The Logic of Action Lattices is Undecidable

Stepan Kuznetsov
Steklov Mathematical Institute of the RAS
8 Gubkina St., Moscow, Russia
National Research University Higher School of Economics
3 Kochnovsky Pr., Moscow, Russia
Email: skuzn@inbox.ru

***Abstract*—We prove algorithmic undecidability of the (in)equational theory of residuated Kleene lattices (action lattices), thus solving a problem left open by D. Kozen, P. Jipsen, W. Buszkowski.**

Published by IEEE as S. Kuznetsov, The logic of action lattices is undecidable. In: Proc. 34th Annual ACM/IEEE Symposium on Logic in Computer Science, IEEE, 2019. https://ieeexplore.ieee.org/document/8785659

I. Introduction

Kleene algebras form one of the classic notions of theoretical computer science, going back to the seminal paper of S. C. Kleene [23]. Kleene algebras use the standard set of regular expression operations: $\cdot$ (product), $+$ (join; further we denote it by $\vee$), and, most interestingly, the iteration operation, * (Kleene star).

Kleene [23] informally interpreted elements of a Kleene algebra as types of *events*. This interpretation gives an intuition of the Kleene algebra operations: $a \cdot b$ means event a followed by event b; $a + b$ means an event which is either a or b; a^* is a repeated several times (maybe zero[1]).

The join operation yields a semi-lattice preorder on the Kleene algebra: $a \preceq b$ iff $a \vee b = b$. In the informal interpretation, $a \preceq b$ means that a is a narrower type of events, than b (every event of type a is also of type b, but maybe not the other way round).

The notion of *action algebra,* or residuated Kleene algebra, was introduced by V. Pratt [30] as an extension of Kleene algebra with division operations (residuals). D. Kozen [18] extended it further to *action lattices* by adding the meet operation ($\wedge$).

Both extensions—adding residuals and meet—were motivated by the fact that the classes of algebras in the extended setting happened to have better properties than the original ones. Namely, residuated Kleene algebras form a finitely based variety [30], while Kleene algebras without residuals do not [31], [7]. For residuated Kleene lattices, the algebra of matrices over such a lattice is also a residuated Kleene lattice, while this does not hold for residuated Kleene algebras [18].

Residuals also fit the original paradigm of Kleene [23], where elements of the algebra represent events (or actions) performed in a system. Namely, $a \to b$ (resp., $b \leftarrow a$) could be interpreted as follows: this is an event which, if preceded (resp., followed) by an event of type a, becomes an event of type b.

The formal definition of action lattice is as follows.

Definition 1. An action lattice is a structure $\langle \mathcal{A}; \preceq, \vee, \wedge, \mathbf{0}, \cdot, \mathbf{1}, \leftarrow, \to, ^* \rangle$, where:[2]

1) $\langle \mathcal{A}; \preceq, \vee, \wedge \rangle$ is a lattice, $\mathbf{0}$ is its minimal element ($\mathbf{0} \preceq a$ for any $a \in \mathcal{A}$);
2) $\langle \mathcal{A}; \cdot, \mathbf{1} \rangle$ is a monoid;
3) $\leftarrow$ and $\to$ are residuals of $\cdot$ w.r.t. $\preceq$, *i.e.,*
$$a \preceq c \leftarrow b \iff a \cdot b \preceq c \iff b \preceq a \to c;$$
4) a^* is the least element b such that $\mathbf{1} \vee a \cdot b \preceq b$ (in other words: $\mathbf{1} \preceq a^*$, $a \cdot a^* \preceq a^*$; if $\mathbf{1} \preceq b$ and $a \cdot b \preceq b$, then $a^* \preceq b$).

The presence of residuals makes many desired properties of our algebras automatically true, so we do not need to postulate them explicitly. These include:

- monotonicity of $\cdot$ w.r.t. $\preceq$ (shown by J. Lambek [26]); residuals are monotone by one argument and anti-monotone by the other one;
- despite the asymmetry of the condition for Kleene star, the dual one also holds: a^* is also the least b such that $\mathbf{1} \vee b \cdot a \preceq b$ (shown by Pratt [30]); without residuals, there exist left and right Kleene algebras [16];
- the zero element is the annihilator w.r.t. $\cdot$: $\mathbf{0} \cdot a = a \cdot \mathbf{0} = \mathbf{0}$ for any $a \in \mathcal{A}$.

Standard examples of action lattices include the algebra of languages over an alphabet and the algebra of binary relations on a set (with * being the reflexive-transitive closure). Action lattices of these two classes are **-continuous* in the sense of the following definition:

Definition 2. An action lattice is *-continuous, if, for any $a \in \mathcal{A}$, $a^* = \sup\{a^n \mid n \in \omega\}$ (where ω denotes the set of all natural numbers, including 0).

