Kleene algebra with commutativity conditions is undecidable

Arthur Azevedo de Amorim, Cheng Zhang, Marco Gaboardi

▶ To cite this version:

Arthur Azevedo de Amorim, Cheng Zhang, Marco Gaboardi. Kleene algebra with commutativity conditions is undecidable. 2024. hal-04534715v2

HAL Id: hal-04534715
https://hal.science/hal-04534715v2

Preprint submitted on 26 Apr 2024 (v2), last revised 24 Nov 2024 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Kleene algebra with commutativity conditions is undecidable

Arthur Azevedo de Amorim Cheng Zhang Marco Gaboardi

April 26, 2024

Abstract

We prove that the equational theory of Kleene algebra with commutativity conditions on atomic terms is undecidable, thereby settling a longstanding open question in the theory of Kleene algebra. In fact, we show that this undecidability result holds even if we drop the *induction axioms* of Kleene algebra, which leads to a simpler equational theory. This complements a recent result of Kuznetsov, who independently established a similar undecidability result, but relying on induction.

1 Introduction

Kleene algebra is an algebraic framework that generalizes the theory of regular languages while retaining its most fundamental equational properties. One of the main features of Kleene algebra is that its equational theory is decidable. This enables numerous applications in program verification, by first translating programs and specifications into Kleene-algebra terms and then checking these terms for equality. Many domains have benefited from this approach to verification, including networked systems [And+14; Fos+15], concurrency [Hoa+09; Kap+20; Kap+18], probabilistic programming [MCM06; MRS11], relational verification [Ant+22], program schematology [AK01], and program incorrectness [ZAG22].

In most applications, it is necessary to extend Kleene algebra with additional axioms. A popular choice of axioms is commutativity conditions of the form $e_1e_2 = e_2e_1$, which state that the terms e_1 and e_2 can be composed in any order. In terms of program analysis, the terms e_1 and e_2 correspond to sub-commands of a larger program, and their commutativity ensures that they can be executed in any order without affecting the final output. Such properties have been proven useful for relational reasoning [Ant+22] and concurrency [DM97].

Unfortunately, it is known that commutativity conditions can pose issues for the decidability of the equational theory. In particular, if we are allowed to add arbitrary commutativity conditions $ab = ba$, where a and b are atomic terms (also known as "primitives"), it is undecidable to test whether two *regular languages* given by Kleene algebra terms are equivalent. Cohen proved this result by encoding the Post correspondence problem (PCP) using such equivalences; Cohen's proof was originally un-

published, but eventually reported by Kozen [Koz96]. In fact, as observed by Kozen [Koz97], the proof shows that the problem turns out to be Π^0_1-complete, or equivalent to the complement of the halting problem.

Despite Cohen's negative result, there was still some hope that 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 equational theory of Kleene algebras is generated by finitely many clauses, the set of valid equations 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. Despite this intuition, the question of decidability of the equational theory of Kleene algebra with commutativity conditions remained open for almost 30 years, since Kozen's work [Koz97].

This paper settles this question *negatively*, proving that it is impossible to decide whether an equation between two terms holds only by using commutativity conditions on atoms and the axioms of Kleene algebra. The question has been independently settled by Kuznetsov [Kuz23], who showed that this problem is, in fact, Σ^0_1-complete (that is, equivalent to the halting problem for Turing machines). Though our techniques overlap, we show actually that the undecidability result extends to weaker version of Kleene algebra where we use only a subset of its axioms. More precisely, we show that the theory of Kleene algebra with commutativity conditions remains undecidable even if we drop 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 Turing 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.

Structure of the paper In Section 2, we recall basic facts about Kleene algebra and related structures, and introduce an abstract framework for stating the problem of Kleene algebra terms modulo commutativity conditions, using the language of 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 can decide such inequalities, we would be able to distinguish recursively inseparable sets, which would lead to a contradiction. This argument hinges on a *completeness result* (Theorem 3.8), 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 5.6) 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 [Kuz23], which proved a similar result for Kleene algebra terms.

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 express Kleene algebra terms equipped with commutativity conditions on atoms.

