A cyclic proof system for Guarded Kleene Algebra with Tests

Jan Rooduijn[1], Alexandra Silva[2], and Dexter Kozen[2]

[1] Institute of Logic, Language and Computation, University of Amsterdam, The Netherlands j.m.w.rooduijn@uva.nl
[2] Cornell University, Ithaca, USA

Abstract. Guarded Kleene Algebra with Tests (GKAT for short) is an efficient fragment of Kleene Algebra with Tests, suitable for reasoning about simple imperative while-programs. Following earlier work by Das and Pous on Kleene Algebra, we study GKAT from a proof-theoretical perspective. The deterministic nature of GKAT makes ordinary sequents sufficient for achieving regular completeness, unlike the situation with Kleene Algebra, where hypersequents are required. Moreover, the proof search induces a coNP decision procedure, rather than one in PSPACE.

Keywords: Kleene Algebra · Guarded Kleene Algebra with Tests · Cyclic proofs

1 Introduction

In this paper we introduce a cyclic proof system for Guarded Kleene Algebra with Tests, or GKAT for short. This formal system is used for reasoning about simple imperative while-programs. It draws from a long tradition, which we sketch below. For more on the history of GKAT we refer the reader to [13, 17, 16].

The first origin for GKAT is *Kleene Algebra*, an algebraic theory thoroughly studied by Conway [5], and dating back to Kleene [9]. Recall that, given a finite alphabet Σ, a *regular expression over* Σ is a string generated by the grammar:

$$e, f ::= 0 \mid 1 \mid a \in \Sigma \mid e + f \mid e \cdot f \mid e^*$$

Under the standard interpretation, regular expressions denote *languages over* Σ, *i.e.* sets of words over Σ. The constant 0 is interpreted as the empty language $\emptyset$, and the constant 1 as the language $\{\epsilon\}$ containing just the empty string. The interpretation of a is the language $\{a\}$ containing only a. The operators $+, \cdot, ^*$ are interpreted as union, pairwise concatenation, and *Kleene closure* respectively. The latter, applied to a language L, gives the smallest language extensions (or superset) L^* of L that is closed under repeated concatenation. A language denoted by some regular expression is said to be a *regular language*.

The research of Jan Rooduijn has been made possible by a grant from the Dutch Research Council NWO, project number 617.001.857.

Although Kleene Algebra appears under different interpretations in various areas of logic and computer science, it is most commonly understood as a generalisation of the algebra of regular languages under the operations $(0, 1, +, \cdot, ^*)$. There has been much interest in finding a nice axiomatisation of the equational validities of this algebra. The fragment without Kleene closure is finitely axiomatised by the axioms of an algebraic structure called an *idempotent semiring*. Extending this axiomatisation to the full language has posed a formidable challenge. Numerous proposed axiomatisations exist, with one of the most prominent coming from Salomaa [15]. To capture the behaviour of the Kleene star *, this system features the following rule:

$$\frac{eg + f \equiv g \qquad e \text{ does not have the empty word property}}{e^* f \equiv g}$$

where e is said to have the *empty word property* if the language denoted by e contains the empty word ϵ. Although Salomaa's system is sound and complete with respect to the algebra of regular languages, it is not entirely satisfactory, because it is not *algebraic*. More precisely, the empty word property is not closed under substitution and therefore itself not equationally axiomatisable. As a consequence, this system does not give rise a notion of "a Kleene algebra", in which equations may be true or false.

A solution to this problem was provided by Kozen in [10]. Using $e \leq f$ as a shorthand for $e + f \equiv f$, Kozen axiomatised $e^* f$ as a least fixed point by adding the following axiom and rule:[3]

$$1 + ee^* \leq e^* \qquad\qquad \frac{eg + f \leq g}{e^* f \leq g}$$

Kozen showed that this system is complete with respect to the algebra of regular languages. With this, a *Kleene algebra* is then defined to be any algebra $(K, 0, 1, +, \cdot, ^*)$ such that $(K, 0, 1, +, \cdot)$ is an idempotent semiring, and the axiom and rule above are satisfied. It turns out that the algebra of regular languages over Σ is then the free Kleene algebra generated by Σ.

Another way to interpret Kleene Algebra is as a semantics of programs. Under this interpretation, characters in Σ are seen as primitive programs, 0 is a program without any valid behaviour, and 1 is *skip*, which simply does nothing. The concatenation $e \cdot f$ is thought of as first running program e, and then running program f. The union $e + f$ non-deterministically runs e or f. Finally, e^* repeats the program e a finite number of times, possibly zero. A shortcoming of using Kleene Algebra for this purpose is that is unable to express common programming constructs such as if-then-else statements and while loops, because of the lack of tests for the control structures. This inspired the development of an extension of Kleene Algebra, called Kleene Algebra with Tests, or `KAT` for

[3] Kozen's original axiomatisation also has a dual axiom and rule characterising fe^* as a least fixed point, but it turns out the other rules suffice to derive all valid equations. The system we describe, without the dual axiom and rule, is often called *left-handed Kleene Algebra*.

short [11]. KAT is a finite quasi-equational theory with two sorts, namely *programs* and a subset thereof consisting of *tests*, such that the programs form a Kleene algebra under the operations $(+, \cdot, *, 0, 1)$ and the tests form a Boolean algebra under the operations $(+, \cdot, ^-, 0, 1)$. The tests allows one to express if-then-else statements and while loops by $b \cdot e + \overline{b} \cdot f$ and $(b \cdot e)^{(b)} \cdot \overline{b}$, respectively. Despite the gain in expressive power, the complexity of deciding KAT-equations remains the same as for Kleene Algebra, namely PSPACE-complete.

Finally, the system GKAT, initially presented in [13, 12], and subsequently coined and further explored in [17], is a fragment of KAT, obtained by replacing the operations $+$ and $*$ by their *guarded* counterparts $+_b$ and $-^{(b)}$. Roughly, this restricts the language to *only* if-then-else statements, rather than general non-deterministic choice, and to *only* while loops, rather than the general (non-deterministic) Kleene star. The main advantage of GKAT is the efficiency of deciding equivalence of its programs, which can be done in *nearly linear* time, *i.e.* in $\mathcal{O}(n \cdot \alpha(n))$, where α is the very slow-growing inverse Ackermann function.

Cyclic proof systems are proof systems in which a leaf is not necessarily required to be an axiom, but may alternatively be justified by one of its ancestors further down in the proof tree. As such, a cyclic proof is a graph (more specifically, a finite tree back edges), rather than a well-founded tree. Usually this leaf must be labelled by exactly the same sequent as its justifying ancestor, and some condition is required to hold between the leaf and this ancestor. Together, these requirements ensure that infinite paths through this cyclic tree with back edges satisfy a certain progress condition, facilitating a soundness proof by infinite descent. Cyclic proof theory is an active field of research for logics with recursive operators, see *e.g.* [3, 14, 2, 1, 8] for a variety of approaches.

In [7], Das & Pous propose a cyclic proof system HKA for Kleene Algebra. Building on their work, we introduce a cyclic proof system SGKAT for GKAT. Below is a summary of the contributions of this paper:

- We propose a new type of sequent tailored to GKAT (Definition 3).
- We propose a proof system SGKAT for GKAT (Figure 3).
- We show that (possibly infinitary) proofs in SGKAT are sound (Theorem 1) and complete (Theorem 2) with respect to the language semantics of GKAT.
- We show that proofs are *finite-state*: each proof contains only finitely many distinct sequents. More precisely, by a more fine-grained analysis than in [7], we give a polynomial bound on the number of distinct sequents occurring in a proof, in terms of the size of its endsequent. It follows that the subset of *regular*[4] proofs is complete. Since regular trees can be represented as finite trees with back edges, SGKAT is indeed a proper *cyclic* proof system.
- We show that the proof search procedure used to establish completeness induces a coNP decision procedure for deciding the language inclusion problem for GKAT-expressions. This gives a (most probably non-optimal) upper bound on the complexity of this decision procedure.

Due to space limitations all proofs are either omitted or sketched. Full proofs can be found in the appendix of the extended version of this paper.

[4] Recall that a labelled tree is *regular* if it has finitely many non-isomorphic subtrees.

2 Preliminaries

2.1 Syntax

The language of GKAT has two sorts, namely *programs* and a subset thereof consisting of *tests*. It is built from a finite and non-empty set T of *primitive tests* and a non-empty set Σ of *primitive programs*, where T and Σ are disjoint. For the rest of this paper we fix such sets T and Σ. We reserve the letters t and p to refer, respectively, to arbitrary primitive tests and primitive programs. The first of the following grammars defines the *tests*, and the second the *expressions*.

$$b, c ::= 0 \mid 1 \mid t \mid \overline{b} \mid b \vee c \mid b \cdot c \qquad e, f ::= b \mid p \mid e \cdot f \mid e +_b f \mid e^{(b)},$$

where $t \in T$ and $p \in \Sigma$. Intuitively, the operator $+_b$ stands for the `if-then-else` construct, and the operator $(-)^{(b)}$ stands for the `while` loop. Note that the tests are simply propositional formulas. It is convention to use $\cdot$ instead of $\wedge$ for conjunction. As usual, we often omit $\cdot$ for syntactical convenience, *e.g.* by writing pq instead of $p \cdot q$.

Example 1. The idea of GKAT is to model imperative programs. For instance, the expression $(p +_b q)^{(a)}$ represents the following imperative program:

```
while a do (if b then p else q)
```

Remark 1. As mentioned in the introduction, GKAT is a fragment of Kleene Algebra with Tests, or KAT [11]. The syntax of KAT is the same as that of GKAT, but with unrestricted union $+$ instead of guarded union $+_b$, and unrestricted iteration $*$ instead of the while loop operator (b). The embedding φ of GKAT into KAT acts on guarded union and guarded iteration as follows, and commutes with all other operators: $\varphi(e +_b f) = b \cdot \varphi(e) + \overline{b} \cdot \varphi(f)$, and $\varphi(e^{(b)}) = (b \cdot \varphi(e))^* \cdot \overline{b}$.

2.2 Semantics

There are several kinds of semantics for GKAT. In [17], a *language* semantics, a *relational* semantics, and a *probabilistic* semantics are given. In this paper we are only concerned with the language semantics, which we shall now describe.

