Kleene Algebra with Tests and Program Schematology

Allegra Angus Dexter Kozen

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501, USA

July 10, 2001

Abstract

The theory of flowchart schemes has a rich history going back to Ianov [6]; see Manna [22] for an elementary exposition. A central question in the theory of program schemes is scheme equivalence. Manna presents several examples of equivalence proofs that work by simplifying the schemes using various combinatorial transformation rules. In this paper we present a purely algebraic approach to this problem using Kleene algebra with tests (KAT). Instead of transforming schemes directly using combinatorial graph manipulation, we regard them as a certain kind of automaton on abstract traces. We prove a generalization of Kleene's theorem and use it to construct equivalent expressions in the language of KAT. We can then give a purely equational proof of the equivalence of the resulting expressions. We prove soundness of the method and give a detailed example of its use.

1 Introduction

Kleene algebra (KA) is the algebra of regular expressions. It was first introduced by Kleene [8]; the name *Kleene algebra* was coined by Conway [5], who developed much of the algebraic theory. Kleene algebra has appeared in computer science in many guises: semantics and logics of programs [9, 23], automata and formal languages [20, 21], and the design and analysis of algorithms [1, 7, 10]. Many authors have contributed over the years to the development of the algebraic theory; see [13] and references therein.

Kleene algebra with tests (KAT), introduced in [13], combines programs and assertions in a purely equational system. A *Kleene algebra with tests* is a Kleene algebra with an embedded Boolean subalgebra. KAT strictly subsumes propositional Hoare Logic (PHL), is of no greater complexity than PHL, and is deductively complete over relational models (PHL is not) [17, 4, 14, 18]. KAT is less expressive than propositional Dynamic Logic (PDL), but also less complex (unless $PSPACE = EXPTIME$, which complexity theorists generally regard as unlikely). Moreover, KAT requires nothing beyond classical equational logic, in contrast to PHL or PDL, which depend on a more complicated syntax involving partial correctness assertions or modalities.

KAT has been applied successfully in a number of low-level verification tasks involving communication protocols, basic safety analysis, concurrency control, and compiler optimization [2, 3, 16]. A useful feature of KAT in this regard is its ability to accommodate certain basic equational assumptions regarding the interaction of atomic instructions. This feature makes KAT ideal for reasoning about the correctness of low-level code transformations.

In this paper we further demonstrate the utility of KAT by showing how it can be used to recast much of the classical theory of flowchart schemes into a purely algebraic framework. A *flowchart scheme* is a vertex-labeled graph that represents an uninterpreted program. The vertices are labeled with atomic assignments and tests that can be performed during execution, and the edges represent control paths. The primitive function and relation symbols appearing in vertex labels are uninterpreted; one is typically interested in the behavior of the program under all possible interpretations.

The theory of flowchart schemes has a rich history going back to Ianov [6]. The text of Manna [22] gives an elementary exposition of the theory. A central question that occupies much of Manna's attention is the problem of scheme equivalence. Manna presents several examples of equivalence proofs that work by applying a sequence of local simplifying transformations to the schemes. These transformations operate on the graph itself and are combinatorial in nature.

In this paper we present a purely algebraic approach to the equivalence problem using KAT. Instead of transforming the graphs of the schemes directly, we regard them as automata of a certain generalized form and use a generalization of Kleene's theorem to construct equivalent expressions in the language of KAT. We can then give a purely equational proof in KAT of the equivalence of the resulting expressions. All our atomic transformations are special cases of four fundamental transformations. Thus the combinatorial graph manipulations of Manna are replaced with simple equational rewriting. This approach succeeds even for the most complicated of Manna's examples, a particularly intricate equivalence due to Paterson (see [22]).

The chief advantage of this approach is that the objects we are manipulating are linguistic expressions as opposed to combinatorial objects such as graphs. This gives a number of benefits. Since the semantics is compositional, it is easier to give rigorous proofs of soundness. Comparison of expressions to be simplified with the rules of KAT helps to suggest tactics for simplification. Finally, since the formalism is based on classical equational logic, it is more amenable to implementation.

2 Definitions

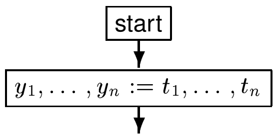
2.1 Flowchart Schemes

We follow Manna's development [22]. Let Σ be a fixed finite ranked alphabet consisting of function symbols $f, g, \ldots$ and relation symbols $P, Q, \ldots$, each with a fixed nonnegative *arity* (number of inputs). We omit any explicit notation for the arity of a symbol, except that the mention of an expression $f(t_1, \dots, t_n)$ or $P(t_1, \dots, t_n)$ carries with it the implicit proviso that f or P is n-ary. Function symbols of arity 0 are called *individual constants* and are denoted $a, b, \dots$. In addition to the symbols in Σ, there is an infinite set of *individual variables* $x, y, z, \dots$.

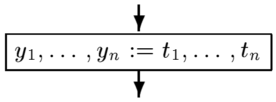
A *term* is a well-formed (i.e., arity-respecting) expression composed of individual variables and function symbols of Σ. For example, if f is unary, g is binary, a is nullary, and x, y are variables, then $g(g(x, a), f(f(y)))$ is a term. An *atomic formula* is an expression of the form $P(t_1, \dots, t_n)$, where the $t_1, \dots, t_n$ are terms.

A *flowchart scheme* S over Σ consists of a finite *flow diagram*, a designated finite set of *input variables*, a designated finite set of *output variables*, and a designated finite set of *work variables*. The sets of input, output, and work variables are pairwise disjoint. The flow diagram is composed of the following five types of statements:

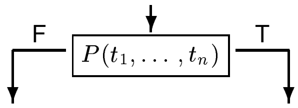
1. Start statement:



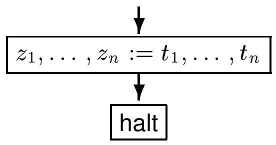
2. Assignment statement:



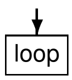
3. Test statement:



4. Halt statement:



5. Loop statement:



Each flowchart scheme begins with a single start statement. Immediately after the start statement, the work variables must be initialized with terms that depend only on the input variables. Similarly, immediately before each halt statement, the output variables must be assigned values depending only on the work variables and input variables. The scheme may contain other assignment or test statements involving the work variables only. These syntactic restrictions ensure that the output values depend functionally on the input values. We might relax these restrictions to require only that each work variable be initialized before it is used and each output variable be set before halting, but without syntactic enforcement these conditions are undecidable.

An *interpretation* for Σ consists of a first-order structure D of signature Σ, which provides meanings for all the symbols in Σ. Each n-ary function symbol $f \in \Sigma$ is interpreted as an n-ary function $f^D : D^n \to D$ and each relation symbol $P \in \Sigma$ is interpreted as an n-ary relation $P^D \subseteq D^n$.

A *flowchart program* is a pair $<S, D>$, where S is a scheme and D is an interpretation. The semantics of $<S, D>$ is operational; see Manna [22] for a formal definition. Under this semantics, each program denotes a partial function from the set of input valuations (valuations of the input variables) over D to the set of output valuations (valuations of the output variables) over D. The assignment statement $y_1, \ldots, y_n := t_1, \ldots, t_n$ denotes the simultaneous assignment of the current values of terms $t_1, \ldots, t_n$ to the variables $y_1, \ldots, y_n$. The loop statement denotes an infinite non-halting computation.

We may compare two schemes only if they have the same input variables and the same output variables. Two such schemes are said to be *compatible*. For a given pair of compatible schemes S and S' and interpretation D, we say that the programs $<S, D>$ and $<S', D>$ are *equivalent* if they denote the same partial function from input valuations to output valuations; that is, if for every input valuation, either

- both programs do not halt, or
- both programs halt and produce the same output valuation.

Two compatible flowchart schemes S and S' are *equivalent* if $<S, D>$ and $<S', D>$ are equivalent for every interpretation D.

In our treatment, we will make some simplifying assumptions that entail no loss of generality.

- We restrict our attention to simple assignments $y := t$ only. Parallel assignments $y_1, \ldots, y_n := t_1, \ldots, t_n$ can be simulated by introducing new work variables if necessary.
- We assume the domain contains a designated neutral element $\perp$ to which all non-input variables are initialized immediately after the start and to which all not-output variables are set immediately before halting. The value of $\perp$ does not matter. This obviates the need for the initialization of work variables and the setting of output variables. Accordingly, our start and halt statements take the simpler form

respectively.

- We dispense with the loop statement, since it can be simulated easily.

2.2 Kleene Algebra with Tests

Kleene algebra was introduced by S. C. Kleene [8] (see also [5]). We define a *Kleene algebra* (KA) to be a structure $(K,\ +,\ \cdot,\ ^*,\ 0,\ 1)$, where $(K,\ +,\ \cdot,\ 0,\ 1)$ is an idempotent semiring, $\mathsf{p}^*\mathsf{q}$ is the least solution to $\mathsf{q} + \mathsf{px} \leq \mathsf{x}$, and qp^* the least solution to $\mathsf{q} + \mathsf{xp} \leq \mathsf{x}$. Here "least" refers to the natural partial order $\mathsf{p} \leq \mathsf{q} \leftrightarrow \mathsf{p} + \mathsf{q} = \mathsf{q}$. The operation $+$ gives the supremum with respect to $\leq$. This particular axiomatization is from [11].

