Guarded Kleene Algebra with Tests: Coequations, Coinduction, and Completeness

Todd Schmid
Department of Computer Science, University College London, UK

Tobias Kappé
Department of Computer Science, Cornell University, Ithaca, NY, USA

Dexter Kozen
Department of Computer Science, Cornell University, Ithaca, NY, USA

Alexandra Silva
Department of Computer Science, University College London, UK

Abstract

Guarded Kleene Algebra with Tests (GKAT) is an efficient fragment of KAT, as it allows for almost linear decidability of equivalence. In this paper, we study the (co)algebraic properties of GKAT. Our initial focus is on the fragment that can distinguish between unsuccessful programs performing different actions, by omitting the so-called *early termination axiom*. We develop an operational (coalgebraic) and denotational (algebraic) semantics and show that they coincide. We then characterize the behaviors of GKAT expressions in this semantics, leading to a coequation that captures the covariety of automata corresponding to these behaviors. Finally, we prove that the axioms of the reduced fragment are sound and complete w.r.t. the semantics, and then build on this result to recover a semantics that is sound and complete w.r.t. the full set of axioms.

2012 ACM Subject Classification Theory of computation → Program reasoning

Keywords and phrases Kleene algebra, program equivalence, completeness, coequations

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.142

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2102.08286

Funding *Tobias Kappé*: DARPA grant HR001120C0107 (Pronto)
Dexter Kozen: NSF grant CCF-2008083
Alexandra Silva: ERC Consolidator Grant AutoProbe (101002697) and a Royal Society Wolfson Fellowship

1 Introduction

Kleene algebra with tests (KAT) [17] was introduced in the early 90's as an extension of Kleene algebra (KA), the algebra of regular expressions. The core idea of the extension was simple: consider regular languages over a two-sorted alphabet, in which one sort represents Boolean tests and the other denotes basic program actions. This seemingly simple extension enables an important application for regular languages in reasoning about imperative programs with basic control flow structures like branches (**if-then-else**) and loops (**while**). KAT largely inherited the properties of KA: a language model [22], a Kleene theorem [19], a sound and complete axiomatization [22], and a PSPACE decision procedure for equivalence [8].

In 2014, a specialized KAT called NetKAT [4] was proposed to program software-defined networks. NetKAT was later extended with a probabilistic choice operator that enabled the modelling of randomized protocols [9]. Interestingly, there exists a decision procedure for NetKAT program equivalence that enables practical verification of reachability in networks

with thousands of nodes and links, which seems to scale almost linearly despite the PSPACE-completeness of this problem [10, 35]. This raised the question: do practical NetKAT programs belong to a fragment of KAT that has more favorable properties than the full language?

Recently, this question was answered positively [34], in the form of *Guarded Kleene Algebra with Tests* (GKAT), a fragment of KAT obtained by adding a Boolean guard to the non-deterministic choice and iteration operators so that they correspond exactly to the standard **if-then-else** and **while** constructs. GKAT is expressive enough to capture all programs used in network verification while allowing for almost linear time[1] decidability of equivalence, thereby explaining the experimental results observed in NetKAT.

The use of GKAT as a framework for program analysis also raises further questions about recovering the properties of KAT on the level of GKAT. Is there a class of automata that provides a Kleene theorem? Is there a sound and complete axiomatization of GKAT equivalence? The original paper [34] gave incomplete answers to these questions. First, it proposed a class of *well-nested* automata that can be used to describe the semantics of all GKAT programs, but left open whether this class covered all automata that accept the behaviors of GKAT programs. Second, GKAT was axiomatized under the assumption of *early termination*: intuitively, referring to a semantics of imperative programs where programs that fail immediately are equated to programs that fail eventually. This semantics, though useful, is too coarse in contexts where program behavior prior to failure matters.

In this paper, we take a new perspective on the semantics of GKAT programs and their corresponding automata, using coequations. Coequations provide the right tool to characterize fragments of languages as they enable a precise way to remove unwanted traces. We are then able to give a precise characterization of the behaviors of GKAT programs and prove a completeness theorem for each of the fragments of interest.

Our contributions. In a nutshell, the contributions of this paper are the following:

1. We give a denotational model for GKAT without early termination by representing the behavior as a certain kind of tree. This allows us to design two coequations: one characterizing the behaviors denoted by GKAT expressions, and another capturing only the behaviors of GKAT expressions that terminate early.
2. We obtain two completeness results for GKAT: one for the model of the previous item and the axiomatization of [34] without the early termination axiom; and building on this, another for the full axiomatization. The former is new; the latter provides an alternative proof to the completeness theorem presented in [34].
3. A concrete example of a well-nested GKAT automaton with a non-well-nested quotient. This settles an open question of [34] and closes the door on an alternative proof of completeness based on well-nested automata.

2 Guarded Kleene Algebra with Tests

At its heart, *Guarded Kleene Algebra with Tests* (GKAT) is an algebraic theory of imperative programs. Expressions in GKAT are concise formulas for WHILE programs [23], which are built inductively from actions and tests with sequential composition and the classic programming constructs of branches and loops: **if** b **then** e **else** f and **while** b **do** e.

[1] $O(n\alpha(n))$, where $\alpha(n)$ is the inverse of Ackermann's function

Union Axioms

U1. $e +_b e \equiv e$

U2. $e +_b f \equiv f +_{\bar{b}} e$

U3. $(e +_b f) +_c g \equiv e +_{b \wedge c} (f +_c g)$

U4. $e +_b f \equiv b \cdot e +_b f$

U5. $e \cdot g +_b f \cdot g \equiv (e +_b f) \cdot g$

Sequence Axioms

S1. $(e \cdot f) \cdot g \equiv e \cdot (f \cdot g)$

S2. $0 \cdot e \equiv 0$

S3. $e \cdot 0 \equiv 0$

S4. $1 \cdot e \equiv e$; S5. $e \equiv e \cdot 1$

S6. $b \cdot c \equiv b \wedge c$

Loop Axioms

W1. $e^{(b)} \equiv e \cdot e^{(b)} +_b 1$

W2. $(ce)^{(b)} \equiv (e +_c 1)^{(b)}$

W3. $\dfrac{E(e) \equiv 0 \quad g \equiv eg +_b f}{g \equiv e^{(b)} \cdot f}$

Figure 1 Axioms for GKAT-expressions. Here, $e, f, g \in \mathrm{Exp}$ and $b, c \in \mathrm{BExp}$.

