A cyclic proof system for Guarded Kleene Algebra with Tests (full version)*

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

[1] Institute of Logic, Language and Computation, University of Amsterdam, The Netherlands janrooduijn@gmail.com
[2] Cornell University, Ithaca, NY, 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 allows for a non-well-founded sequent system whose set of regular proofs is complete with respect to the guarded language model. This is unlike the situation with Kleene Algebra, where hypersequents are required. Moreover, the decision procedure induced by proof search runs in NLOGSPACE, whereas that of Kleene Algebra is in PSPACE.

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

1 Introduction

Guarded Kleene Algebra with Test (GKAT) is the fragment of Kleene Algebra with Tests (KAT) comprised of the deterministic while programs. Those are the programs built up from sequential composition ($e \cdot f$), conditional branching (`if-`b`-then-`e`-else-`f) and loops (`while` b `do` e). For an introduction to KAT we refer the reader to [10]. The first papers focusing on the fragment of KAT that is nowadays called GKAT are Kozen's [11] and Kozen & Tseng's [12], where it is used to study the relative power of several programming constructs.

As GKAT is a fragment of KAT, it directly inherits a rich theory. It admits a language semantics in the form of *guarded strings* and for every expression there is a corresponding KAT-automaton. Already in [12] it was argued that GKAT expressions are more closely related to so-called *strictly deterministic automata*, where every state transition executes a primitive program. Smolka et al. significantly advanced the theory of GKAT in [22], by studying various additional semantics, identifying the precise class of strictly deterministic automata corresponding to GKAT-expressions (proving a *Kleene theorem*), giving a nearly linear decision procedure of the equivalence of GKAT-expressions, and studying its equational

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

axiomatisation. Since then GKAT has received considerable further attention, *e.g.* in [20,17,24,21].

One of the most challenging and intriguing aspects of GKAT is its proof theory. The standard equational axiomatisation of KAT from [10] does not simply restrict to GKAT, since a derivation of an expression that lies within the GKAT-fragment might very well contain expressions that lie outside of it. Moreover, the axiomatisation of KAT contains a least fixed point rule that relies on the equational definability of inclusion, which does not seem to be available in GKAT.

In [22], this problem is circumvented by introducing a custom equational axiomatisation for GKAT that uses a *unique* fixed point rule. While a notable result, this solution is still not entirely satisfactory. First, completeness is only proven under the inclusion of a variant of the unique fixed point rule that operates on entire systems of equations (this problem was recently addressed for the so-called *skip-free* fragment of GKAT in [21]). Moreover, even the ordinary, single-equation, unique fixed point rule contains a non-algebraic side-condition, analogous to the empty word property in Salomaa's axiomatisation of Kleene Algebra [18]. Because of this, a proper definition of 'a GKAT' is still lacking.

In recent years the proof theory of logic with fixed point operators (such as while-b-do-e) has seen increasing interest in *non-well-founded* proofs. In such proofs, branches need not to be closed by axioms, but may alternatively be infinitely deep. To preserve soundness, a progress condition is often imposed on each infinite branch, facilitating a soundness proof by infinite descent. In some cases non-well-founded proofs can be represented by finite trees with back-edges, which are then called *cyclic proofs*. See *e.g.* [5,13,4,2,8] for a variety of such approaches. Often, the non-well-founded proof theory of some logic is closely related to its corresponding automata theory. Taking the proof-theoretical perspective, however, can be advantageous because it is more fine-grained and provides a natural setting for establishing results such as interpolation [14,3], cut elimination [19,1], and completeness by proof transformation [23,6].

In [7], Das & Pous study the non-well-founded proof theory of Kleene Algebra, a close relative of GKAT (for background on Kleene Algebra we refer the reader to [9]). They show that a natural non-well-founded sequent system for Kleene Algebra is not complete when restricting to the subset of cyclic proofs. To remedy this, they introduce a *hypersequent* calculus, whose cyclic proofs *are* complete. They give a proof-search procedure for this calculus and show that it runs in PSPACE. Since deciding Kleene Algebra expressions is PSPACE-complete, their proof-search procedure induces an optimal decision procedure for this problem. In a follow-up paper together with Doumane, left-handed completeness of Kleene Algebra is proven by translating cyclic proofs in the hypersequent calculus to well-founded proofs in left-handed Kleene Algebra [6].

