Kleene Algebra with Commutativity Conditions Is Undecidable

Arthur Azevedo de Amorim
Rochester Institute of Technology, NY, USA

Cheng Zhang[1]
University College London, UK

Marco Gaboardi
Boston University, MA, USA

Abstract

We prove that the equational theory of Kleene algebra with commutativity conditions on primitives (or atomic terms) is undecidable, thereby settling a longstanding open question in the theory of Kleene algebra. While this question has also been recently solved independently by Kuznetsov, our results hold even for weaker theories that do not support the *induction axioms* of Kleene algebra.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Regular languages

Keywords and phrases Kleene Algebra, Hypotheses, Complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2025.36

Funding *Arthur Azevedo de Amorim*: National Science Foundation Grant No. 2314323.
Cheng Zhang: National Science Foundation Award No. 1845803 and No. 2040249.
Marco Gaboardi: National Science Foundation Award No. 1845803 and No. 2040249.

Acknowledgements We want to thank Todd Schmid and Alexandra Silva for the valuable discussion during this work; and also acknowledge the useful feedback provided by the anonymous reviewers of CSL2025. Finally, we want to thank Stepan Kuznetsov for his detailed and valuable feedback on this work.

1 Introduction

Kleene algebra generalizes the algebra of regular languages while retaining many of its pleasant properties, such as having a decidable equational theory. This enables numerous applications in program verification, by translating programs and specifications into Kleene-algebra terms and then checking these terms for equality. This idea has proved fruitful in many domains, including networked systems [1, 7], concurrency [9, 11, 12], probabilistic programming [18, 19], relational verification [4], program schematology [2], and program incorrectness [21].

Many applications require extending Kleene algebra with other axioms. A popular extension is adding commutativity conditions $e_1 e_2 = e_2 e_1$, which state that e_1 and e_2 can be composed in any order. In terms of program analysis, e_1 and e_2 correspond to commands of a larger program, and commutativity ensures that their order does not affect the final output. Such properties have been proven useful for relational reasoning [4] and concurrency [6].

Unfortunately, such extensions can pose issues for decidability. In particular, even the addition of equations of the form $xy = yx$, where x and y are primitives, can make the equivalence of two *regular languages* given by Kleene algebra terms [14, 8, 5] undecidable – in fact, Π^0_1-complete [15], or equivalent to the complement of the halting problem.

[1] Work performed at Boston Univeristy

Despite this negative result, it was still unknown whether we could decide such equations in *arbitrary* Kleene algebras – or, equivalently, decide whether an equation can be derived solely from the Kleene-algebra axioms. Indeed, since the set of Kleene-algebra equations is generated by finitely many clauses, it is recursively enumerable, or Σ^0_1. Since a set cannot be simultaneously Σ^0_1 and Π^0_1-complete, the problem of deciding equations under commutativity conditions for all regular languages is not the same as the problem of deciding such equations for all Kleene algebras. There must be equations that are valid for all regular languages, but not for arbitrary Kleene algebras. Nevertheless, the question of decidability of Kleene-algebra equations with commutativity conditions remained open for almost 30 years [15].

This paper settles this question *negatively*, proving that this problem is undecidable. In fact, undecidability holds even for weaker notions of Kleene algebra that do not validate its *induction axioms*, which are needed to prove many identities involving the iteration operation. At a high level, our proof works as follows. Given a machine M and an input x, we define an inequality between Kleene algebra terms with the following two properties: (1) if M halts on x and accepts, the inequality holds, but (2) if M halts on x and rejects, the inequality does not hold. If such inequalities were decidable, we would be able to computationally distinguish these two scenarios, which is impossible by diagonalization.

On Kuznetsov's Undecidability Proof

As we were finishing this paper, we learned that the question of undecidability had also been settled independently by Kuznetsov in recent work [17]. Though our techniques overlap, there are two noteworthy differences between the two proofs. On the one hand, Kuznetsov's proof uses the induction axioms of Kleene algebra, so it applies to fewer settings. On the other hand, Kuznetsov was able to prove that the equational theory of Kleene algebra with commutativity conditions is, in fact Σ^0_1-complete, by leveraging *effective inseparability*, a standard notion of computability theory. After learning about Kuznetsov's work, we could adapt his argument to derive completeness in our more general setting as well, so this paper can be seen as a synthesis of Kuznetsov's work and our own.

Structure of the paper

In Section 2, we recall basic facts about Kleene algebra, and introduce a framework for stating the problem of equations modulo commutativity conditions using category theory.

In Section 3, we present the core of our undecidability proof. We use algebra terms to model the transition relation of an abstract machine, and construct a set of inequalities that allows us to tell whether a machine accepts a given input or not. If we could decide such inequalities, we would be able to distinguish two effectively inseparable sets, which would lead to a contradiction. This argument hinges on a *completeness result* (Theorem 16), which guarantees that, if a certain machine accepts an input, then a corresponding inequality holds.

In Section 4, we prove that an analog of the completeness result holds for a large class of relations that can be represented with terms, provided that they satisfy a technical condition that allows us to reason about the image of a set by a relation.

In Section 5, we develop techniques to prove that the machine transition relation satisfies the required technical conditions for completeness. In Section 5.1, we show how we can view Kleene algebra terms as automata, proving an *expansion lemma* (Lemma 27) that guarantees that most terms can be expanded so that all of its matched strings bounded by some maximum length can be identified. This framework generalizes the usual definitions of derivative on Kleene algebra terms, but does not rely on the induction axioms of Kleene algebra. In Section 5.2, we show how we can refine the expansion lemma when terms have *bounded-output*, which, roughly speaking, means such terms represent relations that map a string to only finitely many next strings. We prove that the transition relation satisfies these technical conditions (Section 5.3), which concludes the undecidability proof.

We conclude in Section 6, providing a detailed comparison between our work and the independent work of Kuznetsov [17].

2 Kleene Algebra and Commutable Sets

To set the stage for our result, we recall some basic facts about Kleene algebra and establish some common notation that we will use throughout the paper. We also introduce a notion of *commutable set*, which we will use to define algebras with commutativity conditions.

A (left-biased) *pre-Kleene algebra* is an idempotent semiring X equipped with a star operation. Spelled out explicitly, this means that X has operations

$$0 : X \qquad 1 : X$$
$$(-) + (-) : X \times X \to X \qquad (-) \cdot (-) : X \times X \to X \qquad (-)^* : X \to X,$$

which are required to satisfy the following equations:

$$1 \cdot x = x \cdot 1 = x \qquad 0 \cdot x = x \cdot 0 = 0 \qquad x \cdot (y \cdot z) = (x \cdot y) \cdot z$$
$$0 + x = x \qquad x + y = y + x \qquad x + (y + z) = (x + y) + z$$
$$x \cdot (y + z) = x \cdot y + x \cdot z \qquad (x + y) \cdot z = x \cdot z + y \cdot z \qquad x^* = 1 + x \cdot x^*,$$

where the last rule $x^* = 1 + x \cdot x^*$ is named "left unfolding". A pre-Kleene algebra carries the usual ordering relation on idempotent monoids: $x \leq y$ means that $y + x = x$. A *Kleene algebra* is a pre-Kleene algebra that satisfies the following properties:

$$xy \leq y \Rightarrow x^*y \leq y \qquad xy \leq x \Rightarrow xy^* \leq x,$$

dubbed left and right induction. A $*$-*continuous Kleene algebra* is a pre-Kleene algebra where, for all p, q and r, $\sup_{n \geq 0} pq^n r$ exists and is equal to pq^*r. Every $*$-continuous algebra satisfies the induction axioms, so it is, in fact, a Kleene algebra.

▶ **Example 1.** Though many pre-Kleene algebras that we'll consider are actually proper Kleene algebras, the two theories do not coincide. The following algebra, adapted from Kozen [13], validates all the pre-Kleene algebra axioms, but not the induction ones. The carrier set of the algebra is $\mathbb{N} + \{\bot, \top\}$, ordered by posing $\bot \leq n \leq \top$ for all $n \in \mathbb{N}$. The addition operation computes the maximum of two elements. Multiplication is defined as:

$$x \cdot \bot \triangleq \bot \cdot x \triangleq \bot \qquad x \cdot \top \triangleq \top \cdot x \triangleq \top \text{ when } x \neq \bot \qquad x \cdot y \triangleq x +_{\mathbb{N}} y$$

where $+_{\mathbb{N}}$ is the usual addition operation on natural numbers. The neutral elements of addition and multiplication are respectively $\bot$ and 0. The star operation is defined as follows:

$$x^* \triangleq \begin{cases} 0 & \text{if } x = \bot \\ \top & \text{otherwise} \end{cases}$$

We can verify the unfolding rule by case analysis:

$$\text{when } x = \bot\text{:} \quad \bot^* = 0 = \max(0, \bot) = \max(0, \bot \cdot 0) = \max(0, \bot \cdot \bot^*);$$
$$\text{when } x \neq \bot\text{:} \quad x^* = \top = \max(0, \top) = \max(0, x \cdot \top) = \max(0, x \cdot x^*).$$

However, this algebra is not a proper Kleene algebra, because it doesn't satisfy $(x^*)^* = x^*$ (which must hold in any Kleene algebra). Indeed, $(\bot^*)^* = 0^* = \top \neq 0 = \bot^*$.

▶ Remark 2. Weak Kleene algebras [16] are algebraic structures that sit between proper Kleene algebras and pre-Kleene algebras. They need not validate the induction axioms of Kleene algebra, but satisfy more rules than just left unfolding – in particular, $(x^*)^* = x^*$. Thus, Example 1 also shows that the theory of pre-Kleene algebras is strictly weaker than that of weak Kleene algebras.

Let X and Y be pre-Kleene algebras. A *morphism* of type $X \to Y$ is a function $f : X \to Y$ that commutes with all the operations. This gives rise to a series of categories $\mathsf{KA}^* \subset \mathsf{KA} \subset \mathsf{preKA}$ of $*$-continuous algebras, Kleene algebras, and pre-Kleene algebras. Each category is a strict full subcategory of the next one – strict because some pre-Kleene algebras are not Kleene algebras (Example 1) and because some Kleene algebras are not $*$-continuous [13].

The prototypical example of Kleene algebra is given by the set $\mathcal{L}X$ of regular languages over some alphabet X. In program analysis applications, a regular language describes the possible traces of events performed by some system. We use the multiplication operation to represent the sequential composition of two systems: if two components produce traces t_1 and t_2, then their sequential composition produces the concatenated trace t_1t_2, indicating that the actions of the first component happen first. Thus, by checking if two regular languages are equal, we can assert that the behaviors of two programs coincide. When X is empty, $\mathcal{L}X$ is isomorphic to the booleans $\mathbb{2} \triangleq \{0 \leq 1\}$. The addition operation is disjunction, the multiplication operation is conjunction, and the star operation always outputs 1. This Kleene algebra is the initial object in all three categories preKA, KA and KA^*.

The induction property of Kleene algebra allows us to derive several useful properties for terms involving the star operation. For example, they imply that the star operation is monotonic, a *right-unfolding rule* $x^* = 1 + x^*x$, and also that $x^*x^* = x^*$. This means that many of intuitions about regular languages carry over to Kleene algebra. Unfortunately, when working with pre-Kleene algebras, most of these results cannot be directly applied, making reasoning trickier. In practice, we can only reason about properties of the star operation that involve a finite number of uses of the left-unfolding rule. Dealing with this limitation is at the heart of the challenges we will face when proving our undecidability result.