Formally, these expressions are drawn from a two-sorted language of *tests* and *programs*. The tests are built from a finite set of ***primitive tests*** T, as follows:

$$\mathrm{BExp} \ni b, c ::= 0 \mid 1 \mid t \in T \mid \bar{b} \mid b \wedge c \mid b \vee c.$$

Here, 0 and 1 are understood as the constant tests **false** and **true** respectively, $\bar{b}$ denotes the negation of b, and $\wedge$ and $\vee$ are conjunction and disjunction, respectively. We will use A to denote the set of ***atomic tests*** (or just ***atoms***), Boolean expressions of the form $d_1 \wedge \cdots \wedge d_l$, where $d_i \in \{t_i, \bar{t}_i\}$ for each $i \leqslant l$ and $\{t_i \mid i \leqslant l\}$ is a fixed enumeration of T. It is well known that any $b \in \mathrm{BExp}$ can be written equivalently as the disjunction of the atoms $a \in A$ that imply b under the laws of Boolean algebra. We will often identify each Boolean expression $b \in \mathrm{BExp}$ with this set of atoms and write $b \subseteq A$ or $a \in b$.

Programs are built from tests and a finite set of ***primitive programs*** or ***actions*** Σ, disjoint from T. Formally, programs are generated by the grammar

$$\mathrm{Exp} \ni e, f ::= b \in \mathrm{BExp} \mid p \in \Sigma \mid e \cdot f \mid e +_b f \mid e^{(b)}$$

Here, a test b abbreviates the statement **assert** $\boldsymbol{b}$, the operator $\cdot$ is sequential composition, $e +_b f$ is shorthand for **if** $\boldsymbol{b}$ **then** $\boldsymbol{e}$ **else** $\boldsymbol{f}$ and $e^{(b)}$ is shorthand for **while** $\boldsymbol{b}$ **do** $\boldsymbol{e}$.

GKAT programs satisfy standard properties of imperative programs. For instance, swapping the branches of an **if-then-else** construct should not make a difference, provided that we also negate the condition; that is, the semantics of $e +_b f$ should coincide with that of $f +_{\bar{b}} e$. The rules in Figure 1 axiomatize equivalences between programs. Together with the axioms of Boolean algebra, these generate a congruence $\equiv$ on Exp.

Some remarks are in order for axiom W3. The right-hand premise states that an expression g has some self-similarity in the sense that it is equivalent to checking whether b holds, in which case it runs e followed by recursing at g, and otherwise running f. Intuitively, this says that g is loop-like, matching the conclusion that g is equivalent to $e^{(b)} \cdot f$. However, this conclusion may not make sense when based on just the second premise. Specifically, if we choose e, f, g and b to be 1, we can show that the premise holds and derive $1 \equiv 1^{(1)} \cdot 1$, which is to say that **assert true** is equivalent to **(while true do assert true); assert true**. Intuitively, this should be false: the first program terminates successfully and immediately, but the second program does not. The problem is that the loop body does not perform any actions that affect the state and make progress towards the end of the loop.

This is remedied by the left-hand premise, which distinguishes loop bodies that can accept immediately from those that cannot. It plays the same role as the *empty word property* in Salomaa's axiomatization of the algebra of regular events [31]. Formally, given $e \in \mathrm{Exp}$, the Boolean expression $E(e)$ is defined inductively by setting $E(p) = 0$, $E(b) = b$, and

$$E(e \cdot f) = E(e) \wedge E(f) \qquad E(e +_b f) = (b \wedge E(e)) \vee (\bar{b} \wedge E(f)) \qquad E(e^{(b)}) = \bar{b}$$

We call e ***productive*** if $E(e) \equiv 0$. Axioms W2 and W3 are analogues of Salomaa's axioms A_{11} and R2 [31]. Specifically, W2 says that non-productive loop iterations do not contribute to the semantics. This allows the use of W3 to reason about loops in general, for instance to prove $e^{(b)} \equiv e^{(b)} \cdot \overline{b}$, which says that the loop condition is false when a loop ends [34].

Axiom S3 identifies a program that fails eventually with the program that fails immediately. As a consequence, $\equiv$ cannot distinguish between processes that loop forever, like $p^{(1)}$ and $q^{(1)}$, even though they perform different actions [34]. Consequently, GKAT can be seen as a theory of *computation* schemata, i.e., programs that need to halt successfully to be meaningful.

In contrast, it is also useful to be able to reason about *process* schemata, i.e., programs that perform meaningful tasks, even when they do not terminate successfully. To this end, we define the ***reduced congruence*** $\equiv_0$ generated by the axioms of Figure 1 except S3.

Let $[\![-]\!] : \mathrm{Exp} \to S$ be a semantics of GKAT. We say that $[\![-]\!]$ is ***sound w.r.t.*** $\equiv$ if for all $e, f \in \mathrm{Exp}$ with $e \equiv f$, it holds that $[\![e]\!] = [\![f]\!]$. Similarly, $[\![-]\!]$ is ***sound w.r.t.*** $\equiv_0$ if $e \equiv_0 f$ implies that $[\![e]\!] = [\![f]\!]$.

Since $\equiv$ encodes common program laws, one might wonder whether there is a single interpretation in which programs are related by $\equiv$ if and only if they have the same image. Such an interpretation is called ***free w.r.t.*** $\equiv$. This question is not just of theoretical interest: a free interpretation can help decide whether programs are provably equivalent, and hence the same under any sound interpretation, by checking whether their free semantics coincide. Naturally, the same question can be asked for $\equiv_0$: is there a semantics that is ***free w.r.t.*** $\equiv_0$, i.e., where $e \equiv_0 f$ if and only if e and f have the same interpretation?

The remainder of this paper is organized as follows. In Section 3, we describe the operational structure for GKAT expressions in terms of GKAT-automata, as in [34]. In Section 4, we provide an explicit construction of a GKAT-automaton in which all other automata can be uniquely interpreted. We then build a semantics that is sound w.r.t. $\equiv_0$ in Section 5. In Section 6 we relate our coequational description of GKAT expressions to the *well-nested* GKAT-*automata* of [34]. In Section 7, we prove that this semantics is in fact complete w.r.t. $\equiv_0$ and, building on this, obtain a semantics that is complete w.r.t. $\equiv$. Omitted proofs are included in the extended version [32].

3 An operational model: GKAT-automata

In this section we discuss the small-step operational model for GKAT programs from [34]. The operational perspective provides us with the tools to describe a semantics that is complete w.r.t. $\equiv_0$ and paves the way to a decision procedure.

We can think of a GKAT-program as a machine that evolves as it reads a string of atomic tests. Depending on the most recently observed atomic test, the program either accepts, rejects, or emits an action label and changes to a new state. For example, feeding **if** b **do** p **else** q an atomic test $a \in b$ causes it to perform the action p and then terminate successfully.