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

$$\begin{aligned} 1 &: X \\ 0 &: X \\ (-)+(-) &: X \times X \to X \\ (-)\cdot(-) &: X \times X \to X \\ (-)^* &: X \to X, \end{aligned}$$

which are required to satisfy the following equations:

$$\begin{aligned} 1 \cdot x &= x \\ x \cdot 1 &= x \\ 0 \cdot x &= 0 \\ x \cdot 0 &= 0 \\ x \cdot (y \cdot z) &= (x \cdot y) \cdot z \\ 0 + x &= x \\ x + y &= y + x \\ x + (y + z) &= (x + y) + z \\ x \cdot (y + z) &= x \cdot y + x \cdot z \\ (x + y) \cdot z &= x \cdot z + y \cdot z \\ x^* &= 1 + x \cdot x^* && \text{left unfolding.} \end{aligned}$$

A weak Kleene algebra carries the usual ordering relation on idempotent monoids: $x \leq y$ means that $y + x = x$. A *Kleene algebra* is a weak Kleene algebra that satisfies the following properties:

$$xy \leq y \Rightarrow x^* y \leq y \qquad \text{left induction}$$
$$xy \leq x \Rightarrow xy^* \leq x \qquad \text{right induction.}$$

A $*$-*continuous Kleene algebra* is a weak Kleene algebra where, for all p, q and r, $\sup_{n \geq 0} pq^n r$ exists and is equal to $pq^* r$. We can show that every $*$-continuous algebra is, in fact, a Kleene algebra.

Let X and Y be weak Kleene algebras. A *morphism* of type $X \to Y$ is a function $f : X \to Y$ that commutes with all the algebra operations. This gives rise to a series of categories $\mathsf{KA}^* \subseteq \mathsf{KA} \subseteq \mathsf{WKA}$ of $*$-continuous algebras, Kleene algebras, and weak Kleene algebras. Each category is a full subcategory of the next one.

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 $t_1 t_2$, 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 WKA, 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 allows us to apply many of our intuitions about regular languages to the context of a general Kleene algebra. Unfortunately, when working with a weak Kleene algebra, most of these results cannot be directly applied, making reasoning about its elements trickier. In practice, we can only reason about properties of the star operation that involve a finite amount 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 where it is not possible to check the equality of two terms algorithmically. The notion of *commutable set*, which we introduce next, allows us to discuss such commutativity hypotheses in generality.

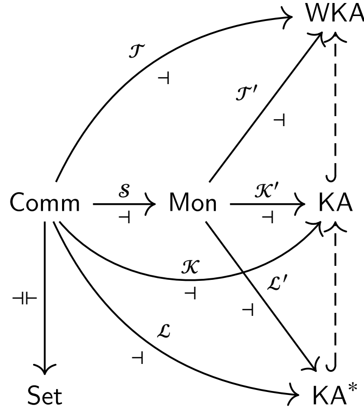


Figure 1: Algebraic constructions on commutable sets

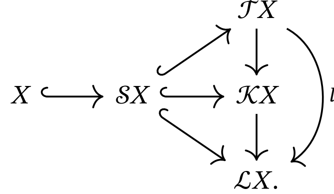
Definition 2.1. A *commuting relation* on a set X is a reflexive symmetric relation on X. A *commutable set* is a carrier set endowed with a commuting relation $\sim$. We say that two elements x and y commute if $x \sim y$. A commutable set is *commutative* if all two elements commute; it is *discrete* if the commuting relation is equality. A morphism of commutable sets is a function between the carriers that preserves the commuting relation. This data defines 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 obtained by restricting the corresponding relation of X. A commutable subset of X is the same thing as a regular subobject of X in Comm—i.e., it is the equalizer of a pair of maps; more precisely, the obvious maps into the two-element set.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.

Given a commutable set, we have various ways of building new 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 corresponding right adjoint functors 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 of elements of X, where we equate two strings if they can be obtained from each other by swapping adjacent elements that commute in X, according to $\sim$. 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 weak Kleene algebra, and where we identify two terms if they can be obtained from each other

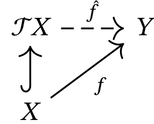
by applying the weak Kleene algebra equations. The construction $\mathcal{K}'$ is obtained by imposing further equations on terms, while $\mathcal{L}'$ is given by the algebra of regular languages over a monoid [Koz97], which we'll define soon. The right adjoints of these constructions view an algebra 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.