2.1 Commuting conditions

Sometimes, we would like to reason about a system where two actions can be reordered without affecting its behavior. For example, we might want to say that a program can perform assignments to separate variables in any order, or that actions of separate threads can be executed concurrently. To model this, we can work with algebra terms where some elements can be composed in any order. As we will see, unfortunately, adding such hypotheses indiscriminately can lead to algebras with an undecidable equational theory. The notion of *commutable set*, which we introduce next, allows us to discuss such hypotheses in generality.

▶ **Definition 3.** *A* commuting relation *on a set X is a reflexive symmetric relation. A* commutable set *is a carrier set endowed with a commuting relation $\sim$. We say that two elements x and y commute if $x \sim y$. A commutable set is* commutative *if all elements commute; it is* discrete *if the commuting relation is equality. A morphism of commutable sets is a function that preserves the commuting relation, which leads to a category* Comm*. A* commutable subset *of a commutable set X is a commutable set Y whose carrier is a subset of X, and whose commuting relation is the restriction of $\sim$ to Y. We'll often abuse notation and treat a subobject $Y \hookrightarrow X$ as a commutable subset if its image in X is a commutable subset.*

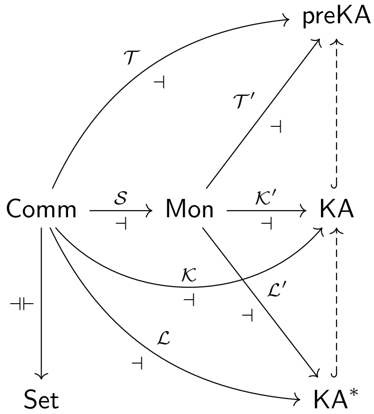
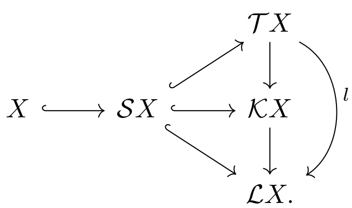


Figure 1 Algebraic constructions on commutable sets.

Given a commutable set, we have various ways of building algebraic structures, which can be summarized in the diagram of Figure 1 (which is commutative, except for the dashed arrows). The right-pointing arrows, marked with a $\dashv$, denote free constructions, in the sense that they have right adjoints that forget structure. The first construction, $\mathcal{S}$, is a functor from Comm to the category Mon of monoids and monoid morphisms. It maps a commutable set X to the monoid $\mathcal{S}X$ of strings over X, where we equate two strings if they can be obtained from each other by swapping adjacent elements that commute in X. The monoid operation is string concatenation, and the neutral element is the empty string. The corresponding right adjoint views a monoid Y as a commutable set where $x \sim y$ if and only if $xy = yx$.

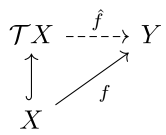
Another group of constructions extends a monoid X with the other Kleene algebra operations, and quotient the resulting terms by the equations we desire. For example, the elements of $\mathcal{T}'X$ are terms formed with Kleene algebra operations, where we identify the monoid operation with the multiplication operation of the pre-Kleene algebra, and where we identify two terms if they can be obtained from each other by applying the pre-Kleene algebra equations. The construction $\mathcal{K}'$ is obtained by imposing further equations on terms, while $\mathcal{L}'$ is given by the algebra of regular languages over a monoid [15], which we'll define soon. The right adjoints of these constructions view an algebra as a multiplicative monoid. By composing these constructions with $\mathcal{S}$, we obtain free constructions $\mathcal{T}$, $\mathcal{K}$ and $\mathcal{L}$ that turn any commutable set into some Kleene-algebra-like structure.

Being a free construction means, in particular, that we can embed the elements of a commutable set X into $\mathcal{S}X$, $\mathcal{T}X$, $\mathcal{K}X$ and $\mathcal{L}X$, as depicted in this commutative diagram:



By abuse of notation, we'll usually treat X as a proper subset of the free algebras. The vertical arrows take the elements of some algebra and impose the additional identities required by a stronger algebra. The composite l computes the *language interpretation* of a term, and will play an important role in our development, as we will see.

Free constructions, like $\mathcal{T}$, also allow us to define a morphism out of an algebra $\mathcal{T}X$ simply by specifying how the morphism acts on X. In other words, if $f : X \to Y$ is a morphism mapping a commutable set X to a pre-Kleene algebra Y, there exists a unique algebra morphism $\hat{f} : \mathcal{T}X \to Y$ such that the following diagram commutes:



Since f and $\hat{f}$ correspond uniquely to each other, we will not bother distinguishing between the two. We'll employ similar conventions for other left adjoints such as $\mathcal{S}$ or $\mathcal{L}$.

The last construction on Figure 1 allows us to turn any commutable set into a plain set by forgetting its commuting relation. This construction has both a left and a right adjoint: the right adjoint views a set as a commutative commutable set, by endowing it with the total relation; the left adjoint views a set as a discrete commutative set, by endowing it with the equality relation. By turning a set into a commutable set, discrete or commutative, and then building an algebra on top of that commutable set, we are able to express the usual notions of free algebra over a set, or of a free algebra where all symbols are allowed to commute.

▶ Remark 4 (Embedding algebras). We introduce some notation for embedding algebras into larger ones. Suppose that X is a commutable set and $Y \subseteq X$ is a commutable subset. By functoriality, this inclusion gives rise to morphisms of algebras of types $\mathcal{S}Y \to \mathcal{S}X$, $\mathcal{T}Y \to \mathcal{T}X$, etc. These morphisms are all injective, because they can be inverted: we can define a projection π_Y that maps $x \in X$ to itself, if $x \in Y$, or to 1, if $x \notin Y$. This definition is valid because, since Y inherits the commuting relation from X, and since 1 commutes with everything in $\mathcal{S}Y$, $\mathcal{T}Y$, etc., we can check that the commuting relation in X is preserved.

2.2 Regular Languages

If X is a monoid, we can view its power set $\mathcal{P}X$ as a $*$-continuous algebra equipped with the following operations:

$$0 \triangleq \emptyset \qquad 1 \triangleq \{1\}$$

$$A + B \triangleq A \cup B \qquad A \cdot B \triangleq \{xy \mid x \in A, y \in B\} \qquad A^* \triangleq \bigcup_{n \in \mathbb{N}} A^n.$$

The $*$-continuous algebra $\mathcal{L}'X$ of regular languages over X is the smallest subalgebra of $\mathcal{P}X$ that contains the singletons. The language interpretation $l : \mathcal{T}X \to \mathcal{L}X$ is the morphism that maps a symbol $x \in X$ to the singleton set $\{x\}$. This allows us to view a term as a set of strings over X, and we will often do this to simplify the notation; for example, if e is a term, we'll write $X \subseteq e$ to mean $X \subseteq l(e)$. Indeed, as the next few results show, it is often safe for us to view a term as a set of strings.

▶ **Theorem 5.** *If $s \in \mathcal{S}X$ is a string and $e \in \mathcal{T}X$ a term, then $s \leq e$ is equivalent to $s \in l(e)$.*

▶ **Theorem 6.** *We say that $e \in \mathcal{T}X$ is* finite *if its language $l(e)$ is. In this case, then $e = \sum l(e)$.*

▶ **Corollary 7.** *The language interpretation l is injective on finite terms: if $l(e_1) = l(e_2)$ and both e_1 and e_2 are finite, then $e_1 = e_2$.*

These results allow us to unambiguously view a finite set of strings over X as a finite term over X. We'll extend this convention to other sets: Y is a (pre-)Kleene algebra, we are going to view a finite set of elements $A \subseteq Y$ as the element $\sum_{a \in A} a \in Y$.

▶ **Corollary 8.** *For every term $e \neq 0$, there exists some string s such that $s \leq e$.*

One useful property of Kleene algebra is that, if X is finite, then $X^* \in \mathcal{K}X$ is the top element of the algebra. This result is generally not valid for $\mathcal{T}X$, but the following property will be good enough for our purposes.

▶ **Theorem 9.** *If X is finite and $e \in \mathcal{T}X$ is finite, then $eX^* \leq X^*$.*

To conclude our analogy between languages and terms, as far as $*$-continuous algebras are concerned, elements of $\mathcal{T}X$ are just as good as their corresponding languages – if Y is $*$-continuous, then every morphism of algebras $f : \mathcal{T}X \to Y$ can be factored through the language interpretation l:

$$\begin{array}{ccc} X & \hookrightarrow & \mathcal{L}X \\ \downarrow & \overset{l}{\nearrow} & \downarrow \\ \mathcal{T}X & \xrightarrow[f]{} & Y. \end{array}$$

This has some pleasant consequences. For example, let $[-]_0 : \mathcal{T}X \to \mathbb{2}$ be the morphism that maps every $x \in X$ to 0. Then $[e]_0 = 1$ if and only if $1 \leq e$. Indeed, this morphism must factor through $\mathcal{L}X$. The corresponding factoring $\mathcal{L}X$ must map any nonempty string to 0 and the empty string to 1. Thus, $[e]_0 = 1$ if and only if $1 \in l(e)$, which is equivalent to $1 \leq e$.

3 Undecidability via Effective Inseparability

Our undecidability result works by using pre-Kleene algebra equations to encode the execution of *two-counter machines*. Roughly speaking, a two-counter machine M is an automaton that has a control state and two counters. The machine can increment each counter, test if their values are zero, and halt. Two-counter and Turing machines are equivalent in expressive power: any two-counter machine can simulate the execution of a Turing machine, and vice versa; see Hopcroft et al. [10, §8.5.3,§8.5.4] for an idea of how this simulation works. In particular, given a Turing machine M, there exists a two-counter machine that halts on every input where M halts, and yields the same output for that input. For this reason, we'll tacitly use two-counter machines to implement computable functions in what follows.

▶ **Definition 10.** *A* two-counter *machine is a tuple $M = (Q_M, \dot{q}, \iota)$, where Q_M is a finite set of* control states, *$\dot{q} \in Q_M$ is an initial state, and $\iota : Q_M \to I_M$ is a transition function. The set I_M is the set of* instructions *of the machine, defined by the grammar*

$$I_M \ni i := \mathsf{Inc}(r, q) \mid \mathsf{If}(r, q, q) \mid \mathsf{Halt}(x) \qquad (r \in \{1, 2\}, q \in Q_M, x \in \{0, 1\}).$$

Two-counter machines act on configurations, which are strings of the form $a^n b^m q$, where q is a control state and a and b are counter symbols: the number of symbol occurrences determines which number is stored in a counter. When the machine halts, it outputs either 1 or 0 to indicate whether its input was accepted or rejected.

▶ **Definition 11.** *Let M be a two-counter machine. We define the following discrete commutable sets and terms:*

$$\begin{aligned} \Sigma_M &\triangleq Q_M + \{a, b, c_0, c_1\} && \textit{symbols} \\ \mathcal{T}\Sigma_M \ni C_M &\triangleq a^* b^* Q_M && \textit{running configurations} \\ \mathcal{T}\Sigma_M \ni T_M &\triangleq C_M + \{c_0, c_1\} && \textit{all configurations.} \end{aligned}$$

Normally, we would define the semantics of a two-counter machine directly, as a relation on configurations. However, it'll be more convenient to instead define the semantics through algebra terms that describe the graph of this relation, since we'll use these terms to analyze the execution of a machine with equations in pre-Kleene algebra. Our definition relies on the following construction.