▶ **Definition 3.1.** *A* GKAT**-automaton** *[34, 23] is a pair $\mathcal{X} = (X, \delta)$, where X is a set of* **states** *and $\delta : X \times A \to 2 + \Sigma \times X$ is a* **transition function**. *We use $x \xrightarrow{a|p}_{\mathcal{X}} x'$ as a notation for $\delta(x, a) = (p, x')$. Similarly, $x \Rightarrow_{\mathcal{X}} a$ denotes that $\delta(x, a) = 1$, and $x \downarrow_{\mathcal{X}} a$ denotes that $\delta(x, a) = 0$. We drop the subscript $\mathcal{X}$ when the automaton is clear from context.*

Intuitively, X represents the states of an abstract machine running a GKAT program, with dynamics encoded in δ. When the machine is in state $x \in X$ and observes $a \in A$, there are three possibilities: if $x \downarrow a$, the machine rejects; if $x \Rightarrow a$, it accepts; and if $x \xrightarrow{a|p} x'$, it performs the action p followed by a transition to the state x'.

$$\frac{a \in b}{b \Rightarrow a} \qquad \frac{}{p \xrightarrow{a|p} 1} \qquad \frac{a \in b \qquad e \Rightarrow a}{e +_b f \Rightarrow a} \qquad \frac{a \in \bar{b} \qquad f \Rightarrow a}{e +_b f \Rightarrow a} \qquad \frac{a \in b \qquad e \xrightarrow{a|p} e'}{e +_b f \xrightarrow{a|p} e'} \qquad \frac{a \in \bar{b} \qquad f \xrightarrow{a|p} f'}{e +_b f \xrightarrow{a|p} f'}$$

$$\frac{e \Rightarrow a \qquad f \Rightarrow a}{e \cdot f \Rightarrow a} \qquad \frac{e \Rightarrow a \qquad f \xrightarrow{a|p} f'}{e \cdot f \xrightarrow{a|p} f'} \qquad \frac{e \xrightarrow{a|p} e'}{e \cdot f \xrightarrow{a|p} e' \cdot f} \qquad \frac{a \in b \qquad e \xrightarrow{a|p} e'}{e^{(b)} \xrightarrow{a|p} e' \cdot e^{(b)}} \qquad \frac{a \in \bar{b}}{e^{(b)} \Rightarrow a}$$

Figure 2 The transition structure of $\mathcal{E}$. Here, $e, e', f, f' \in \mathrm{Exp}$, $b \subseteq A$, $a \in A$, and $p \in \Sigma$. Transitions that are not explicitly defined above are assumed to be failed termination.

▶ Remark 3.2. The reader familiar with coalgebra will recognize that GKAT-automata are precisely coalgebras for the functor $G = (2 + \Sigma \times \mathsf{Id})^A$ [34]. Indeed, the notions relating to GKAT-automata, such as homomorphism, bisimulation, and semantics to follow are precisely those that arise from G as prescribed by universal coalgebra [27].

We can impose an automaton structure on Exp yielding the ***syntactic* GKAT*-automaton*** $\mathcal{E} = (\mathrm{Exp}, D)$, where D is the transition map given by Brzozowski derivatives [34] as specified in Figure 2. For instance, the operational behavior of $p^{(b)}$ as a state of $\mathcal{E}$ could be drawn as follows, where $x \xrightarrow{b|p} y$ denotes that $x \xrightarrow{a|p} y$ for every $a \in b$ and rejecting transitions $x \downarrow a$ are left implicit:

$$\bar{b} \Longleftarrow p^{(b)} \xrightarrow{b|p} 1 \cdot p^{(b)} \Longrightarrow \bar{b} \qquad (1)$$

The operational structure of $\mathcal{E}$ is connected to $\equiv_0$ as follows.

▶ **Theorem 3.3** (Fundamental theorem of GKAT). *For any $e \in \mathrm{Exp}$, $e \equiv_0 1 +_{E(e)} D(e)$ where*

$$D(e) = \bigplus_{e \xrightarrow{a|p_a} e_a} p_a \cdot e_a \qquad \text{and} \qquad \bigplus_{a \in b} e_a = \begin{cases} 0 & \text{if } b = 0, \\ e_a +_a \left(\bigplus_{a' \in b \setminus a} e_{a'} \right) & \text{some } a \in b, \text{ otherwise.} \end{cases}$$

The generalized guarded union above is well defined, in that the order of atoms does not matter up to $\equiv_0$. See [34] for more details about the generalised guarded union.

States of GKAT-automata have the same behavior if reading the same sequence of atoms leads to the same sequence of actions, acceptance, or rejection. This happens when one state mimics the moves of the other, performing the same actions in response to the same stimuli. For instance, consider the GKAT-automaton in (1): the behavior of $p^{(b)}$ can be replicated by the behavior of $1 \cdot p^{(b)}$, in that both either consume an $a \in \bar{b}$ and terminate or consume $a \in b$ and emit p before transitioning to $1 \cdot p^{(b)}$. This can be made precise.

▶ **Definition 3.4.** *Let $R \subseteq X \times Y$ be a relation between the state spaces of* GKAT*-automata $\mathcal{X}$ and $\mathcal{Y}$. Then R is a* **bisimulation** *if for any $(x, y) \in R$ and $a \in A$,*
(1) *$x \downarrow_{\mathcal{X}} a$ if and only if $y \downarrow_{\mathcal{Y}} a$; and (2) $x \Rightarrow_{\mathcal{X}} a$ if and only if $y \Rightarrow_{\mathcal{Y}} a$; and*
(3) *if $x \xrightarrow{a|p}_{\mathcal{X}} x'$ and $y \xrightarrow{a|q}_{\mathcal{Y}} y'$ for some x' and y', then $p = q$ and $(x', y') \in R$.*

If a pair of states $(x, y) \in X \times Y$ is contained in a bisimulation, we say that x and y are **bisimilar**. *If a bisimulation R is the graph of a function $\varphi : X \to Y$, we write $\varphi : \mathcal{X} \to \mathcal{Y}$ and call φ a* GKAT***-automaton homomorphism*** *[27].*

Indeed, bisimulations are designed to formally witness behavioral equivalence. We use the term ***behavior*** as a synonym for the phrase *bisimilarity (equivalence) class.*

4 The final GKAT-automaton

One way of assigning semantics to GKAT expressions is to find a sufficiently large GKAT-automaton $\mathcal{Z}$ that contains the behavior of every other GKAT-automaton. In this section, we provide a concrete explicit description of such a "semantic" GKAT-automaton – this is a crucial step towards being able to devise a completeness proof.

Concretely, $\mathcal{Z}$ represents the behavior of a state as a tree that holds information about acceptance, rejection, and transitions to other states (which are subtrees). Essentially, this tree is an unfolding of the transition graph from that state.

