether the transition rules of M are followed. The expression e generates all strings iff M does not accept x. This construction is quite standard (see for example [11, 21, 29]), so we omit further details. □

4. Applications

4.1 The Böhm-Jacopini Theorem

A well-studied problem in program schematology is that of transforming unstructured flowgraphs to structured form. An early seminal result is the Böhm–Jacopini theorem [6], which states that any deterministic flowchart program is equivalent to a deterministic while program. This theorem has reappeared in many contexts and has been reproved by many different methods[3, 26–28, 30].

Like most early work in program schematology, the Böhm–Jacopini theorem is usually formulated at the first-order level. This allows auxiliary individual or Boolean variables to be introduced to preserve information across computations. This is an essential ingredient of the Böhm–Jacopini construction, and they asked whether it was strictly necessary. This question was answered affirmatively by Ashcroft and Manna [3] and Kosaraju [15].

In [24], a purely propositional account of this negative result was given. A class of automata called *strictly deterministic automata* was presented, an abstraction of deterministic flowchart schemes. The three-state strictly deterministic automaton of Fig. 4 was shown not to be equivalent to any deterministic while program, where the α_i are mutually exclusive and exhaustive tests and the p_{ij} are primitive actions.

With strictly deterministic automata, Boolean values are provided by the environment in the form of an input string consisting of an infinite sequence of atoms, and the program responds with actions, including halting or failing. This is the correct propositional semantics: it allows all possible interpretations of the actions that

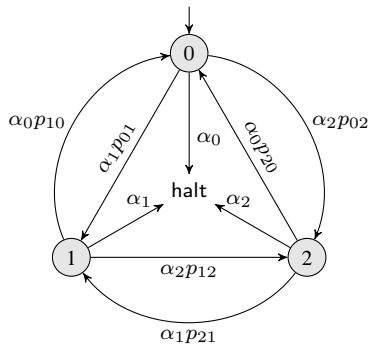

Figure 4. A strictly deterministic automaton not equivalent to any while program

could cause tests to become true or false. Two strictly deterministic automata are considered equivalent if they generate the same set of finite guarded strings (see [24] for formal definitions and details).

The Böhm–Jacopini theorem is true in the presence of mutable tests. The technique is well known, so rather than give a general account, we illustrate with the strictly deterministic automaton of Fig. 4. We introduce mutable tests t_0, t_1, and t_2, which serve as program counters. An equivalent deterministic while program with mutable tests is shown in Fig. 5. The major difference here is that

$t_0!; \bar{t}_1!; \bar{t}_2!;$ //start state is 0
while true {
  if $t_0?$ then
    $\bar{t}_0!$; if α_1 then $p_{01}; t_1!$; else if α_2 then $p_{02}; t_2!$; else halt;
  else if $t_1?$ then
    $\bar{t}_1!$; if α_2 then $p_{12}; t_2!$; else if α_0 then $p_{10}; t_0!$; else halt;
  else //must be t_2
    $\bar{t}_2!$; if α_0 then $p_{20}; t_0!$; else if α_1 then $p_{21}; t_1!$; else halt;
}

Figure 5. A while program with mutable tests equivalent to Fig. 4

the mutable tests are under the control of the program instead of the environment.

We have not given the formal definition of the set of guarded strings generated by a strictly deterministic automaton with mutable tests, but under the appropriate definition, it can be shown that this while program and the strictly deterministic automaton of Fig. 4 generate the same set of guarded strings.

4.2 A Folk Theorem

In this section we illustrate how KAT+B! can be used in practice. We will show, reasoning equationally in KAT+B!, a classical result of program schematology: Every while program can be simulated by a while program with at most one while loop, assuming that we allow extra Boolean variables. Many of the proofs are long sequences of simple equational inferences and are not very interesting in themselves, so we relegate them to an appendix.

We work with a programming language that has atomic programs (written $a, b, \ldots$), the constant programs skip and fail, atomic tests, as well as the constructs: sequential composition $f; g$, conditional test if p then f else g, and iteration while p do f. These constructs are modeled in KAT as follows:

$$\mathsf{skip} = 1 \qquad \mathsf{fail} = 0 \qquad f; g = fg$$
$$\mathsf{if}\ e\ \mathsf{then}\ f\ \mathsf{else}\ g = ef + \bar{e}g \qquad \mathsf{while}\ e\ \mathsf{do}\ f = (ef)^*\bar{e}$$

There is a semantic justification for these translations, using the standard relation-theoretic semantics for the input-output behavior of while programs. Intuitively, to show the result we introduce extra Boolean variables that encode the control structure of the program. These variables are modeled in KAT+B! using mutable tests $t_1, t_2, \ldots$, which are taken to be disjoint from any mutable tests that might already appear in the program.