▶ **Definition 12.** *Let X and Y be commutable sets. We define a commutable set*

$$X \oplus Y \triangleq \{x_l \mid x \in X\} \uplus \{y_r \mid y \in Y\},$$

where the commuting relation on $X \oplus Y$ is generated by the following rules:

$$\frac{}{x_l \sim y_r} \qquad \frac{x \sim x'}{x_l \sim x'_l} \qquad \frac{y \sim y'}{y_r \sim y'_r}$$

The canonical injections $(-)_l : X \to X \oplus Y$ and $(-)_r : Y \to X \oplus Y$ are morphisms in Comm *(and present commutable subsets). We abbreviate $X \oplus X$ as $\ddot{X}$.*

If X and Y are commutable sets, we abuse notation and view the functions $(-)_l : X \to X \oplus Y$ and $(-)_r : Y \to X \oplus Y$ as having types $\mathcal{T}X \to \mathcal{T}(X \oplus Y)$ and $\mathcal{T}Y \to \mathcal{T}(X \oplus Y)$. We have the corresponding projection functions $\pi_l : \mathcal{T}(X \oplus Y) \to \mathcal{T}X$ and $\pi_r : \mathcal{T}(X \oplus Y) \to \mathcal{T}Y$, where $\pi_l(y_r) = 1$ for $y \in Y$, and similarly for π_r (cf. Remark 4). If X is a commutable set, view a term $e \in \mathcal{T}X$ as an element $\mathcal{T}\ddot{X}$ by mapping each symbol $x \in X$ in e to $x_l x_r$. We'll use a similar convention for strings $\mathcal{S}$.

The idea behind this construction is that any string over $X \oplus Y$ can be seen as a pair of strings over X and Y. More precisely, the monoids $\mathcal{S}(X \oplus Y)$ and $\mathcal{S}X \times \mathcal{S}Y$ are isomorphic via the mappings

$$\mathcal{S}(X \oplus Y) \ni s \mapsto (\pi_l(s), \pi_r(s)) \qquad \mathcal{S}X \times \mathcal{S}Y \ni (s_1, s_2) \mapsto (s_1)_l (s_2)_r.$$

Since a term e over $X \oplus Y$ can be seen as a set of strings over $X \oplus Y$, we can also view it as a set of pairs of strings over X and Y – in other words, as a relation from $\mathcal{S}X$ to $\mathcal{S}Y$. We write $s \to_e s'$ if two strings are related in this way; that is, if $s_l s'_r \leq e$.

▶ **Definition 13** (Running a two-counter machine)**.** *We interpret each instruction $i \in I_M$ as an element $[\![i]\!] \in \mathcal{T}\ddot{\Sigma}_M$:*

$$\begin{aligned} [\![\mathsf{Inc}(1,q)]\!] &\triangleq a_r a^* b^* q_r & [\![\mathsf{If}(1,q_1,q_2)]\!] &\triangleq b^*(q_1)_r + a_l a^* b^* (q_2)_r \\ [\![\mathsf{Inc}(2,q)]\!] &\triangleq a^* b_r b^* q_r & [\![\mathsf{If}(2,q_1,q_2)]\!] &\triangleq a^*(q_1)_r + a^* b_l b^* (q_2)_r \\ [\![\mathsf{Halt}(x)]\!] &\triangleq (c_x)_r. \end{aligned}$$

The transition relation *of M, $R_M \in \mathcal{T}\ddot{\Sigma}_M$, is defined as*

$$R_M \triangleq \sum \{[\![\iota(q)]\!] q_l \mid q \in Q_M\}.$$

We say that M halts *on n if $a^n b^0 \dot{q} \to^*_{R_M} c_x$ for some $x \in \{0, 1\}$. We refer to x as the* output *of M on n.*

▶ **Lemma 14.** *The relation R_M satisfies the following property: for every $s \to_{R_M} s'$, s is of the form $a^n b^m q \leq C_M$. Moreover, for any s of this form, we have $s' = [\![\iota(q)]\!]_f(n,m)$, where the function $[\![i]\!]_f : \mathbb{N} \times \mathbb{N} \to T_M$ is defined as follows:*

$$[\![\mathsf{Inc}(1,q)]\!]_f(n,m) \triangleq a^{n+1}b^m q \qquad [\![\mathsf{If}(1,q_1,q_2)]\!]_f(n,m) \triangleq \begin{cases} a^n b^m q_1 & \text{if } n = 0 \\ a^p b^m q_2 & \text{if } n = p+1 \end{cases}$$

$$[\![\mathsf{Inc}(2,q)]\!]_f(n,m) \triangleq a^n b^{m+1} q \qquad [\![\mathsf{If}(2,q_1,q_2)]\!]_f(n,m) \triangleq \begin{cases} a^n b^m q_1 & \text{if } m = 0 \\ a^n b^p q_2 & \text{if } m = p+1 \end{cases}$$

$$[\![\mathsf{Halt}(x)]\!]_f(n,m) \triangleq c_x.$$

In particular, R_M defines a (partial) functional relation on T_M.

This means that Definition 13 accurately describes the standard semantics of two-counter machines [10], which allows us to analyze their properties algebraically. Combining this encoding with Theorem 17, we can show that KA inequalities over $\ddot{\Sigma}_M$ cannot be decided. More precisely, in the remainder of the paper, our aim is to prove the following results:

▶ **Theorem 15** (Soundness). *Given a two-counter machine M and a configuration $s \leq T_M$, suppose that the following inequality holds in $\mathcal{L}\ddot{\Sigma}_M$:*

$$s_r R_M^* \leq \Sigma^*(C_M + c_1)_r + \Sigma_M^* \Sigma_M^{\neq} \ddot{\Sigma}_M^*,$$

where $\Sigma_M^{\neq} \triangleq \sum_{\substack{x,y\in\Sigma \\ x\neq y}} x_l y_r$. If $s \to_{R_M}^* c_x$, then $x = 1$.

▶ **Theorem 16** (Completeness). *Given a two-counter machine M and some configuration $s \leq T_M$, we can compute a term ρ with the following property. If $s \to_{R_M}^* c_1$, then the following inequality is valid in* pre-*Kleene algebra: $sR_M^* \leq \Sigma_M^*(C_M + c_1)_r + \Sigma_M^* \Sigma_M^{\neq} \rho$.*

The soundness theorem can be shown by establishing a correspondence between traces of two-counter machines and the languages, then arguing that the language inequality implies that M halts and outputs 1. However, an inequality between terms is always stronger than the same inequality on languages. Thus, for completeness, we need to establish a stronger inequality between terms.

To obtain undecidability from soundness and completeness, we leverage *effective inseparability*, a notion from computability theory. In what follows, we use the notation $\langle x \rangle$ to refer to some effective encoding of the object x as a natural number.[2]

▶ **Theorem 17.** *The following two languages are effectively inseparable:*

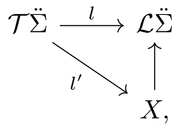
$$A \triangleq \{\langle M, x\rangle \mid \textit{The two-counter machine } M \textit{ halts on } x \textit{ and outputs 1}\}$$
$$B \triangleq \{\langle M, x\rangle \mid \textit{The two-counter machine } M \textit{ halts on } x \textit{ and outputs 0}\}.$$

In other words, there is a partial computable function f with the following property. Given a machine M, let W_M be the set of inputs accepted by M. Suppose that M_1 and M_0 are such that $W_{M_1} \cap W_{M_0} = \emptyset$, $A \subseteq W_{M_1}$ and $B \subseteq W_{M_0}$. Then $f\langle M_1, M_0\rangle$ is defined and does not belong to $W_{M_1} \cup W_{M_0}$.

[2] Note that we do not assume that this encoding is a functional relation. For example, we will need to encode pre-Kleene algebra terms as numbers. Such a term is an equivalence class of syntax trees quotiented by provable equality. Thus, each term can be encoded as multiple natural numbers, one for each syntax tree in its equivalence class. Nevertheless, by abuse of notation, we'll use the encoding notation as if it denoted a unique number.

Effective inseparability is a strengthening of the notion of inseparability, which says that two sets cannot be distinguished by a total computable function. If we are just interested in the undecidability of the equational theory, then basic inseparability is enough, as the following argument shows:

▶ **Theorem 18** (Undecidability). *Let $\Sigma \triangleq \{0,1\}$ be a discrete commutable set. Suppose that we have a diagram of sets*

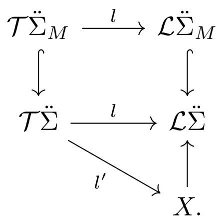


where l' is computable. Then equality on X is undecidable. In particular, equality is undecidable on $\mathcal{T}\ddot{\Sigma}$, $\mathcal{K}\ddot{\Sigma}$ and $\mathcal{L}\ddot{\Sigma}$.

Proof. Let A and B be the sets of Theorem 17. Let's define a computable function $\eta : \Sigma^* \to \Sigma^*$ with the following properties:

- if $s \in A$, then $\eta(s) \in X_=$, where $X_= \triangleq \{\langle x, y\rangle \mid x \text{ and } y \text{ encode the same element of } X\}$.
- if $s \in B$, then $\eta(s) \notin X_=$.

Find a suitable encoding of the characters of Σ_M as binary strings, which leads to the following injective embeddings:



In what follows, we'll treat $\mathcal{T}\ddot{\Sigma}_M$ and $\mathcal{L}\ddot{\Sigma}_M$ as subsets of $\mathcal{T}\ddot{\Sigma}$ and $\mathcal{L}\ddot{\Sigma}$, to simplify the notation.

Suppose that we are given some string $s \in \Sigma^*$. We define $\eta(s)$ as follows. We can assume that s is of the form $\langle M, n\rangle$, where M is a machine and $n \in \mathbb{N}$ (if s is not of this form, we define the output as $\eta(s) = \langle l'(0), l'(1)\rangle$). First, we compute the term ρ of Theorem 16, using $a^n b^0 \dot{q}$ as the initial configuration. Next, let e_L and e_R be the left- and right-hand sides of the inequality of Theorem 16. We pose $\eta\langle M, n\rangle \triangleq \langle l'(e_L + e_R), l'(e_R)\rangle$.

If $\langle M, n\rangle \in A$, the inequality of Theorem 16 is valid. Thus, $e_L \leq e_R$ holds, or, equivalently, $e_L + e_R = e_R$. Thus, $l'(e_L + e_R) = l'(e_R)$ is valid, which implies that $\eta(s) \in X_=$.

If, on the other hand, M outputs 0 on n (that is, $\langle M, n\rangle \in B$), we claim that $\eta(s) \notin X_=$. It suffices to prove $l'(e_L + e_R) \neq l'(e_R)$. Aiming for a contradiction, suppose that $l'(e_L + e_R) = l'(e_R)$. This implies $l(e_L + e_R) = l(e_L) + l(e_R) = l(e_R)$, which is equivalent to the inequality $l(e_L) \leq l(e_R)$. Let $e'_R \in \mathcal{T}\ddot{\Sigma}_M$ be the right-hand side of the inequality of Theorem 15. We have $l(e_R) \leq l(e'_R)$ because $l(\rho) \leq l(\Sigma_M^*)$. Thus, $l(e_L) \leq l(e'_R)$. However, by Theorem 15, this can only hold if M outputs 1, which contradicts our assumption.