We describe these trees using partial functions. Let us write A^+ for the set of all non-empty words consisting of atoms. The state space Z of $\mathcal{Z}$ is the set of all partial functions $t : A^+ \rightharpoonup 2 + \Sigma$ with $A \subseteq \mathrm{dom}(t)$, such that the following hold for all $a \in A$ and $x \in A^+$.

$$\frac{w \in \mathrm{dom}(t) \quad t(w) \in \Sigma}{wa \in \mathrm{dom}(t)} \qquad \frac{w \in \mathrm{dom}(t) \quad t(w) \in 2}{wx \notin \mathrm{dom}(t)}$$

The transition structure of $\mathcal{Z}$ is defined by the inferences

$$\frac{t(a) = 0}{t \downarrow a} \qquad \frac{t(a) = 1}{t \Rightarrow a} \qquad \frac{t(a) = p \in \Sigma}{t \xrightarrow{a|p} \lambda w.t(aw)}$$

When $t(w) \in \Sigma$, we will write $\partial_w t$ for $\lambda u.t(wu)$. We can think of $t \in Z$ as a tree where the root has leaves for atoms $a \in A$ with $t(a) = 1$, and a subtree for every $a \in A$ with $t(a) \in \Sigma$.

▶ Remark 4.1. Trees correspond to *deterministic* (possibly *infinite*) *guarded languages* [34, 23]. More precisely, every tree can be identified with a language $L \subseteq (A \cdot \Sigma)^* \cdot A \cup (A \cdot \Sigma)^\omega$ satisfying (i) if $wap\sigma, waq\sigma' \in L$, then $p = q$; and (ii) if $wa \in L$, then $wap\sigma \notin L$ for any $p\sigma$. We forgo a description in terms of guarded languages in favor of trees because these trees have the constraint about determinism built in.

A ***node*** of t is a word $w \in A^*$ such that either $w = \epsilon$ (the empty word), or $w \in \mathrm{dom}(t)$ and $t(w) \in \Sigma$. We write $\mathrm{Node}(t)$ for the set of nodes of t. A ***subtree*** of t is a tree t' such that $t' = \partial_w t$ for some $w \in \mathrm{Node}(t)$. A ***leaf*** of t is a word $w \in \mathrm{dom}(t)$ such that $t(w) \in 2$.

Next, we specialize Definition 3.4 to $\mathcal{Z}$ (c.f. [28, Theorem 3.1]).

▶ **Lemma 4.2.** *$R \subseteq Z \times Z$ is a bisimulation on $\mathcal{Z}$ iff for any $(t, s) \in R$ and $a \in A$, (1) $t(a) = s(a)$; and (2) if either $\partial_a t$ or $\partial_a s$ is defined, then both are defined and $(\partial_a t, \partial_a s) \in R$.*

We can now prove that bisimilar trees in Z coincide.

▶ **Lemma 4.3** (Coinduction)**.** *If $s, t \in Z$ are bisimilar, then $s = t$.*

Thus, to show that two trees are equal, it suffices to demonstrate a bisimulation that relates them. This proof method is called ***coinduction***. We can also use Lemma 4.2 to define algebraic operations on Z, and such definitions are said to be ***coinductive***. Many of the results in the sequel are argued using coinduction, and many of the constructions are coinductive. With this in mind, we are now ready to prove that $\mathcal{Z}$ contains every behavior that can be represented by a GKAT-automaton, as follows.

▶ **Theorem 4.4.** *$\mathcal{Z}$ is the final GKAT-automaton. In other words, for every GKAT-automaton $\mathcal{X}$, there exists a unique GKAT-automaton homomorphism $!_{\mathcal{X}}$ from $\mathcal{X}$ to $\mathcal{Z}$.*

Given a GKAT-automaton $\mathcal{X}$, the unique map $!_{\mathcal{X}}$ assigns a tree from Z to each of its states. In particular, recalling that the syntactic GKAT-automaton $\mathcal{E}$ has Exp as its set of states, $!_{\mathcal{E}}$ is a semantics of GKAT programs in terms of trees. The following lemma states that bisimulation is sound and complete with respect to this semantics.

▶ **Lemma 4.5.** *States x and x' of a GKAT-automaton $\mathcal{X}$ are bisimilar iff $!_{\mathcal{X}}(x) = !_{\mathcal{X}}(x')$.*

5 Trees form an algebra

So far, we have seen that the behavior of a GKAT-program is naturally interpreted as a certain kind of tree, and that each such tree is the state of the final GKAT-automaton $\mathcal{Z}$. In this section, we show that the trees in Z can themselves be manipulated and combined using the programming constructs of GKAT. These operations satisfy all of the axioms that build $\equiv_0$, but fail the *early-termination axiom* S3. This gives rise to an inductive semantics of GKAT-programs $[\![-]\!] : \mathrm{Exp} \to Z$ that is sound w.r.t. $\equiv_0$. As a matter of fact, we will see that $[\![-]\!]$ coincides with the unique GKAT-automaton homomorphism $!_{\mathcal{E}} : \mathrm{Exp} \to Z$.

We begin by interpreting the tests. Given $b \subseteq A$, we define $[\![b]\!]$ as the characteristic function of b as a subset of A^+, i.e., $[\![b]\!](a) = 1$ if $a \in b$, and $[\![b]\!](a) = 0$ otherwise.

On the other hand, primitive action symbols denote programs that perform an action in one step and then terminate successfully in the next. For $p \in \Sigma$, this behavior is described by the unique tree $[\![p]\!]$ such that $[\![p]\!](a) = p$ and $\partial_a [\![p]\!] = [\![1]\!]$ for any $a \in A$. When context can disambiguate, we write b in place of $[\![b]\!]$ and p in place of $[\![p]\!]$.

Each operation is defined using a ***behavioral differential equation (BDE)*** consisting of a set of ***initial conditions*** $t(a) = \xi_a \in 2 + \Sigma$ indexed by $a \in A$ and a set of ***step equations*** $\partial_a t = s_a$ indexed by the $a \in A$ with $t(a) \in \Sigma$. This is possible because every BDE describes a unique automaton, which (by Theorem 4.4) has a unique interpretation in Z [28]. Each BDE below can be read more or less directly from Figure 2.

The first operation that we interpret in Z is sequential composition. For any $s, t \in Z$, the tree $s \cdot t$ models sequential composition of programs by replacing each non-zero leaf of s by the nodal subtree of t given by the corresponding atomic test. This can formally be defined as the unique operation satisfying the following behavioral differential equation.

$$(s \cdot t)(a) = \begin{cases} t(a) & \text{if } s(a) = 1, \\ s(a) & \text{otherwise} \end{cases} \qquad \partial_a(s \cdot t) = \begin{cases} \partial_a t & \text{if } s(a) = 1, \\ \partial_a s \cdot t & \text{otherwise.} \end{cases}$$

Here, $\partial_a s \cdot t = (\partial_a s) \cdot t$. Using this operation, we define $[\![e \cdot f]\!] = [\![e]\!] \cdot [\![f]\!]$.

To interpret the guarded union operation, define $+_b$ to be the unique operation such that

$$(s +_b t)(a) = \begin{cases} s(a) & \text{if } a \in b, \\ t(a) & \text{otherwise} \end{cases} \qquad \partial_a(s +_b t) = \begin{cases} \partial_a s & \text{if } a \in b, \\ \partial_a t & \text{otherwise.} \end{cases}$$

As before, we define $[\![e +_b f]\!] = [\![e]\!] +_b [\![f]\!]$.