The goal of the present paper is to study the non-well-founded proof theory of GKAT. This is interesting in its own right, for instance because, as we will see, it has some striking differences with Kleene Algebra. Moreover, we hope it opens up new avenues for exploring the completeness of algebraic proof systems for GKAT, through the translation of our cyclic proofs.

Outline Our paper is structured as follows.

- In Section 2 we introduce preliminary material: the syntax of GKAT and its language semantics.
- Section 3 introduces our non-well-founded proof system SGKAT for GKAT.
- In Section 4 we show that (possibly infinitary) proofs in SGKAT are sound. That is, the interpretation of each derivable sequent - a GKAT-*inequality* - is true in the language model (which means that a certain *inclusion* of languages holds).
- In Section 5 we show that proofs are *finite-state*: each proof contains only finitely many distinct sequents. More precisely, by employing a more fine-grained analysis than in [7], we give a quadratic 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 cyclic proofs proves exactly the same sequents as the set of all non-well-founded proofs.
- Section 6 deals with completeness and complexity. We first use a proof-search procedure to show that SGKAT is complete: every sequent whose interpretation is valid in the language model, can be derived. We then show that this proof-search procedure runs in coNLOGSPACE. This gives an NLOGSPACE upper bound on the complexity of the language inclusion problem for GKAT-expressions.

Our contributions Our paper closely follows the treatment of Kleene Algebra in [7]. Nevertheless, we make the following original contributions:

- Structure of sequents: we devise a form of sequents bespoke to GKAT, by labelling the sequents by sets of atoms. This is similar to how the appropriate automata for GKAT are not simply KAT-automata. In contrast to Kleene Algebra, it turns out that we do not need to extend our sequents to hypersequents in order to obtain completeness for the fragment of cyclic proofs.
- Soundness argument: our modest contribution here is the notion of priority of rules and the fact that our rules are all invertible when they have priority. The soundness argument for finite proofs is, of course, slightly different, because our rules are different. (The step from the soundness of finite proofs, towards the soundness of infinite proofs, is completely analogous to that of [7].)
- Regularity: this concerns showing that every proof contains only finitely many distinct sequents. As in [7], our argument views each expression in a proof as a subexpression of an expression in the proof's root . A modest contribution is that our argument is made more formal by considering these expressions as nodes in a syntax tree. More importantly, the bound on the number of distinct cedents we obtain is sharper: where in [7] it is exponential in the size of the syntax tree, our bound is linear (yielding a quadratic bound on the number of sequents).
- Completeness: the structure of the argument is identical to that in [7], but the details differ due to the different rules and different type of sequents. This for instance shows in our proof of Lemma 9 (which is analogous to

Lemma 20 in [7]), where we make crucial use of the set of atoms annotating a sequent.
– Complexity: our complexity argument is necessarily different because it applies to a different system and is designed to give a different upper bound.

Due to space limitations several proofs are only sketched or omitted entirely. Full versions of these proofs can be found in the extended version of this paper [15].

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 [10]. 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 [22], 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

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

We state two simple facts that will be useful later on.

Lemma 1. *For any two languages L, K of guarded strings, and primitive program p, we have:*

(i) $L^{n+1} = L \diamond L^n$; *(ii)* $[\![p]\!] \diamond L = [\![p]\!] \diamond K$ *implies* $L = K$.

Remark 3. The fact that GKAT models deterministic programs is reflected in the fact that sets of guarded strings arising as 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 the reader to [22] for more details.

Remark 4. The language semantics of GKAT is the same as that of KAT (see [10]), 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.

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 latter.

Definition 1 (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.

Remark 5. As the system in [7] only deals with Kleene Algebra, it does not include tests. We choose the deal with the tests present in GKAT by augmenting each sequent by a set of atoms. This tucks away the Boolean content, as is usual in the GKAT literature, allowing us to omit propositional rules.

Definition 2 (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

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

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

Right logical rules

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

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

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. 1: 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 $\alpha_{n+1} \leq \overline{b}$. 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 6. 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 [22].