We denote by At the set of *atoms* of the free Boolean algebra generated by $T = \{t_1, \ldots t_n\}$. That is, At consists of all tests of the form $c_1 \cdots c_n$, where $c_i \in \{t_i, \overline{t_i}\}$ for each $1 \leq i \leq n$. Lowercase Greek letters $(\alpha, \beta, \gamma, \ldots)$ will be used to denote elements of At. A *guarded string* is an element of the regular set $\mathsf{At} \cdot (\Sigma \cdot \mathsf{At})^*$, that is, a string of the form $\alpha_1 p_1 \alpha_2 p_2 \cdots \alpha_n p_n \alpha_{n+1}$. We will interpret expressions as languages (formally just sets) of guarded strings. The sequential composition operator $\cdot$ is interpreted by the *fusion product* $\diamond$, given by $L \diamond K := \{x\alpha y \mid x\alpha \in L \text{ and } \alpha y \in K\}$. For the interpretation of $+_b$, we define for every set of atoms $B \subseteq \mathsf{At}$ the following operation of *guarded union* on languages: $L +_B K := (B \diamond L) \cup (\overline{B} \diamond K)$, where $\overline{B}$ is $\mathsf{At} \setminus B$. For the interpretation of (b), we stipulate:

$$L^0 := \mathsf{At} \qquad L^{n+1} := L^n \diamond L \qquad L^B := \bigcup_{n \geq 0} (B \diamond L)^n \diamond \overline{B}$$

Finally, the semantics of GKAT is inductively defined as follows:

$$[\![b]\!] := \{\alpha \in \mathsf{At} : \alpha \leq b\} \qquad [\![p]\!] := \{\alpha p \beta : \alpha, \beta \in \mathsf{At}\} \qquad [\![e \cdot f]\!] := [\![e]\!] \diamond [\![f]\!]$$
$$[\![e +_b f]\!] := [\![e]\!] +_{[\![b]\!]} [\![f]\!] \qquad [\![e^{(b)}]\!] := [\![e]\!]^{[\![b]\!]}$$

Note that the interpretation of $\cdot$ between tests is the same whether they are regarded as tests or as programs, *i.e.* $[\![b]\!] \cap [\![c]\!] = [\![b]\!] \diamond [\![c]\!]$.

Remark 2. While the semantics of expressions is explicitly defined, the semantics of tests is derived implicitly through the free Boolean algebra generated by T. It is standard in the GKAT literature to address the Boolean content in this manner.

Example 2. In a guarded string, atoms can be thought of as states of a machine, and programs as executions. For instance, in case of the guarded string $\alpha p \beta$, the machine starts in state α, then executes program p, and ends in state β. Let us briefly check which guarded strings of, say, the form $\alpha p \beta q \gamma$ belong to the interpretation $[\![(p +_b q)^{(a)}]\!]$ of the program of Example 1. First, we must have $\alpha \leq a$, for otherwise we would not enter the loop. Moreover, we have $\alpha \leq b$, for otherwise q rather than p would be executed. Similarly, we find that $\beta \leq a, \overline{b}$. Since the loop is exited after two iterations, we must have $\gamma \leq \overline{a}$. Hence, we find

$$\alpha p \beta q \gamma \in [\![(p +_b q)^{(a)}]\!] \Leftrightarrow \alpha \leq a, b \text{ and } \beta \leq a, \overline{b} \text{ and } \gamma \leq \overline{a}.$$

The following lemmas will be useful later on. The first lemma follows from the fact that $\diamond$ is associative and At is its identity element.

Lemma 1. *$L^{n+1} = L \diamond L^n$ for every language L of guarded strings.*

Lemma 2. *Let p be a primitive program and let L and K be languages of guarded strings. Then $[\![p]\!] \diamond L = [\![p]\!] \diamond K$ implies $L = K$.*

Remark 3. The fact that GKAT models deterministic programs is reflected in the fact that interpretations of GKAT-expressions satisfy a certain *determinacy property*. Namely, for every $x\alpha y$ and $x\alpha z$ in L, either y and z are both empty, or both begin with the same primitive program. We refer to [17] for more details.

Remark 4. The language semantics of GKAT is the same as that of KAT (see [11]), in the sense that $[\![e]\!] = [\![\varphi(e)]\!]$, where φ is the embedding from Remark 1, the semantic brackets on the right-hand side denote the standard interpretation in KAT, and e is any GKAT-expression.

2.3 Foundational results

In this subsection we briefly summarise some of the foundational results presented in [17]. Reading this subsection is not strictly necessary for understanding the rest of this paper, but it may provide some helpful context and intuition.

Automaton model Automata for GKAT are given as coalgebras for the functor $G : X \mapsto (2 + \Sigma \times X)^{\mathsf{At}}$. That is, a state $s \in X$ of a G-coalgebra, when given an atom $\alpha \in \mathsf{At}$, does one of three things: halt and accept, halt and reject, or execute a program $p \in \Sigma$ and move to a new state in X. A *G-automaton* is simply a G-coalgebra with a designated initial state. In [17], it is shown that the languages of guarded strings accepted by some G-automaton, possibly with infinitely many states, are precisely the languages which satisfy the determinacy property of Remark 3. However, there are G-automata, even finite ones, whose language is not denoted by any GKAT-program. To remedy this situation, the authors of [17] introduce the notion of *well-nestedness* of G-automata, the definition of which falls outside the scope of this paper. They show for any GKAT-expression e how to construct a (finite) well-nested G-automaton $\mathbb{A}_e$ such that the language accepted by $\mathbb{A}_e$ is precisely $[\![e]\!]$. Conversely, they describe for any given well-nested G-automaton $\mathbb{A}$ a GKAT-expression $e_{\mathbb{A}}$ such that the language of $\mathbb{A}$ is precisely $[\![e_{\mathbb{A}}]\!]$.

Decision procedure One of the main advantages of GKAT over KAT lies in the efficiency of deciding program equivalence, *i.e.* whether $[\![e]\!] = [\![f]\!]$ holds for two given expressions e and f. Roughly, the decision procedure for GKAT-expressions presented in [17] works by first converting e and f into G-automata $\mathbb{A}_e$ and $\mathbb{A}_f$. By construction the number of states of $\mathbb{A}_e$ and $\mathbb{A}_f$ will be linear in, respectively, the sizes of the expressions e and f. After applying a certain normalisation procedure on $\mathbb{A}_e$ and $\mathbb{A}_f$, a general algorithm for checking bisimilarity of coalgebras can be used to check whether their initial states are bisimilar. Since G is a so-called *polynomial functor*, and the set of deterministic languages carries a G-coalgebra structure under which it is the final coalgebra for normal coalgebras, general coalgebraic theory entails that bisimilarity and language equivalence coincide on normal coalgebras. By virtue of the relatively small size of automata for GKAT-expressions, the decision procedure runs in time $\mathcal{O}(n \cdot \alpha(n))$, when $|\mathsf{At}|$ is constant, n is the sum of the sizes of the expressions e and f, and α is the inverse of the Ackermann function. Recall that this is a computable function which grows so fast that it is not definable by primitive recursion. Hence, its inverse is an extremely slow-growing function. One therefore says that this procedure runs in *nearly linear* time. This is much more efficient than deciding KAT-equivalence, which is PSPACE-complete, even when the number of atoms is constant.

Axiomatisation In [17] an axiomatisation for GKAT-equivalence was put forward. While it is there presented from a more algebraic perspective, we will present it explicitly as a proof system. For this will use the following definition.

Definition 1. *A substitution is a function $\sigma : \Sigma \to$ GKAT, assigning a* GKAT*-expression to each primitive program.*

Given a substitution σ, we let $\widehat{\sigma}$: GKAT $\to$ GKAT be the unique map which extends σ such that $\widehat{\sigma}$ commutes with the guarded union, concatenation and while-loop operators, and such that $\sigma(b) = b$ for every test b. The system is based on *equational logic*, of which the axioms and rules are given in Figure 1.

For background we refer the reader to [4]. The system moreover contains axioms from Figure 2 (cf. [17, Figure 1]).

$$\frac{}{e \equiv e} \qquad \frac{e \equiv f}{f \equiv e} \qquad \frac{e \equiv f \qquad f \equiv g}{e \equiv g} \qquad \frac{e \equiv f}{\widehat{\sigma}(e) \equiv \widehat{\sigma}(g)}$$

$$\frac{e_1 \equiv f_1 \qquad e_2 \equiv f_2}{e_1 +_b e_2 \equiv f_1 +_b f_2} \qquad \frac{e_1 \equiv f_1 \qquad e_2 \equiv f_2}{e_1 \cdot e_2 \equiv f_1 \cdot f_2} \qquad \frac{e \equiv f}{e^{(b)} \equiv f^{(b)}}$$

Fig. 1: The axioms and rules of equational logic in the signature of GKAT.

U1.	$e +_b e \equiv e$	S1.	$(e \cdot f) \cdot g \equiv e \cdot (f \cdot g)$
U2.	$e +_b f \equiv f +_{\overline{b}} e$	S2.	$0 \cdot e \equiv 0$
U3.	$(e +_b f) +_c g \equiv e +_{bc} (f +_c g)$	S3.	$e \cdot 0 \equiv 0$
U4.	$e +_b f \equiv be +_b f$	S4.	$1 \cdot e \equiv e$
U5.	$eg +_b fg \equiv (e +_b f) \cdot g$	S5.	$e \cdot 1 \equiv e$
W1.	$e^{(b)} \equiv ee^{(b)} +_b 1$	W2.	$(e +_c 1)^{(b)} \equiv (ce)^{(b)}$

Fig. 2: The GKAT axioms from [17, Fig. 1].

Definition 2. *The system* EGKAT *consists of the following axioms and rules.*

1. *All axioms and rules of equational logic, as given in Figure 1.*
2. *For all tests b, c, every $b \equiv c$ derivable from the axioms of Boolean algebra.*
3. *All axioms from [17, Fig. 1],* i.e. *all axioms in Figure 2 above.*
4. *A fixed point rule with a side condition (†):*

$$\frac{g \equiv eg +_b f}{g \equiv e^{(b)} f} \; (\dagger)$$

We will not go into the technical details of the side condition (†) in the above definition. In short, it is a syntactic restriction on the loop body e, called *strict productivity*, which is analogous to Salomaa's empty word property. The soundness of the above axiomatisation, *i.e.* that $\mathsf{EGKAT} \vdash e \equiv f$ implies $[\![e]\!] = [\![f]\!]$ is not hard to show by induction on the length of derivations. The completeness is an open question, although completeness has been shown for an extension by a stronger fixed point axiom (see [17, Section 6]). Note, however, that even if the above system were complete, it would still suffer from the same drawback as Salomaa's system: it is not algebraic, because the strict productivity condition in the fixed point rule is not closed under substitution.

3 The non-well-founded proof system SGKAT$^\infty$

In this section we commence our proof-theoretical study of GKAT. We will present a cyclic sequent system for GKAT, inspired by the cyclic sequent system for Kleene Algebra presented in [7]. In passing, we will compare our system to the one in [7].

Definition 3 (Sequent). *A* sequent *is a triple* (Γ, A, Δ)*, written* $\Gamma \Rightarrow_A \Delta$*, where* $A \subseteq \mathsf{At}$ *and* Γ *and* Δ *are (possibly empty) lists of* GKAT*-expressions.*

The list on the left-hand side of a sequent is called its *antecedent*, and the list on the right-hand side its *succedent*. In general we refer to lists of expressions as *cedents*. The symbol ϵ refers to the empty cedent.

Definition 4 (Validity). *We say that a sequent* $e_1, \ldots, e_n \Rightarrow_A f_1, \ldots, f_m$ *is* valid *whenever* $A \diamond [\![e_1 \cdots e_n]\!] \subseteq [\![f_1 \cdots f_n]\!]$.

We often abuse notation writing $[\![\Gamma]\!]$ instead of $[\![e_1 \cdots e_n]\!]$, where $\Gamma = e_1, \ldots, e_n$.