Finally, we interpret the guarded exponential operation. Following Figure 2, $t^{(b)}$ can be defined as the unique tree satisfying

$$t^{(b)}(a) = \begin{cases} 1 & \text{if } a \notin b, \\ t(a) & \text{if } a \in b \text{ and } t(a) \in \Sigma, \\ 0 & \text{otherwise.} \end{cases} \qquad \partial_a(t^{(b)}) = \partial_a t \cdot t^{(b)}$$

Similar to the other operators, we set $[\![e^{(b)}]\!] = [\![e]\!]^{(b)}$. This completes our definition of the algebraic homomorphism $[\![-]\!] : \mathrm{Exp} \to Z$.

As it happens, $[\![-]\!]$ is also a GKAT automaton homomorphism from $\mathcal{E}$ to $\mathcal{Z}$. By uniqueness of such homomorphisms (Theorem 4.4), we can conclude that $[\![-]\!]$ and $!_{\mathcal{E}}$ are the same.

▶ **Proposition 5.1.** *For any $e \in \mathrm{Exp}$, $[\![e]\!] = !_{\mathcal{E}}(e)$.*

This allows us to treat the algebraic and coalgebraic semantics as synonymous. Using Lemma 4.5, we can then show soundness w.r.t. $\equiv_0$ by arguing that $\equiv_0$ is a bisimulation on $\mathcal{E}$.

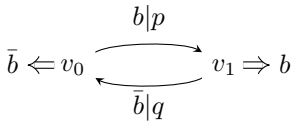

Figure 3 A GKAT-automaton without GKAT behaviors.

▶ **Theorem 5.2.** *The semantics* $[\![-]\!]$ *is sound w.r.t.* $\equiv_0$.

On the other hand, Z does not satisfy S3. For instance, $[\![p \cdot 0]\!] \neq [\![0]\!]$ for any $p \in \Sigma$. We will adapt the model to overcome this in Section 7.3.

6 Well-nested automata and nested behavior

Not all behaviors expressible in terms of finite GKAT-automata occur in $\mathcal{E}$. For example, the two-state automaton in Figure 3 fails to exhibit any behavior of the form $[\![e]\!]$, with $e \in \mathrm{Exp}$, when $b, \bar{b} \neq 0$. This is proven in the extended version [32] where we show that no branch of a GKAT behavior can accept both b and $\bar{b}$ infinitely often. For another example, see [23], where a particular three-state automaton is shown to exhibit no GKAT behavior.

Intuitively, both of the examples above fail to exhibit the behaviors of GKAT programs because GKAT lacks a **goto**-statement that allows control to transfer to an arbitrary position in the program; instead, GKAT automata corresponding to GKAT expressions are structured by branches and loops. The question then arises: can we characterize the "shapes" of automata whose behavior is **goto**-free, i.e., described by a GKAT expression?

In [34], the authors proposed the class of *well-nested* GKAT automata, consisting of automata built inductively by applying a series of operations designed to mimic the structural effects of loops. It was shown that the behavior of every GKAT expression can be described by some well-nested automaton. Moreover, they proved that the class of well-nested automata constitutes a sufficient condition: the behavior of a well-nested GKAT automaton is described by a GKAT expression. Whether this condition is also *necessary*, i.e., whether every automaton with behavior corresponding to a GKAT expression is well-nested, was left open.

Thus, a positive answer to the latter question amounts to showing that every GKAT automaton whose behavior is the same as a well-nested GKAT automaton is itself well-nested. Such a class of automata closed under behavioral equivalence is known as a ***covariety***. Covarieties have desirable structural properties. In particular, they are closed under homomorphic images [27, 12, 3]. Unfortunately, well-nested automata do not satisfy this property: we have found a well-nested automaton whose homomorphic image is not well-nested, depicted in Figure 4. In other words, there exists a non-well-nested automaton whose behavior is still described by a GKAT expression. This also closes the door on a simpler approach to completeness described in [34].

Thus, well-nested automata do not constitute a characterization of the GKAT automata that correspond to GKAT expressions. To obtain such a characterization, we take a slightly different approach: rather than describing shapes of these automata, we describe the shapes of the trees that they denote. We refer to a set of trees $\mathsf{U} \subseteq Z$ as a ***coequation***, and treat it as a predicate: a GKAT-automaton $\mathcal{X}$ ***satisfies*** U, written $\mathcal{X} \models \mathsf{U}$, if every behavior present in $\mathcal{X}$ appears in U – in other words, if $!_{\mathcal{X}}$ factors through U. We write $\mathrm{Cov}(\mathsf{U})$ to denote the class of all GKAT-automata that satisfy U. It is easily shown that $\mathrm{Cov}(\mathsf{U})$ is a covariety.

The coequation that we give to describe the covariety of automata whose behavior corresponds to a GKAT expression is driven by the intuition behind well-nested automata: the trees in this coequation are built using compositions that enforce **while**-like behavior,

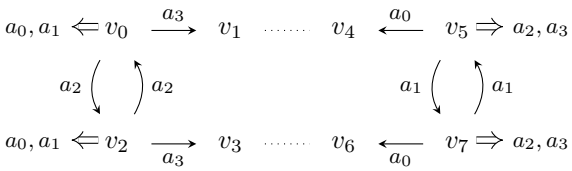

Figure 4 As depicted, this automaton is well-nested. However, identifying v_1 with v_4, and v_3 with v_6, we obtain an automaton that is not well-nested.

and do not permit the construction of **goto**-like behavior. To this end, we need to define a new *continuation* operation, as follows. Given $s, t \in Z$, the ***continuation*** $s \rhd t$ of s along t is the unique tree satisfying the behavioral differential equation

$$(s \rhd t)(a) = \begin{cases} t(a) & \text{if } s(a) = 1, \\ s(a) & \text{otherwise} \end{cases} \qquad \partial_a(s \rhd t) = \begin{cases} \partial_a t \rhd t & \text{if } s(a) = 1, \\ \partial_a s \rhd t & \text{otherwise.} \end{cases}$$

Intuitively, $s \rhd t$ is the tree that attaches infinitely many copies of t to s. This operation can be thought of as the dual to Kleene's original $*$-operation [16], which loops on its first argument some number of times before continuing in the second.

▶ **Definition 6.1.** *The* **nesting** *coequation* W *is the smallest subset of* Z *containing the* **discrete** *coequation* $\mathsf{D} := \{[\![b]\!] \mid b \subseteq A\}$ *and closed under the* **nesting** *rules below:*

$$\frac{t, s \in \mathsf{W}}{t \cdot s \in \mathsf{W}} \qquad \frac{(\forall a \in A)\ t(a) \in \Sigma \implies \partial_a t \in \mathsf{W}}{t \in \mathsf{W}} \qquad \frac{t, s \in \mathsf{W}}{t \rhd s \in \mathsf{W}}$$

The first and third nesting rules say that W is closed under composition and continuation; the second rule says that integrals over nested trees are nested.