Being a free construction allows 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 weak 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 allow 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 2.2 (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 the set $\mathcal{P}X$ as a $*$-continuous algebra by using the following operations:

$$\begin{aligned} 0 &\triangleq \varnothing \\ 1 &\triangleq \{1\} \\ A + B &\triangleq A \cup B \\ A \cdot B &\triangleq \{xy \mid x \in A, y \in B\} \\ A^* &\triangleq \bigcup_{n \in \mathbb{N}} A^n. \end{aligned}$$

The $*$-continuous algebra $\mathcal{L}'X$ of regular sets 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, in the sense that we do not lose much information by doing so.

Theorem 2.3. *Let $s \in \mathcal{S}X$ be a string and $e \in \mathcal{T}X$ be an arbitrary term. Then $s \leq e$ is equivalent to $s \in l(e)$.*

Proof. 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 $e = e_1 e_2$, we must have $s = s_1 s_2$, 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 $(s_i)_{i \in \{1,\dots,n\}}$ 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 2.4. *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. 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 $e_1 e_2$ and $l(e) = \varnothing$, this need not be the case, but at least one of the factors e_i satisfies $l(e_i) = \varnothing$, which is good enough. □

Corollary 2.5. *The language interpretation l is injective on finite terms: if $l(e_1) = l(e_2)$ is finite, then $e_1 = e_2$.*

Proof. We have $e_1 = \sum l(e_1) = \sum l(e_2) = e_2$. □

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 (weak) 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 2.6. *For every term $e \neq 0$, there exists some string s such that $s \leq e$.*

Proof. Note that $l(e) \neq \varnothing$. Indeed, if $l(e) = \varnothing = l(0)$, then $e = 0$ by Corollary 2.5, 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 2.3. □

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 2.7. *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 Inseparability

It is commonplace to prove that a language is undecidable by reducing the halting problem to it. Our proof will use a variation on this theme, based on the following basic fact from computability theory (we use the notation $\langle x \rangle$ to refer to some effective encoding of the object x as a binary string).

Theorem 3.1. *The following two languages are recursively inseparable:*

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

In other words, there does not exist a total computable function $f : 2^* \to 2$ such that $f(A) = \{1\}$ and $f(B) = \{0\}$.

Proof. By diagonalization. Suppose that such an f exists. Consider the following decision procedure. Given an input $x \in 2^*$, proceed as follows.

- If $f(\langle x, x\rangle) = 1$, then we output 0.
- If $f(\langle x, x\rangle) = 0$, then we output 1.

This decision procedure is total assuming that f is. Therefore, it must be implemented by some Turing machine M. Since the machine halts on every input, $M(\langle M\rangle)$ must be defined and be either 0 or 1.

- If $M(\langle M\rangle) = 1$, then $\langle\langle M\rangle, \langle M\rangle\rangle \in A$. By assumption, $f(\langle\langle M\rangle, \langle M\rangle\rangle) = 1$. But by the definition of M, this means that $M(\langle M\rangle) = 0$; contradiction.
- If $M(\langle M\rangle) = 0$, then $\langle\langle M\rangle, \langle M\rangle\rangle \in B$. By assumption, $f(\langle\langle M\rangle, \langle M\rangle\rangle) = 0$. But by the definition of M, this means that $M(\langle M\rangle) = 1$; contradiction.

Since M has no behavior on this input, f cannot be total. □

We will prove that, if the language of equations on terms with commutativity conditions were decidable, we could construct such an f, thus contradicting Theorem 3.1. To simplify our construction, we'll use as a stepping stone the notion of *two-counter machine*. 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 machines 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, Motwani, and Ullman [HMU01, §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. To conclude, it suffices to reduce the output language for two-counter machines to the problem of solving weak Kleene algebra inequalities.