Left logical rules

$$\frac{\Gamma \Rightarrow_{A\restriction b} \Delta}{b, \Gamma \Rightarrow_A \Delta}\, b\text{-}l \qquad \frac{e, \Gamma \Rightarrow_{A\restriction b} \Delta \qquad f, \Gamma \Rightarrow_{A\restriction \bar{b}} \Delta}{e +_b f, \Gamma \Rightarrow_A \Delta}\, +_b\text{-}l$$

$$\frac{e, g, \Gamma \Rightarrow_A \Delta}{e \cdot g, \Gamma \Rightarrow_A \Delta}\, \cdot\text{-}l \qquad \frac{e, e^{(b)}, \Gamma \Rightarrow_{A\restriction b} \Delta \qquad \Gamma \Rightarrow_{A\restriction \bar{b}} \Delta}{e^{(b)}, \Gamma \Rightarrow_A \Delta}\, (b)\text{-}l$$

Right logical rules

$$(\dagger)\ \frac{\Gamma \Rightarrow_A \Delta}{\Gamma \Rightarrow_A b, \Delta}\, b\text{-}r \qquad \frac{\Gamma \Rightarrow_{A\restriction b} e, \Delta \qquad \Gamma \Rightarrow_{A\restriction \bar{b}} f, \Delta}{\Gamma \Rightarrow_A e +_b f, \Delta}\, +_b\text{-}r$$

$$\frac{\Gamma \Rightarrow_A e, f, \Delta}{\Gamma \Rightarrow_A e \cdot f, \Delta}\, \cdot\text{-}r \qquad \frac{\Gamma \Rightarrow_{A\restriction b} e, e^{(b)}, \Delta \qquad \Gamma \Rightarrow_{A\restriction \bar{b}} \Delta}{\Gamma \Rightarrow_A e^{(b)}, \Delta}\, (b)\text{-}r$$

Axioms and modal rules

$$\frac{}{\epsilon \Rightarrow_A \epsilon}\, \mathsf{id} \qquad \frac{}{\Gamma \Rightarrow_\emptyset \Delta}\, \bot \qquad \frac{\Gamma \Rightarrow_{\mathsf{At}} \Delta}{p, \Gamma \Rightarrow_A p, \Delta}\, \mathsf{k} \qquad \frac{\Gamma \Rightarrow_{\mathsf{At}} 0}{p, \Gamma \Rightarrow_A \Delta}\, \mathsf{k}_0$$

Fig. 3: The rules of SGKAT. The side condition (†) requires that $A \restriction b = A$.

Example 3. An example of a valid sequent is given by $(cp)^{(b)} \Rightarrow_{\mathsf{At}} (p(cp +_b 1))^{(b)}$. The antecedent denotes guarded strings $\alpha_1 p \alpha_2 p \cdots \alpha_n p \alpha_{n+1}$ where $\alpha_i \leq b, c$ for each $1 \leq i \leq n$, and $\alpha_{n+1} \leq \bar{b}$. The succedent denotes such strings where $\alpha_i \leq c$ is only required for those $1 \leq i \leq n$ where i is even.

Remark 5. Like the sequents for Kleene Algebra in [7], our sequents express language *inclusion*, rather than language equivalence. For Kleene Algebra this difference is insignificant, as the two notions are interdefinable using unrestricted union: $[\![e]\!] \subseteq [\![f]\!] \Leftrightarrow [\![e + f]\!] = [\![f]\!]$. For GKAT, however, it is not clear how to define language inclusion in terms of language equivalence. As a result, an advantage of axiomatising language inclusion rather than language equivalence, is that the while-operator can be axiomatised as a *least* fixed point, eliminating the need for a *strict productivity* requirement as is present in the axiomatisation in [17].

Given a set of atoms A and a test b, we write $A \restriction b$ for $A \diamond [\![b]\!]$, *i.e.* the set of atoms $\{\alpha \in A : \alpha \leq b\}$. The rules of the sequent system SGKAT are given in Figure 3. Importantly, the rules are always applied to the leftmost expression in a list (whether in the antecedent or in the succedent). Also note that the system has no propositional rules for tests, since the propositional reasoning is tucked away in the set of atoms labelling a sequent. This simplifies the system, and aligns with the usual treatment of the (finitely many) tests in the GKAT literature.

Remark 6. Following [7], we call k a 'modal' rule. The reason is simply that it looks like the rule k (sometimes called K or $\Box$) in the standard sequent calculus for basic modal logic. Our system also features a second modal rule, called k_0. Like k, this rule adds a primitive program p to the antecedent of the sequent. Since the premiss of k_0 entails that $[\![\Gamma]\!] = [\![0]\!]$, the antecedent of its conclusion will denote the empty language, and is therefore included in any antecedent Δ.

Remark 7. Note that the rules of SGKAT are highly symmetric. Indeed, the only rules that behave differently on the left than on the right, are the b-rules and k_0. For the b-rules, note that b-l changes the set of atoms, while b-r uses a side condition. The asymmetry of k_0 is clear: the succedent of the premiss has a 0, whereas the antecedent does not. A third and final asymmetry will be introduced in Definition 5, where a soundness condition is imposed on infinite branches which is sensitive to (b)-l but not to (b)-r.

Remark 8. In [16] a variant of GKAT is studied which omits the axiom (S3) in Figure 2. This axiom, also called the *early termination axiom*, equates all programs which eventually fail. A denotational model of this variant of GKAT is given in the form of certain kinds of trees. We conjecture that our proof system without the rule k_0 is sound and complete with respect to this denotational model.

An SGKAT^∞-*derivation* is a (possibly infinite) tree generated by the rules of SGKAT. Such a derivation is said to be *closed* if every leaf is an axiom.

Definition 5 (Proof). *A closed* SGKAT^∞*-derivation is said to be an* SGKAT^∞-proof *if every infinite branch is* fair *for* (b)*-*l*,* i.e. *contains infinitely many applications of the rule* (b)*-*l*.*

We write $\mathsf{SGKAT} \vdash^\infty \Gamma \Rightarrow_A \Delta$ if there is an SGKAT^∞-proof of $\Gamma \Rightarrow_A \Delta$.

Example 4. Let $\Delta_1 := (p(cp +_b 1))^{(b)}$ and $\Delta_2 := cp +_b 1, \Delta_1$. The following proof Π_1 is an example of an SGKAT^∞-proof of the sequent of Example 3. We use $(\bullet)$

to indicate that the proof repeats itself at this leaf and, for the sake of readability, omit branches that can be immediately closed by an application of $\bot$.

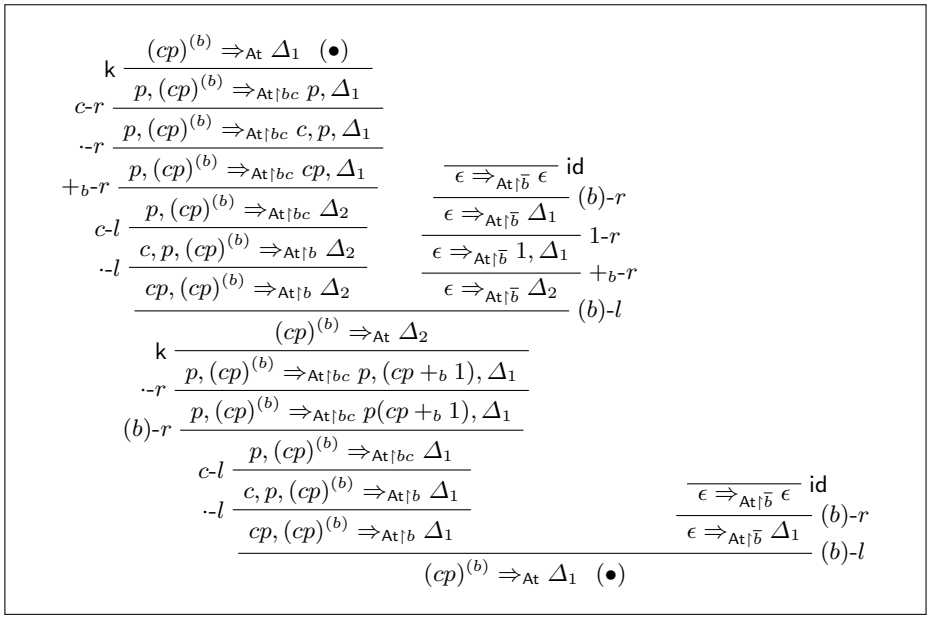


Fig. 4: The SGKAT-proof Π_1.

To illustrate the omission of branches that can be immediately closed by an application of $\bot$, let us write out the two applications of $+_b$-r in Π_1.

$$\dfrac{\epsilon \Rightarrow_{\mathsf{At}\restriction bc} cp, \Delta_1 \qquad \dfrac{}{\epsilon \Rightarrow_{\emptyset} 1, \Delta_1}\,\bot}{\epsilon \Rightarrow_{\mathsf{At}\restriction bc} \Delta_2}\,{+_b}r \qquad \dfrac{\bot\,\dfrac{}{\epsilon \Rightarrow_{\emptyset} cp, \Delta_1} \qquad \epsilon \Rightarrow_{\mathsf{At}\restriction \overline{b}} 1, \Delta_1}{\epsilon \Rightarrow_{\mathsf{At}\restriction \overline{b}} \Delta_2}\,{+_b}r$$

It can also be helpful to think of the set of atoms as *selecting* one of the premisses.

We close this section with a useful definition and a lemma.

Definition 6 (Exposure). *A list Γ of expressions is said to be* exposed *if it is either empty or begins with a primitive program.*

Recall that the sets of primitive tests and primitive programs are disjoint. Hence an exposed list Γ cannot start with a test.

Lemma 3. *Let Γ and Δ be exposed lists of expressions. Then:*

(i) $\alpha x \in [\![\Gamma]\!] \Leftrightarrow \beta x \in [\![\Gamma]\!]$ *for all* $\alpha, \beta \in \mathsf{At}$
(ii) $\Gamma \Rightarrow_{\mathsf{At}} \Delta$ *is valid if and only if* $\Gamma \Rightarrow_A \Delta$ *is valid for some* $A \neq \emptyset$.

4 Soundness

In this section we prove that SGKAT^∞ is sound. We will first prove that *well-founded* (that is, finite) SGKAT^∞-proofs are sound. For the sake of readability we will write $[\![\Gamma]\!]$ to abbreviate $[\![\gamma_1 \cdots \gamma_n]\!]$ for some list Γ of expressions $\gamma_1, \ldots, \gamma_n$. The following lemma is useful for the soundness proof.

Lemma 4. *For any set A of atoms, test b, and cedent Θ, we have:*

1. $[\![e +_b f, \Theta]\!] = ([\![b]\!] \diamond [\![e, \Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![f, \Theta]\!])$;
2. $[\![e^{(b)}, \Theta]\!] = ([\![b]\!] \diamond [\![e, e^{(b)}, \Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![\Theta]\!])$.

We will moreover use the following property of the system SGKAT, which follows from direct inspection of the rules and the fact that sequents are lists.

Lemma 5. *Let $\Gamma \Rightarrow_A \Delta$ be a sequent, and let r be any rule of SGKAT. Then there is at most one rule instance of r with conclusion $\Gamma \Rightarrow_A \Delta$.*

We prioritise the rules of SGKAT in order of occurrence in Figure 3, reading left-to-right, top-to-bottom. Hence, each left logical rule is of higher priority than each right logical rule, which is of higher priority than each axiom or modal rule.