To conclude, suppose that $d : \Sigma^* \to \{0,1\}$ is a decider for $X_=$ (that is, suppose that equality on X is decidable). Then $d \circ \eta$ can separate the sets A and B, which contradicts Theorem 17 because two effectively inseparable sets are also computationally inseparable. Therefore, such a d cannot exist. ◀

However, if we also want a more precise characterization of the complexity of this theory, the notion of effective inseparability is crucial. The following argument, which refines the previous proof, is based on Kuznetsov's work [17].

▶ **Theorem 19** (Complexity). *If equalities in X (from Theorem 18) are recursive enumerable, then equalities in X are Σ^0_1-complete. In particular, equalities in $\mathcal{T}\ddot{\Sigma}$ and $\mathcal{K}\ddot{\Sigma}$ are Σ^0_1-complete.*

Proof. Let $A' \triangleq \{\langle M, n\rangle \mid l'(e_L + e_R) = l'(e_R)\}$, where (M, n) is a machine-input pair and e_L and e_R are defined as in the proof of Theorem 18. By arguments in the proof of Theorem 18, if $\langle M, n\rangle \in A$, then $l'(e_L + e_R) = l'(e_R)$, which means $A' \supseteq A$. By similar arguments, $A' \cap B = \emptyset$.

By folklore [17, Proposition 9], A' and B are effectively inseparable because A and B are. Moreover, note that both A' and B are in Σ^0_1 – membership in A' is recursively enumerable because we can compute e_L and e_R from (M, n) and enumerate the possible proofs of $l'(e_L + e_R) = l'(e_R)$. This implies that A' is Σ^0_1-complete [17, Proposition 7], and therefore Σ^0_1-hard.

The function η in the proof of Theorem 18 has the property that $\eta(s) \in X_=$ if and only if $s \in A'$. (This relies on the fact that we defined $\eta(s) = \langle l'(0), l'(1)\rangle$ when s is not the encoding of a machine-input pair, and that $l'(0) \neq l'(1)$ because $l(0) \neq l(1)$). In other words, η is a reduction from A' to $X_=$, which proves $X_=$ is Σ^0_1-hard. We conclude because we assumed that equality on X is in Σ^0_1. ◀

Thus, to establish undecidability, we need to prove soundness and completeness. The easiest part is proving soundness: we just need to adapt the proof of undecidability of equations of $*$-continuous Kleene algebras with commutativity conditions [14]. For completeness, however, we need to do some more work. Roughly speaking, we first prove that R_M satisfies an analogue of the completeness theorem for a single transition, and then show that this version implies a more general one for an arbitrary number of transitions (Section 4).

The main challenge for proving the single-step version of completeness is that we can no longer leverage properties of regular languages, and must reason solely using the laws of pre-Kleene algebra. Our strategy is to show that R_M is just as good as its corresponding regular language if we want to reason about *prefixes* of matched strings. Given any string $s' \leq R_M$ and a current state s, we can tell whether s' encodes a valid sequence of transitions or not simply by looking at some finite prefix determined by s. This finite prefix can be extracted by unfolding R_M finitely many times, which can be done in the setting of preKA.

4 Representing Relations

In this section, we show that we can reduce the statement of completeness to a similar statement about single transitions. If $e \in \mathcal{T}\ddot{\Sigma}$ and $\Lambda \subseteq \mathcal{S}\Sigma$ is a set of strings, we write $\mathsf{Next}_e(\Lambda)$ to denote the image of Λ by $\to_e$; that is, the set $\bigcup_{s\in\Lambda}\{s' \mid s \to_e s'\}$.

▶ **Definition 20.** *Let $L \in \mathcal{T}\Sigma$ be term. We say that a term $e \in \mathcal{T}\ddot{\Sigma}$ is a* representable relation *on L if the following conditions hold:*

- $\pi_l(e) \leq L$;
- $\pi_r(e) \leq L$;
- $\mathsf{Next}_e(\Lambda)$ *is finite if Λ is (note that we must have $\mathsf{Next}_e(\Lambda) \leq \pi_r(e) \leq L$);*
- *there exists some* residue term ρ *such that $\Lambda_r e \leq \Lambda\, \mathsf{Next}_e(\Lambda)_r + \Sigma^*\Sigma^{\neq}\rho$ for every finite Λ.*

We write $e : \mathsf{Rel}(L)$ to denote the type of e.

Given a representable relation e, we can iterate the above inequality several times when reasoning about its reflexive transitive closure e^*:

▶ **Lemma 21.** *Suppose that* $e : \mathsf{Rel}(L)$*. There exists some* ρ *such that, for every* $n \in \mathbb{N}$ *and every finite* $\Lambda \leq L$*, we have the inequality* $\Lambda_r e^* \leq \Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda)_r + \Sigma^* \,\mathsf{Next}_e^n(\Lambda)_r e^* + \Sigma^* \Sigma^{\neq} \rho$*, where* $\mathsf{Next}_e^{<n} = \bigcup_{i<n} \mathsf{Next}_e^i(\Lambda)$*.*

If we know that the number of transitions from a given set of initial states is bounded, we obtain the following result.

▶ **Theorem 22.** *If* $e : \mathsf{Rel}(L)$*, there exists* ρ *such that, given* $n \in \mathbb{N}$ *and a finite* $\Lambda \leq L$*, if* $\mathsf{Next}_e^n(\Lambda) = \emptyset$*, then* $\Lambda_r e^* \leq \Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda)_r + \Sigma^* \Sigma^{\neq} \rho$*.*

5 Proving Representability

In this section, we prove that the transition relation R_M of a two-counter machine is a representable relation, which will allow us to derive completeness from Theorem 22. To do this, we need to show how we can use finite unfoldings of a relation to pinpoint certain terms that definitely match the "error" term $\Sigma_M^* \Sigma_M^{\neq} \rho$.

5.1 Automata theory

One of the pleasant consequences of working with Kleene algebra is that many intuitions about regular languages carry over. In particular, we can analyze terms by characterizing them as automata. This can be done algebraically by posing certain *derivative operations* δ_x on terms, which satisfy a *fundamental theorem* [20]: given a term $e \in \mathcal{K}X$, we have $e = e_0 + \sum_{x \in X} x \cdot \delta_x(e)$, where $e_0 \in \{0, 1\}$. Intuitively, each term in this equation corresponds to a state of some automaton. The term e corresponds to the starting state of the automaton, the null term e_0 states whether the starting state is accepting, and each $\delta_x(e)$ the state we transition too after observing the character $x \in X$. Derivatives can be iterated, describing the behavior of the automaton as it reads larger and larger strings, and which of those strings are accepted by it. This would be useful for our purposes, because such iterated derivatives would allow us to compute all prefixes up to a given length that can match an expression. Unfortunately, this theory of derivatives hinges on the induction properties of Kleene algebra, and it is unlikely that it can be adapted in all generality to the preKA setting. For example, the closest we can get to an expansion for 1^* is $1^* = 1 + 1 \cdot 1^* = 1 + 1^*$, which does not have the required form. Indeed, as demonstrated by Example 1, the star operation no longer preserves the multiplicative identity in preKA.

To remedy this issue, we are going to carve out a set of so-called *finite-state terms* of a pre-Kleene algebra, for which this type of reasoning is sound. Luckily, most regular operations preserve finite-state terms; we just need to be a little bit careful with the star operation. We start by defining *derivable* terms, which can be derived at least once. Finite-state terms will then allow us to iterate derivatives.

▶ **Definition 23.** *Let* $e \in \mathcal{T}X$ *be a term, where* X *is finite. We say that* e *is* derivable *if there exists a family of terms* $\{\delta_x(e)\}_{x \in X}$ *such that* $e = [e]_0 + \sum_x x\delta_x(e)$*. Recall* $[-]_0$ *is the homomorphism* $[-]_0 : \mathcal{T}X \to 2$ *such that* $[e]_0 = 1 \iff e \geq 1$*. We refer to the term* $\delta_x(e)$ *as the* derivative with respect to x*.*

The family $\delta_x(e)$ is not necessarily unique. Nevertheless, we'll use the notation $\delta_x(e)$ to refer to specific derivatives of x when it is clear from the context which one we mean.

▶ **Lemma 24.** *Derivable terms are closed under all the pre-Kleene algebra operations, with the following caveats: for e^*, we also require that $[e]_0 = 0$; for e_1e_2, the term is also derivable if e_2 isn't, provided that $[e_1]_0 = 0$. We have the following choices of derivatives:*

$$\delta_x(0) = 0 \qquad \delta_x(1) = 0$$
$$\delta_x(x) = 1 \qquad \delta_x(y) = 0 \quad \text{if } y \neq x$$
$$\delta_x(e_1 + e_2) = \delta_x(e_1) + \delta_x(e_2) \qquad \delta_x(e_1e_2) = [e_1]_0\delta_x(e_2) + \delta_x(e_1)e_2$$
$$\delta_x(e^*) = \delta_x(e)e^*,$$

where, by abuse of notation, we treat $[e_1]_0\delta_x(e_2)$ as 0 when e_2 is not necessarily derivable (since, by assumption, $[e_1]_0 = 0$ in that case).

▶ **Definition 25.** *Suppose that X is finite. A* finite-state automaton *is a finite set S of elements of $\mathcal{T}X$ (the* states*) that contains 1, is closed under finite sums and under derivatives (that is, every $e \in S$ is derivable, and each $\delta_x(e)$ is a state). We say that a term e is* finite state *if it is a state of some finite-state automaton S.*

Requiring that the states of an automaton be closed under sums means, roughly speaking, that we are working with non-deterministic rather than deterministic automata, generalizing the notion of Antimirov's derivative [3]. This treatment is convenient for the commutative setting, since a given string could be matched by choosing different orderings of its characters.

Finite-state terms can, in fact, be inductively constructed from the operations of pre-Kleene algebra, thus making the identification of a finite-state term trivial.

▶ **Lemma 26.** *Let X be a finite commutable set. Finite-state terms are preserved by all the pre-Kleene algebra operations (for e^*, we additionally require that $[e]_0 = 0$). Moreover, the set of states of the corresponding automata can be effectively computed.*

Furthermore, since terms in a finite-state automaton are closed under derivatives, we can unfold them via derivatives k times. This unfolding will turn a term into a sum of some strings that are shorter than k; and some strings s with length exact k, followed the residual expressions e_s indexed by s. Formally, we can express this property as follows.

▶ **Lemma 27.** *Let $e \in \mathcal{T}X$ be a state of a finite-state automaton S, and $k \in \mathbb{N}$. We can write*

$$e = \sum\{s \mid s \in \mathcal{S}X, s \leq e, |s| < k\} + \sum\{se_s \mid s \in \mathcal{S}X, |s| = k\},$$

where each $e_s \in S$ for all s, and the size $|s| \in \mathbb{N}$ of a string s is defined by mapping every symbol of s to $1 \in \mathbb{N}$.