Given a set of atoms A and a test b, we write $A \upharpoonright b$ for $A \diamond [\![b]\!]$, *i.e.* the set of atoms $\{\alpha \in A : \alpha \leq b\}$. The rules of SGKAT are given in Figure 1. Importantly, the rules are always applied to the leftmost expression in a cedent. As a result, we have the following lemma, that later will be used in the completeness proof.

Lemma 2. *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 $\Gamma \Rightarrow_A \Delta$.*

Remark 7. 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 succedent Δ.

Remark 8. 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. 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 asymmetry will be introduced in Definition 3, with a condition on infinite branches that is sensitive to (b)-l but not to (b)-r.

Remark 9. The authors of [20] study a variant of GKAT that omits the so-called *early termination axiom*, which equates all programs that eventually fail. They give a denotational model of this variant in the form of certain kinds of trees. We conjecture that omitting the rule k_0 from our system will make it 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 3 (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. Not every SGKAT^∞-derivation is a proof. Consider for instance the following derivation, where $(\bullet)$ indicates that the derivation repeat itself.

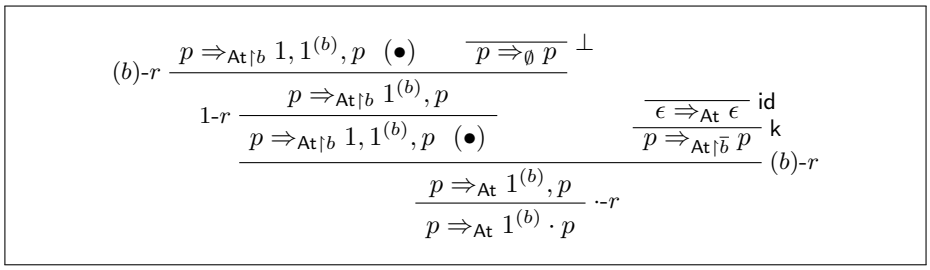


Fig. 2: An SGKAT^∞-derivation that is not a proof.

Example 5. Let $\Delta_1 := (p(cp +_b 1))^{(b)}$ and $\Delta_2 := cp +_b 1, \Delta_1$. The following proof Π_1 is an example SGKAT^∞-proof of the sequent of Example 3. We again use $(\bullet)$ to indicate that the proof repeats itself at this leaf and, for the sake of readability, omit branches that can be closed immediately by an application of $\bot$.

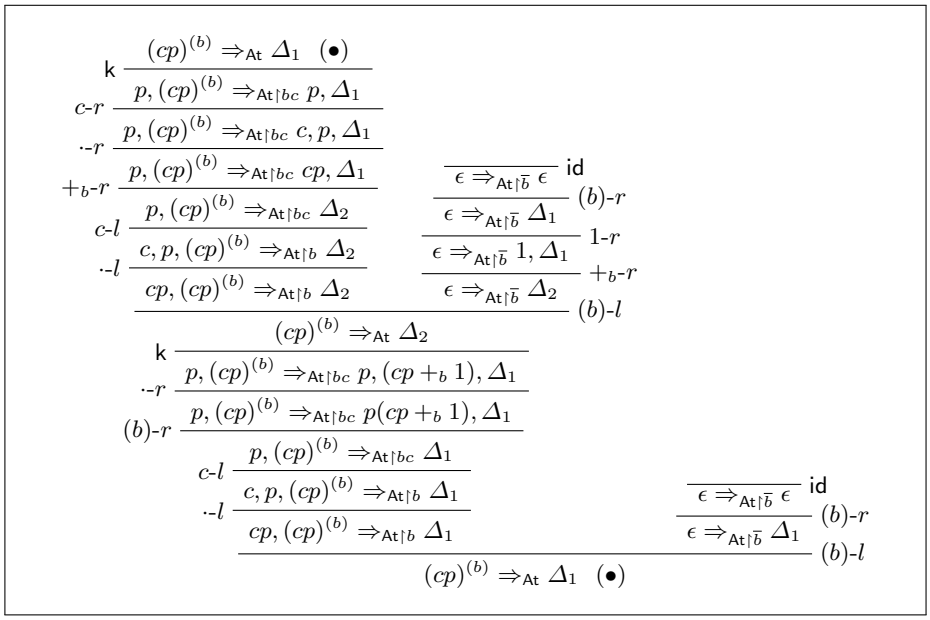


Fig. 3: The SGKAT^∞-proof Π_1.