We normally omit the $\cdot$, writing pq for $\mathsf{p} \cdot \mathsf{q}$. The precedence of the operators is $^* > \cdot > +$. Thus $\mathsf{p} + \mathsf{qr}^*$ should be parsed $\mathsf{p} + (\mathsf{q}(\mathsf{r}^*))$.

Typical models include the family of regular sets of strings over a finite alphabet, the family of binary relations on a set, and the family of $n \times n$ matrices over another Kleene algebra.

The following are some elementary theorems of KA.

$$\mathsf{p}^* \;=\; 1 + \mathsf{pp}^* \;=\; 1 + \mathsf{p}^*\mathsf{p} \;=\; \mathsf{p}^*\mathsf{p}^* \;=\; \mathsf{p}^{**} \tag{1}$$

$$\mathsf{p}(\mathsf{qp})^* \;=\; (\mathsf{pq})^*\mathsf{p} \tag{2}$$

$$\mathsf{p}^*(\mathsf{qp}^*)^* \;=\; (\mathsf{p} + \mathsf{q})^* \;=\; (\mathsf{p}^*\mathsf{q})^*\mathsf{p}^* \tag{3}$$

$$\mathsf{px} = \mathsf{xq} \;\rightarrow\; \mathsf{p}^*\mathsf{x} = \mathsf{xq}^* \tag{4}$$

The identities (2) and (3) are called the *sliding rule* and the *denesting rule*, respectively. These rules are particularly useful in program equivalence proofs. The property (4) is a kind of bisimulation property. It plays a prominent role in the completeness proof of [11]. We refer the reader to [11] for further definitions and basic results.

A *Kleene algebra with tests* (KAT) [13] is a Kleene algebra with an embedded Boolean subalgebra. More precisely, it is a two-sorted structure $(K, B, +, \cdot, ^*, ^-, 0, 1)$, where $^-$ is a unary operator defined only on B, such that $B \subseteq K$, $(K,\ +,\ \cdot,\ ^*,\ 0,\ 1)$ is a Kleene algebra, and $(B,\ +,\ \cdot,\ ^-,\ 0,\ 1)$ is a Boolean algebra. The elements of B are called *tests*. We reserve the letters $\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}, \ldots$ for arbitrary elements of K and $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ for tests.

When applied to arbitrary elements of K, the operators $+, \cdot, 0, 1$ refer to nondeterministic choice, composition, fail and skip, respectively. 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, sequentially testing b and c is the same as testing their conjunction bc.

The following are some basic theorems of KAT.

$$\mathsf{bq} = \mathsf{qb} \;\rightarrow\; \mathsf{bq}^* \;=\; (\mathsf{bq})^*\mathsf{b} \;=\; \mathsf{q}^*\mathsf{b} \;=\; \mathsf{b}(\mathsf{qb})^*$$

$$\mathsf{bq} = \mathsf{qb} \;\leftrightarrow\; \overline{\mathsf{b}}\mathsf{q} = \mathsf{q}\overline{\mathsf{b}} \;\leftrightarrow\; \mathsf{bq}\overline{\mathsf{b}} + \overline{\mathsf{b}}\mathsf{qb} = 0.$$

See [13] for further definitions and basic results.

2.3 Semantics of KAT

For applications in program verification, we usually interpret programs and tests as sets of traces or sets of binary relations on a set of states. Both these classes of algebras are defined in terms of *Kripke frames*. A Kripke frame over a set of atomic programs P and a set of atomic tests B (not necessarily finite) is a structure $(K,\ \mathfrak{m}_K)$, where

$$\mathfrak{m}_K : \mathsf{P} \to 2^{K\times K} \qquad\qquad \mathfrak{m}_K : \mathsf{B} \to 2^K.$$

Elements of K are called *states*. A *trace* in K is a sequence $u_0\mathsf{p}_0u_1 \cdots u_{n-1}\mathsf{p}_{n-1}u_n$, where $n \geq 0$, $u_i \in K$, $\mathsf{p}_i \in \mathsf{P}$, and $(u_i, u_{i+1}) \in \mathfrak{m}_K(\mathsf{p}_i)$ for $0 \leq i \leq n-1$. We denote traces by $\sigma, \tau, \ldots$. The first and last states of a trace σ are denoted **first**(σ) and **last**(σ), respectively. If **last**$(\sigma) =$ **first**(τ), we can fuse σ and τ to get the trace $\sigma\tau$.

The powerset of the set of all traces in K forms a KAT in which $+$ is set union, $\cdot$ is the operation

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

and A^* is the union of all finite powers of A. The Boolean elements are the sets of traces of length 0, i.e. traces consisting of a single state. A canonical interpretation for KAT expressions over P and B is given by

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

extended homomorphically. A set of traces is *regular* if it is $[\![\mathsf{p}]\!]_K$ for some KAT expression p. The subalgebra of all regular sets of traces of K is denoted $\mathbf{Tr}_K$ [19].

The set of all binary relations on K also forms a KAT under the standard interpretation of the KAT operators (see [19]). The operator $\cdot$ is ordinary relational composition. The Boolean elements are subsets of the identity relation. As above, one can define a canonical interpretation

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

A binary relation is *regular* if it is $[\mathsf{p}]_K$ for some KAT expression p. The relational algebra consisting of all regular sets of binary relations on K is denoted $\mathbf{Rel}_K$.

These classes of algebras are related by the KAT homomorphism

$$\mathrm{Ext} : X \ \mapsto\ \{(\mathbf{first}(\sigma), \mathbf{last}(\sigma)) \mid \sigma \in X\},$$

which maps $\mathbf{Tr}_K$ canonically onto $\mathbf{Rel}_K$ in the sense that

$$\mathrm{Ext}([\![\mathsf{p}]\!]_K) \ =\ [\mathsf{p}]_K \tag{5}$$

for all expressions p [19, §3.4].

When B is finite, a language-theoretic interpretation is given by the algebra of regular sets of *guarded strings* [17]. Let $\mathcal{A}_{\mathsf{B}}$ denote the set of atoms (minimal nonzero elements) of the free Boolean algebra generated by B. A *guarded string* is a sequence $\alpha_0\mathsf{p}_0\alpha_1\cdots\alpha_{n-1}\mathsf{p}_{n-1}\alpha_n$, where $\alpha_i \in \mathcal{A}_\mathsf{B}$ and $\mathsf{p} \in \mathsf{P}$. The algebra of regular sets of guarded strings is most easily described as the regular trace algebra $\mathbf{Tr}_G$ of the Kripke frame G whose states are $\mathcal{A}_\mathsf{B}$ and

$$\begin{aligned} \mathfrak{m}_G(\mathsf{p}) &\stackrel{\mathrm{def}}{=} \mathcal{A}_\mathsf{B} \times \mathcal{A}_\mathsf{B}, \quad \mathsf{p} \in \mathsf{P} \\ \mathfrak{m}_G(\mathsf{b}) &\stackrel{\mathrm{def}}{=} \{\alpha \in \mathcal{A}_\mathsf{B} \mid \alpha \leq \mathsf{b}\}, \quad \mathsf{b} \in \mathsf{B}. \end{aligned}$$

There is a natural homomorphism from this algebra to the regular trace algebra $\mathbf{Tr}_K$ of any Kripke frame K over P, B [19, Lemma 3.2].

3 Schematic KAT

Schematic KAT (SKAT) is a version of KAT whose intended semantics coincides with the semantics of flowchart schemes over a ranked alphabet Σ as described in Section 2.1. The atomic programs P of SKAT are *assignments* $x := t$, where x is a variable and t is a Σ-term, and the atomic tests B are $P(t_1, \ldots, t_n)$, where P is an n-ary relation symbol of Σ and $t_1, \ldots, t_n$ are Σ-terms.

3.1 Semantics of Schematic KAT

We are primarily interested in interpretations over Kripke frames of a special form defined with respect to a first-order structure D of signature Σ. Such Kripke frames are called *Tarskian*. The structure D provides arity-respecting interpretations for the atomic function and relation symbols in Σ. *States* are *valuations*, or functions that assign a value from D to each individual variable. Valuations over D are denoted $\theta, \eta, \ldots$. The set of all such valuations is denoted Val_D. The action of the assignment $x := t$ is to change the state in the following way: the expression t is evaluated in the input state and assigned to x, and the resulting valuation is the output state. Formally,

$$\begin{aligned} \mathfrak{m}_D(x := t) &\stackrel{\mathrm{def}}{=} \{(\theta, \eta) \mid \eta = \theta[x/\theta(t)]\} \\ \mathfrak{m}_D(P(t_1, \ldots, t_n)) &\stackrel{\mathrm{def}}{=} \{\theta \mid P^D(\theta(t_1), \ldots, \theta(t_n))\}, \end{aligned}$$

where $\theta[x/a]$ denotes the valuation that agrees with θ on all variables except possibly x, on which it takes the value a. As described in Section 2.3, a Tarskian frame D gives rise to a regular trace algebra $\mathbf{Tr}_D$ and a regular relational algebra $\mathbf{Rel}_D$.

3.2 Schematic Automata

To obtain an expression of SKAT from a given scheme, we will view a scheme as a kind of generalized finite automaton called a *schematic automaton*. Schematic automata are a generalization of automata on guarded strings (AGS) introduced in [15]. AGS are like ordinary finite automata, except that they take guarded strings as inputs. In turn, schematic automata over Σ are like AGS, except that they take traces of Kripke frames over Σ as inputs.