Commutativity axioms: KAT+B! has axioms that say that primitive actions commute with the mutable test symbols, that is, $t?a = at?$ and $t!a = at!$. Moreover, $t!p = pt!$ and $t!\bar{p} = \bar{p}t!$ for every atomic KAT test p, since tests commute. The following claims establish that using the axioms of KAT+B! more commutativity equations can be shown.

CLAIM 1. *If the mutable test symbols $t, \bar{t}$ do not appear in the KAT+B! test term p, then we have that $t!p = pt!$ and $t!\bar{p} = \bar{p}t!$.*

CLAIM 2. *If the mutable test symbols $t, \bar{t}$ do not appear in the KAT+B! term f, then $t?f = ft?$ and $t!f = ft!$.*

The theorem that follows is a normal form theorem, from which the result we want to show follows immediately. Working in a bottom-up fashion, every while program term is brought in the normal form. That the transformed program in normal form is equivalent to the original one is shown in KAT+B!.

THEOREM 6. *For any while program f, there are while-free u, p, φ and a finite collection $t_1, \ldots, t_k$ of extra mutable tests such that $f; z = u; \mathsf{while}\, p\, \mathsf{do}\, \varphi; z$, where $z = \bar{t}_1!; \ldots; \bar{t}_k!$.*

Proof. In the normal form given above, the pre-computation u, the while-guard p, and the while-body φ may involve the extra mutable test symbols $t_1, \ldots, t_k, \bar{t}_1, \ldots, \bar{t}_k$. These symbols do not appear in f. The post-computation $z = \bar{t}_1!; \ldots; \bar{t}_k$ "zeroes out" all the extra mutable Boolean variables. Its rôle is in some sense to simply project out this extra finite state. The proof proceeds by induction on the structure of the while program term f.

Base case: Suppose that f is a while-free program term, and let t be a fresh mutable test symbol. Intuitively, $t?$ holds if f has not been executed yet, and $\bar{t}?$ holds when f has been executed. Reasoning in KAT+B!:

CLAIM 3. $f; z = t!; \mathsf{while}\, t?\, \mathsf{do}\, (f; \bar{t}!); z$, *where* $z = \bar{t}!$.

Induction step: From the induction hypothesis, we can bring the programs f and g in normal form so that $f; z = u; \mathsf{while}\, p\, \mathsf{do}\, \varphi; z$ and $g; z = v; \mathsf{while}\, q\, \mathsf{do}\, \psi; z$, where z sets to zero all the mutable tests that appear in the transformations of f and g. For the cases of a conditional test $\mathsf{if}\, e\, \mathsf{then}\, f\, \mathsf{else}\, g$ and composition $f; g$, we introduce a fresh mutable test symbol t.

Conditional: We handle the case $\mathsf{if}\, e\, \mathsf{then}\, f\, \mathsf{else}\, g$. Intuitively, the Boolean variable corresponding to the symbol t records the branch to be taken. So, $t?$ holds when f should be executed, and $\bar{t}?$ holds when g should be executed. Reasoning in KAT+B!:

CLAIM 4. *The program* $(\mathsf{if}\, e\, \mathsf{then}\, f\, \mathsf{else}\, g); z; \bar{t}!$ *is equal to*

$$\begin{array}{l} \mathsf{if}\, e\, \mathsf{then}\, (t!; u)\, \mathsf{else}\, (\bar{t}!; v); \\ \mathsf{while}\, ((t? \wedge p) \vee (\bar{t}? \wedge q))\, \mathsf{do}\, (\mathsf{if}\, t?\, \mathsf{then}\, \varphi\, \mathsf{else}\, \psi); \\ z; \bar{t}!. \end{array}$$

Sequential composition: We handle the case $f; g$. Intuitively, the Boolean variable t records the current position of execution. So, $t?$ holds when we are executing f, and $\bar{t}?$ when we are executing g.