Definition 7. *A rule instance of r with conclusion $\Gamma \Rightarrow_A \Delta$ is said to* have priority *if any other rule instance, say of r', with conclusion $\Gamma \Rightarrow_A \Delta$ is of lower priority (that is, the rule r' appears after r in Figure 3).*

Recall that a rule is *sound* if the validity of all its premisses implies the validity of its conclusion. Conversely, a rule is *invertible* if the validity of its conclusion implies the validity of all of its premisses.

The above notion of priority will be used in the completeness proof of Section 6 to guide a proof search procedure. Conveniently, the following proposition entails that every rule instance which has priority is invertible, allowing this proof search procedure to be deterministic.

Proposition 1. *Every rule of SGKAT is sound. Moreover, every rule is invertible except for k and k_0, which are invertible whenever they have priority.*

Proposition 1 entails that all finite proofs are sound. We will now extend this result to non-well-founded proofs, closely following the treatment in [7]. We first recursively define a syntactic abbreviation: $[e^{(b)}]^0 := \bar{b}$ and $[e^{(b)}]^{n+1} := be[e^{(b)}]^n$.

Lemma 6. *For every $n \in \mathbb{N}$: if we have $\mathsf{SGKAT} \vdash^\infty e^{(b)}, \Gamma \Rightarrow_A \Delta$, then we also have $\mathsf{SGKAT} \vdash^\infty [e^{(b)}]^n, \Gamma \Rightarrow_A \Delta$.*

We let the *while-height* $\mathsf{wh}(e)$ be the maximal nesting of while loops in a given expression e. Formally,

- $\mathsf{wh}(b) = \mathsf{wh}(p) = 0$;
- $\mathsf{wh}(e^{(b)}) = \mathsf{wh}(e) + 1$.
- $\mathsf{wh}(e \cdot f) = \mathsf{wh}(e +_b f) = \max\{\mathsf{wh}(e), \mathsf{wh}(f)\}$;

Given a list Γ, the *weighted while-height* $\mathsf{wwh}(\Gamma)$ of Γ is defined to be the multiset $[\mathsf{wh}(e) : e \in \Gamma]$. We order such multisets using the Dershowitz–Manna ordering (for linear orders): we say that $N < M$ if and only if $N \neq M$ and for the greatest n such that $N(n) \neq M(n)$, it holds that $N(n) < M(n)$.

Note that in any SGKAT-derivation the weighted while-height of the antecedent does not increase when reading bottom-up. Moreover, we have:

Lemma 7. $\mathsf{wwh}([e^{(b)}]^n, \Gamma) < \mathsf{wwh}(e^{(b)}, \Gamma)$ *for every* $n \in \mathbb{N}$.

Finally, we can prove the soundness theorem using induction on $\mathsf{wwh}(\Gamma)$.

Theorem 1 (Soundness). *If* $\mathsf{SGKAT} \vdash^{\infty} \Gamma \Rightarrow_A \Delta$, *then* $A \diamond [\![\Gamma]\!] \subseteq [\![\Delta]\!]$.

5 Finite-stateness

Before we show that SGKAT^{∞} is not only sound, but also complete, we will first show that every SGKAT^{∞}-proof is *finite-state*, *i.e.* that it contains at most finitely many distinct sequents. Our treatment differs from that in [7] in two major ways. First, we formalise the notion of (sub)occurrence using the standard notion of a *syntax tree*. Secondly, we obtain a polynomial bound on the number of distinct sequents occurring in a proof, rather than an exponential one.

Definition 8. *The* syntax tree (T_e, l_e) *of an expression* e *is a well-founded, labelled and ordered tree, defined by the following induction on* e.

- *If* e *is a test or primitive program, its syntax tree only has a root node* ϵ, *with label* $l_e(\epsilon) := e$.
- *If* $e = f_1 \circ f_2$ *where* $\circ = \cdot$ *or* $\circ = +_b$, *its syntax tree again has a root node* ϵ *with label* $l_e(\epsilon) = e$, *and with two outgoing edges. The first edge connects* ϵ *to* (T_{f_1}, l_{f_1}), *the second edge connects it to* (T_{f_2}, l_{f_2}).
- *If* $e = f^{(b)}$, *its syntax tree again has a root node* ϵ *with label* $l_e(\epsilon) = e$, *but now with just one outgoing edge. This edge connects* ϵ *to* (T_f, l_f).

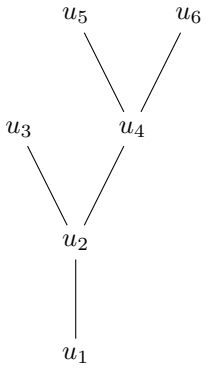
Definition 9. *An* e-cedent *is a list of nodes in the syntax tree of* e. *The* realisation *of an* e*-cedent* $u_1, \ldots, u_n$ *is the cedent* $l_e(u_1), \ldots, l_e(u_n)$.

Definition 10. *Let* u *be a node in the syntax tree of* e. *We define the* e*-cedent* $\mathsf{tail}(u)$ *inductively as follows:*

- *For the root* ϵ *of* T_e, *we set* $\mathsf{tail}(\epsilon) := \epsilon$.
- *For every node* u *of* T_e, *we define* tail *on its children by a case distinction on the main connective* mc *of* u*:*
  - *if* $\mathsf{mc} = \cdot$, *let* u_1 *and* u_2 *be, respectively, the first and second child of* u. *We set* $\mathsf{tail}(u_1) := u_2, \mathsf{tail}(u)$ *and* $\mathsf{tail}(u_2) := \mathsf{tail}(u)$.
  - *if* $\mathsf{mc} = +_b$, *let* u_1 *and* u_2 *again be its first and second child. We set* $\mathsf{tail}(u_1) := \mathsf{tail}(u_2) := \mathsf{tail}(u)$.
  - *if* $\mathsf{mc} = (b)$, *let* v *be the single child of* u. *We set* $\mathsf{tail}(v) := u, \mathsf{tail}(u)$.

An e-cedent is called tail*-generated if it is empty or of the form $u, \mathsf{tail}(u)$ for some node u in the syntax tree of e.*

Example 5. Below is the syntax tree of $(p(p+_b 1))^{(b)}$ and a calculation of $\mathsf{tail}(u_3)$.



$$
\begin{aligned}
l(u_1) &= (p(p +_b 1))^{(b)} \\
l(u_2) &= p(p +_b 1) \\
l(u_3) &= p \\
l(u_4) &= p +_b 1 \\
l(u_5) &= p \\
l(u_6) &= 1
\end{aligned}
\qquad
\begin{aligned}
\mathsf{tail}(u_3) &= u_4, \mathsf{tail}(u_2) \\
&= u_4, u_1, \mathsf{tail}(u_1) \\
&= u_4, u_1
\end{aligned}
$$

The following lemma embodies the key idea for establishing finite-stateness.

Lemma 8. *Let π be an* SGKAT^∞*-derivation of a sequent of the form $e \Rightarrow_A f$. Then every antecedent in π is the realisation of a* tail*-generated e-sequent, and every succedent is the realisation of a* tail*-generated f-sequent or 0-sequent.*

The number of realisations of tail-generated e-sequents is clearly linear in the size of the syntax tree of e, for every expression e. Hence we obtain:

Corollary 1. *The number of distinct sequents in an* SGKAT^∞*-proof of $e \Rightarrow_A f$ is polynomial in $|T_e| + |T_f|$.*

Note that the above lemma and corollary can easily be generalised to arbitrary (rather than singleton) cedents, by rewriting each cedent $e_1, \ldots, e_n$ as $e_1 \cdots e_n$. The following corollary follows by a standard argument in the literature.

Corollary 2. *If $\Gamma \Rightarrow_A \Delta$ has an* SGKAT^∞*-proof, then it has a regular one.*

We define a *cyclic* SGKAT*-proof* as a regular SGKAT^∞-proof. Cyclic proofs can be equivalently described using finite trees with back edges, but this is not needed for the purposes of the present paper.

6 Completeness and complexity

In this section we prove the completeness of SGKAT^∞. Our argument uses a proof search procedure, which we will show to induce a coNP decision procedure. The material in this section is again inspired by [7], but requires several modifications to treat the tests present in `GKAT`. We first verify:

Lemma 9. *Any valid sequent is the conclusion of some rule application.*

Note that in the following lemma A and B may be distinct.

Lemma 10. *Let π be a derivation using only right logical rules and containing a branch of the form:*

$$\dfrac{\begin{array}{c}\Gamma \Rightarrow_B e^{(b)}, \Delta \\ \vdots \end{array}}{\Gamma \Rightarrow_A e^{(b)}, \Delta}\ (b)\text{-}r \qquad (*)$$

such that (1) $\Gamma \Rightarrow_A e^{(b)}, \Delta$ is valid, and (2) every succedent on the branch has $e^{(b)}, \Delta$ as a final segment. Then $\Gamma \Rightarrow_B 0$ is valid.

Lemma 11. *Let $(\Gamma_n \Rightarrow_{A_n} \Delta_n)_{n\in\omega}$ be an infinite branch of some SGKAT^∞-derivation on which the rule (b)-r is applied infinitely often. Then there are n, m with $n < m$ such that the following hold:*

(i) the sequents $\Gamma_n \Rightarrow_{A_n} \Delta_n$ and $\Gamma_m \Rightarrow_{A_m} \Delta_m$ are equal;
(ii) the sequent $\Gamma_n \Rightarrow_{A_n} \Delta_n$ is the conclusion of (b)-r in π;
(iii) for every $i \in [n, m)$ it holds that Δ_n is a final segment of Δ_i.

With the above lemmas in place, we are ready for the completeness proof.

Theorem 2 (Completeness). *Every valid sequent is provable in SGKAT^∞.*

Proof (sketch). Given a valid sequent, we apply the following bottom-up proof search procedure. At each step we apply the rule that has priority. By Lemma 9 such a rule exists, and by Lemma 5 it is unique. Moreover, by Lemma 1, the rule application is invertible, whence every sequent encountered throughout our procedure is valid. There is one exception: if at some point Lemma 10 is applicable, and k_0 is as well, then we apply k_0. Note that k_0 does not have priority at this point, because (b)-r is applicable as well. However, by Lemma 10 the premiss of this application of k_0 is nonetheless valid. Now suppose that our procedure creates an infinite branch. We claim that this branch must be fair for (b)-l. Suppose, towards a contradiction, that it is not. Since every rule, except for the axioms and the right logical rules, shortens the antecedent, there is some point on the branch at which only right logical rules are applied. From Lemma 11 it follows that Lemma 10 is at some point applicable, at which point we apply k_0. But after that the succedent is 0, contradicting the assumption that infinitely many right logical rules are applied.

By Corollary 2 we obtain that the subset of cyclic SGKAT-proofs is also complete.

Corollary 3. *Every valid sequent has a regular SGKAT^∞-proof.*

Proposition 2. *The proof search procedure of Theorem 2 induces a coNP decision procedure for the language inclusion problem for $\mathtt{GKAT}$-expressions.*

Proof (sketch). The idea is that if some branch of the proof search fails, then there is a failing branch which is polynomial in the size of the endsequent. Note first that any failing branch must be finite, for every infinite branch constructed by the proof search is fair for (b)-l. Given such a branch, we delete all segments between identical sequents, except possibly for the segment that together with Lemma 10 justifies an application of k_0 that does not have priority (note there is at most one such application of k_0). The resulting branch still fails but, by Corollary 1, it is also of length polynomial in the size of the endsequent.