It is not too hard to see that W is a subautomaton of $\mathcal{Z}$. In other words, if $t \in \mathsf{W}$, then the derivatives of t are in W as well. In fact, W is a subalgebra of Z in that it is closed under the operations of GKAT. This can be seen from the following observations: first, $\partial_a p = 1$ for all $a \in A$, so $p \in \mathsf{W}$ for any $p \in \Sigma$ by the second nesting rule. Second, W is closed under sequential composition by definition. Third, if $s, t \in \mathsf{W}$ and $b \subseteq A$, then every derivative of $s +_b t$ is either a derivative of s or a derivative of t. Lastly, closure under the guarded exponential is a consequence of the identity

$$t^{(b)} = 1 \rhd (\tilde{t} +_b 1), \qquad \text{where} \qquad \tilde{t} := +_{t \xrightarrow{a|p_a} t_a} p_a \cdot t_a.$$

This identity can be shown to hold for all $t \in Z$ and $b \subseteq A$ using a coinductive argument. It follows that the nesting coequation contains the image of $[\![-]\!]$. A similar argument can be used to establish the reverse containment as well, which leads to the following.

▶ **Proposition 6.2.** W *is the set of* GKAT *program behaviors, i.e,* $\mathsf{W} = \{[\![e]\!] \mid e \in \mathrm{Exp}\}$.

Proposition 6.2 characterizes W as the the set of behavioral patterns exhibited by GKAT expressions: the states of a GKAT-automaton $\mathcal{X}$ behave like GKAT programs if and only if $\mathcal{X}$ satisfies W, or, in other words, if $\mathcal{X}$ can be found in the covariety $\mathrm{Cov}(\mathsf{W})$. Since every well-nested automaton has the behavior of some GKAT expression [34], it must satisfy W.

▶ **Proposition 6.3.** *Well-nested* GKAT*-automata satisfy the nesting coequation.*

7 Completeness

This section contains two completeness theorems for GKAT . As in [34], we need to assume that W3 is generalized to arbitrary (linear) systems of equations. This *uniqueness axiom*, discussed in Section 7.1, will allow us to prove that the semantics $[\![-]\!]$ from Section 5 is free with respect to $\equiv_0$ – that is, $[\![e]\!] = [\![f]\!]$ implies $e \equiv_0 f$ – in Section 7.2. This will then provide an alternative route to completeness for GKAT in Section 7.3.

7.1 Uniqueness of solutions for Salomaa systems

In part, W3 from Figure 1 ensures that the equation $g \equiv e \cdot g +_b f$ with indeterminate g has at most one solution in $\mathrm{Exp}/\equiv_0$ for any $e, f \in \mathrm{Exp}$ under the condition that e denotes a productive program. In fact, we could have stated the axiom this way from the beginning, as W1 provides the existence of a solution to this equation (even without the restriction on productivity). As we will see, the uniqueness axiom makes a more general statement than W3 about *systems* of equations with an arbitrary number of indeterminates.

▶ **Definition 7.1.** *A* **system of (n left-affine) equations** *is a sequence of n equations of the form $x_i = e_{i1} \cdot x_1 +_{b_{i1}} \cdots +_{b_{i(n-1)}} e_{in} \cdot x_n +_{b_{in}} c_i$, indexed by $i \leqslant n$, such that (1) x_i is an indeterminate variable; (2) $(b_{ij})_{j \leqslant n}$ is a sequence of* **disjoint** *Boolean expressions, i.e. $b_{ij} \wedge b_{ik} \equiv 0$ for any $j \neq k$; (3) c_i is a Boolean expression disjoint from b_{ij} for all $j \leqslant n$; and (4) e_{ij} is a* GKAT *expression for any $j \leqslant n$.*

Given any congruence $\doteqdot$ satisfying the axioms of $\equiv_0$, a **solution in $\mathrm{Exp}/\doteqdot$** *to such a system is an n-tuple of* GKAT *expressions $(g_i)_{i \leqslant n}$ such that the equivalence $g_i \doteqdot e_{i1} \cdot g_1 +_{b_{i1}} \cdots +_{b_{i(n-1)}} e_{in} \cdot g_n +_{b_{in}} c_i$ holds for all $i \leqslant n$.*

For example, the equation in the premise of W3 is a system of one left-affine equation, and the conclusion prescribes a unique solution (in $\mathrm{Exp}/\equiv_0$) to the premise. Every finite GKAT-automaton $\mathcal{X}$ gives rise to a system of equations with variables indexed by $X = \{x_i \mid i \leqslant n\}$ and coefficients indexed by the transition map, as follows:

$$e_{ij} = \bigplus_{x_i \xrightarrow{a|p_a} x_j} p_a \qquad c_i = \{a \in A \mid x_i \Rightarrow a\} \qquad b_{ij} = \{a \in A \mid x_i \xrightarrow{a|p} x_j\}.$$

Solving this system of equations uncovers the GKAT-constructs the automaton implements.

The uniqueness axiom states that certain systems of equations, like the one in the premise of W3, admit at most one solution. Choosing which systems the axiom should apply to must be done carefully for the same reason that necessitates the side-condition on W3. Crucially, we require that the system have *productive coefficients*, i.e. $E(e_{ij}) \equiv 0$ for all $i, j \leqslant n$, to admit a unique solution. As this condition is analogous to Salomaa's *empty word property* [31], a system of equations with productive coefficients is called ***Salomaa*** [34]. The ***uniqueness axiom (for $\doteqdot$)*** states that every Salomaa system of equations has at most one solution in $\mathrm{Exp}/\doteqdot$. It is sound with respect to the semantics $[\![-]\!]$ from Section 5.

▶ **Theorem 7.2.** *For any $i, j \leqslant n$, let $s_{ij} \in Z$ satisfy $s_{ij}(a) \neq 1$ for any $a \in A$, $(b_{ij})_{j \neq n}$ be a sequence of disjoint Boolean expressions for any $i \leqslant n$, and $c_i \subseteq A$ be disjoint from b_{ij} for each $i \leqslant n$. The system of equations $x_i = s_{i1} \cdot t_1 +_{b_{i1}} \cdots +_{b_{i(n-1)}} s_{in} \cdot t_n +_{b_{in}} c_i$, indexed by $i \leqslant n$ has a unique solution in Z^n.*

7.2 Completeness with respect to $\equiv_0$

Next, we present a completeness theorem w.r.t. $\equiv_0$. We have already seen that the behavior of a program takes the form of a tree, and that the programming constructs of GKAT apply to trees in such a way that equivalence up to the axioms of $\equiv_0$ is preserved (Theorem 5.2). The completeness theorem in this section shows that up to $\equiv_0$-equivalence, GKAT programs can be identified with the trees they denote.