Definition 3.2. 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 as follows:

$$\begin{aligned} I_M \triangleq\ & \{\mathsf{Inc}(r, q) \mid r \in \{1,2\}, q \in Q_M\} \\ & \cup \{\mathsf{If}(r, q_1, q_2) \mid r \in \{1,2\}, q_1, q_2 \in Q_M\} \\ & \cup \{\mathsf{Halt}(x) \mid x \in \{0,1\}\}. \end{aligned}$$

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 3.3. Let M be a two-counter machine. We define the following discrete commutable sets and terms:

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

To describe the execution of two-counter machines, we'll use the following construction.

Definition 3.4. 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'll 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 2.2). 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

$$\begin{aligned} \mathcal{S}(X \oplus Y) \ni s &\mapsto (\pi_l(s), \pi_r(s)) \in \mathcal{S}X \times \mathcal{S}Y \\ \mathcal{S}X \times \mathcal{S}Y \ni (s_1, s_2) &\mapsto (s_1)_l (s_2)_r \in \mathcal{S}(X \oplus Y). \end{aligned}$$

Since 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 $s_l s'_r \le e$.

Definition 3.5 (Running a two-counter machine)**.** We interpret each instruction $i \in I_M$ as an element $(\!|i|\!) \in F\ddot{\Sigma}_M$:

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

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

$$R_M \triangleq \sum_{q \in Q_M} (\!|\iota(q)|\!) q_l.$$

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 3.6. *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 $s' = [\![\iota(q)]\!](n,m)$, where the function $[\![i]\!] : \mathbb{N} \times \mathbb{N} \to T_M$ is defined as follows:*

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

In particular, this defines a functional relation.

This means that the encoding in Definition 3.5 accurately describes the execution of two-counter machines, which allows us to analyze their properties algebraically. Combining this encoding with Theorem 3.1, we can show that KA inequalities over $\ddot{\Sigma}_M$ cannot be decided. More precisely, our aim is to prove the following results:

Theorem 3.7 (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

$$\Sigma_M^{\neq} \triangleq \sum_{\substack{x,y \in \Sigma \\ x \neq y}} x_l y_r,$$

then the machine accepts the input if it terminates. Formally, if $s \to^*_{R_M} c_x$, then $x = 1$.

Theorem 3.8 (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* weak *Kleene algebra:*

$$sR_M^* \leq \Sigma_M^*(C_M + c_1)_r + \Sigma_M^*\Sigma_M^{\neq}\rho.$$

Recall that an inequality between terms is always stronger than the corresponding inequality on languages. For soundness, we only need to assume that the inequality holds between languages, but for completeness, we want to prove that we can establish a stronger inequality between terms. These two results combined yield our main result.

Theorem 3.9 (Undecidability)**.** *Let $\Sigma \triangleq \{0, 1\}$ be a discrete commutable set. Suppose that we have a diagram of sets*

$$\begin{array}{ccc} \mathcal{T}\ddot{\Sigma} & \xrightarrow{l} & \mathcal{L}\ddot{\Sigma} \\ & {\scriptstyle e}\searrow & \uparrow \\ & & X, \end{array}$$

where e 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. Assume that equality on X is decidable. We are going to define a distinguishing function f for the languages A and B of Theorem 3.1. We can assume that those sets are defined using two-counter machines instead of Turing machines. Given a pair $\langle\langle M\rangle, \langle n\rangle\rangle$, where M is a machine and $n \in \mathbb{N}$, we proceed as follows. First, we compute the term ρ of Theorem 3.8, using $a^n b^0 \dot{q}$ as the initial configuration. Next, we find a suitable encoding of the characters of Σ_M as binary strings, which leads to the following injective embeddings:

$$\begin{array}{ccc} \mathcal{T}\ddot{\Sigma}_M & \xrightarrow{l} & \mathcal{L}\ddot{\Sigma}_M \\ \downarrow & & \downarrow \\ \mathcal{T}\ddot{\Sigma} & \xrightarrow{l} & \mathcal{L}\ddot{\Sigma} \\ & {\scriptstyle e}\searrow & \uparrow \\ & & X. \end{array}$$

Since the inequality in Theorem 3.8 is defined as an equality, we can convert that inequality into a related equality $a = b$ in X by applying e on both sides. We then check if this equality is valid. It suffices to show that this yields a valid distinguishing function.