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.

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

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 4 (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. The following easy lemma will be useful later on.

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. The following straightforward facts will be useful in the soundness proof.

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

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

We prioritise the rules of SGKAT in order of occurrence in Figure 1, 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. 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.

We say that a rule application *has priority* of there is no higher-priority rule with the same conclusion. Conveniently, the following proposition entails that every rule instance which has priority is invertible. This will aid our proof search procedure in Section 6 .

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.*

Proof (sketch). We treat two illustrative cases. For the rule $+_b$-r, we find

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

where the first equivalence holds due to Lemma 4.(ii), and the second due to $A \diamond [\![\Gamma]\!] = ([\![b]\!] \diamond A \diamond [\![\Gamma]\!]) \cup ([\![\bar{b}]\!] \diamond A \diamond [\![\Gamma]\!])$ and Lemma 4.(i).

The other rule we will treat is 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 $\bot$. In this case $A = \emptyset$, which means that the conlusion must be valid. Hence any application of k that does not have priority is vacuously sound. It need, however, not be 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

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

as required. The step marked by † is the following property of guarded languages: $[\![p]\!] \diamond L = [\![p]\!] \diamond K$ implies $L = K$.

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: $[e^{(b)}]^0 := \overline{b}$ and $[e^{(b)}]^{n+1} := be[e^{(b)}]^n$.

Lemma 5. *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,

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

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 6. $\mathsf{wwh}([e^{(b)}]^n, \Gamma) < \mathsf{wwh}(e^{(b)}, \Gamma)$ *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 $A \diamond [\![\Gamma]\!] \subseteq [\![\Delta]\!]$.*

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 5, we have that $[e^{(b)}]^n, \Gamma' \Rightarrow_{A'} \Delta'$ is provable for every n. Lemma 6 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.

5 Regularity

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.

The results of this section crucially depend on the fact that we are only applying rules to the leftmost expressions of cedents. Indeed, otherwise one could easily create infinitely many distinct sequents by simply unravelling the same while loop $e^{(b)}$ infinitely often.

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, and more importantly, we obtain a quadratic bound on the number of distinct sequents occurring in a proof, rather than an exponential one. In fact, we will show that the number of distinct antecedents (succedents) is *linear* in the size of the syntax tree of the antecedent (succedent) of the root. We will do this by showing that each leftmost expression of a cedent in the proof (given as node of the syntax tree of a root cedent) can only occur in the proof as the leftmost expresson of that *unique* cedent.

Definition 5. *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* ρ*, with label* $l_e(\rho) := e$*.*
- *If* $e = f_1 \circ f_2$ *where* $\circ = \cdot$ *or* $\circ = +_b$*, its syntax tree again has a root node* ρ *with label* $l_e(\rho) = e$*, and with two outgoing edges. The first edge connects* ρ *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* ρ *with label* $l_e(\rho) = e$*, but now with just one outgoing edge. This edge connects* ρ *to* (T_f, l_f)*.*

Definition 6. *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* $l_e(u_1), \ldots, l_e(u_n)$*.*

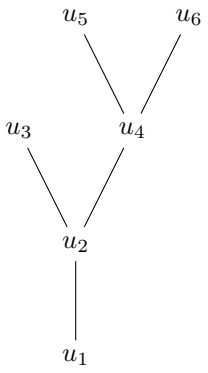
Given the leftmost expression of a cedent, we will now explicitly define the cedent that it must be the leftmost expression of.

Definition 7. *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* ρ *of* T_e*, we set* $\mathsf{tail}(\rho)$ *to be the empty list* ϵ*.*
- *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 6. 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 the main result of this section: every leftmost expression is the leftmost expression of a unique cedent.

Lemma 7. *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.

We prove this by induction on the syntax tree of e. For the root ρ, we have $\mathsf{tail}(\rho) = \epsilon$, 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 bottom-up induction on π. For the base case, note that e and f are realisations of the roots of their respective syntax trees. Such a root ρ is tail-generated, since $\rho = \rho, \epsilon = \rho, \mathsf{tail}(\rho)$. The induction step follows by direct inspection of the rules of SGKAT.