5.2 Bounded-Output Terms

Lemma 27 gives us almost what we need to prove that the transition term R_M is a representable relation. It allows us to partition R_M into strings s of length bounded by k and terms of the form se_s, which match strings prefixed by s of length greater than k. The first component, the strings s, can be easily shown to satisfy the upper bound required for being representable. However, the prefixes s that appear in the terms se_s are arbitrary, and, since we are working with pre-Kleene algebra, there isn't much we can leverage to show that such prefixes will yield a similar bound. The issue is that, in principle, in order to tell whether $s'_r se_s \leq \Sigma_M^* \Sigma_M^{\neq} \rho$, we might need to unfold e_s arbitrarily deep, which we cannot do in the preKA setting. To rule out these issues, we introduce a notion of *bounded-output terms*, which guarantee that only a finite amount of unfolding is necessary.

▶ **Definition 28.** *Let $e \in \mathcal{T}\ddot{\Sigma}$ be a term. We say that e has* bounded output *if there exists some $k \in \mathbb{N}$ (the* fanout*) such that, for every string $s \leq e$, $|\pi_r(s)| \leq (|\pi_l(s)| + 1)k$.*

▶ **Lemma 29.** *Let e have bounded output with fanout k and let Λ be finite. If $s \in \mathsf{Next}_e(\Lambda)$, then $|s| \leq (m+1)k$, where $m = \max\{|s'| \mid s' \in \Lambda\}$. Thus, since Σ is finite, $\mathsf{Next}_e(\Lambda)$ is finite.*

▶ **Lemma 30.** *Bounded-output terms are closed under all the pre-Kleene algebra operations. For e^*, we additionally require that $|\pi_l(s)| \geq 1$ for all strings $s \leq e$.*

For bounded-output terms, we can improve the expansion of Lemma 27.

▶ **Lemma 31.** *Let $e \in \mathcal{T}\ddot{\Sigma}$ be a bounded-output term that is the state of some automaton S. There exists some $k \in \mathbb{N}$ such that e has fanout k and such that, for every $n \in \mathbb{N}$, we can write*

$$e = \sum\{s \mid s \leq e, |s| < n\} + \sum\{se_s \mid s \in \mathcal{S}\ddot{\Sigma}, |s| = n, |\pi_r(s)| \leq (|\pi_l(s)| + 1)k\},$$

where $e_s \in S$ for every s.

▶ **Definition 32.** *A term L over Σ is* prefix free *if for all strings $s_1 \leq L$ and $s_2 \leq L$, if s_1 is a prefix of s_2, then $s_1 = s_2$.*

▶ **Lemma 33** (normal)**.** *Let s and s' be two strings over Σ such that one is not a prefix of the other, or vice versa. Then we can write $s = s_0 x s_1$ and $s' = s_0 x' s_1'$ with $x \neq x'$. Thus, $s_r s_l' \ddot{\Sigma}^* \leq \Sigma^* \Sigma^{\neq} \ddot{\Sigma}^*$.*

▶ **Lemma 34.** *Suppose that $e \in \mathcal{T}\ddot{\Sigma}$ is such that $\pi_l(e) \leq L$ and $\pi_r(e) \leq L$, where L is prefix free. Suppose, moreover, that e is finite-state and has bounded output. Then $e : \mathsf{Rel}(L)$.*

5.3 Putting Everything Together

To derive completeness for two-counter machines (Theorem 16), it suffices to show that the hypotheses of Lemma 34 are satisfied.

▶ **Lemma 35.** *We have the following properties:*
- *T_M is prefix free.*
- *$\pi_l(R_M) \leq C_M \leq T_M$.*
- *$\pi_r(R_M) \leq T_M$.*
- *R_M is finite state (Definition 25).*
- *R_M has bounded output (Definition 28).*

Thus, by Lemma 34, the term R_M is a representable relation of type $\mathsf{Rel}(T_M)$.

Proof. To show that R_M is finite state and had bounded output, we just appeal to the closure properties of such terms Lemmas 26 and 30. The rest is routine. ◀

We can finally conclude with the proof of completeness, thus establishing undecidability (Theorem 18).

Proof of Theorem 16. If $s = s_0 \to_{R_M} \cdots \to_{R_M} s_n = c_1$, we can show that $\mathsf{Next}_e^i(s)$ is $\{s_i\}$ for $i \leq n$ and $\emptyset$ when $i > n$, because the transition relation is deterministic and because c_1 does not transition. Moreover, by Lemma 35, we have $s_i \leq C_M$ for every $i < n$ (since $(s_i)_l(s_{i+1})_r \leq R_M$).

Choose ρ as in Theorem 22. We have

$$\begin{aligned} sR_M^* &\leq \Sigma^* \,\mathsf{Next}_e^{<n+1}(s)_r + \Sigma^*\Sigma^{\neq}\rho \\ &= \Sigma^*(\mathsf{Next}_e^{<n}(s) + \mathsf{Next}_e^{n}(s))_r + \Sigma^*\Sigma^{\neq}\rho \\ &\leq \Sigma^*(C_M + c_1)_r + \Sigma^*\Sigma^{\neq}\rho. \end{aligned}$$ ◀

6 Conclusion and Related Work

In his seminal work, Kozen [15] established several hardness and completeness results for variants of Kleene algebra. He noted that deciding equality in $*$-continuous Kleene algebras with commutativity conditions on primitives was not possible – more precisely, the problem is Π^0_1-complete, by reduction from the complement of the Post correspondence problem (PCP). However, at the time, it was unknown whether a similar result applied to the pure theory of Kleene algebra with commutativity conditions ($\mathcal{K}X$). The question had been left open since then. Our work provides a solution, proving that the problem is undecidable, even for a much weaker theory $\mathcal{T}X$, which omits the induction axioms of Kleene algebra.

As we were about to post publicly this work, we became aware of the work of Kuznetsov [17], who independently proved a similar result. There are two main differences between our results and his. Originally, our proof only established the undecidability of the theory of Kleene algebra with commutativity conditions, whereas Kuznetsov's work proved its Σ^0_1-completeness as well by leveraging the notion of *effective inseparability*. Since learning about his work, we managed to adapt his ideas to our setting, thus obtaining completeness as well. On the other hand, Kuznetsov's proof requires the induction axiom of Kleene algebra to simplify some of the inequalities involving starred terms – specifically, he needs the identity $A^*(A^*)^+ \leq A^*$ and the monotonicity of $(-)^*$, whereas our proof also applies to the weaker theory of pre-Kleene algebra. In this sense, we can view the results reported here as a synthesis of Kuznetsov's work and ours.

In terms of techniques, both of our works draw inspiration from the proof of Π^0_1-completeness of the equational theory of $*$-continuous KA. Leveraging the reduction of the halting problem to the PCP, Kuznetsov used Kleene-algebra inequalities to describe *self-looping* Turing machines – that is, Turing machines that run forever by reaching a designated configuration that steps to itself. He then showed that the set of machine-input pairs $\langle M, x\rangle$ where machines M that reach a self-looping state on input x is recursively inseparable from the set of such pairs where M halts on the input, which implies that such inequalities cannot decidable.

The inequalities used by Kuznetsov are similar to ours, and can be proved by unfolding finitely many times the starred term that defines the execution of Turing machines, and by applying standard Kleene algebra inequalities that follow from induction. One important difference is that, in Kuznetsov's work, this starred term contains only $*$-free terms, which arise from the reduction of the halting problem to the PCP. This requires some more work to establish that the inequality indeed encodes the execution of the Turing machine, but this work just replicates the ideas behind the standard reduction from the halting problem to the PCP, so it does not need to be belabored. On the other hand, we leverage the language of Kleene algebra to define an execution model for two-counter machines, which can be encoded more easily. The downside of our approach is that our relation R_M involves starred terms, which require our notion of bounded output to be analyzed effectively.

References

1 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: semantic foundations for networks. *ACM SIGPLAN Notices*, 49(1):113–126, January 2014. doi:10.1145/2578855.2535862.

2 Allegra Angus and Dexter Kozen. Kleene Algebra with Tests and Program Schematology. Technical Report, Cornell University, USA, June 2001.

3 Valentin Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, March 1996. doi:10.1016/0304-3975(95)00182-4.

4 Timos Antonopoulos, Eric Koskinen, Ton Chanh Le, Ramana Nagasamudram, David A. Naumann, and Minh Ngo. An Algebra of Alignment for Relational Verification. *Proceedings of the ACM on Programming Languages*, 7(POPL):20:573–20:603, January 2023. doi:10.1145/3571213.

5 Jean Berstel. *Transductions and Context-Free Languages.* Vieweg+Teubner Verlag, Wiesbaden, 1979. doi:10.1007/978-3-663-09367-1.

6 Volker Diekert and Yves Métivier. Partial Commutation and Traces. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 457–533. Springer, Berlin, Heidelberg, 1997. doi:10.1007/978-3-642-59126-6_8.

7 Nate Foster, Dexter Kozen, Mae Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for netkat. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 343–355, New York, NY, USA, January 2015. Association for Computing Machinery. doi:10.1145/2676726.2677011.

8 Alan Gibbons and Wojciech Rytter. On the decidability of some problems about rational subsets of free partially commutative monoids. *Theoretical Computer Science*, 48:329–337, January 1986. doi:10.1016/0304-3975(86)90101-5.

9 C. A. R. Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene Algebra. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 - Concurrency Theory*, pages 399–414, Berlin, Heidelberg, 2009. Springer. doi:10.1007/978-3-642-04081-8_27.

10 J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley series in computer science. Addison-Wesley, 2001. URL: https://books.google.com/books?id=omIPAQAAMAAJ.

11 Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. *Concurrent Kleene Algebra with Observations: From Hypotheses to Completeness*, volume 12077 of *Lecture Notes in Computer Science*, pages 381–400. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-45231-5_20.

12 Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. Concurrent kleene algebra: Free model and completeness. In Amal Ahmed, editor, *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 856–882, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-89884-1_30.

13 Dexter Kozen. On kleene algebras and closed semirings. In Branislav Rovan, editor, *Mathematical Foundations of Computer Science 1990*, volume 452, pages 26–47. Springer-Verlag, Berlin/Heidelberg, 1990. doi:10.1007/BFb0029594.

14 Dexter Kozen. *Kleene algebra with tests and commutativity conditions*, volume 1055 of *Lecture Notes in Computer Science*, pages 14–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. doi:10.1007/3-540-61042-1_35.

15 Dexter Kozen. On the complexity of reasoning in kleene algebra. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 195–202. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614947.

16 Dexter Kozen and Alexandra Silva. Left-handed completeness. *Theoretical Computer Science*, 807:220–233, February 2020. doi:10.1016/j.tcs.2019.10.040.

17 Stepan L. Kuznetsov. On the complexity of reasoning in kleene algebra with commutativity conditions. In Erika Ábrahám, Clemens Dubslaff, and Silvia Lizeth Tapia Tarifa, editors, *Theoretical Aspects of Computing - ICTAC 2023 - 20th International Colloquium, Lima, Peru, December 4-8, 2023, Proceedings*, volume 14446 of *Lecture Notes in Computer Science*, pages 83–99. Springer, 2023. doi:10.1007/978-3-031-47963-2_7.

18 A. K. McIver, E. Cohen, and C. C. Morgan. *Using Probabilistic Kleene Algebra for Protocol Verification*, volume 4136 of *Lecture Notes in Computer Science*, pages 296–310. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/11828563_20.

19 Annabelle McIver, Tahiry M. Rabehaja, and Georg Struth. On probabilistic kleene algebras, automata and simulations. In *Proceedings of the 12th International Conference on Relational and Algebraic Methods in Computer Science*, RAMICS'11, pages 264–279, Berlin, Heidelberg, May 2011. Springer-Verlag. doi:10.1007/978-3-642-21070-9_20.