AGS are defined as follows. Let $\mathcal{B}$ be the set of tests built from a finite set of atomic tests B. Note that $\mathcal{B}$ is infinite in general. An AGS over P, B is a labeled transition graph with labels $\mathsf{P} \cup \mathcal{B}$ and designated sets of start and accept states. Transitions labeled with atomic programs are called *action transitions* and those labeled with tests are called *test transitions*.

Informally, a nondeterministic AGS works as follows. An input is a guarded string x over P, B. We start with a pebble on an input state with the input pointer reading **first**(x). At any point in the computation, the pebble is occupying a state, and the input pointer is pointing to an atom somewhere in x. If there is an action transition from the current state labeled with $\mathsf{p} \in \mathsf{P}$, and the next program symbol in x is p, then we may nondeterministically choose to move the pebble along that transition and advance the input pointer beyond p. If there is a test transition from the current state labeled with a test $\mathsf{b} \in \mathcal{B}$, then that transition is *enabled* if the current atom α in x satisfies b, where we regard α as a truth assignment to B. We may nondeterministically choose to move the pebble along any enabled test transition, but we do not advance the input pointer. The input is accepted if the pebble ever occupies an accept state while the input pointer is reading **last**(x).

Ordinary finite automata with ε-transitions can be regarded as the special case in which $\mathsf{B} = \varnothing$, giving the two-element Boolean algebra $\{0, 1\}$. An ε-transition is just a test transition with Boolean label 1. In this case the only atom is 1, which is suppressed in the input string.

It was shown in [15, Theorem 3.1] that a version of Kleene's theorem holds for automata on guarded strings: given any AGS, one can efficiently construct an equivalent KAT expression, and vice versa. Here equivalence means that the AGS and the KAT expression represent the same set of guarded strings. Moreover, the constructions in both directions are the same as in the usual version of Kleene's theorem. A more formal treatment and further details can be found in [15].

AGS are unsuitable for schematic interpretations over Σ. One complication is that the set B of atomic tests is infinite in general. It is true that only finitely many tests occur in any fixed program, but this is insufficient for our purposes because our transformation rules of Section 4.1 do not respect this restriction.

For schematic automata M over Σ, B is the (possibly infinite) set of atomic formulas over Σ, $\mathcal{B}$ is the set of quantifier-free first-order formulas, and the inputs are no longer guarded strings but traces $u_0\mathsf{p}_0u_1 \cdots u_{n-1}\mathsf{p}_{n-1}u_n$ of a Kripke frame K over Σ. Let us call the states of K *nodes* and the states of M *states* to avoid confusion. At any point in time, the input pointer is reading a node u of the input trace, and the atomic formulas true at that node determine which test transitions from the current state are enabled. Specifically, a test transition with label b is enabled if $u \vDash \mathsf{b}$; that is, if b is a logical consequence of

$$\begin{aligned} &\{P(t_1, \ldots, t_n) \mid u \in \mathfrak{m}_K(P(t_1, \ldots, t_n))\} \\ &\quad \cup \quad \{\neg P(t_1, \ldots, t_n) \mid u \notin \mathfrak{m}_K(P(t_1, \ldots, t_n))\}. \end{aligned}$$

The formal definition of acceptance for schematic automata is similar to AGS. Let $R(M)$ be the set of strings over the alphabet $(\mathsf{P} \cup \mathcal{B})^*$ accepted by M under the classical definition of finite automaton acceptance. Each string $\mathsf{x} \in (\mathsf{P} \cup \mathcal{B})^*$ is a KAT

expression, therefore represents a set of traces $[\![\mathsf{x}]\!]_K$. A trace σ of K is *accepted* by M if there exists an $\mathsf{x} \in R(M)$ such that $\sigma \in [\![\mathsf{x}]\!]_K$; that is, if $\sigma \in H_K(R(M))$, where

$$H_K(A) \quad = \quad \bigcup_{\mathsf{p} \in A} [\![\mathsf{p}]\!]_K.$$

We denote by $L_K(M)$ the set of traces of K accepted by M.

The following lemma is Kleene's theorem for schematic automata. It is a direct generalization of [15, Theorem 3.1] for AGS. The proof is a straightforward modification of the proof of that theorem.

Lemma 3.1 *Schematic automata accept all and only regular sets of traces. That is, for each schematic automaton M over Σ, one can construct an equivalent* KAT *expression* p *over Σ, and for each* KAT *expression* p *over Σ, one can construct an equivalent schematic automaton M over Σ. Here* equivalent *means that for any Kripke frame K over Σ, $L_K(M) = [\![\mathsf{p}]\!]_K$.*

Proof. Given p, consider it as a regular expression over the alphabet $\mathsf{P} \cup \mathcal{B}$ with the classical interpretation, and construct an equivalent finite automaton M with input alphabet $\mathsf{P} \cup \mathcal{B}$ as in the usual proof of Kleene's theorem (see e.g. [12]). Conversely, given a finite automaton M with input alphabet $\mathsf{P} \cup \mathcal{B}$, construct an equivalent regular expression p. Let $R(\mathsf{p})$ denote the regular subset of $(\mathsf{P} \cup \mathcal{B})^*$ denoted by p under the classical interpretation of regular expressions. In both constructions, $R(\mathsf{p}) = R(M)$.

We claim that in both constructions, $[\![\mathsf{p}]\!]_K = L_K(M)$. To show this, it suffices to show that $L_K(M) = H_K(R(M))$ and $[\![\mathsf{p}]\!]_K = H_K(R(\mathsf{p}))$. The former equation is just the definition of acceptance for schematic automata. For the latter, it is easily shown that the map H_K is a homomorphism with respect to the regular operators. Moreover, the maps $[\![\text{-}]\!]_K$ and $H_K \circ R$ agree on the generators P and $\mathcal{B}$, since $H_K(R(\mathsf{p})) = H_K(\{\mathsf{p}\}) = [\![\mathsf{p}]\!]_K$ for $\mathsf{p} \in \mathsf{P} \cup \mathcal{B}$, and $H_K(R(0)) = [\![0]\!]_K = \varnothing$. It follows by induction that $[\![\text{-}]\!]_K$ and $H_K \circ R$ agree on all regular expressions over $\mathsf{P} \cup \mathcal{B}$. $\square$

4 Soundness of the Method

Our method of proving the equivalence of compatible flowchart schemes proceeds as follows. We assume that all non-input variables of both schemes are initialized with $y := \bot$ at the start in both schemes and that all non-output variables of both schemes are set by $y := \bot$ in both schemes before halting. This does not affect the semantics of the scheme. (In practice, we do not write the assignments $y := \bot$ explicitly, but we assume that they are there.) We view each scheme as a schematic automaton as defined in Section 3.2. Schemes and schematic automata are essentially notational variants of each other; the only difference is that the labels are on the edges in schematic automata instead of on the nodes as in flowchart schemes. The start state is the start statement and the accept states are the halt statements. See Figs. 1–4 for a comparison of these two views. We use the construction of Kleene's theorem (Lemma 3.1) to write an equivalent SKAT expression for each automaton. We then use the axioms of KAT in conjunction with certain additional valid identities given below in Section 4.1 to prove the equivalence of the resulting expressions.

The soundness of our method is based on the following theorem.

Theorem 4.1 *Let S and T be two compatible flowchart schemes over Σ, and let* p *and* q *be the* SKAT *expressions obtained by applying the above procedure to S and T. For any interpretation D over Σ,* $[\mathtt{p}]_D = [\mathtt{q}]_D$ *if and only if* $<S, D>$ *and* $<T, D>$ *are equivalent in the sense of Section 2.1.*

Proof. Since schemes are deterministic, each pair $<S, D>$ and starting valuation $\theta \in \mathrm{Val}_D$ determine a unique finite or infinite trace σ such that $\textbf{first}(\sigma) = \theta$. The trace σ is finite iff the program $<S, D>$ halts on input valuation θ, and in that case its output valuation is $\textbf{last}(\sigma)$. By the semantics of schematic automata, the set of all such finite traces is just $L_D(S)$, the set of traces accepted by S viewed as a schematic automaton, and $<S, D>$ and $<T, D>$ are equivalent iff the partial functions $\mathrm{Ext}(L_D(S))$ and $\mathrm{Ext}(L_D(T))$ from input valuations to output valuations are the same. (Our convention regarding the initialization of non-input variables and the setting of non-output variables to the neutral element $\perp$ makes the distinction between the different types of variables irrelevant.) By Lemma 3.1 and (5), this occurs iff $[\mathtt{p}]_D = [\mathtt{q}]_D$. $\square$

The significance of Theorem 4.1 is that scheme equivalence amounts to the equational theory of the regular relation algebras $\mathbf{Rel}_D$ of Tarskian frames D over Σ. From a practical standpoint, this theorem justifies the use of the KAT axioms in scheme equivalence proofs. Since the $\mathbf{Rel}_D$ are Kleene algebras with tests, any theorem $\mathsf{p} = \mathsf{q}$ of KAT holds under the interpretation $[\text{-}]_D$. Moreover, any additional identities that can be determined to hold under all such interpretations can be used in scheme equivalence proofs. We identify some useful such identities in the next section.