If M outputs 1 on n, the inequality of Theorem 3.8 is valid, and thus $a = b$ is valid, which means that $f\langle\langle M\rangle, \langle n\rangle\rangle = 1$.

Otherwise, if M outputs 0, $a = b$ cannot be valid. If the equation were valid, we would get a corresponding valid inequality in $\mathcal{L}\ddot{\Sigma}$. By diagram chasing, this would yield a corresponding valid inequality in $\mathcal{L}\ddot{\Sigma}_M$. Since $\rho \leq \ddot{\Sigma}_M^*$ is valid in languages, the inequality of Theorem 3.7 would hold, which would imply that M actually outputs 1 on n; contradiction. □

Thus, to establish undecidability, we need to prove soundness and completeness. The easiest part is proving soundness: we just need to adapt Cohen's proof of undecidability of equations of regular languages [Koz96].

Proof of Theorem 3.7. Suppose that we have some finite sequence of transitions $s = s_0 \to \cdots \to s_n = c_x$. By definition, $(s_i)_l(s_{i+1})_r \le R_M$ for every $i \in \{0, \ldots, n-1\}$. Thus, we have the following inequality on languages:

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

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

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

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 $p \le \Sigma_M^*(C_M + c_1)_r$. By projecting out the right components, we find that $\pi_r(p) = s_0 \cdots s_n \le \Sigma_M^*(C_M + c_1)_r$. We cannot have $\pi_r(p) \le \Sigma_M^* C_M$, since the last character c_x cannot appear in a string in C_M. Therefore, $\pi_r(p) \le \Sigma_M^* c_1$, from which we conclude. □

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 weak 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 $s' \le R_M$ 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 finitely unfolding R_M, which can be done in the weak setting.

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 4.1. 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 4.2. *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. 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

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

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

Theorem 4.3. *If $e : \mathsf{Rel}(L)$, there exists ρ such that, given $n \in \mathbb{N}$ and a finite $\Lambda \leq L$, if $\mathsf{Next}_e^{n}(\Lambda) = \varnothing$, then*

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

Proof. Choose the same ρ as in Lemma 4.2. Then

$$\begin{aligned}
&\Lambda_r e^* \\
&\leq \Sigma^* \operatorname{Next}_e^{<n}(\Lambda)_r + \Sigma^* \operatorname{Next}_e^{n}(\Lambda)_r e^* + \Sigma^*\Sigma^{\neq}\rho && \text{by Lemma 4.2} \\
&= \Sigma^* \operatorname{Next}_e^{<n}(\Lambda)_r + \Sigma^*\Sigma^{\neq}\rho.
\end{aligned}$$

□

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 4.3. 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 can be applied to reason about them. In particular, we can analyze terms by characterizing them as automata. This connection can be defined algebraically by posing certain *derivative operations* δ_x on terms, which satisfy a *fundamental theorem*: given a term $e \in \mathcal{K}X$, we have $e = e_0 + \sum_x x\delta_x(e)$, 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 weak setting. For example, the closest we can get to an expansion for 1^* is $1^* = 1 + 1^*$, which is not quite right.

To remedy this issue, we are going to carve out a set of so-called *finite-state terms* of a weak Kleene algebra terms that enables this type of reasoning. 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 5.1. 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 $e = [e]_0 + \sum_x x\delta_x(e)$. 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 5.2. *Derivable terms are closed under all the weak Kleene algebra operations, with the following caveats: for e^*, we also require that $[e]_0 = 0$; for $e_1 e_2$, the term is also derivable if e_2 isn't, provided that $[e_1]_0 = 0$. We have the following choices of derivatives:*

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

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. 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:

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

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. □

Definition 5.3. 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 finite sum of states 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. This is convenient for the commutative setting, since a given string could be matched by choosing different orderings of its characters.

However, finite-state automata, as defined in Definition 5.3 can be hard to construct directly. We remedy this difficulty by defining the notion of *pre-automaton*, which is more flexible, and then prove that every pre-automaton can be naturally extended to an automaton.