20 A.M Silva. *Kleene coalgebra*. s.n.; UB Nijmegen host, S.l.; Nijmegen, 2010. URL: http://hdl.handle.net/2066/83205.

21 Cheng Zhang, Arthur Azevedo de Amorim, and Marco Gaboardi. On incorrectness logic and kleene algebra with top and tests. *arxiv preprint*, August 2022. arXiv:2108.07707, doi:10.48550/arXiv.2108.07707.

A Detailed Proofs

▶ **Theorem 5.** *If $s \in \mathcal{S}X$ is a string and $e \in \mathcal{T}X$ a term, then $s \leq e$ is equivalent to $s \in l(e)$.*

Proof of Theorem 5. Suppose that $s \leq e$. Then $s \in \{s\} = l_X(s) \subseteq l_X(e)$ by monotonicity.

Conversely, suppose that $s \in l_X(e)$. We proceed by induction on e.

- If $e = x \in X$, then $s \in l_X(x)$ means that $s = x$. Thus, we get $s \leq e$.
- If $e = 0$, we get a contradiction.
- If $e = 1$, we must have $s = 1$, thus $s \leq e$.
- If $e = e_1e_2$, we must have $s = s_1s_2$, with $s_i \in l_X(e_i)$. By the induction hypotheses, $s_i \leq e_i$, and thus $s \leq e$.
- If $e = e_1 + e_2$, then there is some i such that $s \in l_X(e_i)$. By the induction hypothesis, $s \leq e_i$, and thus $s \leq e_1 + e_2$.
- Finally, suppose that $e = e_1^*$. Thus, there exists some n such that $s \in l_x(e_1)^n$. This means that we can find a family $(s_i)_{i\in\{1,\ldots,n\}}$ such that $s = \prod_i s_i$ and $s_i \in l_x(e_1)$ for every i. By the induction hypothesis, $s_i \leq e_1$ for every i. Therefore, $s = \prod_i s_i \leq e_1^n \leq e_1^* = e$. ◀

▶ **Theorem 6.** *We say that $e \in \mathcal{T}X$ is* finite *if its language $l(e)$ is. In this case, then $e = \sum l(e)$.*

Proof of Theorem 6. By induction on e. We note that, if $l(e)$ is finite, then $l(e')$ is also finite for every immediate subterm e', which allows us to apply the relevant induction hypotheses. If e is of the form e_1e_2 and $l(e) = \emptyset$, this need not be the case, but at least one of the factors e_i satisfies $l(e_i) = \emptyset$, which is good enough. ◀

▶ **Corollary 7.** *The language interpretation l is injective on finite terms: if $l(e_1) = l(e_2)$ and both e_1 and e_2 are finite, then $e_1 = e_2$.*

Proof of Corollary 7. We have $e_1 = \sum l(e_1) = \sum l(e_2) = e_2$. ◀

▶ **Corollary 8.** *For every term $e \neq 0$, there exists some string s such that $s \leq e$.*

Proof of Corollary 8. Note that $l(e) \neq \emptyset$. Indeed, if $l(e) = \emptyset = l(0)$, then $e = 0$ by Corollary 7, which contradicts our hypothesis. Therefore, we can find some s such that $s \in l(e)$. But this is equivalent to $s \leq e$ by Theorem 5. ◀

▶ **Lemma 14.** *The relation R_M satisfies the following property: for every $s \to_{R_M} s'$, s is of the form $a^n b^m q \leq C_M$. Moreover, for any s of this form, we have $s' = [\![\iota(q)]\!]_f(n,m)$, where the function $[\![i]\!]_f : \mathbb{N} \times \mathbb{N} \to T_M$ is defined as follows:*

$$[\![\mathsf{Inc}(1,q)]\!]_f(n,m) \triangleq a^{n+1}b^m q \qquad [\![\mathsf{If}(1,q_1,q_2)]\!]_f(n,m) \triangleq \begin{cases} a^n b^m q_1 & \text{if } n = 0 \\ a^p b^m q_2 & \text{if } n = p+1 \end{cases}$$

$$[\![\mathsf{Inc}(2,q)]\!]_f(n,m) \triangleq a^n b^{m+1} q \qquad [\![\mathsf{If}(2,q_1,q_2)]\!]_f(n,m) \triangleq \begin{cases} a^n b^m q_1 & \text{if } m = 0 \\ a^n b^p q_2 & \text{if } m = p+1 \end{cases}$$

$$[\![\mathsf{Halt}(x)]\!]_f(n,m) \triangleq c_x.$$

In particular, R_M defines a (partial) functional relation on T_M.

▶ **Theorem 15** (Soundness)**.** *Given a two-counter machine M and a configuration $s \leq T_M$, suppose that the following inequality holds in $\mathcal{L}\ddot{\Sigma}_M$:*

$$s_r R_M^* \leq \Sigma^*(C_M + c_1)_r + \Sigma_M^* \Sigma_M^{\neq} \ddot{\Sigma}_M^*,$$

where $\Sigma_M^{\neq} \triangleq \sum_{\substack{x,y \in \Sigma \\ x \neq y}} x_l y_r$. If $s \to_{R_M}^* c_x$, then $x = 1$.

Proof of Theorem 15. Suppose that we have some finite sequence of transitions $s = s_0 \to \cdots \to s_n = c_x$. By definition, $(s_i)_l (s_{i+1})_r \leq R_M$ for every $i \in \{0, \ldots, n-1\}$. Thus, we have the following inequality on languages:

$$\begin{aligned} p &\triangleq (s_0)_r \cdot (s_0)_l (s_1)_r \cdots (s_{n-1})_l (s_n)_r \\ &\leq (s_0)_r \cdot R_M \cdot \cdots \cdot R_M \\ &\leq (s_0)_r R_M^* \\ &\leq \Sigma_M^*(C_M + c_1)_r + \Sigma_M^* \Sigma_M^{\neq} \ddot{\Sigma}_M^*. \end{aligned}$$

On the other hand, by shuffling left and right characters,

$$\begin{aligned} p &= (s_0)_r \cdot (s_0)_l (s_1)_r \cdots (s_{n-1})_l (s_n)_r \\ &= (s_0)_r (s_0)_l \cdot (s_1)_r (s_1)_l \cdots (s_{n-1})_r (s_{n-1})_l \cdot (s_n)_r \\ &= s_0 \cdots s_{n-1} (s_n)_r \\ &\leq \Sigma_M^* (\Sigma_M)_r^+. \end{aligned}$$

We can check that the languages $\Sigma_M^*(\Sigma_M)_r^+$ and $\Sigma_M^* \Sigma_M^{\neq} \ddot{\Sigma}_M^*$ are disjoint. Therefore, it must be the case that $p \leq \Sigma_M^*(C_M + c_1)_r$. By projecting out the right components, we find that $\pi_r(p) = s_0 \cdots s_n \leq \Sigma_M^*(C_M + c_1)_r$. We cannot have $\pi_r(p) \leq \Sigma_M^* C_M$, since the last character c_x cannot appear in a string in C_M. Therefore, $\pi_r(p) \leq \Sigma_M^* c_1$, from which we conclude. ◀

▶ **Theorem 17.** *The following two languages are effectively inseparable:*

$$A \triangleq \{\langle M, x\rangle \mid \textit{The two-counter machine } M \textit{ halts on } x \textit{ and outputs 1}\}$$

$$B \triangleq \{\langle M, x\rangle \mid \textit{The two-counter machine } M \textit{ halts on } x \textit{ and outputs 0}\}.$$

In other words, there is a partial computable function f with the following property. Given a machine M, let W_M be the set of inputs accepted by M. Suppose that M_1 and M_0 are such that $W_{M_1} \cap W_{M_0} = \emptyset$, $A \subseteq W_{M_1}$ and $B \subseteq W_{M_0}$. Then $f\langle M_1, M_0\rangle$ is defined and does not belong to $W_{M_1} \cup W_{M_0}$.

Proof of Theorem 17. We implement f as follows. Given an input x, if x does not encode a pair of machines, then the output is undefined. Otherwise, suppose that $x = \langle M_1, M_0\rangle$. Construct a machine M_η as follows. On an input x, run M_1 and M_0 on $\langle x, x\rangle$ in parallel. If M_i accepts first, then halt and output $1 - i$. If neither accept, then just run forever. We pose $f(x) = \langle M_\eta, M_\eta\rangle$.

We need to show that $f(x) \notin W_{M_1} \cup W_{M_0}$ when $x = \langle M_1, M_0\rangle$ and the two machines satisfy the above hypotheses. Suppose that $f(x) = \langle M_\eta, M_\eta\rangle \in W_{M_1}$. By the definition of M_η, this means that M_η outputs 0 on $\langle M_\eta\rangle$. Thus $\langle M_\eta, M_\eta\rangle \in B \subseteq W_{M_0}$. This contradicts the hypothesis that $W_{M_1} \cap W_{M_0} = \emptyset$. Thus, $f(x) \notin W_{M_1}$. An analogous reasoning shows that $f(x) \notin W_{M_0}$, which allows us to conclude. ◀

▶ **Lemma 21.** *Suppose that* $e : \mathsf{Rel}(L)$*. There exists some* ρ *such that, for every* $n \in \mathbb{N}$ *and every finite* $\Lambda \leq L$*, we have the inequality* $\Lambda_r e^* \leq \Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda)_r + \Sigma^* \,\mathsf{Next}_e^{n}(\Lambda)_r e^* + \Sigma^*\Sigma^{\neq}\rho$*, where* $\mathsf{Next}_e^{<n} = \bigcup_{i<n} \mathsf{Next}_e^{i}(\Lambda)$*.*

Proof of Lemma 21. Let $\rho \triangleq \rho' e^*$, where ρ' is the residue of e. Abbreviate $\Sigma^*\Sigma^{\neq}\rho$ as ε. We proceed by induction on n. If $n = 0$, then the goal becomes $\Lambda_r e^* \leq \Sigma^* \,\mathsf{Next}_e^{0}(\Lambda)_r e^* + \varepsilon$, which holds because $\mathsf{Next}_e^{0}(\Lambda) = \Lambda$.