4.1 Additional Schematic Identities

In manipulating SKAT expressions, we will need to make use of extra equational postulates that are valid in all relational algebras $\mathbf{Rel}_D$ of Tarskian frames. These play the same role as the assignment axiom of Hoare Logic and may be regarded as axioms of SKAT. All such postulates we will need in this paper are instances of the following four identities:

$$x := s \,;\, y := t \quad = \quad y := t[x/s] \,;\, x := s \qquad (y \notin \mathrm{FV}(s)) \tag{6}$$

$$x := s \,;\, y := t \quad = \quad x := s \,;\, y := t[x/s] \qquad (x \notin \mathrm{FV}(s)) \tag{7}$$

$$x := s \,;\, x := t \quad = \quad x := t[x/s] \tag{8}$$

$$\varphi[x/t] \,;\, x := t \quad = \quad x := t \,;\, \varphi \tag{9}$$

where in (6) and (7), x and y are distinct variables and $\mathrm{FV}(s)$ denotes the set of variables occurring in s. Special cases of (6) and (9) are the commutativity conditions

$$x := s \,;\, y := t \quad = \quad y := t \,;\, x := s \qquad (x \notin \mathrm{FV}(t),\ y \notin \mathrm{FV}(s)) \tag{10}$$

$$\varphi \,;\, x := t \quad = \quad x := t \,;\, \varphi \qquad (x \notin \mathrm{FV}(\varphi)) \tag{11}$$

The notation $\varphi[x/t]$ or $s[x/t]$ denotes the result of substituting the term t for all occurrences of x in the formula φ or term s, respectively. This is not to be confused with $\theta[x/a]$ for valuations θ defined previously in Section 3.1. We use both notations below. They are related by the equation

$$\theta(t[x/s]) \quad = \quad \theta[x/\theta(s)](t), \tag{12}$$

which is easily proved by induction on t.

We will prove the soundness of (6)–(11) below (Theorem 4.3). Note that these identities are not valid in the trace algebras $\mathbf{Tr}_D$, but they are valid in the corresponding relational algebras $\mathbf{Rel}_D$.

It is interesting to compare (9) with the assignment axiom of Hoare Logic. The Hoare partial correctness assertion $\{\mathsf{b}\}\mathsf{p}\{\mathsf{c}\}$ is encoded in KAT by the equation $\mathsf{bp\overline{c}} = 0$, or equivalently, $\mathsf{bpc} = \mathsf{bp}$. Intuitively, the former says that the program p with preguard b and postguard $\overline{\mathsf{c}}$ has no halting execution, and the latter says that testing c after executing bp is always redundant.

The assignment axiom of Hoare Logic is

$$\{\varphi[x/t]\}\; x := t\; \{\varphi\},$$

which is represented in schematic KAT by either of the two equivalent equations

$$\varphi[x/t]\;;\; x := t\;;\;\varphi \qquad = \qquad \varphi[x/t]\;;\; x := t \tag{13}$$

$$\varphi[x/t]\;;\; x := t\;;\;\neg\varphi \qquad = \qquad 0. \tag{14}$$

The interesting fact is that (9) is equivalent to two applications of the Hoare assignment rule, one for φ and one for $\neg\varphi$. This can be seen from the following lemma by taking b, p, and c to be $\varphi[x/t]$, $x := t$, and φ, respectively.

Lemma 4.2 *The following equations of* KAT *are equivalent:*

(i) $\mathsf{bp} = \mathsf{pc}$

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

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

Proof. We prove the equivalence of (i) and (iii); the equivalence of (ii) and (iii) is symmetric. If $\mathsf{bp} = \mathsf{pc}$, then $\mathsf{bp}\overline{\mathsf{c}} + \overline{\mathsf{b}}\mathsf{pc} = \mathsf{pc}\overline{\mathsf{c}} + \overline{\mathsf{b}}\mathsf{bp} = 0$. Conversely, suppose $\mathsf{bp}\overline{\mathsf{c}} + \overline{\mathsf{b}}\mathsf{pc} = 0$. Then $\mathsf{bp}\overline{\mathsf{c}} = 0$ and $\overline{\mathsf{b}}\mathsf{pc} = 0$, therefore

$$\mathsf{bp} \;=\; \mathsf{bp}(\mathsf{c} + \overline{\mathsf{c}}) \;=\; \mathsf{bpc} + \mathsf{bp}\overline{\mathsf{c}} \;=\; \mathsf{bpc} \;=\; \overline{\mathsf{b}}\mathsf{pc} + \mathsf{bpc} \;=\; (\overline{\mathsf{b}} + \mathsf{b})\mathsf{pc} \;=\; \mathsf{pc}.$$

□

We conclude this section with a proof of soundness of the identities (6)–(11).

Theorem 4.3 *Equations* (6)–(11) *hold under the interpretation* `[-]`$_D$ *for any Tarskian frame* D *over* Σ.

Proof. We need only prove the result for (6)–(9), since (10) and (11) are special cases.

An instance of (6) is of the form

$$x := s\ ;\ y := t \quad = \quad y := t[x/s]\ ;\ x := s$$

where x and y are distinct variables and $y \notin \mathrm{FV}(s)$. We need to show that for any Tarskian frame D,

$$[\![x := s\ ;\ y := t]\!]_D \quad = \quad [\![y := t[x/s]\ ;\ x := s]\!]_D.$$

We have

$$\begin{aligned}
&[\![x := s\ ;\ y := t]\!]_D \\
&= [\![x := s]\!]_D \circ [\![y := t]\!]_D \\
&= \{(\theta, \theta[x/\theta(s)]) \mid \theta \in \mathrm{Val}_D\} \circ \{(\eta, \eta[y/\eta(t)]) \mid \eta \in \mathrm{Val}_D\} \\
&= \{(\theta, \theta[x/\theta(s)][y/\theta[x/\theta(s)](t)]) \mid \theta \in \mathrm{Val}_D\}
\end{aligned}$$

and similarly,

$$\begin{aligned}
&[\![y := t[x/s]\ ;\ x := s]\!]_D \\
&= [\![y := t[x/s]]\!]_D \circ [\![x := s]\!]_D \\
&= \{(\theta, \theta[y/\theta(t[x/s])]) \mid \theta \in \mathrm{Val}_D\} \circ \{(\eta, \eta[x/\eta(s)]) \mid \eta \in \mathrm{Val}_D\} \\
&= \{(\theta, \theta[y/\theta(t[x/s])][x/\theta[y/\theta(t[x/s])](s)]) \mid \theta \in \mathrm{Val}_D\}
\end{aligned}$$

so it suffices to show that for all $\theta \in \mathrm{Val}_D$,

$$\theta[x/\theta(s)][y/\theta[x/\theta(s)](t)] \quad = \quad \theta[y/\theta(t[x/s])][x/\theta[y/\theta(t[x/s])](s)].$$

Starting from the right-hand side,

$$\begin{aligned}
&\theta[y/\theta(t[x/s])][x/\theta[y/\theta(t[x/s])](s)] \\
&= \theta[y/\theta(t[x/s])][x/\theta(s)] && y \notin \mathrm{FV}(s) \\
&= \theta[x/\theta(s)][y/\theta(t[x/s])] && x \text{ and } y \text{ are distinct} \\
&= \theta[x/\theta(s)][y/\theta[x/\theta(s)](t)] && \text{by (12).}
\end{aligned}$$

The proofs for (7) and (8) are similar. For (9),

$$\begin{aligned}
[\![\varphi[x/t]\ ;\ x := t]\!]_D &= [\![\varphi[x/t]]\!]_D \circ [\![x := t]\!]_D \\
&= \{(\theta, \theta) \mid \theta \in \mathrm{Val}_D,\ \theta \vDash \varphi[x/t]\} \circ \{(\theta, \theta[x/\theta(t)]) \mid \theta \in \mathrm{Val}_D\} \\
&= \{(\theta, \theta[x/\theta(t)]) \mid \theta \in \mathrm{Val}_D,\ \theta \vDash \varphi[x/t]\},
\end{aligned}$$

$$\begin{aligned}
[\![x := t\ ;\ \varphi]\!]_D &= [\![x := t]\!]_D \circ [\![\varphi]\!]_D \\
&= \{(\theta, \theta[x/\theta(t)]) \mid \theta \in \mathrm{Val}_D\} \circ \{(\eta, \eta) \mid \eta \in \mathrm{Val}_D,\ \eta \vDash \varphi\} \\
&= \{(\theta, \theta[x/\theta(t)]) \mid \theta \in \mathrm{Val}_D,\ \theta[x/\theta(t)] \vDash \varphi\},
\end{aligned}$$

and $\theta \vDash \varphi[x/t]$ iff $\theta[x/\theta(t)] \vDash \varphi$ by (12). $\square$

In addition to our previous usage, we will also use the name SKAT to refer to the axiomatization consisting of the axioms of KAT plus the identities (6)–(9). Henceforth, the equality symbol $=$ between schematic KAT expressions denotes provable equality in this system; that is, equality in the free KAT over the language Σ modulo (6)–(9).

4.2 Some Useful Metatheorems

The following is a general metatheorem with several useful applications.

Lemma 4.4 *Let f and g be homomorphisms of KAT expressions, and let x be a fixed KAT expression. Consider the condition*

$$f(\mathsf{p})\mathsf{x} \quad = \quad \mathsf{x}g(\mathsf{p}). \tag{15}$$

If (15) *holds for all atomic* p*, then* (15) *holds for all* p.