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 quadratic 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 $e_1 \cdots e_n$.

Recall that a non-well-founded tree is *regular* if it contains only finitely many pairwise non-isomorphic subtrees. The following corollary follows by a standard argument in the literature (see *e.g* [16, Corollary I.2.23]).

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 $\mathsf{NLOGSPACE}$ decision procedure for the language inclusion problem of `GKAT` expressions. The material in this section is again inspired by [7], but requires several modifications to treat the tests present in `GKAT`.

First note the following fact.

Lemma 8. *Any valid sequent is the conclusion of some rule application.*

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

Lemma 9. *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 8). 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 10. *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.

Since the proof is finite-state (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.

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

Theorem 2 (Completeness). *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 10. In this case Lemma 9 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 8, 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 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).
Similarly to stage (3), stage (4) only comprises a single step and thus always terminates.

We claim that the constructed derivation is fair for (b)-l. 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.

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 runs in* $\mathsf{coNLOGSPACE}$*. Hence proof search, and thus also the language inclusion problem for* `GKAT`*-expressions, is in* $\mathsf{NLOGSPACE}$*.*

Proof (sketch). Assume without loss of generality that the initial sequent is of the form $e \Rightarrow_A f$. We non-deterministically search for a failing branch, at each iteration storing only the last sequent. By Lemma 7 this can be done by storing two pointers to, respectively, the syntax trees T_e and T_f, together with a set of atoms. The loop check of stage (2) can be replaced by a counter. Indeed, stage (2) must always hit a repetition after $|\mathsf{At}| \cdot |T_f|$ steps, where m is the number of nodes in the syntax tree. After this repetition there must be a continuation that reaches a repetition to which Lemma 9 applies before this stage has taken $2 \cdot |\mathsf{At}| \cdot |T_f|$ steps in total. Finally, a global counter can be used to limit the depth of the search. Indeed, a failing branch needs at most one repetition (in stage (2), to which k_0 is applied) and all other repetitions can be cut out. Hence if there is a failing branch, there must be one of size at most $4 \cdot |T_e| \cdot |\mathsf{At}| \cdot |T_f|$.

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 an NLOGSPACE decision procedure, whereas that of Kleene Algebra is in PSPACE.

The most natural 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 [22, Figure 1]), 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 a similar 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 NLOGSPACE, but do not know whether it is optimal.

Finally, it would be interesting to verify the conjecture in Remark 9 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 wants to acknowledge Sonia Marin, who some years ago proposed a similar master project at UCL. We moreover thank the reviewers for their helpful comments, in particular for pointing out that our complexity result could be sharpened. 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. Acclavio, M., Curzi, G., Guerrieri, G.: Infinitary cut-elimination via finite approximations. In: 32nd Annual Conference on Computer Science Logic, CSL. LIPIcs, vol. 288, pp. 8:1–8:19. Schloss Dagstuhl (2024)
2. Afshari, B., Enqvist, S., Leigh, G.E.: Cyclic proofs for the first-order μ-calculus. Logic Journal of the IGPL (2022)

3. Afshari, B., Leigh, G.E., Turata, G.M.: Uniform interpolation from cyclic proofs: The case of modal mu-calculus. In: 30th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX. Lecture Notes in Computer Science, vol. 12842, pp. 335–353. Springer (2021)
4. 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)
5. 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)
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. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation **110**(2), 366–390 (1994)
10. Kozen, D.: Kleene algebra with tests. ACM Transactions on Programming Languages and Systems **19**(3), 427–443 (1997)
11. 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)
12. 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)
13. 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)
14. Marti, J., Venema, Y.: Focus-style proof systems and interpolation for the alternation-free μ-calculus (2021), arXiv preprint 2103.01671
15. Rooduijn, J., Kozen, D., Silva, A.: A cyclic proof system for Guarded Kleene Algebra with Tests (full version) (2024), available at `alexandrasilva.org/files/ijcar24-full.pdf`
16. Rooduijn, J.: Fragments & Frame Classes. Ph.D. thesis, University of Amsterdam (2024)
17. Rozowski, W., Kappé, T., Kozen, D., Schmid, T., Silva, A.: Probabilistic guarded KAT modulo bisimilarity: Completeness and complexity. In: 50th International Colloquium on Automata, Languages, and Programming, ICALP. LIPIcs, vol. 261, pp. 136:1–136:20. Schloss Dagstuhl (2023)
18. Salomaa, A.: Two complete axiom systems for the algebra of regular events. Journal of the ACM **13**(1), 158–169 (1966)
19. Savateev, Y., Shamkanov, D.S.: Cut-elimination for the modal Grzegorczyk logic via non-well-founded proofs. In: 24th International Workshop on Logic, Language,