CLAIM 5. *The program $f; g; z; \bar{t}!$ is provably equal to*

$$\begin{array}{l} t!; u; \\ \mathsf{while}\, (t? \vee (\bar{t}? \wedge q))\, \mathsf{do} \\ \quad \mathsf{if}\, t?\, \mathsf{then}\, (\mathsf{if}\, p\, \mathsf{then}\, \varphi\, \mathsf{else}\, (z; \bar{t}!; v))\, \mathsf{else}\, \psi; \\ z; \bar{t}!. \end{array}$$

Loop: It remains to handle the case of the while loop $\mathsf{while}\, e\, \mathsf{do}\, f$. First, we observe that

CLAIM 6. $\mathsf{while}\, e\, \mathsf{do}\, f; z = \mathsf{while}\, e\, \mathsf{do}\, (f; z); z$.

Using the above claim, we can bring the program in a more convenient form:

CLAIM 7. *The program* $(\mathsf{while}\, e\, \mathsf{do}\, f); z$ *is provably equal to*

$$\mathsf{if}\, e\, \mathsf{then}\, \Big(u; \mathsf{while}\, (e + p)\, \mathsf{do}\, \mathsf{if}\, p\, \mathsf{then}\, \varphi\, \mathsf{else}\, (z; u)\Big); z.$$

But we already know how to transform conditional statements, so we apply that transformation to bring the term in the desired normal form. □

5. Conclusion

We have shown how to axiomatically extend Kleene algebra with tests with a finite amount of mutable state. This extra feature allows certain program transformations to be effected at the propositional level without passing to a full first-order system. The extension is conservative and deductively complete relative to the theory of the underlying algebra. The full theory is decidable and complete for EXPSPACE. We have given a representation theorem of the free models in terms of matrices.

An intriguing open problem is whether the coproduct of two KATs is injective. We have shown that it is if one of the two cofactors is a KAT of binary relations on a finite set.

6. Acknowledgments

Thanks to Bob Constable, Nate Foster, Fritz Henglein, Mark Reitblatt, Ross Tate, and Laure Thompson for valuable conversations and insights. This work was funded by the National Security Agency. The DIKU-affiliated author expresses his thanks to the Department of Computer Science at Cornell University for hosting him in the Fall 2013 and to the Danish Council for Independent Research for financial support for this work under Project 11-106278, "Kleene Meets Church (KMC): Regular Expressions and Types."

References

[1] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. In *Proc. 41st ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'14)*, pages 113–126, San Diego, California, USA, January 2014. ACM.

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

[3] E. Ashcroft and Z. Manna. The translation of goto programs into while programs. In C. Freiman, J. Griffith, and J. Rosenfeld, editors, *Proceedings of IFIP Congress 71*, volume 1, pages 250–255. North-Holland, 1972.

[4] P. Balbiani, A. Herzig, and N. Troquard. Dynamic logic of propositional assignments: A well-behaved variant of PDL. In *Proc. 28th Symp. Logic in Computer Science (LICS'13)*, pages 143–152. ACM/IEEE, 2013.

[5] A. Barth and D. Kozen. Equational verification of cache blocking in LU decomposition using Kleene algebra with tests. Technical Report TR2002-1865, Computer Science Department, Cornell University, June 2002.

[6] C. Böhm and G. Jacopini. Flow diagrams, Turing machines and languages with only two formation rules. *Communications of the ACM*, pages 366–371, May 1966.

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

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

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

[10] E. Cohen, D. Kozen, and F. Smith. The complexity of Kleene algebra with tests. Technical Report TR96-1598, Computer Science Department, Cornell University, July 1996.

[11] J. Ferrante and C. Rackoff. *The computational complexity of logical theories*, volume 718 of *Lecture Notes in Mathematics*. Springer-Verlag, 1979.

[12] C. Hardin and D. Kozen. On the elimination of hypotheses in Kleene algebra with tests. Technical Report TR2002-1879, Computer Science Department, Cornell University, October 2002.

[13] D. Harel. On folk theorems. *Comm. Assoc. Comput. Mach.*, 23(7): 379–389, July 1980.

[14] K. Hirose and M. Oya. General theory of flowcharts. *Comment. Math. Univ. St. Pauli*, 21(2):55–71, 1972.