Otherwise, for the inductive step, suppose that the goal is valid for n. We need to prove that it is valid for $n + 1$. Recall that $\Lambda' \triangleq \mathsf{Next}_e(\Lambda) \leq L$. We have

$$
\begin{aligned}
&\Lambda_r e^* \\
&= \Lambda_r + \Lambda_r e e^* \\
&\leq \Lambda_r + \Lambda \,\mathsf{Next}_e(\Lambda)_r e^* + \varepsilon && (e \text{ is representable}) \\
&= \Lambda_r + \Lambda \Lambda'_r e^* + \varepsilon \\
&\leq \Lambda_r + \Lambda \left(\Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda')_r + \Sigma^* \,\mathsf{Next}_e^{n}(\Lambda')_r e^* + \varepsilon\right) + \varepsilon && \text{I.H.} \\
&= \Lambda_r + \Lambda\Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda')_r + \Lambda\Sigma^* \,\mathsf{Next}_e^{n}(\Lambda')_r e^* + \Lambda\varepsilon + \varepsilon \\
&\leq \Sigma^*\Lambda_r + \Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda')_r + \Sigma^* \,\mathsf{Next}_e^{n}(\Lambda')_r e^* + \varepsilon + \varepsilon && (\Lambda \text{ is finite}) \\
&= \Sigma^* \,\mathsf{Next}_e^{0}(\Lambda)_r + \Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda')_r + \Sigma^* \,\mathsf{Next}_e^{n}(\Lambda')_r e^* + \varepsilon \\
&= \Sigma^* \,\mathsf{Next}_e^{<n+1}(\Lambda)_r + \Sigma^* \,\mathsf{Next}_e^{n+1}(\Lambda)_r e^* + \varepsilon.
\end{aligned}
$$
◀

▶ **Theorem 22.** *If* $e : \mathsf{Rel}(L)$*, there exists* ρ *such that, given* $n \in \mathbb{N}$ *and a finite* $\Lambda \leq L$*, if* $\mathsf{Next}_e^{n}(\Lambda) = \emptyset$*, then* $\Lambda_r e^* \leq \Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda)_r + \Sigma^*\Sigma^{\neq}\rho$*.*

Proof of Theorem 22. Choose the same ρ as in Lemma 21. Then

$$
\begin{aligned}
&\Lambda_r e^* \\
&\leq \Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda)_r + \Sigma^* \,\mathsf{Next}_e^{n}(\Lambda)_r e^* + \Sigma^*\Sigma^{\neq}\rho && \text{by Lemma 21} \\
&= \Sigma^* \,\mathsf{Next}_e^{<n}(\Lambda)_r + \Sigma^*\Sigma^{\neq}\rho.
\end{aligned}
$$
◀

▶ **Lemma 24.** *Derivable terms are closed under all the pre-Kleene algebra operations, with the following caveats: for* e^**, we also require that* $[e]_0 = 0$*; for* e_1e_2*, the term is also derivable if* e_2 *isn't, provided that* $[e_1]_0 = 0$*. We have the following choices of derivatives:*

$$
\begin{aligned}
\delta_x(0) &= 0 & \delta_x(1) &= 0 \\
\delta_x(x) &= 1 & \delta_x(y) &= 0 \quad \text{if } y \neq x \\
\delta_x(e_1 + e_2) &= \delta_x(e_1) + \delta_x(e_2) & \delta_x(e_1e_2) &= [e_1]_0\delta_x(e_2) + \delta_x(e_1)e_2 \\
\delta_x(e^*) &= \delta_x(e)e^*,
\end{aligned}
$$

where, by abuse of notation, we treat $[e_1]_0\delta_x(e_2)$ *as 0 when* e_2 *is not necessarily derivable (since, by assumption,* $[e_1]_0 = 0$ *in that case).*

Proof of Lemma 24. We prove the closure property for products and star. For products, we start by expanding e_1:

$$\begin{aligned} e_1 e_2 &= \left([e_1]_0 + \sum_x x\delta_x(e_1) \right) e_2 \\ &= [e_1]_0 e_2 + \sum_x x\delta_x(e_1)e_2. \end{aligned}$$

If $[e_1]_0 = 0$, the first term gets canceled out, and we obtain $\sum_x x\delta_x(e_1)e_2 = [e_1]_0[e_2]_0 + \sum_x x\delta_x(e_1)e_2$. Otherwise, we know that e_2 is derivable, and we proceed as follows:

$$\begin{aligned} e_1 e_2 &= [e_1]_0 \left([e_2]_0 + \sum_x x\delta_x(e_2) \right) + \sum_x x\delta_x(e_1)e_2 \\ &= [e_1]_0[e_2]_0 + \sum_x [e_1]_0 x\delta_x(e_2) + \sum_x x\delta_x(e_1)e_2 \\ &= [e_1]_0[e_2]_0 + \sum_x x([e_1]_0\delta_x(e_2) + \delta_x(e_1)e_2) \qquad \text{(because } [e_1]_0 x = x[e_1]_0\text{)}, \end{aligned}$$

which allows us to conclude.

For star, assuming that $[e]_0 = 0$, we note that $e^* = 1 + ee^*$, and we apply the closure properties for the other operations. ◀

▶ **Lemma 26.** *Let X be a finite commutable set. Finite-state terms are preserved by all the pre-Kleene algebra operations (for e^*, we additionally require that $[e]_0 = 0$). Moreover, the set of states of the corresponding automata can be effectively computed.*

Proof of Lemma 26. Let's consider all the cases.

- The set $\{0, 1\}$ is an automaton by Lemma 24. Therefore, 0 and 1 are finite state.
- By Lemma 24, if x is a symbol, the set $S = \{x\}$ is a pre-automaton. Therefore, x is finite state because it belongs to the automaton $\bar{S}$.
- Suppose that S_1 and S_2 are finite automata. By Lemma 24, the set $S = \{e_1 + e_2 \mid e_1 \in S_1, e_2 \in S_2\}$ is a pre-automaton. Therefore, if we have finite-state terms e_1 and e_2 of S_1 and S_2, their sum $e_1 + e_2$ is finite state because it belongs to the automaton $\bar{S}$.
- Suppose that S_1 and S_2 are finite automata. By Lemma 24, the set $S = \{e_1 e_2 \mid e_1 \in S_1, e_2 \in S_2\}$ is a pre-automaton. Indeed, $\delta_x(e_1 e_2) = [e_1]_0\delta_x(e_2) + \delta_x(e_1)e_2$ is a sum of elements of S, since

$$\begin{aligned} [e_1]_0 &\in S_1 \\ \delta_x(e_2) &\in S_2 \\ \delta_x(e_1) &\in S_1 \\ e_2 &\in S_2. \end{aligned}$$

  Therefore, if we have finite-state terms e_1 and e_2 of S_1 and S_2, their product $e_1 e_2$ is finite state because it belongs to the automaton $\bar{S}$.
- Suppose that e is a state of some automaton S such that $[e]_0 = 0$. Define $S' = \{e'e^* \mid e' \in S\}$. By Lemma 24, this set is a pre-automaton. Indeed,

$$\begin{aligned} \delta_x(e'e^*) &= [e']_0\delta_x(e^*) + \delta_x(e')e^* \\ &= [e']_0\delta_x(e)e^* + \delta_x(e')e^* \\ &= ([e']_0\delta_x(e) + \delta_x(e'))e^*. \end{aligned}$$

  The terms $\delta_x(e)$ and $\delta_x(e')$ are in S. Thus, $[e']_0\delta_x(e) \in S$ and $\delta_x(e'e^*)$ is a sum of terms of S'. Since $e^* = 1e^*$ is an element of S', then it is a state of $\bar{S}'$, and e^* is finite state. ◀

▶ **Lemma 27.** *Let $e \in \mathcal{T}X$ be a state of a finite-state automaton S, and $k \in \mathbb{N}$. We can write*

$$e = \sum\{s \mid s \in \mathcal{S}X, s \leq e, |s| < k\} + \sum\{se_s \mid s \in \mathcal{S}X, |s| = k\},$$

where each $e_s \in S$ for all s, and the size $|s| \in \mathbb{N}$ of a string s is defined by mapping every symbol of s to $1 \in \mathbb{N}$.

Proof of Lemma 27. By induction on k. When $k = 0$, the equation is equivalent to $e = e$, and we are done. Otherwise, suppose that the result is valid for k. We need to prove that it is also valid for $k + 1$. Write

$$e = \sum_{\substack{s \in \mathcal{S}X \\ s \leq e \\ |s| < k}} s + \sum_{\substack{s \in \mathcal{S}X \\ |s| = k}} se_s.$$

By deriving each e_s, we can rewrite this as

$$\begin{aligned} e &= \sum_{\substack{s \in \mathcal{S}X \\ s \leq e \\ |s| < k}} s + \sum_{\substack{s \in \mathcal{S}X \\ |s| = k}} s \left([e_s]_0 + \sum_{x \in X} x\delta_x(e_s) \right) \\ &= \sum_{\substack{s \in \mathcal{S}X \\ s \leq e \\ |s| < k}} s + \sum_{\substack{s \in \mathcal{S}X \\ |s| = k}} s[e_s]_0 + \sum_{\substack{s \in \mathcal{S}X \\ |s| = k}} \sum_{x \in X} sx\delta_x(e_s). \end{aligned} \tag{1}$$

We can see that $[e_s]_0 = 1$ if and only if $s \leq e$: by taking the language interpretation of (1), we can see that a string of size k can only belong to the middle term, since the left and right terms can only account for strings of strictly smaller or larger size, respectively. Thus, we can rewrite (1) as

$$\begin{aligned} e &= \sum_{\substack{s \in \mathcal{S}X \\ s \leq e \\ |s| < k}} s + \sum_{\substack{s \in \mathcal{S}X \\ |s| = k \\ s \leq e}} s[e_s]_0 + \sum_{s, |s| = k} \sum_{x \in X} sx\delta_x(e_s) \\ &= \sum_{\substack{s \in \mathcal{S}X \\ s \leq e \\ |s| < k+1}} s + \sum_{\substack{s \in \mathcal{S}X \\ |s| = k}} \sum_{x \in X} sx\delta_x(e_s). \end{aligned} \tag{2}$$

Given some string s with $|s| = k + 1$, define

$$e'_s \triangleq \sum_{\substack{(s',x) \in \mathcal{S}X \times X \\ s = s'x}} \delta_x(e_{s'}).$$

This sum is well defined because there are only finitely many s' and $x \in X$ such that $s = s'x$: s' must be of size k, and there are only finitely many such strings. Moreover, e'_s is an element of S, since S is closed under taking derivatives and finite sums. We have

$$\begin{aligned} se'_s &= \sum_{\substack{(s',x) \\ |s'| = k \\ s = s'x}} s\delta_x(e_{s'}) \\ &= \sum_{\substack{(s',x) \\ |s'| = k \\ s = s'x}} s'x\delta_x(e_{s'}). \end{aligned}$$

Therefore,

$$\begin{aligned}\sum_{\substack{s\\|s|=k+1}} se'_s &= \sum_{\substack{s\\|s|=k+1}} \sum_{\substack{(s',x)\\|s'|=k\\s=s'x}} s'x\delta_x(e_{s'})\\&= \sum_{\substack{(s',x)\\|s'|=k}} s'x\delta_x(e_{s'})\\&= \sum_{\substack{s'\\|s'|=k}} \sum_{x\in X} s'x\delta_x(e_{s'}).\end{aligned}$$

Putting everything together, (2) becomes

$$e = \sum_{s\leq e,|s|<k+1} s + \sum_{\substack{s\\|s|=k+1}} se'_s, \tag{3}$$

which completes the inductive case. ◀

▶ **Lemma 29.** *Let e have bounded output with fanout k and let Λ be finite. If $s \in \mathsf{Next}_e(\Lambda)$, then $|s| \leq (m+1)k$, where $m = \max\{|s'| \mid s' \in \Lambda\}$. Thus, since Σ is finite, $\mathsf{Next}_e(\Lambda)$ is finite.*

Proof of Lemma 29. If $s \in \mathsf{Next}_e(\Lambda)$, by definition, there exists $s' \in \Lambda$ such that $s'_l s_r \leq e$. Since e has fanout k, we have

$$|s| = |\pi_r(s'_l s_r)| \leq (|\pi_l(s'_l s_r)| + 1)k = (|s| + 1)k \leq (n+1)k.$$ ◀

▶ **Lemma 30.** *Bounded-output terms are closed under all the pre-Kleene algebra operations. For e^*, we additionally require that $|\pi_l(s)| \geq 1$ for all strings $s \leq e$.*

Proof of Lemma 30. Let's focus on the last point. Suppose that e has fanout k and that $|\pi_l(s)| \geq 1$ for every $s \leq e$. We are going to show that e^* has bounded output with fanout $2k$.