In the presence of residuals, we do not need the context in the definition of *-continuity (for Kleene algebras without

[1] Wishing to avoid the empty event ("nothing happened"), Kleene considered a compound connective a^*b, meaning "several times a followed by b."

[2] Here we use the notation $\to$ and $\leftarrow$ for residuals, as Pratt, Kozen, and Buszkowski do; J. Lambek [26] denotes them as division operations: $\backslash$, $/$. For other operations, $\cdot$, $\vee$, and $\wedge$, we follow Buszkowski; Kozen denotes them as ;, $+$, and $\cdot$ respectively. The Kleene star is always denoted by *.

978-1-7281-3608-0/19/$31.00

residuals, the condition is as follows: $b \cdot a^* \cdot c = \sup\{b \cdot a^n \cdot c \mid n \in \omega\}$); *-continuity makes other conditions on the Kleene star (item 4 in Definition 1) redundant. Non-*-continuous action lattices also do exist; concrete examples are given in [25].

In this paper, we are interested in (in)equational theories, or *algebraic logics,* of action lattices. Statements of such theories are of the form $\alpha \preceq \beta$, where α and β are terms (formulae) constructed from variables and constants $\mathbf{1}$ and $\mathbf{0}$, using the operations of action lattices: $\cdot$, $\to$, $\leftarrow$, $\vee$, $\wedge$, *. Statements which are true under all interpretations of variables over arbitrary action lattices form *action logic,* denoted by $\mathbf{ACT}$. If we consider only *-continuous action lattices, we get $\mathbf{ACT}_\omega$, as an extension of $\mathbf{ACT}$.

For Kleene algebras, without residuals and meet, these theories were exhaustively studied by Kozen [17] and Krob [24]. The equational theory of *-continuous Kleene algebras coincides with that of all Kleene algebras, and the derivability problem for this theory is algorithmically decidable and belongs to PSPACE.

In computer science, the usage of Kleene algebras is connected to reasoning about program correctness. Such applications arise not in the pure setting of Kleene algebras, but rather utilise extensions of those—remarkable examples include Kleene algebras with tests (KAT) [21], concurrent Kleene algebras (CKA) [12], nominal Kleene algebras [10], [22], [2], and others. In the view of these applications, decidability of natural algorithmic problems dealing with a given extension of Kleene algebra becomes a desired property. In this paper, we focus on decidability of equational theories, as one of the simplest algorithmic problems. For KAT and CKA, equational theories are decidable and belong, respectively, to PSPACE [19] and to EXPSPACE [3].

Residuated Kleene algebras or lattices, theoretically, could have also found their place in this series of extensions of Kleene algebra. However, the situation with residuals is dramatically different. Though the inequational theories for both Kleene algebras without residuals and meet and residuated lattices without the Kleene star belong to PSPACE (the latter is due to its connection to non-commutative linear logic [27], [14]), the combination of them rises algorithmic complexity to a higher level. The negative results presented in this paper, in a sense, work against possible applications of residuated Kleene lattices in the framework of reasoning about programs, or at least require such applications to be more sophisticated and accurate.

Logics of action lattices, $\mathbf{ACT}$ and $\mathbf{ACT}_\omega$, can be axiomatized using Gentzen-style sequent calculi. In these calculi, derivable objects are *sequents* of the form $\Pi \vdash \beta$, where β is a formula and Π is a finite linearly ordered sequence of formulae (possibly empty: the empty sequence is denoted by Λ). The sequent $\alpha_1, \dots, \alpha_n \vdash \beta$ means $\alpha_1 \cdot \ldots \cdot \alpha_n \preceq \beta$ (due to associativity, we need no brackets here); $\Lambda \vdash \beta$ is interpreted as $\mathbf{1} \preceq \beta$.

Both $\mathbf{ACT}$ and $\mathbf{ACT}_\omega$ are extensions of the multiplicative-additive Lambek calculus ($\mathbf{MALC}$), which is the logic of residuated lattices without Kleene star [28]. Axioms and inference rules of $\mathbf{MALC}$ are presented in Figure 1; $\mathbf{ACT}$ and $\mathbf{ACT}_\omega$ are extensions of $\mathbf{MALC}$ by inference rules presented in Figures 2 and 3 respectively.[3] Rules of $\mathbf{ACT}$ are admissible in $\mathbf{ACT}_\omega$, therefore the latter is an extension of the former (if we consider them as sets of theorems, we can write $\mathbf{ACT} \subseteq \mathbf{ACT}_\omega$).

Let us comment a bit more on the $(^*\vdash)_\omega$ rule. The notation α^n in it is a shortcut for $\alpha, \dots, \alpha$ (n times). This rule has infinitely many premises, indexed by natural numbers: $\Gamma, \Delta \vdash \gamma$; $\Gamma, \alpha, \Delta \vdash \gamma$; $\Gamma, \alpha, \alpha, \Delta \vdash \gamma$; …, in other words, it is an ω-rule. In the presence of the ω-rule, the notion of derivation needs to be clarified. Now a derivation tree can be infinite, but is still required to be well-founded, *i.e.,* it is not allowed to include infinite paths: each path going upwards from the goal sequent should reach an axiom instance in a finite number of steps.