Lemma 5.4. *Given a finite commutable set X, a* pre-automaton *is a finite set S of terms over X such that every $e \in S$ is derivable, and $\delta_x(e)$ is a sum of some elements in $S \cup \{1\}$. The set $\bar{S} \triangleq \{\sum_{i=1}^{n} e_i \mid n \in \mathbb{N}, e \in (S \cup \{1\})^n\}$ is a finite-state automaton. We refer to $\bar{S}$ as the automaton generated by the pre-automaton S.*

Proof. It is easy to show that $\bar{S}$ is finite, contains 1 and is closed under finite sums. We just need to show that it is closed under taking derivatives. This follows from Lemma 5.2. ☐

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

Lemma 5.5. *Let X be a finite commutable set. Finite-state terms are preserved by all the weak 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. Let's consider all the cases.

- The set $\{0, 1\}$ is an automaton by Lemma 5.2. Therefore, 0 and 1 are finite state.
- By Lemma 5.2, 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 5.2, 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 5.2, 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 5.2, 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. □

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

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

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. 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

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

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. □

5.2 Bounded-Output Terms

Lemma 5.6 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 theupper bound required for being representable. However, the prefixes s that appear in the terms se_s are arbitrary, and, since we are working with weak 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 weak 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 5.7. 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 5.8. *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. 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

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

Lemma 5.9. *Bounded-output terms are closed under all the weak Kleene algebra operations. For e^*, we additionally require that $|\pi_l(s)| \geq 1$ for all strings $s \leq e$.*

Proof. 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,

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

For bounded-output terms, we can improve the expansion of Lemma 5.6.

Lemma 5.10. *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*

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

where $e_s \in S$ for every s.

Proof. 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 5.6, we have

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

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

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

Thus,

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

□

Definition 5.11. 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 5.12. *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. By induction on the length of s. □

Lemma 5.13. *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. We have already seen that $\mathsf{Next}_e(\Lambda)$ is finite when Λ is (Lemma 5.8). 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

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

Let k be the constant of Lemma 5.10 for e, $n = |s|$, and let $p = (k+1)(n+1)$. Let

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

By applying Lemma 5.10 to e, we can write

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

Thus, to prove the inequality, it suffices to prove

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

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

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

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 5.8, we have $|s'| \leq (n+1)k$, and thus $|s_l s'_r| = |s| + |s'| \leq (n+1)(k+1) = p$.

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

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

Next, let us look at (5). Suppose that s' is such that $s' \leq e$ and $\pi_l(s') \neq s$. Since L is prefix free, and $\pi_l(s') \leq L$, Lemma 5.12 applied to s and s' yields

$$s_l s' \leq \Sigma^* \Sigma^{\neq} \ddot{\Sigma}^* \leq \Sigma^* \Sigma^{\neq} \ddot{\Sigma}^* \rho_e = \Sigma^* \Sigma^{\neq} \rho,$$

where we use the fact that $\rho_e \geq 1$ 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 $s' \in \Lambda$,

$$s_r s' e_{s'} \leq \Sigma^* \Sigma^{\neq} \rho.$$

If $e_{s'} = 0$, we are done. Otherwise, by Corollary 2.6, we can find some string $s'' \leq e_{s'}$. We have $s' s'' \leq s' e_{s'} \leq e$.

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

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

which is a contradiction.

Since $\pi_l(s' s'') \leq \pi_l(e) \leq L$ and L is prefix free, by Lemma 5.12, 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 $x \neq x'$. 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 $s_r s' = s_r \pi_l(s') \pi_r(s') \leq \Sigma^* \Sigma^{\neq} \ddot{\Sigma}^*$, and thus

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

5.3 Putting Everything Together

To derive completeness for two-counter machines (Theorem 3.8), it suffices to show that the hypotheses of Lemma 5.13 are satisfied.

Lemma 5.14. *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 5.3).*
- R_M *has bounded output (Definition 5.7).*

Thus, by Lemma 5.13, the term R_M *is a representable relation of type* $\mathsf{Rel}(T_M)$.

Proof. To show that T_M is prefix free, we note that every string $s \leq T_M$ must be of the form $s'x$, where $x \in Q_M \cup \{c_1, c_0\}$ and s' does not contain any such symbols. Thus, any proper prefix of such a string cannot lie in T_M.