[15] S. R. Kosaraju. Analysis of structured programs. In *Proc. 5th ACM Symp. Theory of Computing (STOC'73)*, pages 240–252, New York, NY, USA, 1973. ACM. .

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

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

[18] D. Kozen. Halting and equivalence of schemes over recursive theories. Technical Report TR2002-1881, Computer Science Department, Cornell University, October 2002.

[19] D. Kozen. Automata on guarded strings and applications. *Matématica Contemporânea*, 24:117–139, 2003.

[20] D. Kozen. Some results in dynamic model theory. *Science of Computer Programming*, 51(1–2):3–22, May 2004. Special issue: *Mathematics of Program Construction (MPC 2002)*. Eerke Boiten and Bernhard Möller (eds.).

[21] D. Kozen. *Theory of Computation*. Springer, New York, 2006. ISBN 10: 1-84628-297-7.

[22] D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. 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.

[23] D. Kozen and F. 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.

[24] D. Kozen and W.-L. D. Tseng. The Böhm-Jacopini theorem is false, propositionally. In P. Audebaud and C. Paulin-Mohring, editors, *Proc. 9th Int. Conf. Mathematics of Program Construction (MPC'08)*, volume 5133 of *Lecture Notes in Computer Science*, pages 177–192. Springer, July 2008.

[25] G. Mirkowska. *Algorithmic Logic and its Applications*. PhD thesis, University of Warsaw, 1972. in Polish.

[26] G. Oulsnam. Unraveling unstructured programs. *The Computer Journal*, 25(3):379–387, 1982.

[27] W. Peterson, T. Kasami, and N. Tokura. On the capabilities of while, repeat, and exit statements. *Comm. Assoc. Comput. Mach.*, 16(8):503–512, 1973.

[28] L. Ramshaw. Eliminating goto's while preserving program structure. *Journal of the ACM*, 35(4):893–920, 1988.

[29] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. 5th Symp. Theory of Computing*, pages 1–9, New York, 1973. ACM.

[30] M. Williams and H. Ossher. Conversion of unstructured flow diagrams into structured form. *The Computer Journal*, 21(2):161–167, 1978.

A. Appendix

Proof of Claim 1. By induction on p. If p is an atomic KAT test, then the claim follows directly from axioms. The cases of the constants 0 and 1 are trivial. If p is a mutable test $s?$, then by our assumption we have that $s \neq t, \bar{t}$ and therefore $t!s? = s?t!$ and $t!\bar{s}? = \bar{s}?t!$ are axioms of B!. For the induction step, consider the case $p + q$:

$$t!(p+q) = t!p + t!q = pt! + qt! = (p+q)t!$$
$$t!(\overline{p+q}) = t!\bar{p}\bar{q} = \bar{p}t!\bar{q} = \bar{p}\bar{q}t! = (\overline{p+q})t!$$

The case pq is similar. For the case of $\bar{p}$, the equation $t!\bar{p} = \bar{p}t!$ follows from the induction hypothesis for p. Similarly, $t!\bar{\bar{p}} = t!p = pt! = \bar{\bar{p}}t!$. □

Proof of Claim 2. We only show the part involving $t!$, for $t?$ the proof is essentially the same. We argue by induction on the structure of f. If f is a test, then the result follows from Claim 1. If f is an atomic program a, then from the stipulated axioms we have that $t!a = at!$. For composition and choice we have using the induction hypothesis: $t!fg = ft!g = fgt!$, and

$$t!(f+g) = t!f + t!g = ft! + gt! = (f+g)t!.$$

It remains to show that $t!f^* = f^*t!$. By virtue of the bisimulation rule, it suffices to see that $t!f = ft!$, which is the induction hypothesis. □

Proof of Claim 3. First, we unravel the expression $(f?f\bar{t}!)^*$ twice and observe that

$$\begin{aligned}(t?f\bar{t}!)^* &= 1 + t?f\bar{t}!(t?f\bar{t}!)^* \\ &= 1 + t?f\bar{t}!(1 + t?f\bar{t}!(t?f\bar{t}!)^*) \\ &= 1 + t?f\bar{t}! + t?f\bar{t}!t?f\bar{t}!(t?f\bar{t}!)^* \\ &= 1 + t?f\bar{t}!,\end{aligned}$$

because $\bar{t}!t? = \bar{t}!\bar{t}?t? = 0$. So, we conclude that

$$\begin{aligned}\text{RHS} &= t!(t?f\bar{t}!)^*\bar{t}?\bar{t}! \\ &= t!(1 + t?f\bar{t}!)\bar{t}? \\ &= t!\bar{t}? + t!t?f\bar{t}!\bar{t}?,\end{aligned}$$

which is equal to $t!f\bar{t}! = ft!\bar{t}! = f\bar{t}! = f; z$, since t was chosen to be fresh (Claim 2). □