▶ **Theorem 7.3** (Completeness for $\equiv_0$). *Assume the uniqueness axiom for $\equiv_0$ and let $e, f \in \mathrm{Exp}$. If $[\![e]\!] = [\![f]\!]$, then $e \equiv_0 f$.*

Proof sketch. Since $[\![e]\!] = [\![f]\!]$, e and f are bisimilar as expressions. This bisimulation gives rise to a Salomaa system of equations, which can be shown to admit both the derivatives of e and f as solutions. By the unique solutions axiom, it then follows that $e \equiv_0 f$. ◀

7.3 Completeness with respect to $\equiv$

Having found a semantics that is sound and complete w.r.t. $\equiv_0$, we proceed to extend this result to find a semantics that is sound and complete w.r.t. $\equiv$. Recall that the only difference between these equivalences was S3, which equates programs that fail eventually with programs that fail immediately. To coarsen our semantics, we need an operation on labelled trees that forces early termination in case an accepting state cannot be reached.

▶ **Definition 7.4.** *We say $t \in Z$ is* **dead** *when for all $w \in \mathrm{dom}(t)$ it holds that $t(w) \neq 1$. The* **normalization** *operator is defined coinductively, as follows:*

$$t^{\wedge}(a) = \begin{cases} 0 & t(a) \in \Sigma \wedge \partial_a t \text{ is dead,} \\ t(a) & \text{otherwise} \end{cases} \qquad \partial_a(t^{\wedge}) = (\partial_a t)^{\wedge}.$$

▶ **Example 7.5.** Normalizing the tree $[\![p +_b p \cdot 0]\!]$ prunes the branch corresponding to $\bar{b}$, since it has no accepting leaves. This yields the tree $[\![b \cdot p]\!]$.

We can compose the normalization operator with the semantics $[\![-]\!]$ to obtain a new semantics $[\![-]\!]^{\wedge}$, which replaces dead subtrees with early termination. Composing normalization with the earlier semantics of GKAT, we obtain the ***normalized semantics*** $[\![-]\!]^{\wedge}$. This semantics is sound w.r.t. $\equiv$.

▶ **Proposition 7.6.** *If $e \equiv f$, then $[\![e]\!]^{\wedge} = [\![f]\!]^{\wedge}$.*

For the corresponding completeness property, we need a way of "normalizing" a given expression in Exp. The following observation gives us a way to do this.

▶ **Lemma 7.7.** W *is closed under normalization.*

When $e \in \mathrm{Exp}$, we have that $[\![e]\!] \in \mathsf{W}$. Moreover, by the above, $[\![e]\!]^{\wedge} \in \mathsf{W}$, which means that there is an $e' \in \mathrm{Exp}$ such that $[\![e']\!] = [\![e]\!]^{\wedge}$. We write $e^{\wedge}$ for this ***normalized expression***. As it turns out, we can derive the equivalence $e^{\wedge} \equiv e$ from the uniqueness axiom for $\equiv$. This gives an alternative proof of the completeness result of [34] that highlights the role of coequational methods in reasoning about failure modes.

▶ **Corollary 7.8** ([34]). *Assume the uniqueness axiom for $\equiv$ and $\equiv_0$. If $[\![e]\!]^{\wedge} = [\![f]\!]^{\wedge}$, then $e \equiv f$.*

Proof sketch. If $[\![e]\!]^{\wedge} = [\![f]\!]^{\wedge}$, then $[\![e^{\wedge}]\!] = [\![f^{\wedge}]\!]$. By completeness of $\equiv_0$ w.r.t. $[\![-]\!]$, we can then derive that $e \equiv e^{\wedge} \equiv_0 f^{\wedge} \equiv f$, and since $\equiv_0$ is contained in $\equiv$, also $e \equiv f$. ◀

By normalizing the trees in W, we obtain the coequation $\mathsf{W}^\wedge = \{t^\wedge \mid t \in \mathsf{W}\}$. This coequation precisely characterizes GKAT programs with forced early termination. In particular, since $\mathsf{W}^\wedge \subseteq \mathsf{W}$, neither state in Figure 3 has a semantics described by $[\![e]\!]^\wedge$ for some $e \in \mathrm{Exp}$.

8 Related work

This paper builds on [34], where GKAT was proposed together with a language semantics based on guarded strings [15] and an axiomatization closely related to Salomaa's axiomatization of regular expressions based on unique fixpoints [31]. Note that the language of *propositional while programs* from [23, 20] is closely related to GKAT in terms of semantics, although the compact syntax and axiomatization were only introduced in [34].

Some GKAT-automata have behavior that does not correspond to any GKAT expression, such as the example in [23]. The upshot is that the Böhm-Jacopini theorem [6, 13], which states that every deterministic flowchart corresponds to a WHILE program, does not hold propositionally, i.e., when we abstract from the meaning of individual actions and tests [23].

In contrast with [34, 23], our work provides a precise characterization of the behaviors denoted by GKAT programs using trees. In other words, we characterize the image of the semantic map inside the space of all behaviors. This explicit characterization was essential for proving completeness of the full theory of GKAT, including the early termination axiom. KAT equivalence without early termination has been investigated by Mamouras [24].

Brzozowski derivatives [7] appear in the completeness proof of KA [18, 21, 14]. We were more directly inspired by Silva's coalgebraic analogues of Brzozowski derivatives used in the context of completeness [33]. Rutten [28] and Pavlovic and Escardo [26] document the connection between the differential calculus of analysis and coalgebraic derivatives.

Coequations have appeared in the coalgebra literature in a variety of contexts, e.g. [3, 1, 5, 29, 30], and notably in the proof of generalized Eilenberg theorems [36, 2]. The use of coequations in completeness proofs is, as far as we are aware, new.

9 Discussion

GKAT was introduced in [23] under the name *propositional while programs* and extensively studied in [34] as an algebraic framework to reason about simple imperative programs. We presented a new perspective on the theory of GKAT, which allowed us to isolate a fragment of the original axiomatization that captures the purely behavioral properties of GKAT programs. We solved an open problem from [34], providing a proof that well-nested automata are not closed under homomorphisms, thereby making it unlikely that these automata can be used in a completeness proof that does not rely on uniqueness axioms. Finally, we proved completeness for the full theory, respecting the early-termination property, in which programs that fail immediately are equated with programs that fail eventually.

There are several directions for future work that are worth investigating. First, it was conjectured in [34] that the uniqueness axiom follows from the other axioms of GKAT. This remains open, but at the time of writing we think this conjecture might be false. Secondly, the technique we use, based on coequations, can serve as basis for a general approach to completeness proofs. We plan to investigate other difficult problems where our technique might apply. Of particular interest is an open problem posed by Milner in [25], which consists of showing that a certain set of axioms are complete w.r.t. bisimulation equivalence for regular expressions. Recently, Grabmeyer and Fokkink [11] provided a partial solution. We believe our technique can simplify their proofs and shed further light on Milner's problem.