Information, and Computation WoLLIC. Lecture Notes in Computer Science, vol. 10388, pp. 321–335. Springer (2017)
20. 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)
21. Schmid, T., Kappé, T., Silva, A.: A complete inference system for skip-free guarded Kleene algebra with tests. In: 32nd European Symposium on Programming, ESOP. Lecture Notes in Computer Science, vol. 13990, pp. 309–336. Springer (2023)
22. 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)
23. Sprenger, C., Dam, M.: On the structure of inductive reasoning: Circular and tree-shaped proofs in the μ-calculus. In: 6th International Conference on Foundations of Software Science and Computation Structures, FOSSACS. Lecture Notes in Computer Science, vol. 2620, pp. 425–440. Springer (2003)
24. Zetzsche, S., Silva, A., Sammartino, M.: Guarded Kleene algebra with tests: Automata learning. In: Proceedings of the 38th Conference on the Mathematical Foundations of Programming Semantics, MFPS. EPTICS, vol. 1. EpiSciences (2022)

A Full proofs

This appendix contains full versions of all the proofs that were either omitted or sketched in the body of the paper.

A.2 ... of Section 2

Lemma 1. *For any two languages L, K of guarded strings, and primitive program p, we have:*

(i) $L^{n+1} = L \diamond L^n$; *(ii)* $[\![p]\!] \diamond L = [\![p]\!] \diamond K$ *implies* $L = K$.

Proof. (i). 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.

(ii). 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 the required:

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

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:*

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

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

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

as required.

Proposition 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.*

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

$$\begin{aligned} A \diamond [\![e +_b f, \Gamma]\!] &= A \diamond [\![e +_b f]\!] \diamond [\![\Gamma]\!] && \text{(sequent int.)} \\ &= A \diamond (([\![b]\!] \diamond [\![e, \Gamma]\!]) \cup ([\![\bar{b}]\!] \diamond [\![f, \Gamma]\!])) && \text{(Lem. 4.(ii))} \\ &= (A \diamond [\![b]\!] \diamond [\![e, \Gamma]\!]) \cup (A \diamond [\![\bar{b}]\!] \diamond [\![f, \Gamma]\!]) && \text{(distrib.)} \\ &= (A \upharpoonright b \diamond [\![e, \Gamma]\!]) \cup (A \upharpoonright \bar{b} \diamond [\![f, \Gamma]\!]) && \text{(Lem. 4.(i))} \end{aligned}$$

($+_b$-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.(ii), and the second due to $A \diamond [\![\Gamma]\!] = ([\![b]\!] \diamond A \diamond [\![\Gamma]\!]) \cup ([\![\bar{b}]\!] \diamond A \diamond [\![\Gamma]\!])$ and Lemma 4.(i).

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

$$\begin{aligned} A \diamond [\![e^{(b)}, \Gamma]\!] &= A \diamond [\![e^{(b)}]\!] \diamond [\![\Gamma]\!] && \text{(sequent int.)} \\ &= A \diamond (([\![b]\!] \diamond [\![e, e^{(b)}, \Gamma]\!]) \cup ([\![\bar{b}]\!] \diamond [\![f, \Gamma]\!])) && \text{(Lem. 4.(iii))} \\ &= (A \diamond [\![b]\!] \diamond [\![e, e^{(b)}, \Gamma]\!]) \cup (A \diamond [\![\bar{b}]\!] \diamond [\![f, \Gamma]\!]) && \text{(distrib.)} \\ &= (A \upharpoonright b \diamond [\![e, e^{(b)}, \Gamma]\!]) \cup (A \upharpoonright \bar{b} \diamond [\![f, \Gamma]\!]) && \text{(Lem. 4.(i))} \end{aligned}$$