Proof of Claim 4. The while-free pre-computation in the normal form translation is equal to $et!u + \bar{e}\bar{t}!v$. The guard of the while loop is $t?p + \bar{t}?q$, and the body is $t?\varphi + \bar{t}?\psi$. So,

$$\begin{aligned}&((t? \wedge p) \vee (\bar{t}? \wedge q)); (\mathsf{if}\ t?\ \mathsf{then}\ \varphi\ \mathsf{else}\ \psi) = \\ &(t?p + \bar{t}?q)(t?\varphi + \bar{t}?\psi) = \\ &t?p\varphi + \bar{t}?q\psi.\end{aligned}$$

The negation of the guard of the loop is $\neg(t?p + \bar{t}?q) = (\bar{t}? + \bar{p})(t? + \bar{q}) = \bar{t}?\bar{q} + t?\bar{p} + \bar{p}\bar{q}$.

First, we claim that $t?(t?p\varphi)^* = t?(p\varphi)^*$. Since $t? \leq 1$ and * is monotone, we have that $(t?p\varphi)^* \leq (p\varphi)^*$, and therefore $t?(t?p\varphi)^* \leq t?(p\varphi)^*$. In order to show that $t?(p\varphi)^* \leq$

$t?(t?p\varphi)^*$, it suffices to see that $t? \leq t?(t?p\varphi)^*$, and that

$$\begin{aligned} t?(t?p\varphi)^* p\varphi &= t?(1 + (t?p\varphi)^* t?p\varphi)p\varphi \\ &= t?p\varphi + t?(t?p\varphi)^* t?p\varphi p\varphi \\ &= t?t?p\varphi + t?(t?p\varphi)^* t?t?p\varphi p\varphi \\ &= t?t?p\varphi + t?(t?p\varphi)^* t?p\varphi t?p\varphi \\ &= t?(1 + (t?p\varphi)^* t?p\varphi)t?p\varphi \\ &= t?(t?p\varphi)^* t?p\varphi \leq t?(t?p\varphi)^*. \end{aligned}$$

Now, we want to show that $t?(t?p\varphi + \bar{t}?q\psi)^* = t?(t?p\varphi)^*$. By monotonicity of *, the right-hand side is less than or equal to the left-hand side. For the other part, we need to show that

$$\begin{aligned} &t?(t?p\varphi)^*(t?p\varphi + \bar{t}?q\psi) = \\ &t?t?(t?p\varphi)^*(t?p\varphi + \bar{t}?q\psi) = && \text{[prev. claim]} \\ &t?t?(p\varphi)^*(t?p\varphi + \bar{t}?q\psi) = && [t \text{ not in } p, \varphi] \\ &t?(p\varphi)^* t?(t?p\varphi + \bar{t}?q\psi) = \\ &t?(p\varphi)^* t?p\varphi = && \text{[prev. claim]} \\ &t?(t?p\varphi)^* t?p\varphi, \end{aligned}$$

which is $\leq t?(t?p\varphi)^*$.

Let W abbreviate the entire while loop of the normal form translation. We have already seen that

$$W = (t?p\varphi + \bar{t}?q\psi)^*(\bar{t}?\bar{q} + t?\bar{p} + \bar{p}\bar{q})$$

and therefore

$$\begin{aligned} t?W &= t?(t?p\varphi)^*(\bar{t}?\bar{q} + t?\bar{p} + \bar{p}\bar{q}) \\ &= t?(p\varphi)^*(\bar{t}?\bar{q} + t?\bar{p} + \bar{p}\bar{q}) \\ &= (p\varphi)^* t?(\bar{t}?\bar{q} + t?\bar{p} + \bar{p}\bar{q}) \\ &= (p\varphi)^*(t?\bar{p} + t?\bar{p}\bar{q}) \\ &= (p\varphi)^* t?\bar{p}, \end{aligned}$$