Proof. The proof is by induction on p. If p is atomic, then (15) holds for p by assumption. For the case of $\cdot$, by two applications of the induction hypothesis and the fact that f and g are homomorphisms,

$$f(\mathsf{pq})\mathsf{x} \;=\; f(\mathsf{p})f(\mathsf{q})\mathsf{x} \;=\; f(\mathsf{p})\mathsf{x}g(\mathsf{q}) \;=\; \mathsf{x}g(\mathsf{p})g(\mathsf{q}) \;=\; \mathsf{x}g(\mathsf{pq}).$$

The cases of $+$, 0, and 1 are equally straightforward. The case of $\bar{}$ follows from Lemma 4.2. Finally, for *, we have $f(\mathsf{p}^*)\mathsf{x} = f(\mathsf{p})^*\mathsf{x}$ and $\mathsf{x}g(\mathsf{p}^*) = \mathsf{x}g(\mathsf{p})^*$ since f and g are homomorphisms. The result is then immediate from the induction hypothesis and (4). $\square$

One immediate consequence of Lemma 4.4 is that if x commutes with all atomic expressions of p, then x commutes with p. This is obtained by taking f and g to be the identity maps.

Here are two more useful applications.

Lemma 4.5 (Elimination of useless variables.) *Let p be an expression of SKAT and y a variable such that y does not appear on the right-hand side of any assignment or in any test of p. (Thus the only occurrences of y in p are on the left-hand sides of assignments.) Let $g(\mathsf{p})$ be the expression obtained by deleting all assignments to y. Then*

$$\mathsf{p}\ ;\ y := \perp \quad = \quad g(\mathsf{p})\ ;\ y := \perp.$$

Proof. We apply Lemma 4.4 with f the identity, g the erasing homomorphism defined in the statement of the lemma, and x the assignment $y := \perp$. The assumptions of Lemma 4.4 are

$$\begin{aligned} x := t\ ;\ y := \perp \quad &= \quad y := \perp\ ;\ x := t, \quad \text{if } x \text{ and } y \text{ are distinct} \\ y := t\ ;\ y := \perp \quad &= \quad y := \perp \\ \varphi\ ;\ y := \perp \quad &= \quad y := \perp\ ;\ \varphi \end{aligned}$$

which hold by (10), (8), and (11), respectively. By Lemma 4.4,

$$\mathsf{p} \,;\, y := \bot \quad = \quad y := \bot \,;\, g(\mathsf{p}).$$

Also, since y does not appear in $g(\mathsf{p})$, we have by (10) and (11) that $y := \bot$ commutes with all atomic programs and tests of $g(\mathsf{p})$. Again by Lemma 4.4,

$$y := \bot \,;\, g(\mathsf{p}) \quad = \quad g(\mathsf{p}) \,;\, y := \bot.$$

$\square$

Lemma 4.6 (Change of work variable.) *Let* p *be an expression of* SKAT *and* x, y *variables such that* y *does not occur in* p*. Let* $\mathsf{p}[x/y]$ *be the expression obtained by changing all occurrences of* x *in* p *to* y*. Then*

$$\begin{aligned} & x := \bot \,;\, y := \bot \,;\, \mathsf{p} \,;\, x := \bot \,;\, y := \bot \\ & \quad = \quad x := \bot \,;\, y := \bot \,;\, \mathsf{p}[x/y] \,;\, x := \bot \,;\, y := \bot. \end{aligned}$$

Proof. We first apply Lemma 4.4 with f the identity, g the homomorphism defined by

$$\begin{aligned} g(z := t) & \stackrel{\text{def}}{=} z := t[x/y], \quad z \text{ different from } x \\ g(x := t) & \stackrel{\text{def}}{=} x := t[x/y] \,;\, y := t[x/y] \\ g(\varphi) & \stackrel{\text{def}}{=} \varphi[x/y], \end{aligned}$$

and x the assignment $y := x$. The preconditions of Lemma 4.4 are

$$z := t \,;\, y := x \quad = \quad y := x \,;\, z := t[x/y] \tag{16}$$

$$x := t \,;\, y := x \quad = \quad y := x \,;\, x := t[x/y] \,;\, y := t[x/y] \tag{17}$$

$$\varphi \,;\, y := x \quad = \quad y := x \,;\, \varphi[x/y]. \tag{18}$$

Equations (16) and (18) are immediate from (6) and (9), respectively, since y does not occur in t or φ. For (17), since y does not occur in t,

$$\begin{aligned} x := t \,;\, y := x & = y := t \,;\, x := t && \text{by (6)} \\ & = y := t \,;\, x := y && \text{by (7)} \\ & = y := x \,;\, y := t[x/y] \,;\, x := y && \text{by (8)} \\ & = y := x \,;\, x := t[x/y] \,;\, y := t[x/y] && \text{by (6).} \end{aligned}$$

Then

$$\begin{aligned} & x := \bot \,;\, \mathsf{p} \,;\, y := \bot \\ & = x := \bot \,;\, \mathsf{p} \,;\, y := x \,;\, y := \bot && \text{by (8)} \\ & = x := \bot \,;\, y := x \,;\, g(\mathsf{p}) \,;\, y := \bot && \text{by Lemma 4.4} \\ & = x := \bot \,;\, y := \bot \,;\, g(\mathsf{p}) \,;\, y := \bot && \text{by (7),} \end{aligned}$$

therefore

$$
\begin{aligned}
& x := \bot \,;\, y := \bot \,;\, \mathsf{p} \,;\, x := \bot \,;\, y := \bot \\
= \quad & x := \bot \,;\, y := \bot \,;\, g(\mathsf{p}) \,;\, x := \bot \,;\, y := \bot.
\end{aligned}
$$

But $g(\mathsf{p})$ is just $\mathsf{p}[x/y]$ with some extra useless assignments to x, which can be eliminated by Lemma 4.5. $\square$

5 An Extended Example

In this section we illustrate the equivalence technique by applying it to an example of Paterson (see Manna [22, pp. 253–258]).

The two schemes to be proved equivalent are called S_{6A} and S_{6E}. They are illustrated in Figs. 1 and 2, respectively. This is a particularly intricate equivalence problem requiring considerable ingenuity. Manna's proof technique is to apply various local graph transformations that work directly on the flow diagram itself. He transforms S_{6A} through a chain of equivalent schemes S_{6B}, S_{6C}, and S_{6D}, finally arriving at S_{6E}.

Define the following abbreviations:

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
\mathsf{a}_i & \stackrel{\text{def}}{=} & P(y_i) & \mathsf{a} & \stackrel{\text{def}}{=} & P(y) \\
\mathsf{b}_i & \stackrel{\text{def}}{=} & P(f(y_i)) & \mathsf{b} & \stackrel{\text{def}}{=} & P(f(y)) \\
\mathsf{c}_i & \stackrel{\text{def}}{=} & P(f(f(y_i))) & \mathsf{c} & \stackrel{\text{def}}{=} & P(f(f(y))) \\
 & & & \mathsf{d} & \stackrel{\text{def}}{=} & P(f(x)) \\
\mathsf{x}_i & \stackrel{\text{def}}{=} & y_i := x & \mathsf{x} & \stackrel{\text{def}}{=} & y := x \\
\mathsf{p}_{ij} & \stackrel{\text{def}}{=} & y_i := f(y_j) & \mathsf{p} & \stackrel{\text{def}}{=} & y := f(y) \\
\mathsf{q}_{ijk} & \stackrel{\text{def}}{=} & y_i := g(y_j, y_k) & \mathsf{q} & \stackrel{\text{def}}{=} & y := g(y, y) \\
\mathsf{r}_{ij} & \stackrel{\text{def}}{=} & y_i := f(f(y_j)) & \mathsf{r} & \stackrel{\text{def}}{=} & y := f(f(y)) \\
\mathsf{s}_i & \stackrel{\text{def}}{=} & y_i := f(x) & \mathsf{s} & \stackrel{\text{def}}{=} & y := f(x) \\
\mathsf{t}_i & \stackrel{\text{def}}{=} & y_i := g(f(x), f(x)) & \mathsf{t} & \stackrel{\text{def}}{=} & y := g(f(x), f(x)) \\
\mathsf{u}_{ijk} & \stackrel{\text{def}}{=} & y_i := g(f(f(y_j)), f(f(y_k))) & \mathsf{u} & \stackrel{\text{def}}{=} & y := g(f(f(y)), f(f(y))) \\
\mathsf{z}_i & \stackrel{\text{def}}{=} & z := y_i & \mathsf{z} & \stackrel{\text{def}}{=} & z := y \\
\mathsf{y}_i & \stackrel{\text{def}}{=} & y_i := \bot & \mathsf{y} & \stackrel{\text{def}}{=} & y := \bot.
\end{array}
$$

Our first step is to convert the two schemes to schematic automata. As previously noted, the schematic and automatic forms are merely notational variants of each other. The resulting schematic automata are shown in Figs. 3 and 4.