7 Conclusion and future work

In this paper we have presented a non-well-founded proof system SGKAT^∞ for GKAT. We have shown that the system is sound and complete with respect to the language model. In fact, the fragment of *regular* proofs is already complete, which means one can view SGKAT as a cyclic proof system. Our system is similar to the system for Kleene Algebra in [7], but the deterministic nature of GKAT allows us to use ordinary sequents rather than hypersequents. To deal with the tests of GKAT every sequent is annotated by a set of atoms. Like in [7], our completeness argument makes use of a proof search procedure. Here again the relative simplicity of GKAT pays off: the proof search procedure induces a coNP decision procedure, whereas that of Kleene Algebra is in PSPACE.

Perhaps the most interesting question for future work is whether our system could be used to prove the completeness of some (ordered)-algebraic axiomatisation of GKAT. We envision using the original GKAT axioms (see Figure 2 above), but basing it on *inequational* logic rather than equational logic. This would allow one to use a *least* fixed point rule of the form

$$\frac{eg +_b f \leq g}{e^{(b)} f \leq g}$$

eliminating the need for a Salomaa-style side condition. We hope to be able to prove the completeness of such an inequational system by translating cyclic SGKAT-proofs into well-founded proofs in the inequational system. This is inspired by the paper [6], where the same strategy is used to give an alternative proof of the left-handed completeness of Kleene Algebra.

Another relevant question is the exact complexity of the language inclusion problem for GKAT-expressions. We have obtained an upper-bound of coNP, but this is unlikely to be optimal. This can probably be improved to at least P, by regarding proofs as graphs (more specifically as parity games), rather than trees. Another, related, approach would be to turn GKAT-expressions into automata and to try to construct a *simulation* between them (cf. the decision procedure in [17]).

Finally, it would be interesting to verify the conjecture in Remark 8 above.

Acknowledgments. Jan Rooduijn thanks Anupam Das, Tobias Kappé, Johannes Marti and Yde Venema for insightful discussions on the topic of this paper. Alexandra Silva further wants to acknowledge Sonia Marin, who some years ago proposed a similar master project at UCL. Lastly, Jan Rooduijn is grateful for the inspiring four-week research visit at the Computer Science department of Cornell in the summer of 2022.

References

1. Afshari, B., Enqvist, S., Leigh, G.E.: Cyclic proofs for the first-order μ-calculus. Logic Journal of the IGPL (2022)
2. Afshari, B., Wehr, D.: Abstract cyclic proofs. In: 28th International Workshop on Logic, Language, Information, and Computation, WoLLIC. Lecture Notes in Computer Science, vol. 13468, pp. 309–325. Springer (2022)
3. Brotherston, J.: Cyclic proofs for first-order logic with inductive definitions. In: 14th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX. Lecture Notes in Computer Science, vol. 3702, pp. 78–92. Springer (2005)
4. Burris, S., Sankappanavar, H.P.: A course in universal algebra, Graduate Texts in Mathematics, vol. 78. Springer (1981)
5. Conway, J.H.: Regular algebra and finite machines. Chapman and Hall (1971)
6. Das, A., Doumane, A., Pous, D.: Left-handed completeness for Kleene algebra, via cyclic proofs. In: 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR. EPiC Series in Computing, vol. 57, pp. 271–289 (2018)
7. Das, A., Pous, D.: A cut-free cyclic proof system for Kleene algebra. In: 26th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX. Lecture Notes in Computer Science, vol. 10501, pp. 261–277. Springer (2017)
8. Dekker, M., Kloibhofer, J., Marti, J., Venema, Y.: Proof systems for the modal μ-calculus obtained by determinizing automata. In: 32nd International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX. pp. 242–259. Springer (2023)
9. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Automata Studies, pp. 3–41. No. 34 in Annals of Mathematics Studies, Princeton University Press (1956)
10. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation **110**(2), 366–390 (1994)
11. Kozen, D.: Kleene algebra with tests. ACM Transactions on Programming Languages and Systems **19**(3), 427–443 (1997)
12. Kozen, D.: Nonlocal flow of control and Kleene algebra with tests. In: 23rd Annual Symposium on Logic in Computer Science, LICS. pp. 105–117. IEEE (2008)
13. Kozen, D., Tseng, W.D.: The Böhm-Jacopini theorem is false, propositionally. In: 9th International Conference on Mathematics of Program Construction, MPC. Lecture Notes in Computer Science, vol. 5133, pp. 177–192. Springer (2008)
14. Kuperberg, D., Pinault, L., Pous, D.: Cyclic proofs, system T, and the power of contraction. 48th Annual Symposium on Principles of Programming Languages, POPL pp. 1–28 (2021)
15. Salomaa, A.: Two complete axiom systems for the algebra of regular events. Journal of the ACM **13**(1), 158–169 (1966)
16. Schmid, T., Kappé, T., Kozen, D., Silva, A.: Guarded Kleene algebra with tests: Coequations, coinduction, and completeness. In: 48th International Colloquium on Automata, Languages, and Programming, ICALP. LIPIcs, vol. 198, pp. 142:1–142:14. Schloss Dagstuhl (2021)
17. Smolka, S., Foster, N., Hsu, J., Kappé, T., Kozen, D., Silva, A.: Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time. 47th Annual Symposium on Principles of Programming Languages, POPL pp. 61:1–61:28 (2020)

A Omitted proofs

A.1 ... of Section 1

A.2 ... of Section 2

Lemma 1. *$L^{n+1} = L \diamond L^n$ for every language L of guarded strings.*

Proof. Since At is the identity element for the fusion operator, we have

$$L^{n+1} = \mathsf{At} \diamond \underbrace{L \diamond \cdots \diamond L}_{n+1 \text{ times}} = L \diamond \mathsf{At} \diamond \underbrace{L \diamond \cdots \diamond L}_{n \text{ times}} = L \diamond L^n,$$

as required.

Lemma 2. *Let p be a primitive program and let L and K be languages of guarded strings. Then $[\![p]\!] \diamond L = [\![p]\!] \diamond K$ implies $L = K$.*

Proof. Since $[\![p]\!] = \{\alpha p \beta : \alpha, \beta \in \mathsf{At}\}$, we have

$$\gamma y \in L \Leftrightarrow \gamma p \gamma y \in [\![p]\!] \diamond L \Leftrightarrow \gamma p \gamma y \in [\![p]\!] \diamond K \Leftrightarrow \gamma y \in K,$$

as required.

A.3 ... of Section 3

Lemma 3. *Let Γ and Δ be exposed lists of expressions. Then:*

(i) $\alpha x \in [\![\Gamma]\!] \Leftrightarrow \beta x \in [\![\Gamma]\!]$ *for all* $\alpha, \beta \in \mathsf{At}$
(ii) $\Gamma \Rightarrow_{\mathsf{At}} \Delta$ *is valid if and only if* $\Gamma \Rightarrow_A \Delta$ *is valid for some* $A \neq \emptyset$.

Proof. For item (i), we make a case distinction on whether $\Gamma = \epsilon$ or $\Gamma = p, \Theta$ for some list Θ. If $\Gamma = \epsilon$, the result follows immediately from the fact that $[\![\epsilon]\!] = \mathsf{At}$. If $\Gamma = p, \Theta$, we have

$$[\![\Gamma]\!] = [\![p]\!] \diamond [\![\Theta]\!] = \{\gamma p \delta y : \gamma \in \mathsf{At}, \delta y \in [\![\Theta]\!]\},$$

which also suffices.

For item (ii), the only non-trivial implication is the one from right to left. So suppose $\Gamma \Rightarrow_A \Delta$ for some $A \neq \emptyset$. Let $\alpha \in \mathsf{At}$ and let $\beta \in A$ be arbitrary. We find:

$$\begin{aligned} \alpha x \in [\![\Gamma]\!] &\Rightarrow \beta x \in [\![\Gamma]\!] && \text{(item (i))} \\ &\Rightarrow \beta x \in [\![\Delta]\!] && (\beta \in A, \text{ hypothesis}) \\ &\Rightarrow \alpha x \in [\![\Delta]\!], && \text{(item (i))} \end{aligned}$$

as required.

A.4 ... of Section 4

Lemma 4. *Let A be a set of atoms, let b be a test, and let Θ be a list of expressions. We have:*

1. $[\![e +_b f, \Theta]\!] = ([\![b]\!] \diamond [\![e, \Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![f, \Theta]\!])$;
2. $[\![e^{(b)}, \Theta]\!] = ([\![b]\!] \diamond [\![e, e^{(b)}, \Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![\Theta]\!])$.

Proof. Both items are shown by simply unfolding the definitions. We will use the fact $\diamond$ distributes over $\cup$. Note that $\cup$ is not the same as *guarded* union, over which $\diamond$ is merely right-distributive.

For the first item, we calculate

$$\begin{aligned}
[\![e +_b f, \Theta]\!] &= [\![e +_b f]\!] \diamond [\![\Theta]\!] && \text{(sequent interpretation)}\\
&= (([\![b]\!] \diamond [\![e]\!]) \cup (\overline{[\![b]\!]} \diamond [\![f]\!])) \diamond [\![\Theta]\!] && \text{(interpretation of } +_b)\\
&= ([\![b]\!] \diamond [\![e]\!] \diamond [\![\Theta]\!]) \cup (\overline{[\![b]\!]} \diamond [\![f]\!] \diamond [\![\Theta]\!]) && (\diamond \text{ distributes over } \cup)\\
&= ([\![b]\!] \diamond [\![e]\!] \diamond [\![\Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![f]\!] \diamond [\![\Theta]\!]) && (\overline{[\![b]\!]} = [\![\bar{b}]\!])\\
&= ([\![b]\!] \diamond [\![e, \Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![f, \Theta]\!]). && \text{(sequent interpretation)}
\end{aligned}$$

For the second item, we have

$$\begin{aligned}
[\![e^{(b)}, \Theta]\!] &= [\![e^{(b)}]\!] \diamond [\![\Theta]\!] && \text{(sequent int.)}\\
&= \bigcup_{n\geq 0} ([\![b]\!] \diamond [\![e]\!])^n \diamond \overline{[\![b]\!]} \diamond [\![\Theta]\!] && \text{(int. } -^{(b)})\\
&= (\bigcup_{n\geq 1} ([\![b]\!] \diamond [\![e]\!])^n \cup \mathsf{At}) \diamond \overline{[\![b]\!]} \diamond [\![\Theta]\!] && \text{(split } \textstyle\bigcup)\\
&= (\bigcup_{n\geq 1} ([\![b]\!] \diamond [\![e]\!])^n \diamond \overline{[\![b]\!]} \diamond [\![\Theta]\!]) \cup (\mathsf{At} \diamond \overline{[\![b]\!]} \diamond [\![\Theta]\!]) && (\diamond \text{ dist. } \cup)\\
&= (\bigcup_{n\geq 1} ([\![b]\!] \diamond [\![e]\!])^n \diamond [\![\bar{b}]\!] \diamond [\![\Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![\Theta]\!]) && (\overline{[\![b]\!]} = [\![\bar{b}]\!])\\
&= (\bigcup_{n\geq 0} [\![b]\!] \diamond [\![e]\!] \diamond ([\![b]\!] \diamond [\![e]\!])^n \diamond [\![\bar{b}]\!] \diamond [\![\Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![\Theta]\!]) && \text{(Lem. 1)}\\
&= ([\![b]\!] \diamond [\![e]\!] \diamond \bigcup_{n\geq 0} ([\![b]\!] \diamond [\![e]\!])^n \diamond [\![\bar{b}]\!] \diamond [\![\Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![\Theta]\!]) && (\diamond \text{ dist. } \textstyle\bigcup)\\
&= ([\![b]\!] \diamond [\![e]\!] \diamond [\![e^{(b)}]\!] \diamond [\![\Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![\Theta]\!]) && \text{(int. } -^{(b)})\\
&= ([\![b]\!] \diamond [\![e, e^{(b)}, \Theta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![\Theta]\!]), && \text{(sequent int.)}
\end{aligned}$$

as required.