The assertions about $\pi_l(R_M)$ and $\pi_r(R_M)$ have similar proofs, so we focus on the second one. First, we prove that, for any instruction $i \in I_M$, $\pi_r((\!|i|\!)) \leq T_M$. Let us consider the example of Inc; the others are analogous:

$$\begin{aligned}
\pi_r(\mathsf{Inc}(1, q)) &= \pi_r(a_r a^* b^* q_r) \\
&= aa^* b^* q \\
&\leq a^* b^* q && \text{because } aa^* \leq a^* \\
&\leq C_M \\
&\leq T_M.
\end{aligned}$$

Thus,

$$\begin{aligned}
\pi_r(R_M) &= \sum_{q \in Q_M} \pi_r((\!|\iota(q)|\!) q_l) \\
&= \sum_{q \in Q_M} \pi_r((\!|\iota(q)|\!)) \\
&\leq \sum_{q \in Q_M} T_M \\
&= T_M.
\end{aligned}$$

To show the next two points, we just have to appeal to the closure properties of finite-state and bounded-output terms (Lemmas 5.5 and 5.9). These lemmas say that these properties are always preserved by all the algebra operations, except possibly for star. For star, we need to check that the starred sub-terms do not contain 1 and that they only contain strings with at least one left symbol. The starred sub-terms are just $a_l a_r$ and $b_l b_r$, both of which satisfy this property.

□

We can finally conclude with the proof of completeness, thus establishing undecidability (Theorem 3.9).

Proof of Theorem 3.8. 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 5.14, 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 4.3. 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 [Koz97] established several hardness and completeness results for variants of Kleene algebra. He noted that a proof of Cohen's [Koz96] showed that deciding equality of regular languages with commutativity conditions on atoms ($\mathcal{L}X$) was not possible—more precisely, the problem is Π_1^0-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 [Kuz23], who independently proved a similar undecidability result. In terms of techniques, both of our works draw inspiration from Cohen's proof of Π_1^0-completeness. Leveraging the reduction of the halting problem to the PCP, Kuznetsov extends Cohen's idea and uses 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. We can show that the language of pairs $\langle\langle M\rangle, x\rangle$ of 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 we cannot decide such inequalities.

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. 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.

In terms of the end results, Kuznetsov's are stronger, in that he proves also Σ^0_1-completeness of the equational theory of $\mathcal{KX}$ (not just undecidability), but also weaker, in that his results are stated for $\mathcal{KX}$, and do not work as is for a more general theory like $\mathcal{TX}$. However, we believe that the two results could be reconciled. Kuznetsov's proof of Σ^0_1-completeness uses *effective inseparability*, a refinement of recursive inseparability. Roughly speaking, two sets A and B are effectively inseparable if there exists a computable function f such that, when each set is overapproximated by disjoint recursively enumerable sets A' and B', the function produces an output that does not belong to either A' or B' (using codes for these sets as an input). According to a standard result of computability theory (see Kuznetsov [Kuz23] for references), two effectively inseparable sets that are recursively enumerable are Σ^0_1-complete. The sets involved in Theorem 3.1 are known to be effectively inseparable, so we believe that our results could be adapted to show Σ^0_1-completeness as well. On the other hand, Kuznetsov does require 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 $(-)^*$. We believe that it might be possible to adapt his inequalities so that these identities are not needed, perhaps by reusing our idea of keeping a residue term ρ in Theorem 3.8.

References

[AK01] Allegra Angus and Dexter Kozen. *Kleene Algebra with Tests and Program Schematology.* USA, June 2001.

[And+14] Carolyn Jane Anderson et al. "NetKAT: semantic foundations for networks". In: *ACM SIGPLAN Notices* 49.1 (Jan. 2014), pp. 113–126. ISSN: 0362-1340. DOI: 10.1145/2578855.2535862.

[Ant+22] Timos Antonopoulos et al. "An algebra of alignment for relational verification". In: arXiv:2202.04278 (July 2022). arXiv:2202.04278 [cs]. DOI: 10.48550/arXiv.2202.04278. URL: http://arxiv.org/abs/2202.04278.