By the standard Lindenbaum – Tarski construction, these axiomatizations are sound and complete w.r.t. $\mathbf{ACT}$ and $\mathbf{ACT}_\omega$ respectively.

Notice that we include cut as an official rule of the system only in $\mathbf{ACT}$. Indeed, in $\mathbf{ACT}_\omega$, as shown by Palka [29], cut is eliminable, while for $\mathbf{ACT}$ no cut-free system is known. An attempt to construct such a system was taken by P. Jipsen [13], but Buszkowski [5] showed that in Jipsen's system cut is not eliminable. Nevertheless, both $\mathbf{ACT}$ and $\mathbf{ACT}_\omega$ are conservative over $\mathbf{MALC}$:

Proposition 1. *If $\Pi \vdash \beta$ does not include *, then the following are equivalent:*

1) *$\Pi \vdash \beta$ is derivable in $\mathbf{MALC}$;*
2) *$\Pi \vdash \beta$ is derivable in $\mathbf{ACT}$;*
3) *$\Pi \vdash \beta$ is derivable in $\mathbf{ACT}_\omega$.*

Proof. Implications $1 \Rightarrow 2$ and $2 \Rightarrow 3$ are trivial, and $3 \Rightarrow 1$ is due to cut elimination in $\mathbf{ACT}_\omega$: all formulae appearing in a cut-free derivation are subformulae of the goal sequent; thus, if we do not have * in $\Pi \vdash \beta$, we do not need it in the derivation. □

Admissibility of cut also yields invertibility of some rules in $\mathbf{ACT}_\omega$:

Proposition 2. *Rules $(\vdash\to)$, $(\vdash\leftarrow)$, $(\cdot\vdash)$, $(\vdash\wedge)$, $(\vee\vdash)$, and $(^*\vdash)_\omega$ in $\mathbf{ACT}_\omega$ are invertible, i.e.:*

1) *if $\Pi \vdash \alpha \to \beta$ is derivable, then so is $\alpha, \Pi \vdash \beta$;*
2) *if $\Pi \vdash \beta \leftarrow \alpha$ is derivable, then so is $\Pi, \alpha \vdash \beta$;*
3) *if $\Gamma, \alpha \cdot \beta, \Delta \vdash \gamma$ is derivable, then so is $\Gamma, \alpha, \beta, \Delta \vdash \gamma$;*
4) *if $\Pi \vdash \alpha_1 \wedge \alpha_2$ is derivable, then so are both $\Pi \vdash \alpha_1$ and $\Pi \vdash \alpha_2$;*
5) *if $\Gamma, \alpha_1 \vee \alpha_2, \Delta \vdash \gamma$ is derivable, then so are both $\Gamma, \alpha_1, \Delta \vdash \gamma$ and $\Gamma, \alpha_2, \Delta \vdash \gamma$;*
6) *if $\Gamma, \alpha^*, \Delta \vdash \gamma$ is derivable, then so is $\Gamma, \alpha^n, \Delta \vdash \gamma$ for any natural n.*

[3] Conditions on the Kleene star in the definitions of action lattices given by Kozen [18], by Buszkowski [5], and in the present paper, are slightly different, so are the corresponding rules in $\mathbf{ACT}$. Equivalence of these variants of definition, however, was shown already by Pratt [30].

$$\frac{}{\alpha \vdash \alpha}\ (\mathrm{ax})$$

$$\frac{\Pi \vdash \alpha \quad \Gamma, \beta, \Delta \vdash \gamma}{\Gamma, \Pi, \alpha \to \beta, \Delta \vdash \gamma}\ (\to\vdash) \qquad \frac{\alpha, \Pi \vdash \beta}{\Pi \vdash \alpha \to \beta}\ (\vdash\to)$$

$$\frac{\Pi \vdash \alpha \quad \Gamma, \beta, \Delta \vdash \gamma}{\Gamma, \beta \leftarrow \alpha, \Pi, \Delta \vdash \gamma}\ (\leftarrow\vdash) \qquad \frac{\Pi, \alpha \vdash \beta}{\Pi \vdash \beta \leftarrow \alpha}\ (\vdash\leftarrow)$$

$$\frac{\Gamma, \alpha, \beta, \Delta \vdash \gamma}{\Gamma, \alpha \cdot \beta, \Delta \vdash \gamma}\ (\cdot\vdash) \qquad \frac{\Gamma \vdash \alpha \quad \Delta \vdash \beta}{\Gamma, \Delta \vdash \alpha \cdot \beta}\ (\vdash\cdot)$$