Proposition 2. *Every rule of* SGKAT *is sound. Moreover, every rule is invertible except for* k *and* k_0*, which are invertible whenever they have priority.*

Proof. We will cover the rules of SGKAT one-by-one.

(b-l) This is immediate by Lemma 4.1.

(b-r) We have:

$$\begin{aligned} A \diamond [\![\Gamma]\!] \subseteq [\![b, \Delta]\!] &\Leftrightarrow A \diamond [\![\Gamma]\!] \subseteq [\![b]\!] \diamond [\![\Delta]\!] && \text{(sequent int.)} \\ &\Leftrightarrow A \upharpoonright b \diamond [\![\Gamma]\!] \subseteq [\![b]\!] \diamond [\![\Delta]\!] && \text{(by (†))} \\ &\Leftrightarrow A \upharpoonright b \diamond [\![\Gamma]\!] \subseteq [\![\Delta]\!] && (A \upharpoonright b \subseteq [\![b]\!]) \\ &\Leftrightarrow A \diamond [\![\Gamma]\!] \subseteq [\![\Delta]\!] && \text{(by (†))} \end{aligned}$$

($\cdot$-l) Immediate, since $A \diamond [\![e \cdot f, \Gamma]\!] = A \diamond [\![e, f, \Gamma]\!]$.

($\cdot$-r) Likewise, but by $[\![e \cdot f, \Delta]\!] = [\![e, f, \Delta]\!]$.

($+_b$-l) This follows directly from the fact that

$$\begin{aligned} A \diamond [\![e +_b f, \Gamma]\!] &= A \diamond [\![e +_b f]\!] \diamond [\![\Gamma]\!] && \text{(sequent int.)} \\ &= A \diamond (([\![b]\!] \diamond [\![e, \Gamma]\!]) \cup ([\![\bar{b}]\!] \diamond [\![f, \Gamma]\!])) && \text{(Lem. 4.2)} \\ &= (A \diamond [\![b]\!] \diamond [\![e, \Gamma]\!]) \cup (A \diamond [\![\bar{b}]\!] \diamond [\![f, \Gamma]\!]) && \text{(distrib.)} \\ &= (A \upharpoonright b \diamond [\![e, \Gamma]\!]) \cup (A \upharpoonright \bar{b} \diamond [\![f, \Gamma]\!]) && \text{(Lem. 4.1)} \end{aligned}$$

($+_b$-r) We find

$$\begin{aligned} &A \diamond [\![\Gamma]\!] \subseteq [\![e +_b f]\!] \diamond [\![\Delta]\!] \\ &\quad\Leftrightarrow A \diamond [\![\Gamma]\!] \subseteq ([\![b]\!] \diamond [\![e, \Delta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![f, \Delta]\!]) \\ &\qquad\Leftrightarrow A \upharpoonright b \diamond [\![\Gamma]\!] \subseteq [\![e, \Delta]\!] \text{ or } A \upharpoonright \bar{b} \subseteq [\![f, \Delta]\!], \end{aligned}$$

where the first equivalence holds due to Lemma 4.2, and the second due to $A \diamond [\![\Gamma]\!] = ([\![b]\!] \diamond A \diamond [\![\Gamma]\!]) \cup ([\![\bar{b}]\!] \diamond A \diamond [\![\Gamma]\!])$ and Lemma 4.1.

((b)-l) This follows directly from the fact that

$$\begin{aligned} A \diamond [\![e^{(b)}, \Gamma]\!] &= A \diamond [\![e^{(b)}]\!] \diamond [\![\Gamma]\!] && \text{(sequent int.)} \\ &= A \diamond (([\![b]\!] \diamond [\![e, e^{(b)}, \Gamma]\!]) \cup ([\![\bar{b}]\!] \diamond [\![f, \Gamma]\!])) && \text{(Lem. 4.3)} \\ &= (A \diamond [\![b]\!] \diamond [\![e, e^{(b)}, \Gamma]\!]) \cup (A \diamond [\![\bar{b}]\!] \diamond [\![f, \Gamma]\!]) && \text{(distrib.)} \\ &= (A \upharpoonright b \diamond [\![e, e^{(b)}, \Gamma]\!]) \cup (A \upharpoonright \bar{b} \diamond [\![f, \Gamma]\!]) && \text{(Lem. 4.1)} \end{aligned}$$

((b)-r) We find

$$\begin{aligned} &A \diamond [\![\Gamma]\!] \subseteq [\![e^{(b)}, \Delta]\!] \\ &\quad\Leftrightarrow A \diamond [\![\Gamma]\!] \subseteq ([\![b]\!] \diamond [\![e, e^{(b)}, \Delta]\!]) \cup ([\![\bar{b}]\!] \diamond [\![\Delta]\!]) \\ &\qquad\Leftrightarrow A \upharpoonright b \diamond [\![\Gamma]\!] \subseteq [\![b]\!] \diamond [\![e, e^{(b)}, \Delta]\!] \text{ and } A \upharpoonright \bar{b} \subseteq [\![\bar{b}]\!] \diamond [\![\Delta]\!], \end{aligned}$$

where the first equivalence holds due to Lemma 4.3, and the second due to $A \diamond [\![\Gamma]\!] = ([\![b]\!] \diamond A \diamond [\![\Gamma]\!]) \cup ([\![\bar{b}]\!] \diamond A \diamond [\![\Gamma]\!])$ and Lemma 4.1.

(id) This follows from $A \diamond [\![1]\!] = A \diamond \mathsf{At} = A \subseteq \mathsf{At} = [\![1]\!]$.

($\bot$) We have $\emptyset \diamond [\![\Gamma]\!] = \emptyset \subseteq [\![\Delta]\!]$.

(k) Suppose first that some application of k does *not* have priority. The only rule of higher priority than k which can have a conclusion of the form $p, \Gamma \Rightarrow_A p, \Delta$ is $\bot$, whence we must have $A = \emptyset$. As shown in the previous case, this conclusion must be valid. Hence under this restriction the rule application is vacuously sound. It is, however, not invertible, as the following rule instance demonstrates

$$\mathsf{k}\,\frac{1 \Rightarrow_{\mathsf{At}} 0}{p, 1 \Rightarrow_{\emptyset} p, 0}$$

Next, suppose that some application of k does have priority. This means that the set A of atoms in the conclusion $p, \Gamma \Rightarrow_A p, \Delta$ is *not* empty. We will show that under this restriction the rule is both sound and invertible. Let $\alpha \in A$. We have

$$\begin{aligned} A \diamond [\![p, \Gamma]\!] \subseteq [\![p, \Delta]\!] &\Leftrightarrow A \diamond [\![p]\!] \diamond [\![\Gamma]\!] \subseteq [\![p]\!] \diamond [\![\Delta]\!] && \text{(seq. int.)} \\ &\Leftrightarrow \alpha \diamond [\![p]\!] \diamond [\![\Gamma]\!] \subseteq [\![p]\!] \diamond [\![\Delta]\!] && (\alpha \in A, \text{Lem. 3}) \\ &\Leftrightarrow [\![p]\!] \diamond [\![\Gamma]\!] \subseteq [\![p]\!] \diamond [\![\Delta]\!] && \text{(Lem. 3)} \\ &\Leftrightarrow [\![\Gamma]\!] \subseteq [\![\Delta]\!], && \text{(Lem. 2)} \end{aligned}$$

as required.

(k_0) For the final rule k_0, we will first show the soundness of all instances, and then the invertibility of those instances which have priority. For soundness, suppose that the premiss is valid. Since

$$[\![\Gamma]\!] = \mathsf{At} \diamond [\![\Gamma]\!] \subseteq [\![0]\!] = \emptyset,$$

it follows that $[\![\Gamma]\!] = \emptyset$. Hence

$$A \diamond [\![p, \Gamma]\!] = A \diamond [\![p]\!] \diamond [\![\Gamma]\!] = A \diamond [\![p]\!] \diamond \emptyset = \emptyset \subseteq [\![\Delta]\!],$$

as required.

For invertibility, suppose that some instance of k_0 has priority. Then the conclusion $p, \Gamma \Rightarrow_A \Delta$ cannot be the conclusion of any other rule application.

Suppose that $p, \Gamma \Rightarrow_A \Delta$ is valid. We wish to show that $\Gamma \Rightarrow_{\mathsf{At}} 0$ is valid, or, in other words, that $[\![\Gamma]\!] = \emptyset$.

First note that, as in the previous case, from the assumption that our instance of k_0 has priority, it follows that $A \neq \emptyset$.

We now make a case distinction on the shape of Δ. Suppose first that $\Delta = \epsilon$. Then

$$A \diamond [\![p, \Gamma]\!] \subseteq [\![\Delta]\!] = [\![\epsilon]\!] = \mathsf{At}.$$

As $A \diamond [\![p, \Gamma]\!] = \{\alpha p \beta x : \alpha \in A \text{ and } \beta x \in [\![\Gamma]\!]\}$, we must have $[\![\Gamma]\!] = \emptyset$.

Next, suppose that Δ has a leftmost expression e. By the assumption that the rule instance has priority, we know that e is not of the form $e_0 \cdot e_1$, $e_0 +_b e_1$, or $e^{(b)}$, for otherwise a right logical rule could be applied. Hence, the expression e must either be a test or a primitive program.

If e is a test, say b, we know that $A \restriction b \neq A$, for otherwise b-r could be applied. Recall that it suffices to show that $[\![\Gamma]\!] = \emptyset$. So suppose, towards a contradiction, that there is some $\beta x \in [\![\Gamma]\!]$. Let $\alpha \in A$ such that $\alpha \not\leq b$. Then $\alpha p \beta x \in [\![p, \Gamma]\!] \subseteq [\![\Delta]\!]$. But this contradicts the fact that $[\![\Delta]\!] \subseteq \{\alpha y : \alpha \leq b\}$.
Finally, suppose that e is a primitive program, say q. Write $\Delta = q, \Theta$. First note that assumption that the rule instance has priority implies $p \neq q$, for otherwise the rule k could be applied. We have:

$$A \diamond [\![p, \Gamma]\!] \subseteq [\![\Delta]\!] = \{\alpha q \beta x : \beta x \in [\![\Theta]\!]\},$$

As $A \diamond [\![p, \Gamma]\!] = \{\alpha p \beta x : \alpha \in A \text{ and } \beta x \in [\![\Gamma]\!]\}$ and $p \neq q$, we again find that $[\![\Gamma]\!] = \emptyset$.

This finishes the proof.