Now we convert the schematic automata to KAT expressions using the construction of Kleene's theorem (Lemma 3.1). This gives the following expressions:

$$
\begin{aligned}
\mathsf{S}_{6A} \quad = \quad & \mathsf{x}_1\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{214}\mathsf{q}_{311}(\overline{\mathsf{a}}_1\mathsf{p}_{11}\mathsf{q}_{214}\mathsf{q}_{311})^*\mathsf{a}_1\mathsf{p}_{13} \\
& \quad ((\overline{\mathsf{a}}_4 + \mathsf{a}_4(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\overline{\mathsf{a}}_3\mathsf{p}_{41}\mathsf{p}_{11})\mathsf{q}_{214}\mathsf{q}_{311}(\overline{\mathsf{a}}_1\mathsf{p}_{11}\mathsf{q}_{214}\mathsf{q}_{311})^*\mathsf{a}_1\mathsf{p}_{13})^* \\
& \quad\quad \mathsf{a}_4(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\mathsf{a}_3\mathsf{z}_2
\end{aligned} \tag{19}
$$

$$
\mathsf{S}_{6E} \quad = \quad \mathsf{saq}(\overline{\mathsf{a}}\mathsf{raq})^*\mathsf{az}. \tag{20}
$$

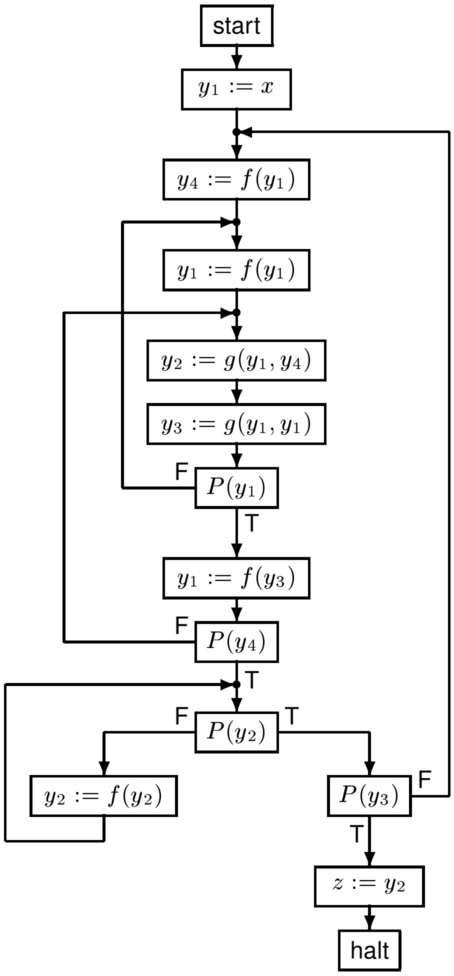


Figure 1: Scheme S_{6A} [22, p. 254]

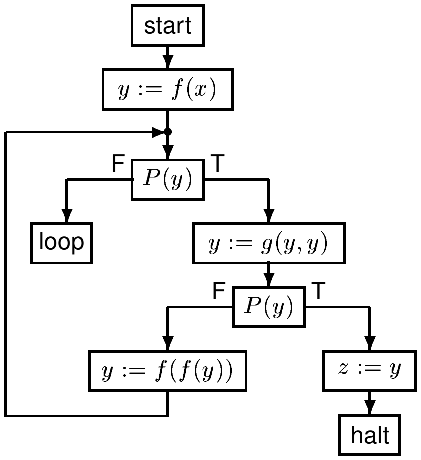


Figure 2: Scheme S_{6E} [22, p. 258]

We now proceed to simplify (19) using the axioms of KAT and the identities (6)–(9) to obtain (20). Our simplifications are guided by the form of the expressions.

First note that (19) has a nested loop. This suggests that we should first try to denest it using the denesting rule (3). By the sliding rule (2), (19) is equivalent to

$$\begin{aligned}
&x_1 p_{41} p_{11} q_{214} q_{311} (\overline{a}_1 p_{11} q_{214} q_{311})^* \\
&\quad (a_1 p_{13} (\overline{a}_4 + a_4 (\overline{a}_2 p_{22})^* a_2 \overline{a}_3 p_{41} p_{11}) q_{214} q_{311} (\overline{a}_1 p_{11} q_{214} q_{311})^*)^* \\
&\qquad a_1 p_{13} a_4 (\overline{a}_2 p_{22})^* a_2 a_3 z_2
\end{aligned} \tag{21}$$

Now using the denesting rule (3), this becomes

$$\begin{aligned}
&x_1 p_{41} p_{11} q_{214} q_{311} \\
&\quad (\overline{a}_1 p_{11} q_{214} q_{311} + a_1 p_{13} (\overline{a}_4 + a_4 (\overline{a}_2 p_{22})^* a_2 \overline{a}_3 p_{41} p_{11}) q_{214} q_{311})^* \\
&\qquad a_1 p_{13} a_4 (\overline{a}_2 p_{22})^* a_2 a_3 z_2
\end{aligned} \tag{22}$$

and by distributivity,

$$\begin{aligned}
&x_1 p_{41} p_{11} q_{214} q_{311} \\
&\quad (\overline{a}_1 p_{11} q_{214} q_{311} + a_1 p_{13} \overline{a}_4 q_{214} q_{311} + a_1 p_{13} a_4 (\overline{a}_2 p_{22})^* a_2 \overline{a}_3 p_{41} p_{11} q_{214} q_{311})^* \\
&\qquad a_1 p_{13} a_4 (\overline{a}_2 p_{22})^* a_2 a_3 z_2
\end{aligned} \tag{23}$$

Using the commutivity rules (10) and (11), distributivity, and the implicit undefining operator y_1 to dispose of the last occurrence of p_{13}, we obtain

$$\begin{aligned}
&x_1 p_{41} p_{11} q_{214} q_{311} \\
&\quad (\overline{a}_1 \overline{a}_4 p_{11} q_{214} q_{311} + \overline{a}_1 a_4 p_{11} q_{214} q_{311} \\
&\quad + a_1 \overline{a}_4 p_{13} q_{214} q_{311} + a_1 a_4 p_{13} (\overline{a}_2 p_{22})^* a_2 \overline{a}_3 p_{41} p_{11} q_{214} q_{311})^* \\
&\qquad (\overline{a}_2 p_{22})^* a_1 a_2 a_3 a_4 z_2
\end{aligned} \tag{24}$$

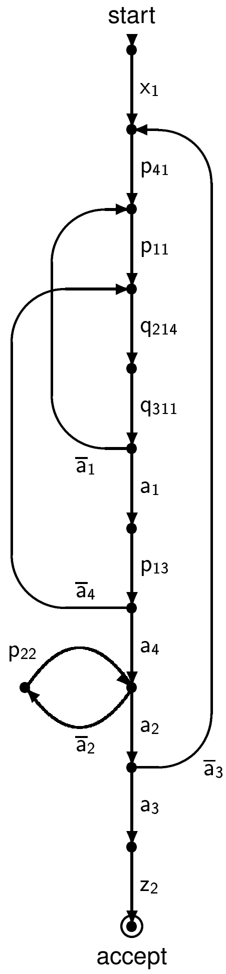


Figure 3: Scheme S_{6A}, automatic form

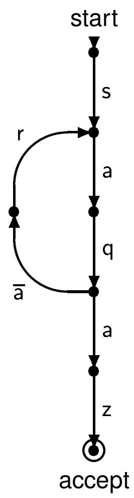


Figure 4: Scheme S_{6E}, automatic form

Now we observe informally that if the outer loop is ever entered with a_4 false, it is impossible to escape, since neither $p_{13}q_{214}q_{311}$ nor $p_{11}q_{214}q_{311}$ changes the value of a_4 and there is a postguard a_4. To exploit this idea formally, we use the following little lemma:

$$uv = uw = 0 \quad \to \quad (u+v)^*w \;=\; v^*w. \tag{25}$$

To prove (25), suppose $uv = uw = 0$. Then $u^*v = v + u^*uv = v$, and similarly $u^*w = w$. Using denesting (3),

$$(u+v)^*w \;\;=\;\; (u^*v)^*u^*w \;\;=\;\; v^*w.$$

Applying (25) to (24) with

$$\begin{aligned}
u &= \bar{a}_1\bar{a}_4p_{11}q_{214}q_{311} + a_1\bar{a}_4p_{13}q_{214}q_{311}\\
v &= \bar{a}_1a_4p_{11}q_{214}q_{311} + a_1a_4p_{13}(\bar{a}_2p_{22})^*a_2\bar{a}_3p_{41}p_{11}q_{214}q_{311}\\
w &= (\bar{a}_2p_{22})^*a_1a_2a_3a_4z_2
\end{aligned}$$

we obtain

$$\begin{aligned}
&x_1p_{41}p_{11}q_{214}q_{311}\\
&\quad(\bar{a}_1a_4p_{11}q_{214}q_{311} + a_1a_4p_{13}(\bar{a}_2p_{22})^*a_2\bar{a}_3p_{41}p_{11}q_{214}q_{311})^*\\
&\qquad(\bar{a}_2p_{22})^*a_1a_2a_3a_4z_2
\end{aligned} \tag{26}$$

The conditions $uv = uw = 0$ hold because $\bar{a}_4$ commutes with everything to its right in u and a_4 commutes with everything to its left in v and w.