$$\frac{\Gamma, \Delta \vdash \gamma}{\Gamma, \mathbf{1}, \Delta \vdash \gamma}\ (\mathbf{1}\vdash) \qquad \frac{}{\Lambda \vdash \mathbf{1}}\ (\vdash \mathbf{1}) \qquad \frac{}{\Gamma, \mathbf{0}, \Delta \vdash \gamma}\ (\mathbf{0}\vdash)$$

$$\frac{\Gamma, \alpha_i, \Delta \vdash \gamma}{\Gamma, \alpha_1 \wedge \alpha_2, \Delta \vdash \gamma}\ (\wedge\vdash)_i,\ i = 1, 2 \qquad \frac{\Pi \vdash \alpha_1 \quad \Pi \vdash \alpha_2}{\Pi \vdash \alpha_1 \wedge \alpha_2}\ (\vdash\wedge)$$

$$\frac{\Gamma, \alpha_1, \Delta \vdash \gamma \quad \Gamma, \alpha_2, \Delta \vdash \gamma}{\Gamma, \alpha_1 \vee \alpha_2, \Delta \vdash \gamma}\ (\vee\vdash) \qquad \frac{\Pi \vdash \alpha_i}{\Pi \vdash \alpha_1 \vee \alpha_2}\ (\vdash\vee)_i,\ i = 1, 2$$

Fig. 1. The multiplicative-additive Lambek calculus (**MALC**)

$$\frac{\Lambda \vdash \beta \quad \alpha, \beta \vdash \beta}{\alpha^* \vdash \beta}\ (^*\vdash)_{\mathrm{fp}} \qquad \frac{}{\vdash \alpha^*}\ (\vdash {}^*)_0 \qquad \frac{\Pi \vdash \alpha \quad \Delta \vdash \alpha^*}{\Pi, \Delta \vdash \alpha^*}\ (\vdash {}^*)_{\mathrm{fp}} \qquad \frac{\Pi \vdash \alpha \quad \Gamma, \alpha, \Delta \vdash \gamma}{\Gamma, \Pi, \Delta \vdash \gamma}\ (\mathrm{cut})$$

Fig. 2. **ACT** rules for Kleene star

$$\frac{\big(\Gamma, \alpha^n, \Delta \vdash \gamma\big)_{n \in \omega}}{\Gamma, \alpha^*, \Delta \vdash \gamma}\ (^*\vdash)_\omega \qquad \frac{\Pi_1 \vdash \alpha \quad \ldots \quad \Pi_n \vdash \alpha}{\Pi_1, \ldots, \Pi_n \vdash \alpha^*}\ (\vdash {}^*)_n,\ n \in \omega$$

Fig. 3. $\mathbf{ACT}_\omega$ rules for Kleene star

Proof. Straightforwardly, using cut. □

For $\mathbf{ACT}_\omega$, algorithmic complexity is established by the following theorem:

Theorem 1 (W. Buszkowski and E. Palka, 2007)**.** $\mathbf{ACT}_\omega$ *is* Π^0_1*-complete and, thus, not decidable and not recursively enumerable.*

In this theorem, Buszkowski [5] proved the lower bound, Π^0_1-hardness, and Palka [29] proved the upper one, $\mathbf{ACT}_\omega \in \Pi^0_1$, or co-recursive-enumerability.

Being Π^0_1-hard, $\mathbf{ACT}_\omega$ is not recursively enumerable (and, therefore, cannot be formulated as a system with finite proofs). On the other side, **ACT** is clearly recursively enumerable (belongs to Σ^0_1), thus it is strictly weaker than $\mathbf{ACT}_\omega$. (A concrete example of a sequent derivable in $\mathbf{ACT}_\omega$, but not in **ACT** is given in [25].)

II. Outline

Decidability of **ACT**, the logic of the whole class of action lattices, remained an open question, raised by Kozen [18], Jipsen [13], and Buszkowski [5], [6]. We give a negative answer:

Theorem 2. **ACT** *is undecidable.*

Before going into technical details, let us sketch the general ideas behind the proof of Theorem 2.

In his proof of Π^0_1-hardness of $\mathbf{ACT}_\omega$, Buszkowski [5] uses an encoding of the totality problem for context-free grammars, that is, the question whether a given context-free grammar generates all (non-empty) words over its terminal alphabet. For the totality problem, in its turn, there is a well-known reduction from the *complement* of the halting problem for Turing machines. Namely, given a deterministic Turing machine $\mathfrak{M}$ and its input word x, one can efficiently construct a context-free grammar $\mathcal{G}_{\mathfrak{M},x}$, which generates all non-empty words if and only if $\mathfrak{M}$ does not halt on x. Next, by Buszkowski's construction, one efficiently constructs a sequent $\Gamma_{\mathfrak{M},x} \vdash S$ (here S is a fixed variable), such that this sequent is derivable in $\mathbf{ACT}_\omega$ if and only if machine $\mathfrak{M}$ does *not* halt (runs infinitely long) on input x. Details of these translations

are presented in Sections IV and V, along with the necessary modifications for our needs. Since the halting problem is Σ_1^0-hard, its complement is Π_1^0-hard, thus so is $\mathbf{ACT}_\omega$.