Lemma 6. *For every $n \in \mathbb{N}$: if we have $\mathsf{SGKAT} \vdash^{\infty} e^{(b)}, \Gamma \Rightarrow_A \Delta$, then we also have $\mathsf{SGKAT} \vdash^{\infty} [e^{(b)}]^n, \Gamma \Rightarrow_A \Delta$.*

Proof. We assume that $A \neq \emptyset$, for otherwise the lemma is trivial. Let π be the assumed SGKAT^{∞}-proof of $e^{(b)}, \Gamma \Rightarrow_A \Delta$. Note that, since all succedents referred to in the lemma are equal to Δ, it suffices to prove the lemma under the assumption that the last rule applied in π is *not* a right logical rule. Hence, we may assume that the last rule applied in π is (b)-l, for that is the only remaining rule with a sequent of this shape as conclusion. This means that π is of the form:

$$\dfrac{\begin{matrix}\pi_1 & \pi_2 \\ e, e^{(b)}, \Gamma \Rightarrow_{A\restriction b} \Delta & \Gamma \Rightarrow_{A\restriction \bar{b}} \Delta\end{matrix}}{e^{(b)}, \Gamma \Rightarrow_A \Delta}\ (b)\text{-}l$$

We show the lemma by induction on n. For the induction base, we take the following proof:

$$\dfrac{\begin{matrix}\pi_2 \\ \Gamma \Rightarrow_{A\restriction \bar{b}} \Delta\end{matrix}}{[e^{(b)}]^0, \Gamma \Rightarrow_A \Delta}\ \bar{b}\text{-}l$$

For the inductive step $n+1$, we construct from π_1 a proof τ of $e, [e^{(b)}]^n, \Gamma \Rightarrow_{A\restriction b} \Delta$. To that end, we first replace in π_1 every occurrence of $e^{(b)}, \Gamma$ as a final segment of the antecedent by $e^{(b)^n}, \Gamma$ and cut off all branches at sequents of the form $[e^{(b)}]^n, \Gamma \Rightarrow_B \Theta$. This may be depicted as follows, where to the left of the arrow $\rightsquigarrow$ we have a branch of π_1, and to right the resulting branch of τ.

$$\dfrac{\begin{matrix}\vdots \\ e^{(b)}, \Gamma \Rightarrow_B \Theta \\ \vdots \\ e, e^{(b)}, \Gamma \Rightarrow_{A\restriction b} \Delta\end{matrix}}{} \quad \rightsquigarrow \quad \dfrac{\begin{matrix}\\ [e^{(b)}]^n, \Gamma \Rightarrow_B \Theta \\ \vdots \\ e, [e^{(b)}]^n, \Gamma \Rightarrow_{A\restriction b} \Delta\end{matrix}}{}$$

Note that every remaining infinite branch in the resulting derivation τ satisfies the fairness condition. Therefore, to turn τ into a proper SGKAT^∞-proof, we only need to close each open leaf, which by construction is of the form $[e^{(b)}]^n, \Gamma \Rightarrow_B \Delta$. Note that π_1 must contain a proof of $e^{(b)}, \Gamma \Rightarrow_B \Delta$, whence by the induction hypothesis the sequent $[e^{(b)}]^n, \Gamma \Rightarrow_B \Delta$ is provable. We can thus close the leaf by simply appending the witnessing proof.

Letting τ be the resulting proof, we finish the induction step by taking:

$$\dfrac{\begin{array}{c}\tau\\ e, [e^{(b)}]^n, \Gamma \Rightarrow_{A\restriction b} \Delta\end{array}}{[e^{(b)}]^{n+1}, \Gamma \Rightarrow_A \Delta}\, b\text{-}l$$

which gives us the required SGKAT^∞-proof.

Lemma 7. $\mathsf{wwh}([e^{(b)}]^n, \Gamma) < \mathsf{wwh}(e^{(b)}, \Gamma)$ *for every* $n \in \mathbb{N}$.

Proof. Let $k := \mathsf{wh}(e^{(b)})$. Note that the maximum while-height in $[e^{(b)}]^n$ is that of e. Hence, we have $\mathsf{wwh}([e^{(b)}]^n)(k) = 0 < 1 = \mathsf{wwh}(e^{(b)})(k)$. Therefore:

$$\begin{aligned}\mathsf{wwh}([e^{(b)}]^n, \Gamma)(k) &= \mathsf{wwh}([e^{(b)}]^n)(k) + \mathsf{wwh}(\Gamma)(k)\\ &< \mathsf{wwh}(e^{(b)})(k) + \mathsf{wwh}(\Gamma)(k) = \mathsf{wwh}(e^{(b)}, \Gamma)(k).\end{aligned}$$

Hence $\mathsf{wwh}([e^{(b)}]^n, \Gamma) \neq \mathsf{wwh}(e^{(b)}, \Gamma)$. Now suppose that for some $l \in \mathbb{N}$ we have $\mathsf{wwh}([e^{(b)}]^n, \Gamma)(l) > \mathsf{wwh}(e^{(b)}, \Gamma)(l)$. We leave it to the reader to verify that in this case we must have $l < k$. As $\mathsf{wwh}([e^{(b)}]^n, \Gamma)(k) < \mathsf{wwh}(e^{(b)}, \Gamma)(k)$, we find $\mathsf{wwh}([e^{(b)}]^n, \Gamma) < \mathsf{wwh}(e^{(b)}, \Gamma)$.

Theorem 1. *If* $\mathsf{SGKAT} \vdash^\infty \Gamma \Rightarrow_A \Delta$*, then* $A \diamond [\![\Gamma]\!] \subseteq [\![\Delta]\!]$.

Proof. We prove this by induction on $\mathsf{wwh}(\Gamma)$. Given a proof π of $\Gamma \Rightarrow_A \Delta$, let $\mathcal{B}$ contain for each infinite branch of π the node of least depth to which a rule (b)-l is applied. Note that $\mathcal{B}$ must be finite, for otherwise, by Kőnig's Lemma, the proof π cut off along $\mathcal{B}$ would have an infinite branch that does not satisfy the fairness condition.

Note that Proposition 1 entails that of every finite derivation with valid leaves the conclusion is valid. Hence, it suffices to show that each of the nodes in $\mathcal{B}$ is valid. To that end, consider an arbitrary such node labelled $e^{(b)}, \Gamma' \Rightarrow_{A'} \Delta'$ and the subproof π' it generates. By Lemma 6, we have that $[e^{(b)}]^n, \Gamma' \Rightarrow_{A'} \Delta'$ is provable for every n. Lemma 7 gives $\mathsf{wwh}([e^{(b)}]^n, \Gamma') < \mathsf{wwh}(e^{(b)}, \Gamma') \leq \mathsf{wwh}(\Gamma)$, and thus we may apply the induction hypothesis to obtain

$$A' \diamond [\![[e^{(b)}]^n]\!] \diamond [\![\Gamma]\!] \subseteq [\![\Delta]\!]$$

for every $n \in \mathbb{N}$. Then by

$$\bigcup_n (A' \diamond [\![[e^{(b)}]^n]\!] \diamond [\![\Gamma]\!]) = A' \diamond \bigcup_n ([\![[e^{(b)}]^n]\!]) \diamond [\![\Gamma]\!] = A' \diamond [\![e]\!]^{[\![b]\!]} \diamond [\![\Gamma]\!],$$

we obtain that $e^{(b)}, \Gamma' \Rightarrow_{A'} \Delta'$ is valid, as required.

A.5 ... of Section 5

Lemma 8. *Let π be an SGKAT^∞-derivation of a sequent of the form $e \Rightarrow_A f$. Then every antecedent in π is the realisation of a tail-generated e-sequent, and every succedent is the realisation of a tail-generated f-sequent or 0-sequent.*

Proof. We first prove the following claim.

> Let e be an expression and let u be a node in its syntax tree. Then $\mathsf{tail}(u)$ is a tail-generated e-sequent.

Proof (of claim). We prove this by induction on the syntax tree of e. For the root ϵ, we have $\mathsf{tail}(\epsilon) = \epsilon$, which is tail-generated by definition. Now suppose that the thesis holds for some arbitrary node u in the syntax tree of e. We will show that the thesis holds for the children of u by a case distinction on the main connective mc of u.

- $\mathsf{mc} = \cdot$. Let u_1 and u_2 be the first and second child of u, respectively. We have $\mathsf{tail}(u_1) = u_2, \mathsf{tail}(u) = u_2, \mathsf{tail}(u_2)$, which is tail-generated by definition. Moreover, we have that $\mathsf{tail}(u_2) = \mathsf{tail}(u)$ is tail-generated by the induction hypothesis.
- $\mathsf{mc} = +_b$. Then for each child v of u, we have $\mathsf{tail}(v) = \mathsf{tail}(u)$ and thus we can again invoke the induction hypothesis.
- $\mathsf{mc} = (b)$. Then for the single child v of u, it holds that $\mathsf{tail}(v) = u, \mathsf{tail}(u)$, which is tail-generated by definition.

Using this claim, the lemma follows by direct inspection of the rules.

A.6 ... of Section 6

Lemma 9. *Any valid sequent is the conclusion of some rule application.*

Proof. We prove this lemma by contraposition. So suppose $\Gamma \Rightarrow_A \Delta$ is *not* the conclusion of *any* rule application. We make a few observations:

- Both Γ and Δ are exposed, for otherwise $\Gamma \Rightarrow_A \Delta$ would be the conclusion of an application of a left, respectively right, logical rule.
- A is non-empty, for otherwise $\Gamma \Rightarrow_A \Delta$ would be the conclusion of an application of $\bot$.
- The leftmost expression of Γ is not a primitive program, for otherwise our sequent $\Gamma \Rightarrow_A \Delta$ would be the conclusion of an application of $\mathsf{k_0}$.
- The leftmost expression of Δ *is* a primitive program, for otherwise, by the previous items, the sequent $\Gamma \Rightarrow_A \Delta$ would be the conclusion of an application of id.

Hence $\Gamma \Rightarrow_A \Delta$ is of the form $\epsilon \Rightarrow_A p, \Theta$. Let $\alpha \in A$. Then $\alpha \in A \diamond [\![\epsilon]\!]$. However, since α is not of the form $\beta p \gamma y$, we have $\alpha \notin [\![p, \Theta]\!]$. This shows that $\Gamma \Rightarrow_A \Delta$ is not valid, as required.

Lemma 10. *Let π be a derivation using only right logical rules and containing a branch of the form:*

$$\dfrac{\begin{array}{c}\Gamma \Rightarrow_B e^{(b)}, \Delta \\ \vdots \end{array}}{\Gamma \Rightarrow_A e^{(b)}, \Delta}\ (b)\text{-}r \qquad (*)$$

such that (1) $\Gamma \Rightarrow_A e^{(b)}, \Delta$ is valid, and (2) every succedent on the branch has $e^{(b)}, \Delta$ as a final segment. Then $\Gamma \Rightarrow_B 0$ is valid.