because $t?\bar{p}\bar{q} \leq t?\bar{p}$. So, we have

$$\begin{aligned} e\text{RHS} &= e(et!u + \bar{e}\bar{t}!v)Wz\bar{t}! = et!uWz\bar{t}! \\ &= et!t?uWz\bar{t}! = et!ut?Wz\bar{t}! \\ &= et!u(p\varphi)^* t?\bar{p}z\bar{t}! = eu(p\varphi)^*\bar{p}z\bar{t}!, \end{aligned}$$

which is equal to $efz\bar{t}!$ by the induction hypothesis. Similarly, it can be shown $\bar{e}\text{RHS} = \bar{e}gz\bar{t}!$. We thus conclude that

$$\begin{aligned} \text{RHS} &= (e + \bar{e})\text{RHS} = e\text{RHS} + \bar{e}\text{RHS} \\ &= efz\bar{t}! + \bar{e}gz\bar{t}! = (ef + \bar{e}g)z\bar{t}!, \end{aligned}$$

which is equal to $(\mathsf{if}\ e\ \mathsf{then}\ f\ \mathsf{else}\ g); z; \bar{t}!$, namely the left-hand size of the equation we wanted to show. □

Proof of Claim 5. The negation of the guard of the while loop is $\neg(t? + \bar{t}?q) = \bar{t}?(t? + \bar{q}) = \bar{t}?\bar{q}$. The body of the loop is equal to $t?(p\varphi + \bar{p}z\bar{t}!v) + \bar{t}?\psi = t?p\varphi + t?\bar{p}z\bar{t}!v + \bar{t}?\psi$. So, the Fisher-Ladner encoding of the while loop is

$$\begin{aligned} &[(t? + \bar{t}?q)(t?p\varphi + t?\bar{p}z\bar{t}!v + \bar{t}?\psi)]^*\bar{t}?\bar{q} \\ &= [t?p\varphi + t?\bar{p}z\bar{t}!v + \bar{t}?q\psi]^*\bar{t}?\bar{q} \\ &= (A + \bar{t}?q\psi)^*\bar{t}?\bar{q} \\ &= A^*(\bar{t}?q\psi A^*)^*\bar{t}?\bar{q}, \end{aligned}$$

where we put $A = t?p\varphi + t?\bar{p}z\bar{t}!v$.

From $\bar{t}?A = \bar{t}?(t?p\varphi + t?\bar{p}z\bar{t}!v) = 0 \leq \bar{t}?$ we obtain that $\bar{t}?A^* \leq \bar{t}?$. Moreover, $\bar{t}? \leq \bar{t}?A$ and hence $\bar{t}?A^* = \bar{t}?$. It follows that $\bar{t}?q\psi A^* = q\psi\bar{t}?A^* = q\psi\bar{t}?$. Now, we claim that $(q\psi\bar{t}?)^*\bar{t}? = \bar{t}?(q\psi)^*$. The inequality $(q\psi\bar{t}?)^*\bar{t}? \leq \bar{t}?(q\psi)^*$ follows from monotonicity of *. For the inequality $\bar{t}?(q\psi)^* \leq (q\psi\bar{t}?)^*\bar{t}?$ we need to show that

$$(q\psi\bar{t}?)^*\bar{t}?q\psi = (q\psi\bar{t}?)^* q\psi\bar{t}? = (q\psi\bar{t}?)^* q\psi\bar{t}?\bar{t}?,$$

which is $\leq (q\psi\bar{t}?)^*\bar{t}?$. We have thus shown that the while loop is equal to $A^*(q\psi\bar{t}?)^*\bar{t}?q = A^*\bar{t}?(q\psi)^*\bar{q}$.