Let us say that "$\mathbf{ACT}_\omega$ proves that $\mathfrak{M}$ does not halt on x," if $\mathbf{ACT}_\omega$ derives $\Gamma_{\mathfrak{M},x} \vdash S$. Our idea is to show that in some cases $\mathbf{ACT}$ is already sufficient to prove non-halting of $\mathfrak{M}$ on x. Namely, after a modification of Buszkowski's encoding, $\mathbf{ACT}$ becomes capable of proving non-halting of Turing machines which are *trivially cycling*.

Definition 3. A state q_C of a Turing machine $\mathfrak{M}$ is called a *cycling* one, if the rules for q_C are $\langle q_C, a\rangle \rightsquigarrow \langle q_C, a, N\rangle$ for any letter a of the internal alphabet of $\mathfrak{M}$; here N stands for "no move." In other words, once $\mathfrak{M}$ reaches q_C, it gets stuck in this state.

Notice that a cycling state is *a priori* not required to be reachable. The definition of a cycling state thus does not refer to the execution of $\mathfrak{M}$, and cycling states can be found algorithmically, given the code of $\mathfrak{M}$. Runtime is handled by the next definition.

Definition 4. A Turing machine $\mathfrak{M}$ *trivially cycles* on input x, if $\mathfrak{M}$ reaches a cycling state q_C while running on x.

Our modification of Buszkowski's construction (presented in Sections IV and V) replaces $\mathcal{G}_{\mathfrak{M},x}$ by $\mathcal{G}'_{\mathfrak{M},x}$ and thus $\Gamma_{\mathfrak{M},x}$ by the corresponding $\Gamma'_{\mathfrak{M},x}$. The new grammar $\mathcal{G}'_{\mathfrak{M},x}$ generates the same language as $\mathcal{G}_{\mathfrak{M},x}$, but has some redundant production rules, which make proving $\Gamma'_{\mathfrak{M},x} \vdash S$ easier in some cases. From the point of view of $\mathbf{ACT}_\omega$, this replacement changes nothing: $\Gamma'_{\mathfrak{M},x} \vdash S$ is still derivable in $\mathbf{ACT}_\omega$ if and only if $\mathfrak{M}$ does not halt on x.

Now, however, the class of pairs $\langle \mathfrak{M}, x\rangle$ for which $\mathbf{ACT}$, the weaker system, proves non-halting (*i.e.,* derives the sequent $\Gamma'_{\mathfrak{M},x} \vdash S$) includes the class $\mathcal{C}$ of all pairs where $\mathfrak{M}$ trivially cycles on x. On the other hand, this class is disjoint with the class $\mathcal{H}$ of all pairs where $\mathfrak{M}$ halts on x: if $\mathbf{ACT}$ proves non-halting for a pair $\langle \mathfrak{M}, x\rangle$, then so does $\mathbf{ACT}_\omega$, and therefore $\mathfrak{M}$ indeed does not halt on x.

Next, there is a folklore fact (see Proposition 10 below), that $\mathcal{H}$ and $\mathcal{C}$ are *recursively inseparable,* that is, there is no decidable class $\mathcal{K}$ of pairs $\langle \mathfrak{M}, x\rangle$ which includes $\mathcal{C}$ and is disjoint with $\mathcal{H}$. Thus, the class of pairs for which $\mathbf{ACT}$ proves non-halting is undecidable, and so is $\mathbf{ACT}$ itself.

This finishes the proof of Theorem 2, and even gives a stronger result:

Theorem 3. *If $\mathbf{ACT} \subseteq \mathcal{L} \subseteq \mathbf{ACT}_\omega$, then $\mathcal{L}$ is undecidable.*

In the following sections, we present the proof of Theorem 3 in detail.

III. The "long rule"

In this section we formulate an admissible rule in $\mathbf{ACT}$, which will be used later. First we introduce a shortcut notation $\psi^+ = \psi \cdot \psi^*$ for positive iteration and recall that $\psi^n = \psi, \ldots, \psi$ (n times; notice that here we use comma, not multiplication).

Next, we prove the following technical lemma:

Lemma 3. *For any natural n, the following "long rule" is admissible in* $\mathbf{ACT}$*:*

$$\frac{\psi \vdash \gamma \quad \psi^2 \vdash \gamma \quad \ldots \quad \psi^n \vdash \gamma \quad \psi^n, \psi^+ \vdash \gamma}{\psi^+ \vdash \gamma}$$

Proof. Induction on n. For $n = 0$ the rule is trivial. For the induction step, we use the fact that $\psi^+ \vdash \psi \vee (\psi \cdot \psi^+)$ is derivable in $\mathbf{ACT}$ (established straightforwardly).