((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

$$\begin{aligned} A \diamond [\![p, \Gamma]\!] \subseteq [\![p, \Delta]\!] &\Leftrightarrow A \diamond [\![p]\!] \diamond [\![\Gamma]\!] \subseteq [\![p]\!] \diamond [\![\Delta]\!] && \text{(seq. int.)} \\ &\Leftrightarrow \alpha \diamond [\![p]\!] \diamond [\![\Gamma]\!] \subseteq [\![p]\!] \diamond [\![\Delta]\!] && (\alpha \in A, \text{Lem. 3}) \\ &\Leftrightarrow [\![p]\!] \diamond [\![\Gamma]\!] \subseteq [\![p]\!] \diamond [\![\Delta]\!] && \text{(Lem. 3)} \\ &\Leftrightarrow [\![\Gamma]\!] \subseteq [\![\Delta]\!], && \text{(Lem. 1.(ii))} \end{aligned}$$

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 5. *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.

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

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 6. $\mathsf{wwh}([e^{(b)}]^n, \Gamma) < \mathsf{wwh}(e^{(b)}, \Gamma)$ *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 $\mathsf{wwh}([e^{(b)}]^n, \Gamma) < \mathsf{wwh}(e^{(b)}, \Gamma)$.

A.6 ... of Section 6

Lemma 8. *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.

Proposition 2. *The proof search procedure of Theorem 2 runs in* coNLOGSPACE*. Hence proof search, and thus also the language inclusion problem for* GKAT*-expressions, is in* NLOGSPACE*.*

Proof. The exact coNLOGSPACE procedure goes as follows. Without loss of generality we assume that the input is a sequent of the form $e \Rightarrow_A f$, given as a triple consisting of the syntax tree T_e of e, the set of atoms A, and the syntax tree of T_f of f. During the procedure we store:

- One pointer to T_e, initialised at the root.
- One pointer to T_f, initialised at the root.
- A set of atoms, initially A.

Note that by Lemma 7 these data suffice to completely describe a sequent. Moreover, we store

- A counter storing how many steps are taken, initially 1.
- A variable $s \in \{1, 2, 3, 4\}$ storing at which of the four stages of the procedure we currently are.

The idea is to non-deterministically apply the proof search procedure of Theorem 2. Note, however, that the fact that we are only storing a single sequent prevents us from performing the loop check that happens in stage (2). As a remedy, we additionally store

- A second counter, counting how many steps are taken in stage (2) (always resetting at the beginning of stage (2)).

At each iteration we apply the rule dictated by the proof search procedure, non-deterministically choosing a premiss and updating the data accordingly. Our non-deterministic procedure searches for a failing branch (*i.e.* one to which no rule can be applied). Whenever the first counter reaches $4 \cdot |T_e| \cdot |\mathsf{At}| \cdot |T_f|$, it gives up the search. Moreover, whenever the second counter reaches $2 \cdot |\mathsf{At}| \cdot |T_f|$, it applies k_0 (this replaces the loop check of stage (2)).

We claim that this procedure succeeds in finding a failing branch if it exists. We first justify the application of k_0 when the second counter hits its limit. Consider a list of sequents $\Gamma \Rightarrow_{A_0} \Delta_0, \ldots, \Gamma \Rightarrow_{A_n} \Delta_n$ in stage (2), where $n = |\mathsf{At}| \cdot |T_f|$. By the pidgeonhole principle, this list must repeat some sequent. Moreover, because the only rule that grows succedents is (b)-r, the segment between the two repetitions must contain a succedent of the form $e^{(b)}, \Delta$ of minimal length. But then there is a variant of this branch where the succedent $e^{(b)}, \Delta$ is repeated and satisfies the conditions of Lemma 9.

Finally, we argue that if there is a failing branch, then there is one of length smaller than the limit of the first counter. This follows from the fact that the second counter can only hit its limit once (because afterwards the succedent becomes trivial) and every repetition that happens before or after the corresponding stage (2) can be cut out of the branch.