We have chosen to adopt the axiomatization from [34], which can be described as a Salomaa-style axiomatization – the loop is a unique fixpoint satisfying a side condition on termination. We would like to generalize the results of the present paper to an axiomatization in which the loop is a least fixpoint w.r.t. an order. The challenge is that there is no natural order in the language because the $+$ of Kleene Algebra has been replaced by $+_b$. However, we hope to devise an order $\leqslant$ directly on expressions and extend the characterizations that we have to the new setting. This new axiomatization would have the advantage of being algebraic (that is, sound under arbitrary substitution), which makes it more suitable for verification purposes as the number of models of the language would increase.

References

1 Jirí Adámek. A logic of coequations. In *CSL*, pages 70–86, 2005. doi:10.1007/11538363_7.

2 Jirí Adámek, Stefan Milius, Robert S. R. Myers, and Henning Urbat. Generalized Eilenberg theorem: Varieties of languages in a category. *ACM Trans. Comput. Log.*, 20(1):3:1–3:47, 2019. doi:10.1145/3276771.

3 Jirí Adámek and Hans-E. Porst. On varieties and covarieties in a category. *Math. Struct. Comput. Sci.*, 13(2):201–232, 2003. doi:10.1017/S0960129502003882.

4 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: semantic foundations for networks. In *POPL*, pages 113–126, 2014. doi:10.1145/2535838.2535862.

5 Adolfo Ballester-Bolinches, Enric Cosme-Llópez, and Jan J. M. M. Rutten. The dual equivalence of equations and coequations for automata. *Inf. Comput.*, 244:49–75, 2015. doi:10.1016/j.ic.2015.08.001.

6 Corrado Böhm and Giuseppe Jacopini. Flow diagrams, Turing machines and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, 1966. doi:10.1145/355592.365646.

7 Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964. doi:10.1145/321239.321249.

8 Ernie Cohen, Dexter Kozen, and Frederick Smith. The complexity of Kleene algebra with tests. Technical Report TR96-1598, Cornell University, July 1996. URL: https://hdl.handle.net/1813/7253.

9 Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic NetKAT. In *ESOP*, pages 282–309, 2016. doi:10.1007/978-3-662-49498-1_12.

10 Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In *POPL*, pages 343–355, 2015. doi:10.1145/2676726.2677011.

11 Clemens Grabmayer and Wan J. Fokkink. A complete proof system for 1-free regular expressions modulo bisimilarity. In *LICS*, pages 465–478, 2020. doi:10.1145/3373718.3394744.

12 H. Gumm. Elements of the general theory of coalgebras, 2000.

13 David Harel. On folk theorems. *Commun. ACM*, 23(7):379–389, 1980. doi:10.1145/358886.358892.

14 Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, pages 375–404, 2006. doi:10.1007/11780274_20.

15 Donald M. Kaplan. Regular expressions and the equivalence of programs. *J. Comput. Syst. Sci.*, 3(4):361–386, 1969. doi:10.1016/S0022-0000(69)80027-9.

16 Stephen C. Kleene. Representation of events in nerve nets and finite automata. In Claude E. Shannon and John McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, 1956.

17 Dexter Kozen. Kleene algebra with tests and commutativity conditions. In *TACAS*, pages 14–33, 1996. doi:10.1007/3-540-61042-1_35.

18 Dexter Kozen. Myhill-Nerode relations on automatic systems and the completeness of Kleene algebra. In *STACS*, pages 27–38, 2001. doi:10.1007/3-540-44693-1_3.

19 Dexter Kozen. Automata on guarded strings and applications. *Matematica Contemporanea*, 24:117–139, 2003.

20 Dexter Kozen. Nonlocal flow of control and Kleene algebra with tests. In *LICS*, pages 105–117, 2008. doi:10.1109/LICS.2008.32.

21 Dexter Kozen. On the coalgebraic theory of Kleene algebra with tests. In Can Başkent, Lawrence S. Moss, and Ramaswamy Ramanujam, editors, *Rohit Parikh on Logic, Language and Society*, volume 11 of *Outstanding Contributions to Logic*, pages 279–298. Springer, 2017. doi:10.1007/978-3-319-47843-2_15.

22 Dexter Kozen and Frederick Smith. Kleene algebra with tests: Completeness and decidability. In *CSL*, pages 244–259, 1996. doi:10.1007/3-540-63172-0_43.

23 Dexter Kozen and Wei-Lung Dustin Tseng. The Böhm-Jacopini theorem is false, propositionally. In *MPC*, pages 177–192, 2008. doi:10.1007/978-3-540-70594-9_11.

24 Konstantinos Mamouras. Equational theories of abnormal termination based on Kleene algebra. In *FOSSACS*, volume 10203 of *Lecture Notes in Computer Science*, pages 88–105, 2017. doi:10.1007/978-3-662-54458-7_6.

25 Robin Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984. doi:10.1016/0022-0000(84)90023-0.

26 Dusko Pavlovic and Martín Hötzel Escardó. Calculus in coinductive form. In *LICS*, pages 408–417, 1998. doi:10.1109/LICS.1998.705675.

27 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

28 Jan J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.*, 308(1-3):1–53, 2003. doi:10.1016/S0304-3975(02)00895-2.

29 Julian Salamanca, Adolfo Ballester-Bolinches, Marcello M. Bonsangue, Enric Cosme-Llópez, and Jan J. M. M. Rutten. Regular varieties of automata and coequations. In *MPC*, pages 224–237, 2015. doi:10.1007/978-3-319-19797-5_11.

30 Julian Salamanca, Marcello M. Bonsangue, and Jurriaan Rot. Duality of equations and coequations via contravariant adjunctions. In Ichiro Hasuo, editor, *CMCS*, pages 73–93, 2016. doi:10.1007/978-3-319-40370-0_6.

31 Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966. doi:10.1145/321312.321326.

32 Todd Schmid, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: Coequations, coinduction, and completeness, 2021. arXiv:2102.08286.

33 Alexandra Silva. *Kleene coalgebra*. PhD thesis, Radboud University, Nijmegen, 2010. URL: https://hdl.handle.net/2066/83205.

34 Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: Verification of uninterpreted programs in nearly linear time. In *POPL*, 2020. doi:10.1145/3371129.

35 Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. Scalable verification of probabilistic networks. In *PLDI*, pages 190–203, 2019. doi:10.1145/3314221.3314639.

36 Henning Urbat, Jirí Adámek, Liang-Ting Chen, and Stefan Milius. Eilenberg theorems for free. In *MFCS*, 2017. doi:10.4230/LIPIcs.MFCS.2017.43.