Next, apply an instance of the long rule for $n-1$, admissible by induction hypothesis:

$$\frac{\psi \vdash \gamma \quad \psi^2 \vdash \gamma \quad \ldots \quad \psi^{n-1} \vdash \gamma \quad \psi^{n-1}, \psi^+ \vdash \gamma}{\psi^+ \vdash \gamma}$$

The first $n-1$ premises of this rule are given. The last one is derived as follows:

$$\dfrac{\psi^+ \vdash \psi \vee (\psi \cdot \psi^+) \qquad \dfrac{\psi^n \vdash \gamma \qquad \dfrac{\psi^n, \psi^+ \vdash \gamma}{\psi^{n-1}, \psi \cdot \psi^+ \vdash \gamma}\,(\cdot\vdash)}{\psi^{n-1}, \psi \vee (\psi \cdot \psi^+) \vdash \gamma}\,(\vee\vdash)}{\psi^{n-1}, \psi^+ \vdash \gamma}\,(\text{cut})$$

□

IV. Encoding I: non-halting of Turing machines to totality of context-free grammars

In this section we recall a standard result—Π_1^0-completeness of the totality problem for context-free grammars. We choose the construction with direct encoding of Turing machines, since further we are going to analyze the behaviour of Turing machines in a more refined way. Such a proof can be found, for example, in textbooks by Kozen [20], Sipser [32], and by Du and Ko [9]. Other proofs could have a detour, for example, via Post's correspondence problem, which would be inconvenient for us. In this section we revisit this proof and modify it for our further needs.

We consider only *deterministic* Turing machines operating on $\{0, 1\}$ as the input alphabet. When talking about *halting,* we consider any situation when the machine cannot perform the next operation: for simplicity, we have no designated "accepting" state(s) in the Turing machine, and every finish of the execution process is considered "successful." Thus, the *complement* to halting is only *infinite execution,* in which the machine, from a given starting configuration, can perform arbitrarily many steps of execution.

For a Turing machine $\mathfrak{M}$, by $\Sigma_{\mathfrak{M}}$ we denote its internal alphabet and $Q_{\mathfrak{M}}$ is the set of the states of $\mathfrak{M}$ (for simplicity we further write just Σ and Q). We suppose that Σ and Q are disjoint ($\Sigma \cap Q = \varnothing$) and that $\#$ is a fresh symbol ($\# \notin \Sigma \cup Q$).

For each pair $\langle \mathfrak{M}, x\rangle$ of a Turing machine and its input word one constructs a context-free grammar $\mathcal{G}_{\mathfrak{M},x}$ over alphabet $\Sigma \cup Q \cup \{\#\}$ with the following property: $\mathcal{G}_{\mathfrak{M},x}$ generates all non-empty words over this alphabet (*i.e.,* has the *totality* property) if and only if $\mathfrak{M}$ does not halt on input x. This construction is going to be efficient in the sense that the function $\langle \mathfrak{M}, x\rangle \mapsto \mathcal{G}_{\mathfrak{M},x}$ is computable.

The *configuration* of $\mathfrak{M}$ when it is in state $q \in Q$ with the word $a_1 a_2 \dots a_n \in \Sigma^*$ in its memory, and observing its letter a_i, is encoded by the following word:

$$k = a_1 \dots a_{i-1} q a_i \dots a_n$$

(if the memory is empty, then $k = q\textvisiblespace$, where $\textvisiblespace$ is the blank symbol). A correct *protocol* of execution of $\mathfrak{M}$ on input x is a sequence of configurations $k_0, k_1, \dots, k_m$, where $k_0 = q_0 x$ is the initial configuration (q_0 is the initial state) and each k_{i+1} is the successor configuration of k_i, according to the program of $\mathfrak{M}$. (Since $\mathfrak{M}$ is deterministic, the successor, if it exists, is unique.) A protocol is encoded by the following word:

$$\begin{array}{ll} \#k_0\#k_1^R\#k_2\#k_3^R\#\dots\#k_m\# & \text{if } m \text{ is even;} \\ \#k_0\#k_1^R\#k_2\#k_3^R\#\dots\#k_m^R\# & \text{if } m \text{ is odd.} \end{array}$$

Notation w^R means w written in the reversed letter order.[4]

Such a protocol is a *halting* one, if k_m has no successor. Otherwise the protocol can be continued further.

We design the context-free grammar $\mathcal{G}_{\mathfrak{M},x}$ that will generate all words over $\Sigma \cup Q \cup \{\#\}$ that *do not include a halting protocol* as a prefix. Thus, $\mathcal{G}_{\mathfrak{M},x}$ generates all non-empty words if and only if no halting protocol exists, *i.e.,* $\mathfrak{M}$ does not halt on x (see Lemma 4 below).