Now we make a similar observation that the first term inside the outer loop of (26) can never be executed, because the program $\mathsf{p}_{41}\mathsf{p}_{11}$ causes a_1 and a_4 to have the same truth value. Formally,

$$\begin{array}{rcll}
\mathsf{p}_{41}\mathsf{p}_{11} & = & y_4 := f(y_1)\ ;\ y_1 := f(y_1) & \\
& = & P(f(y_1)) \leftrightarrow P(f(y_1))\ ;\ y_4 := f(y_1)\ ;\ y_1 := f(y_1) & \text{Boolean algebra} \\
& = & y_4 := f(y_1)\ ;\ P(f(y_1)) \leftrightarrow P(y_4)\ ;\ y_1 := f(y_1) & \text{by (9)} \\
& = & y_4 := f(y_1)\ ;\ y_1 := f(y_1)\ ;\ P(y_1) \leftrightarrow P(y_4) & \text{by (9)} \\
& = & \mathsf{p}_{41}\mathsf{p}_{11}(\mathsf{a}_1 \leftrightarrow \mathsf{a}_4). &
\end{array}$$

To eliminate the first term of the outer loop of (26), we can use our little lemma (25) again. We actually use its dual

$$\mathsf{vu} = \mathsf{wu} = 0 \quad \rightarrow \quad \mathsf{w}(\mathsf{u}+\mathsf{v})^* \ = \ \mathsf{wv}^*$$

with

$$\begin{array}{rcl}
\mathsf{u} & = & \overline{\mathsf{a}}_1\mathsf{a}_4\mathsf{p}_{11}\mathsf{q}_{214}\mathsf{q}_{311} \\
\mathsf{v} & = & \mathsf{a}_1\mathsf{a}_4\mathsf{p}_{13}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\overline{\mathsf{a}}_3\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{214}\mathsf{q}_{311} \\
\mathsf{w} & = & \mathsf{x}_1\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{214}\mathsf{q}_{311},
\end{array}$$

giving

$$\mathsf{x}_1\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{214}\mathsf{q}_{311}(\mathsf{a}_1\mathsf{a}_4\mathsf{p}_{13}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\overline{\mathsf{a}}_3\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{214}\mathsf{q}_{311})^*(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{a}_4\mathsf{z}_2 \qquad (27)$$

To show that the preconditions $\mathsf{vu} = \mathsf{wu} = 0$ are satisfied, it suffices to show

$$\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{214}\mathsf{q}_{311}\overline{\mathsf{a}}_1\mathsf{a}_4 \quad = \quad 0. \qquad (28)$$

This follows from the fact $\mathsf{p}_{41}\mathsf{p}_{11} = \mathsf{p}_{41}\mathsf{p}_{11}(\mathsf{a}_1 \leftrightarrow \mathsf{a}_4)$ proved above, the commutativity of $\mathsf{a}_1 \leftrightarrow \mathsf{a}_4$ with $\mathsf{q}_{214}\mathsf{q}_{311}$, and Boolean algebra. Thus (27) is justified.

At this point we can eliminate the variable y_4 by the following steps:

(i) rewrite assignments to eliminate occurrences of y_4 on the right-hand side;

(ii) show that all tests involving y_4 are redundant;

(iii) use Lemma 4.5 to remove all assignments with y_4 on the left-hand side.

For (i), we show that $\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{214} = \mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}$. Intuitively, after the first two assignments, y_1 and y_4 contain the same value, so we might as well use y_1 in the last assignment instead of y_4. This result would be easy to prove in the presence of equality, but we show how to do it without.

$$\begin{array}{rcll}
\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{214} & = & y_4 := f(y_1)\ ;\ y_1 := f(y_1)\ ;\ y_2 := g(y_1, y_4) & \\
& = & y_1 := f(y_1)\ ;\ y_4 := y_1\ ;\ y_2 := g(y_1, y_4) & \text{by (6)} \\
& = & y_1 := f(y_1)\ ;\ y_4 := y_1\ ;\ y_2 := g(y_1, y_1) & \text{by (7)} \\
& = & y_4 := f(y_1)\ ;\ y_1 := f(y_1)\ ;\ y_2 := g(y_1, y_1) & \text{by (6)} \\
& = & \mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}. &
\end{array}$$

Thus (27) is equivalent to

$$\mathsf{x}_1\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311}(\mathsf{a}_1\mathsf{a}_4\mathsf{p}_{13}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\overline{\mathsf{a}}_3\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311})^*(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{a}_4\mathsf{z}_2 \tag{29}$$

For step (ii), we use sliding (2) to get

$$\mathsf{x}_1(\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{a}_1\mathsf{a}_4\mathsf{p}_{13}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\overline{\mathsf{a}}_3)^*\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{a}_4\mathsf{z}_2 \tag{30}$$

Now $\mathsf{p}_{41}\mathsf{p}_{11} = \mathsf{p}_{41}\mathsf{p}_{11}(\mathsf{a}_1 \leftrightarrow \mathsf{a}_4)$ as shown above. It follows from this, commutativity, and Boolean algebra that $\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{a}_1\mathsf{a}_4 = \mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{a}_1$, therefore the tests a_4 are redundant. This allows us to eliminate all occurrences of a_4 in (30), giving

$$\mathsf{x}_1(\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{a}_1\mathsf{p}_{13}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\overline{\mathsf{a}}_3)^*\mathsf{p}_{41}\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{z}_2 \tag{31}$$

Now Lemma 4.5 applies, allowing us to get rid of all assignments to y_4. This gives

$$\mathsf{x}_1(\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{a}_1\mathsf{p}_{13}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\overline{\mathsf{a}}_3)^*\mathsf{p}_{11}\mathsf{q}_{211}\mathsf{q}_{311}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{z}_2 \tag{32}$$

By sliding and commutativity, we have

$$\mathsf{x}_1\mathsf{p}_{11}(\mathsf{q}_{211}\mathsf{q}_{311}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\overline{\mathsf{a}}_3\mathsf{p}_{13}\mathsf{p}_{11})^*\mathsf{q}_{211}\mathsf{q}_{311}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{z}_2 \tag{33}$$

and by two applications of (8), we have

$$\mathsf{s}_1(\mathsf{q}_{211}\mathsf{q}_{311}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\overline{\mathsf{a}}_3\mathsf{r}_{13})^*\mathsf{q}_{211}\mathsf{q}_{311}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{z}_2 \tag{34}$$

Now we eliminate the variable y_3 as we did for y_4 above. For step (i), use commutativity to get

$$\mathsf{s}_1(\mathsf{a}_1\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{r}_{13}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\overline{\mathsf{a}}_3)^*\mathsf{q}_{211}\mathsf{q}_{311}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{z}_2 \tag{35}$$

We then have

$$\begin{aligned}
\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{r}_{13} &= y_2 := g(y_1,y_1)\ ;\ y_3 := g(y_1,y_1)\ ;\ y_1 := f(f(y_3)) \\
&= y_2 := g(y_1,y_1)\ ;\ y_3 := y_2\ ;\ y_1 := f(f(y_3)) && \text{by (7)} \\
&= y_2 := g(y_1,y_1)\ ;\ y_3 := y_2\ ;\ y_1 := f(f(y_2)) && \text{by (7)} \\
&= y_2 := g(y_1,y_1)\ ;\ y_3 := g(y_1,y_1)\ ;\ y_1 := f(f(y_2)) && \text{by (7)} \\
&= \mathsf{q}_{211}\mathsf{q}_{311}\mathsf{r}_{12},
\end{aligned}$$

which allows us to rewrite (35) as

$$\mathsf{s}_1(\mathsf{a}_1\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{r}_{12}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\overline{\mathsf{a}}_3)^*\mathsf{q}_{211}\mathsf{q}_{311}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{a}_3\mathsf{z}_2 \tag{36}$$

For step (ii), we would like to show that tests involving a_3 are redundant. As above, we have $\mathsf{q}_{211}\mathsf{q}_{311} = \mathsf{q}_{211}\mathsf{q}_{311}(\mathsf{a}_2 \leftrightarrow \mathsf{a}_3)$, thus $\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{a}_2 = \mathsf{q}_{211}\mathsf{q}_{311}\mathsf{a}_3$ and $\mathsf{q}_{211}\mathsf{q}_{311}\overline{\mathsf{a}}_2 = \mathsf{q}_{211}\mathsf{q}_{311}\overline{\mathsf{a}}_3$. By commutativity, (36) is equivalent to

$$\begin{aligned}
&\mathsf{s}_1(\mathsf{a}_1\mathsf{q}_{211}\mathsf{q}_{311}\overline{\mathsf{a}}_3\mathsf{r}_{12}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2)^*\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{a}_3(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{z}_2 \\
&\quad = \mathsf{s}_1(\mathsf{a}_1\mathsf{q}_{211}\mathsf{q}_{311}\overline{\mathsf{a}}_2\mathsf{r}_{12}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2)^*\mathsf{q}_{211}\mathsf{q}_{311}\mathsf{a}_2(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{z}_2.
\end{aligned} \tag{37}$$

The assignments to y_3 are now useless and can be eliminated by Lemma 4.5, giving

$$\mathsf{s}_1(\mathsf{a}_1\mathsf{q}_{211}\overline{\mathsf{a}}_2\mathsf{r}_{12}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2)^*\mathsf{q}_{211}\mathsf{a}_2(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_1\mathsf{a}_2\mathsf{z}_2 \tag{38}$$

Furthermore, because of the preguard $\overline{\mathsf{a}}_2$ and postguard a_2, the loop $(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2$ occurring inside the outer loop of (38) must be executed at least once. Similarly, because of the preguard a_2, the loop $(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2$ occurring outside the outer loop cannot be executed at all. Formally,

$$\begin{aligned}
\overline{\mathsf{a}}_2(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2 &= \overline{\mathsf{a}}_2\mathsf{a}_2 + \overline{\mathsf{a}}_2\overline{\mathsf{a}}_2\mathsf{p}_{22}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2 \\
&= \overline{\mathsf{a}}_2\mathsf{p}_{22}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2, \\
\mathsf{a}_2(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2 &= \mathsf{a}_2\mathsf{a}_2 + \mathsf{a}_2\overline{\mathsf{a}}_2\mathsf{p}_{22}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2 \\
&= \mathsf{a}_2.
\end{aligned}$$