Proof. We claim that $e^{(b)} \Rightarrow_B 0$ is provable. We will show this by exploiting the symmetry of the left and right logical rules of SGKAT (cf. Remark 7). Since on the branch (*) every rule is a right logical rule, and $e^{(b)}, \Delta$ is preserved throughout, we can construct a derivation π' of $e^{(b)} \Rightarrow_B 0$ from π by applying the analogous left logical rules to $e^{(b)}$. Note that the set of atoms B precisely determines the branch (*), in the sense that for every leaf $\Gamma \Rightarrow_C \Theta$ of π it holds that $C \cap B = \emptyset$. Hence, as the root of π' is $e^{(b)} \Rightarrow_B 0$, every branch of π' except for the one corresponding to (*) can be closed directly by an application of $\bot$. The branch corresponding to (*) is of the form

$$\dfrac{\begin{array}{c}e^{(b)} \Rightarrow_B 0 \\ \vdots \end{array}}{e^{(b)} \Rightarrow_B 0}\ (b)\text{-}l \qquad (*)$$

and can thus be closed by a back edge. The resulting finite tree with back edges clearly represents an SGKAT^∞-proof.

Now by soundness, we have $B \diamond [\![e^{(b)}]\!] = \emptyset$. Moreover, by the invertibility of the right logical rules and hypothesis (1), we get

$$B \diamond [\![\Gamma]\!] \subseteq B \diamond [\![e^{(b)}]\!] \diamond [\![\Delta]\!] = \emptyset,$$

as required.

Lemma 11. *Let $(\Gamma_n \Rightarrow_{A_n} \Delta_n)_{n \in \omega}$ be an infinite branch of some SGKAT^∞-derivation on which the rule (b)-r is applied infinitely often. Then there are n, m with $n < m$ such that the following hold:*

(i) the sequents $\Gamma_n \Rightarrow_{A_n} \Delta_n$ and $\Gamma_m \Rightarrow_{A_m} \Delta_m$ are equal;
(ii) the sequent $\Gamma_n \Rightarrow_{A_n} \Delta_n$ is the conclusion of (b)-r in π;
(iii) for every $i \in [n, m)$ it holds that Δ_n is a final segment of Δ_i.

Proof. First note that k_0 is not applied on this branch, because if it were then there could not be infinitely many applications of (b)-r.

By finite-stateness (cf. Corollary 1), there must be a $k \geq 0$ be such that every Δ_i with $i \geq k$ occurs infinitely often on the branch above. Denote by $|\Delta|$ the length of a given list Δ and let l be minimum of $\{|\Delta_i| : i \geq k\}$. In other words, l is the minimal length of the Δ_i with $i \geq k$.

To prove the lemma, we first claim that there is an $n \geq k$ such that $|\Delta_n| = l$ and the leftmost expression in Δ_n is of the form $e^{(b)}$ for some e. Suppose, towards a contradiction, that this is not the case. Then there must be a $u \geq k$ such that $|\Delta_u| = l$ and the leftmost expression in Δ_u is *not* of the form $e^{(b)}$ for any e. Note that (b)-r is the only rule apart from k_0 that can increase the length of the succedent (when read bottom-up). It follows that for no $w \geq u$ the leftmost expression in Δ_w is of the form $e^{(b)}$, contradicting the fact that (b)-r is applied infinitely often.

Now let $n \geq k$ be such that $|\Delta_n| = l$ and the leftmost expression of Δ_n is $e^{(b)}$. Since the rule (b)-r must at some point after Δ_n be applied to $e^{(b)}$, we may assume without loss of generality that $\Gamma_n \Rightarrow_{A_n} \Delta_n$ is the conclusion of an application of (b)-r. By the pigeonhole principle, there must be an $m > n$ such that $\Gamma_n \Rightarrow_{A_n} \Delta_n$ and $\Gamma_m \Rightarrow_{A_m} \Delta_m$ are the same sequents. We claim that these sequents satisfy the three properties above. Properties (i) and (ii) directly hold by construction. Property (iii) follows from the fact that Δ_n is of minimal length and has $e^{(b)}$ as leftmost expression.

Theorem 2. *Every valid sequent is provable in* SGKAT^∞.

Proof. Given a valid sequent, we do a bottom-up proof search with the following strategy. Throughout the procedure all leaves remain valid, in most cases by an appeal to invertibility.

1. Apply left logical rules as long as possible. If this stage terminates, it will be at a leaf of the form $\Gamma \Rightarrow_A \Delta$, where Γ is exposed. We then go to stage (2). If left logical rules remain applicable, we stay in this stage (1) forever and create an infinite branch.
2. Apply right logical rules until one of the following happens:
   (a) We reach a leaf at which no right logical rule can be applied. This means that the leaf must be a valid sequent of the form $\Gamma \Rightarrow_A \Delta$ such that Γ is exposed, and Δ is either exposed or begins with a test b such $A \upharpoonright b \neq A$. We go to stage (4).
   (b) If (a) does not happen, then at some point we must reach a valid sequent of the $\Gamma \Rightarrow_A e^{(b)}, \Delta$ which together with an ancestor satisfies properties (i) - (iii) of Lemma 11. In this case Lemma 10 is applicable. Hence we must be at a leaf of the form $\Gamma \Rightarrow_A e^{(b)}, \Delta$ such that $e^{(b)} \Rightarrow_A 0$ is valid. We then go to stage (3).

   Since at some point either (a) or (b) must be the case, stage (2) always terminates.
3. We are at a valid leaf of the form $\Gamma \Rightarrow_A e^{(b)}, \Delta$, where Γ is exposed. If $A = \emptyset$, we apply $\bot$. Otherwise, if $A \neq \emptyset$, we use the validity of $\Gamma \Rightarrow_A e^{(b)}, \Delta$ and $e^{(b)} \Rightarrow_A 0$ to find:

$$A \diamond [\![\Gamma]\!] \subseteq A \diamond [\![e^{(b)}]\!] \diamond [\![\Delta]\!] = \emptyset.$$

   We claim that $[\![\Gamma]\!] = \emptyset$. Indeed, suppose towards a contradiction that $\alpha x \in [\![\Gamma]\!]$. By the exposedness of Γ and item (i) of Lemma 3, we would have

$\beta x \in [\![\Gamma]\!]$ for some $\beta \in A$, contradicting the statement above. Therefore, the sequent $\Gamma \Rightarrow_{\mathsf{At}} 0$ is valid. We apply the rule k_0 and loop back to stage (1).
Stage (3) only comprises a single step and thus always terminates.

4. Let $\Gamma \Rightarrow_A \Delta$ be the current leaf. By construction $\Gamma \Rightarrow_A \Delta$ is valid, Γ is exposed, and Δ is either exposed or begins with a test b such that $A \upharpoonright b \neq A$. Note that only rules id, $\bot$, k, and k_0 can be applicable. By Lemma 9, at least one of them must be applicable. If id is applicable, apply id. If $\bot$ is applicable, apply $\bot$. If k is applicable, apply k and loop back to stage (1). Note that this application of k will have priority (cf. Definition 7), and is therefore invertible.
Finally, suppose that only k_0 is applicable. We claim that, by validity, the list Γ is not ϵ. Indeed, since A is non-empty, and Δ either begins with a primitive program p or a test b such that $A \upharpoonright b \neq A$, the sequent

$$\epsilon \Rightarrow_A \Delta$$

must be invalid. Hence Γ must be of the form p, Θ. We apply k_0, which has priority and thus is invertible, and loop back to stage (1).
Like stage (3), stage (4) only comprises a single step and thus always terminates.

We claim that the constructed derivation is fair. Indeed, every stage except stage (1) terminates. Therefore, every infinite branch must either eventually remain in stage (1), or pass through stages (3) or (4) infinitely often. Since k and k_0 shorten the antecedent, and no left logical rule other than (b)-l lengthens it, such branches must be fair.

Proposition 3. *The proof search procedure of Theorem 2 induces a* coNP *decision procedure for the language inclusion problem for* GKAT*-expressions.*

Proof. Given an invalid sequent, we will give a polynomial size certificate for its invalidity, which can be verified in polynomial time.

First note that applying the proof search procedure of Theorem 2 to an invalid sequent yields at least one branch which neither ends in an axiom, nor is fair for (b)-l. It is shown in the proof of Theorem 2 that every infinite branch constructed by the proof search procedure is fair for (b)-l, which means that this failing branch is finite.

Now we make a case distinction on whether our branch contains an application of k_0 which does not have priority, *i.e.* on whether it passes through stage 2(b) of the proof search procedure.

We first consider the more easy case, where it does not pass this stage. Define an *extended sequent* to be a sequent annotated by one of the four stages of of the proof search procedure, *i.e.* one of the form $\Gamma \Rightarrow^s_A \Delta$ where $s \in \{1, 2, 3, 4\}$. By Corollary 1 there are polynomially many extended sequents. We can now represent the failing branch of the proof search procedure as a list of extended sequents

$$\Gamma_0 \Rightarrow^{s_0}_{A_0} \Delta_0, \ldots, \Gamma_n \Rightarrow^{s_n}_{A_n} \Delta_n.$$

We will shrink this branch by deleting all segments between two identical extended sequents. More formally, we inductively define the following indices. First, k_0 is maximal such that $\Gamma_{k_0} \Rightarrow^{s_{k_0}}_{A_{k_0}} \Delta_{k_0} = \Gamma_0 \Rightarrow^{s_0}_{A_0} \Delta_0$. Second, if $k_i < n$, then k_{i+1} is maximal such that $\Gamma_{k_{i+1}} \Rightarrow^{s_{k_{i+1}}}_{A_{k_{i+1}}} \Delta_{k_{i+1}} = \Gamma_{k_i+1} \Rightarrow^{s_{k_i+1}}_{A_{k_i+1}} \Delta_{k_i+1}$.

Clearly there must be some m such that $\Gamma_{k_m} \Rightarrow^{s_{k_m}}_{A_{k_m}} \Delta_{k_m} = \Gamma_n \Rightarrow^{s_n}_{A_n} \Delta_n$. We consider the branch

$$\Gamma_{k_0} \Rightarrow^{s_{k_0}}_{A_{k_0}} \Delta_{k_0} \ldots, \Gamma_{k_m} \Rightarrow^{s_{k_m}}_{A_{k_m}} \Delta_{k_m}.$$

By construction this branch also arises in the proof search procedure applied to $\Gamma_0 \Rightarrow_{A_0} \Delta_0$. Moreover, every extended sequent in this branch is distinct, whence it is of polynomial size. It is also failing, because the original branch is failing. Hence it is a polynomially-sized certificate for the invalidity of $\Gamma_0 \Rightarrow_{A_0} \Delta_0$. This can be checked in polynomial time simply by checking that it indeed arises from the proof search procedure.

In case the original branch does pass stage 2(b), the argument is very similar, but we have to take care not to break the the justification for applying Lemma 10. Suppose that k_0 is applied without priority to the sequent $\Gamma_l \Rightarrow_{A_l} \Delta_l$ (note that there is at most one such application of k_0), then there is some $q < l$ such that $\Delta_q = \Delta_l$ and for every $q \leq i \leq l$ it holds that $\Gamma_i = \Gamma_l$ and that Δ_l is a final segment of Δ_i. The idea is now to apply the same shrinking procedure as before, but keeping both $\Gamma_q \Rightarrow_{A_q} \Delta_q$ and $\Gamma_l \Rightarrow_{A_l} \Delta_l$. Again, the resulting branch will be a polynomially-sized certificate for the invalidity of $\Gamma_0 \Rightarrow_{A_0} \Delta_0$. We leave it to the reader to work out the details.