The desired set of words is the union of the following ones:

- words not starting with $\#$;
- words of only one letter;
- $\mathbf{T}$ ("trash"): words starting with $\#$, but such that either between some two occurrences of $\#$ there is a word that is not a code of a configuration or reversed code of a configuration (*i.e.,* has 0 or more than 1 letters from Q), or such that its prefix before the second $\#$ is not $\#q_0x$;
- $\mathbf{E}$ ("error"): starting with $\#$, but includes, between two $\#$'s, a code of a configuration k_i (maybe reversed) whose successor is k_{i+1} (in particular, it *has* a successor), but the next block between $\#$'s is *not* k_{i+1}^R;
- $\mathbf{P}$ ("prefix"): starts with $\#$ and either does not end on $\#$, or ends on $\#k_m\#$ (or $\#k_m^R\#$, depending on parity), where k_m is a configuration that has a successor. Words from $\mathbf{P}$, unless they also belong to $\mathbf{T}$ or $\mathbf{E}$, encode *incomplete* execution protocols.

Notice that languages $\mathbf{T}$ and $\mathbf{P}$ are regular and therefore context-free. The $\mathbf{E}$ language can be generated by a non-deterministic pushdown automaton. This automaton first non-deterministically guesses the number i for which k_i is not followed by its reversed successor. Then it puts k_i to the stack and, after passing $\#$, takes it back from the stack (in the reversed order) and establishes the fact that the next block between $\#$'s is not the reversed successor configuration. This establishes the fact that $\mathbf{E}$ is also context-free. Finally, removing the leftmost $\#$ from these languages does not affect context-freeness.

[4]Kozen [20] uses a different construction, without reversing configurations with odd numbers. This results in a slightly more complicated grammar for the $\mathbf{E}$ language below.

Thus, there exist, and can be efficiently constructed, three context-free grammars for these languages, with the leftmost $\#$ removed. We can suppose that the sets of non-terminal symbols of these grammars are disjoint, and their starting symbols are T, E, and P respectively. Moreover, we translate these grammars into Greibach normal form [11], in which each production rule is of the form $A \Rightarrow aB_1 \dots B_\ell$, where a is a terminal letter from $\Sigma \cup Q \cup \{\#\}$, the symbols A, B_1, $\dots$, B_ℓ are non-terminal ones, and $0 \le \ell \le 2$. Now we put these grammars all together and add a new starting symbol S, another non-terminal U and the following production rules:

$$\begin{array}{ll} S \Rightarrow aU & \text{where } a \neq \# \\ S \Rightarrow a & \text{for any } a \in \Sigma \cup Q \cup \{\#\} \\ S \Rightarrow \#T & \\ S \Rightarrow \#E & \\ S \Rightarrow \#P & \\ U \Rightarrow aU & \text{for any } a \in \Sigma \cup Q \cup \{\#\} \\ U \Rightarrow a & \text{for any } a \in \Sigma \cup Q \cup \{\#\} \end{array}$$

(U generates all non-empty words).

The grammar obtained by this procedure is the needed $\mathcal{G}_{\mathfrak{M},x}$:

Lemma 4. *The grammar $\mathcal{G}_{\mathfrak{M},x}$ defined above generates all non-empty words over $\Sigma \cup Q \cup \{\#\}$ if and only if $\mathfrak{M}$ does not halt on input x.*

Proof. If $\mathfrak{M}$ halts on x, then its halting protocol starts with $\#$, but belongs neither to $\mathbf{T}$, nor to $\mathbf{E}$, nor to $\mathbf{P}$. Hence, the language generated by $\mathcal{G}_{\mathfrak{M},x}$ is not total.

If $\mathfrak{M}$ does not halt on x, then there is no halting protocol. Thus, any word beginning with $\#$ is either malformed (belongs to $\mathbf{T}$), or includes an incorrect execution step (belongs to $\mathbf{E}$), or is an incomplete protocol, *i.e.,* allows further continuation (then it belongs to $\mathbf{P}$). All such words are generated by $\mathcal{G}_{\mathfrak{M},x}$; all non-empty words not beginning with $\#$, as well the one-letter word '$\#$', are also generated explicitly. Therefore, in this case $\mathcal{G}_{\mathfrak{M},x}$ generates all non-empty words. □

This construction is sufficient for Buszkowski's [5] proof of Π_1^0-hardness of $\mathbf{ACT}_\omega$. For our purposes, however, we augment $\mathcal{G}_{\mathfrak{M},x}$ with some extra rules. These rules are redundant in the sense that they do not extend the language generated by the grammar—but they make proving totality easier and enable formalisation of these proofs in the weaker system $\mathbf{ACT}$.