Suppose that $s \leq e^*$. We can write $s = s_1 \cdots s_n$ such that $s_i \leq e$ for every $i \in \{1, \ldots, n\}$. We have, for every $i \in \{1, \ldots, n\}$, $|\pi_r(s_i)| \leq (|\pi_l(s_i)| + 1)k$. Thus,

$$\begin{aligned}|\pi_r(s)| &= \sum_{i=1}^n |\pi_r(s_i)|\\&\leq \sum_{i=1}^n (|\pi_l(s_i)| + 1)k\\&\leq \sum_{i=1}^n 2|\pi_l(s_i)|k && \text{(because } |\pi_l(s_i)| \geq 1)\\&= \left(\sum_{i=1}^n |\pi_l(s_i)|\right) 2k\\&= |\pi_l(s_0) \cdots \pi_l(s_n)|2k\\&= |\pi_l(s_0 \cdots s_n)|2k\\&= |\pi_l(s)|2k\\&\leq (|\pi_l(s)| + 1)2k.\end{aligned}$$ ◀

▶ **Lemma 31.** *Let $e \in \mathcal{T}\ddot{\Sigma}$ be a bounded-output term that is the state of some automaton S. There exists some $k \in \mathbb{N}$ such that e has fanout k and such that, for every $n \in \mathbb{N}$, we can write*

$$e = \sum\{s \mid s \leq e, |s| < n\} + \sum\{se_s \mid s \in \mathcal{S}\ddot{\Sigma}, |s| = n, |\pi_r(s)| \leq (|\pi_l(s)| + 1)k\},$$

where $e_s \in S$ for every s.

Proof of Lemma 31. Let k_0 be the fanout of e. For each $e' \in S$ such that $e' \neq 0$, choose some string $w_{e'} \leq e'$. Define $m \triangleq \max\{|\pi_l(w_{e'})| \mid e' \in S, e' \neq 0\}$ and $k \triangleq (m+1)k_0$. Since $k \geq k_0$, we know that e has fanout k. Moreover, by Lemma 27, we have

$$\begin{aligned} e &= \sum_{\substack{s \leq e \\ |s| < n}} s + \sum_{\substack{s \in \mathcal{S}X \\ |s| = n}} se_s \\ &= \sum_{\substack{s \leq e \\ |s| < n}} s + \sum_{\substack{s \in \mathcal{S}X \\ |s| = n \\ e_s \neq 0}} se_s, \end{aligned}$$

where each e_s is a state of S. If s is such that $|s| = n$ and $e_s \neq 0$, we have $sw_{e_s} \leq e$. Therefore,

$$\begin{aligned} |\pi_r(s)| &\leq |\pi_r(sw_{e_s})| \\ &\leq (|\pi_l(sw_{e_s})| + 1)k_0 \\ &= (|\pi_l(s)| + |\pi_l(w_{e_s})| + 1)k_0 \\ &\leq (|\pi_l(s)| + m + 1)k_0 \\ &\leq (|\pi_l(s)| + 1)(m+1)k_0 \\ &= (|\pi_l(s)| + 1)k. \end{aligned}$$

Thus,

$$\begin{aligned} e &= \sum_{\substack{s \leq e \\ |s| < n}} s + \sum_{\substack{s \\ |s| = n \\ e_s \neq 0 \\ |\pi_r(s)| \leq (|\pi_l(s)|+1)k}} se_s \\ &= \sum_{\substack{s \leq e \\ |s| < n}} s + \sum_{\substack{s \\ |s| = n \\ |\pi_r(s)| \leq (|\pi_l(s)|+1)k}} se_s. \end{aligned}$$ ◀

▶ **Lemma 33** (normal). *Let s and s' be two strings over Σ such that one is not a prefix of the other, or vice versa. Then we can write $s = s_0 x s_1$ and $s' = s_0 x' s_1'$ with $x \neq x'$. Thus, $s_r s_l' \ddot{\Sigma}^* \leq \Sigma^* \Sigma^{\neq} \ddot{\Sigma}^*$.*

Proof of Lemma 33. By induction on the length of s. ◀

▶ **Lemma 34.** *Suppose that $e \in \mathcal{T}\ddot{\Sigma}$ is such that $\pi_l(e) \leq L$ and $\pi_r(e) \leq L$, where L is prefix free. Suppose, moreover, that e is finite-state and has bounded output. Then $e : \mathsf{Rel}(L)$.*

Proof of Lemma 34. We have already seen that $\mathsf{Next}_e(\Lambda)$ is finite when Λ is (Lemma 29). Thus, we need to find some ρ such that, for every finite Λ,

$$\Lambda_r e \leq \Lambda\, \mathsf{Next}_e(\Lambda)_r + \Sigma^* \Sigma^{\neq} \rho.$$

Define $\rho \triangleq \ddot{\Sigma}^* \rho_e$, where ρ_e is the greatest element of the automaton of e. It suffices to prove the result for the case $\Lambda = \{s\}$. Indeed, if the result holds for singletons, we have

$$\begin{aligned}
\Lambda_r e &= \sum_{s \in \Lambda} s_r e \\
&\le \sum_{s \in \Lambda} s\, \mathsf{Next}_e(s)_r + \Sigma^* \Sigma^{\neq} \rho && \text{by assumption} \\
&\le \sum_{s \in \Lambda} \Lambda\, \mathsf{Next}_e(s)_r + \Sigma^* \Sigma^{\neq} \rho \\
&= \Lambda \sum_{s \in \Lambda} \mathsf{Next}_e(s) + \Sigma^* \Sigma^{\neq} \rho \\
&= \Lambda\, \mathsf{Next}_e(\Lambda) + \Sigma^* \Sigma^{\neq} \rho.
\end{aligned}$$

◀

Let k be the constant of Lemma 31 for e, $n = |s|$, and let $p = (k+1)(n+1)$. Let

$$\ddot{\Lambda} \triangleq \{s' \in \mathcal{S}\ddot{\Sigma} \mid |s'| = p+1, |\pi_r(s')| \le (|\pi_l(s')| + 1)k\}.$$

By applying Lemma 31 to e, we can write

$$\begin{aligned}
e &= \sum_{\substack{s' \le e \\ |s'| < p+1}} s' + \sum_{s' \in \ddot{\Lambda}} s' e_{s'} \\
&= \sum_{s' \le e, |s'| \le p} s' + \sum_{s' \in \ddot{\Lambda}} s' e_{s'} \\
&= \sum_{\substack{s' \le e \\ |s'| \le p \\ \pi_l(s') = s}} s' + \sum_{\substack{s' \le e \\ |s'| \le p \\ \pi_l(s') \neq s}} s' + \sum_{s' \in \ddot{\Lambda}} s' e_{s'},
\end{aligned}$$

Thus, to prove the inequality, it suffices to prove

$$s_r \sum_{\substack{s' \le e \\ |s'| \le p \\ \pi_l(s') = s}} s' = s\, \mathsf{Next}_e(s)_r \tag{4}$$

$$s_r \sum_{\substack{s' \le e \\ |s'| \le p \\ \pi_l(s') \neq s}} s' \le \Sigma^* \Sigma^{\neq} \rho \tag{5}$$

$$s_r \sum_{s' \in \ddot{\Lambda}} s' e'_s \le \Sigma^* \Sigma^{\neq} \rho. \tag{6}$$

Let us start with (4). Notice that, for any string s' over $\ddot{\Sigma}$, we have $s' = \pi_l(s')_l \pi_r(s')_r$. Therefore, there is a bijection between the set of indices s' of the sum and the set of strings $\mathsf{Next}_e(s)$. The bijection is given by

$$\begin{aligned}
s' &\mapsto \pi_r(s') \in \mathsf{Next}_e(s) \\
\mathsf{Next}_e(s) \ni s' &\mapsto s_l s'_r.
\end{aligned}$$

To prove that this is a bijection, we must show that the inverse produces indeed a valid index. Notice that, if $s' \in \mathsf{Next}_e(s)$, by Lemma 29, we have $|s'| \le (n+1)k$, and thus $|s_l s'_r| = |s| + |s'| \le (n+1)(k+1) = p$.

By reindexing the sum in (4) with this bijection, we have

$$\begin{aligned}
s_r \sum_{\substack{s' \le e \\ |s'| \le p \\ \pi_l(s') = s}} s' &= s_r \sum_{s' \in \mathsf{Next}_e(s)} s_l s'_r \\
&= s_r s_l \sum_{s' \in \mathsf{Next}_e(s)} s'_r \\
&= s_r s_l \left(\sum_{s' \in \mathsf{Next}_e(s)} s' \right)_r \\
&= s\, \mathsf{Next}_e(s)_r.
\end{aligned}$$

Next, let us look at (5). Suppose that s' is such that $s' \le e$ and $\pi_l(s') \ne s$. Since L is prefix free, and $\pi_l(s') \le L$, Lemma 33 applied to s and s' yields

$$s_l s' \le \Sigma^* \Sigma^{\ne} \ddot{\Sigma}^* \le \Sigma^* \Sigma^{\ne} \ddot{\Sigma}^* \rho_e = \Sigma^* \Sigma^{\ne} \rho,$$

where we use the fact that $\rho_e \ge 1$ because 1 is a state of the automaton of e. Summing over all such s', we get the desired inequality.

To conclude, we must show (6). By distributivity, this is equivalent to showing that, for every $s' \in \Lambda$,

$$s_r s' e_{s'} \le \Sigma^* \Sigma^{\ne} \rho.$$

If $e_{s'} = 0$, we are done. Otherwise, by Corollary 8, we can find some string $s'' \le e_{s'}$. We have $s' s'' \le s' e_{s'} \le e$.

Note that we must have $|\pi_l(s')| > n$. Indeed, suppose that $|\pi_l(s')| \le n$. Since $s' \in \Lambda$, we have

$$\begin{aligned}
|s'| &= |\pi_l(s')| + |\pi_r(s')| \\
&\le |\pi_l(s')| + (|\pi_l(s')| + 1)k \\
&\le (|\pi_l(s')| + 1)(k+1) \\
&\le (n+1)(k+1) \\
&< p + 1 \\
&= |s'|,
\end{aligned}$$

which is a contradiction.

Since $\pi_l(s's'') \le \pi_l(e) \le L$ and L is prefix free, by Lemma 33, we can write $s = s_0 x s_1$ and $\pi_l(s's'') = \pi_l(s')\pi_l(s'') = s_0 x' s'_1$, with $x \ne x'$. But $|\pi_l(s')| > n = |s|$ and $|s_0| < |s|$, thus $\pi_l(s')$ must be of the form $s_0 x' s'_2$. We find that $s_r s' = s_r \pi_l(s') \pi_r(s') \le \Sigma^* \Sigma^{\ne} \ddot{\Sigma}^*$, and thus

$$s_r s' e_{s'} \le \Sigma^* \Sigma^{\ne} \ddot{\Sigma}^* e_{s'} \le \Sigma^* \Sigma^{\ne} \ddot{\Sigma}^* \rho_e = \Sigma^* \Sigma^{\ne} \rho.$$