Now, we focus on simplifying the expression $t?A^*\bar{t}? = t?(t?p\varphi + t?\bar{p}z\bar{t}!v)^*\bar{t}?$. First, we observe that unfolding $(t?\bar{p}z\bar{t}!v)^*$ twice gives us the equation

$$(t?\bar{p}z\bar{t}!v)^* = 1 + t?\bar{p}z\bar{t}!v.$$

Moreover, $\bar{t}?(t?p\varphi)^* = \bar{t}?(1 + t?p\varphi(t?p\varphi)^*) = \bar{t}?$. Therefore, using the denesting rule, we obtain that $t?A^*\bar{t}?$ is provably equal to

$$\begin{aligned} &t?(t?p\varphi)^*(t?\bar{p}z\bar{t}!v(t?p\varphi)^*)^*\bar{t}? = \\ &t?(t?p\varphi)^*(t?\bar{p}z\bar{t}!v\bar{t}?(t?p\varphi)^*)^*\bar{t}? = \\ &t?(t?p\varphi)^*(t?\bar{p}z\bar{t}!v\bar{t}?)^*\bar{t}? = \\ &t?(t?p\varphi)^*(t?\bar{p}z\bar{t}!v)^*\bar{t}? = \\ &t?(t?p\varphi)^*(1 + t?\bar{p}z\bar{t}!v)\bar{t}? = \\ &t?(t?p\varphi)^*\bar{t}? + t?(t?p\varphi)^* t?\bar{p}z\bar{t}!v\bar{t}? = \\ &t?(t?p\varphi)^* t?\bar{p}z\bar{t}!v\bar{t}? = \\ &t?(p\varphi)^*\bar{p}z\bar{t}!v. \end{aligned}$$

Finally, we can work on the right-hand side of the equation we want to establish:

$$\begin{aligned} \text{RHS} &= t!uA^*\bar{t}?(q\psi)^*\bar{q}z\bar{t}! = t!ut?A^*\bar{t}?(q\psi)^*\bar{q}z\bar{t}! \\ &= t!ut?(p\varphi)^*\bar{p}z\bar{t}!v(q\psi)^*\bar{q}z\bar{t}! = u(p\varphi)^*\bar{p}zv(q\psi)^*\bar{q}z\bar{t}!, \end{aligned}$$

which is equal by the induction hypothesis to $fzgz\bar{t}! = fgzz\bar{t}! = f; g; z; \bar{t}!$. □

Proof of Claim 6. The left-hand side is equal to $(ef)^*\bar{e}z$, and the right-hand side equal to $(efz)^*\bar{e}z$. It suffices to show that $(ef)^*z = (efz)^*z$.

$$(efz)^*z \leq (ef)^*z \Longleftarrow efz(ef)^*z \leq (ef)^*z,$$

which holds because $efz(ef)^*z = ef(ef)^*zz \leq (ef)^*z$. Now, we observe that $(efz)^*z = z(efz)^*$ by the bisimulation rule, because $efzz = zefz$ (both are equal to efz). So,

$$(ef)^*z \leq (efz)^*z \Longleftarrow ef(efz)^*z \leq (efz)^*z,$$

which holds because $ef(efz)^*z = ef(efz)^*zz = efz(efz)^*z \leq (efz)^*z$. □

Proof of Claim 7. The above program is equal to

$$\begin{aligned} &\bar{e}z + eu[(e + p)(p\varphi + \bar{p}zu)]^*(\overline{e + p})z = \\ &\bar{e}z + eu[ep\varphi + e\bar{p}zu + p\varphi]^*\bar{e}\bar{p}z = \\ &\bar{e}z + eu(p\varphi + e\bar{p}zu)^*\bar{e}\bar{p}z, \end{aligned}$$

because $ep\varphi \leq p\varphi$. Using the denesting rule (1) and then the sliding rule (2), we see that this is equal to

$$\begin{aligned} &\bar{e}z + eu(p\varphi)^*(e\bar{p}zu(p\varphi)^*)^*\bar{e}\bar{p}z = \\ &\bar{e}z + eu(p\varphi)^*(\bar{p}zeu(p\varphi)^*)^*\bar{e}\bar{p}z = \\ &\bar{e}z + (eu(p\varphi)^*\bar{p}z)^* eu(p\varphi)^*\bar{e}\bar{p}zz = \\ &\bar{e}z + (efz)^* eu(p\varphi)^*\bar{p}z\bar{e}z = \\ &\bar{e}z + (efz)^*(efz)\bar{e}z = \\ &(1 + (efz)^*(efz))\bar{e}z, \end{aligned}$$

which is equal to $(efz)^*\bar{e}z = \mathsf{while}\ e\ \mathsf{do}\ (f; z); z = (\mathsf{while}\ e\ \mathsf{do}\ f); z$. □