[DM97] Volker Diekert and Yves Métivier. "Partial Commutation and Traces". In: *Handbook of Formal Languages: Volume 3 Beyond Words.* Ed. by Grzegorz Rozenberg and Arto Salomaa. Berlin, Heidelberg: Springer, 1997, pp. 457–533. ISBN: 978-3-642-59126-6. DOI: 10.1007/978-3-642-59126-6_8. (Visited on 04/05/2024).

[Fos+15] Nate Foster et al. "A Coalgebraic Decision Procedure for NetKAT". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* POPL '15. New York, NY, USA: Association for Computing Machinery, Jan. 2015, pp. 343–355. ISBN: 978-1-4503-3300-9. DOI: 10.1145/2676726.2677011. URL: https://doi.org/10.1145/2676726.2677011.

[HMU01] 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. ISBN: 9780201441246. URL: https://books.google.com/books?id=omIPAQAAMAAJ.

[Hoa+09] C. Hoare et al. "Concurrent Kleene Algebra". In: vol. 5710. Sept. 2009, pp. 399–414. ISBN: 978-3-642-04080-1. DOI: 10.1007/978-3-642-04081-8_27.

[Kap+18] Tobias Kappé et al. "Concurrent Kleene Algebra: Free Model and Completeness". en. In: *Programming Languages and Systems.* Ed. by Amal Ahmed. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 856–882. ISBN: 978-3-319-89884-1. DOI: 10.1007/978-3-319-89884-1_30.

[Kap+20] Tobias Kappé et al. "Concurrent Kleene Algebra with Observations: From Hypotheses to Completeness". en. In: *Foundations of Software Science and Computation Structures.* Ed. by Jean Goubault-Larrecq and Barbara König. Vol. 12077. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 381–400. ISBN: 978-3-030-45230-8. DOI: 10.1007/978-3-030-45231-5_20. URL: http://link.springer.com/10.1007/978-3-030-45231-5_20.

[Koz96] Dexter Kozen. "Kleene algebra with tests and commutativity conditions". en. In: *Tools and Algorithms for the Construction and Analysis of Systems.* Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 1055. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 14–33. ISBN: 978-3-540-61042-7. DOI: 10.1007/3-540-61042-1_35. URL: http://link.springer.com/10.1007/3-540-61042-1_35.

[Koz97] 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.* IEEE Computer Society, 1997, pp. 195–202. DOI: 10.1109/LICS.1997.614947. URL: https://doi.org/10.1109/LICS.1997.614947.

[Kuz23] Stepan L. Kuznetsov. "On the Complexity of Reasoning in Kleene Algebra with Commutativity Conditions". In: *Theoretical Aspects of Computing - ICTAC 2023 - 20th International Colloquium, Lima, Peru, December 4-8, 2023, Proceedings.* Ed. by Erika Ábrahám, Clemens Dubslaff, and Silvia Lizeth Tapia Tarifa. Vol. 14446. Lecture Notes in Computer Science. Springer, 2023, pp. 83–99. DOI: 10.1007/978-3-031-47963-2_7. URL: https://doi.org/10.1007/978-3-031-47963-2%5C_7.

[MCM06] A. K. McIver, E. Cohen, and C. C. Morgan. "Using Probabilistic Kleene Algebra for Protocol Verification". en. In: *Relations and Kleene Algebra in Computer Science.* Ed. by Renate A. Schmidt. Vol. 4136. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 296–310. ISBN: 978-3-540-37873-0. DOI: 10.1007/11828563_20. URL: http://link.springer.com/10.1007/11828563_20.

[MRS11] Annabelle McIver, Tahiry Rabehaja, and Georg Struth. *On Probabilistic Kleene Algebras, Automata and Simulations*. Vol. 6663. May 2011, p. 279. ISBN: 978-3-642-21069-3. DOI: 10.1007/978-3-642-21070-9_20.

[ZAG22] Cheng Zhang, Arthur Azevedo de Amorim, and Marco Gaboardi. "On Incorrectness Logic and Kleene Algebra with Top and Tests". In: arXiv:2108.07707 (Aug. 2022). arXiv:2108.07707 [cs]. DOI: 10.48550/arXiv.2108.07707. URL: http://arxiv.org/abs/2108.07707.