Thus we can rewrite (38) as

$$\mathsf{s}_1(\mathsf{a}_1\mathsf{q}_{211}\mathsf{r}_{12}\overline{\mathsf{a}}_2\mathsf{p}_{22}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2)^*\mathsf{q}_{211}\mathsf{a}_1\mathsf{a}_2\mathsf{z}_2. \tag{39}$$

Moreover, the remaining inner loop $(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2$ can be executed at most twice. To show this, we use sliding and commutativity to get

$$\begin{aligned}
&\mathsf{s}_1\mathsf{a}_1\mathsf{q}_{211}(\mathsf{r}_{12}\overline{\mathsf{a}}_2\mathsf{p}_{22}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\mathsf{a}_1\mathsf{q}_{211})^*\mathsf{a}_2\mathsf{z}_2 \\
&\quad = \mathsf{s}_1\mathsf{a}_1\mathsf{q}_{211}(\overline{\mathsf{a}}_2\mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*\mathsf{a}_2\mathsf{q}_{211})^*\mathsf{a}_2\mathsf{z}_2
\end{aligned} \tag{40}$$

As above,

$$\begin{aligned}
\mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22}\overline{\mathsf{a}}_2 &\leq \mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\mathsf{p}_{22}\overline{\mathsf{a}}_2 \\
&= \mathsf{r}_{12}\mathsf{a}_1\mathsf{r}_{22}\overline{\mathsf{a}}_2 && \text{by (8)} \\
&= \mathsf{r}_{12}\mathsf{r}_{22}\mathsf{a}_1\overline{\mathsf{a}}_2 \\
&= \mathsf{r}_{12}\mathsf{r}_{22}(\mathsf{a}_1 \leftrightarrow \mathsf{a}_2)\mathsf{a}_1\overline{\mathsf{a}}_2 \\
&= 0,
\end{aligned} \tag{41}$$

therefore

$$\begin{aligned}
&\mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^* \\
&\quad = \mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}(1 + \overline{\mathsf{a}}_2\mathsf{p}_{22} + \overline{\mathsf{a}}_2\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^*) \\
&\quad = \mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22} + \mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22} + \mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22}(\overline{\mathsf{a}}_2\mathsf{p}_{22})^* \\
&\quad = \mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22} + \mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22}
\end{aligned}$$

Thus (40) is equivalent to

$$\mathsf{s}_1\mathsf{a}_1\mathsf{q}_{211}(\overline{\mathsf{a}}_2\mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\mathsf{a}_2\mathsf{q}_{211} + \overline{\mathsf{a}}_2\mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22}\mathsf{a}_2\mathsf{q}_{211})^*\mathsf{a}_2\mathsf{z}_2 \tag{42}$$

Now (41) also implies that $\mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22} = \mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22}\mathsf{a}_2$, therefore (42) can be rewritten

$$\mathsf{s}_1\mathsf{a}_1\mathsf{q}_{211}(\overline{\mathsf{a}}_2\mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\mathsf{a}_2\mathsf{q}_{211} + \overline{\mathsf{a}}_2\mathsf{r}_{12}\mathsf{a}_1\mathsf{p}_{22}\overline{\mathsf{a}}_2\mathsf{p}_{22}\mathsf{q}_{211})^*\mathsf{a}_2\mathsf{z}_2 \tag{43}$$

which by (8) is equivalent to

$$
\begin{aligned}
& \mathsf{s_1 a_1 q_{211}}(\overline{\mathsf{a}}_2 \mathsf{r_{12} a_1 p_{22} a_2 q_{211}} + \overline{\mathsf{a}}_2 \mathsf{r_{12} a_1 p_{22}} \overline{\mathsf{a}}_2 \mathsf{q_{211}})^* \mathsf{a_2 z_2} \\
= \; & \mathsf{s_1 a_1 q_{211}}(\overline{\mathsf{a}}_2 \mathsf{r_{12} a_1 p_{22}}(\mathsf{a_2} + \overline{\mathsf{a}}_2)\mathsf{q_{211}})^* \mathsf{a_2 z_2} \\
= \; & \mathsf{s_1 a_1 q_{211}}(\overline{\mathsf{a}}_2 \mathsf{r_{12} a_1 p_{22} q_{211}})^* \mathsf{a_2 z_2} \\
= \; & \mathsf{s_1 a_1 q_{211}}(\overline{\mathsf{a}}_2 \mathsf{r_{12} a_1 q_{211}})^* \mathsf{a_2 z_2} \qquad (44)
\end{aligned}
$$

The final step is to get rid of the variable y_1. We have

$$
\begin{aligned}
\mathsf{s_1 a_1 q_{211}} &= \mathsf{d s_1 q_{211}} && \text{by (9)} \\
&= \mathsf{d s_1 t_2} && \text{by (7)} \\
\mathsf{r_{12} a_1 q_{211}} &= \mathsf{c_2 r_{12} q_{211}} && \text{by (9)} \\
&= \mathsf{c_2 r_{12} u_{222}} && \text{by (7)}
\end{aligned}
$$

thus (44) is equivalent to

$$
\mathsf{d s_1 t_2}(\overline{\mathsf{a}}_2 \mathsf{c_2 r_{12} u_{222}})^* \mathsf{a_2 z_2} \qquad (45)
$$

We can eliminate the assignments to y_1 by Lemma 4.5, giving

$$
\mathsf{d t_2}(\overline{\mathsf{a}}_2 \mathsf{c_2 u_{222}})^* \mathsf{a_2 z_2} \qquad (46)
$$

Finally, we have

$$
\begin{aligned}
\mathsf{d t_2} &= \mathsf{d s_2 q_{222}} && \text{by (8)} \\
&= \mathsf{s_2 a_2 q_{222}} && \text{by (9)} \\
\mathsf{c_2 u_{222}} &= \mathsf{c_2 r_{22} q_{222}} && \text{by (8)} \\
&= \mathsf{r_{22} a_2 q_{222}} && \text{by (9)}
\end{aligned}
$$

giving

$$
\mathsf{s_2 a_2 q_{222}}(\overline{\mathsf{a}}_2 \mathsf{r_{22} a_2 q_{222}})^* \mathsf{a_2 z_2} \qquad (47)
$$

After a change of variable (Lemma 4.6), we are done.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1975.

[2] Ernie Cohen. Lazy caching. Available as ftp://ftp.telcordia.com/pub/ernie/research/homepage.html, 1994.

[3] Ernie Cohen. Using Kleene algebra to reason about concurrency control. Available as ftp://ftp.telcordia.com/pub/ernie/research/homepage.html, 1994.

[4] Ernie Cohen, Dexter Kozen, and Frederick Smith. The complexity of Kleene algebra with tests. Technical Report 96-1598, Computer Science Department, Cornell University, July 1996.

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

[6] Y. I. Ianov. The logical schemes of algorithms. In *Problems of Cybernetics*, volume 1, pages 82–140. Pergamon Press, 1960. English translation.

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

[8] Stephen C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, N.J., 1956.

[9] Dexter Kozen. On induction vs. *-continuity. In Kozen, editor, *Proc. Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 167–176, New York, 1981. Springer-Verlag.

[10] Dexter Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, New York, 1991.

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

[12] Dexter Kozen. *Automata and Computability*. Springer-Verlag, New York, 1997.

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

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

[15] Dexter Kozen. Automata on guarded strings and applications. Technical Report 2001-1833, Computer Science Department, Cornell University, January 2001.

[16] Dexter Kozen and Maria-Cristina Patron. Certification of compiler optimizations using Kleene algebra with tests. In John Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luis Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proc. 1st Int. Conf. Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 568–582, London, July 2000. Springer-Verlag.

[17] Dexter Kozen and Frederick Smith. Kleene algebra with tests: Completeness and decidability. In D. van Dalen and M. Bezem, editors, *Proc. 10th Int. Workshop Computer Science Logic (CSL'96)*, volume 1258 of *Lecture Notes in Computer Science*, pages 244–259, Utrecht, The Netherlands, September 1996. Springer-Verlag.

[18] Dexter Kozen and Jerzy Tiuryn. On the completeness of propositional Hoare logic. In J. Desharnais, editor, *Proc. 5th Int. Seminar Relational Methods in Computer Science (RelMiCS 2000)*, pages 195–202, January 2000.

[19] Dexter Kozen and Jerzy Tiuryn. Intuitionistic linear logic and partial correctness. In *Proc. 16th Symp. Logic in Comput. Sci. (LICS'01)*, pages 259–268. IEEE, June 2001.

[20] Werner Kuich. The Kleene and Parikh theorem in complete semirings. In T. Ottmann, editor, *Proc. 14th Colloq. Automata, Languages, and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 212–225, New York, 1987. EATCS, Springer-Verlag.

[21] Werner Kuich and Arto Salomaa. *Semirings, Automata, and Languages*. Springer-Verlag, Berlin, 1986.

[22] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, 1974.

[23] Vaughan Pratt. Dynamic algebras as a well-behaved fragment of relation algebras. In D. Pigozzi, editor, *Proc. Conf. on Algebra and Computer Science*, volume 425 of *Lecture Notes in Computer Science*, pages 77–110, Ames, Iowa, June 1988. Springer-Verlag.