First, we notice that $\mathbf{T}$ and $\mathbf{E}$ (but not $\mathbf{P}$) are closed under concatenation with arbitrary words to the right: an incorrect protocol could not get fixed by further extension. Thus, adding the following rules does not add new words to the language:

$$S \Rightarrow \#TU$$
$$S \Rightarrow \#EU$$

Next, if $\mathfrak{M}$ includes a cycling state q_C (see Definition 3), then any protocol in which q_C appears is necessary non-halting (incomplete) and therefore belongs to $\mathbf{P}$. If a word with q_C is not a correct protocol, then it is also already generated by

$\mathcal{G}_{\mathfrak{M},x}$. Hence, any word including q_C belongs to the language, and it is safe to add the following rules:

$$\begin{aligned} &S \Rightarrow \#CU \\ &C \Rightarrow aC \qquad \text{for any } a \in \Sigma \cup Q \cup \{\#\} \\ &C \Rightarrow q_C U \\ &C \Rightarrow q_C \end{aligned}$$

(non-terminal C generates all words with q_C). Notice that these rules are added *only* if $\mathfrak{M}$ actually includes a cycling state.

We denote the augmented grammar by $\mathcal{G}'_{\mathfrak{M},x}$. Since $\mathcal{G}'_{\mathfrak{M},x}$ generates the same language as $\mathcal{G}_{\mathfrak{M},x}$, Lemma 4 holds for $\mathcal{G}'_{\mathfrak{M},x}$ as well.

V. Encoding II: proving non-halting of Turing machines in $\mathbf{ACT}_\omega$ and $\mathbf{ACT}$

Following Buszkowski, we transform $\mathcal{G}'_{\mathfrak{M},x}$ into a Lambek categorial grammar. In order to make our presentation self-contained, here we reexplain the proofs of Buszkowski's lemmata.

Let the set of variables of the Lambek calculus include all non-terminals of $\mathcal{G}'_{\mathfrak{M},x}$. The grammar is in Greibach normal form, so each production rule is of the form $A \Rightarrow aB_1 \dots B_\ell$ (see previous section). For every such rule, we associate the formula $A \leftarrow (B_1 \cdot \ldots \cdot B_\ell)$ with a (if $\ell = 0$, take just A). Next, for every $a \in \Sigma \cup Q \cup \{\#\}$ let $\{\varphi_{a,1}, \dots, \varphi_{a,m}\}$ be the set of all such formulae (obtained from production rules with this a) and let $\varphi_a = \varphi_{a,1} \wedge \ldots \wedge \varphi_{a,m}$.

The following lemma shows that this translation of a context-free grammar into the Lambek calculus is sound and complete:

Lemma 5. *For any non-terminal A of $\mathcal{G}'_{\mathfrak{M},x}$, the word $a_1 \dots a_n$ is generated from A in $\mathcal{G}'_{\mathfrak{M},x}$ if and only if the sequent $\varphi_{a_1}, \dots, \varphi_{a_n} \vdash A$ is derivable in* $\mathbf{MALC}$.

Notice that, due to conservativity (Proposition 1), provability of this sequent is $\mathbf{MALC}$ is equivalent to its provability in $\mathbf{ACT}_\omega$ and/or in $\mathbf{ACT}$.

Proof. The left-to-right implication is easier and is obtained by induction on the context-free derivation of $a_1 \dots a_n$ from A in $\mathcal{G}'_{\mathfrak{M},x}$. Let the first production rule in this derivation be $A \Rightarrow a_1 B_1 \dots B_\ell$. Then by induction hypothesis we have $\varphi_{a_2}, \dots, \varphi_{a_{i_1}} \vdash B_1$, and so on, $\varphi_{a_{i_{\ell-1}+1}}, \dots, \varphi_{a_n} \vdash B_\ell$. By cut from $A \leftarrow (B_1 \cdot \ldots \cdot B_\ell), B_1, \dots, B_\ell \vdash A$ (which is derivable) we get $A \leftarrow (B_1 \cdot \ldots \cdot B_\ell), \varphi_{a_2}, \dots, \varphi_{a_n} \vdash A$. By definition, $A \leftarrow (B_1 \cdot \ldots \cdot B_\ell)$ is a member of the conjunction φ_{a_1}, so several applications of $(\wedge\ \vdash)$ yield $\varphi_{a_1}, \varphi_{a_2}, \dots, \varphi_{a_n} \vdash A$. (In particular, ℓ could be zero, and we get $\varphi_{a_1} \vdash A$ from $A \vdash A$. This is the induction base.)

For the right-to-left implication, consider a cut-free derivation of $\varphi_{a_1}, \dots, \varphi_{a_n} \vdash A$ in $\mathbf{MALC}$. Notice that the only rules which can occur in this derivation are $(\leftarrow\vdash)$, $(\vdash \cdot)$, and $(\wedge \vdash)$. Since $\wedge$'s appear only as the topmost connectives of the formulae φ_i, if $(\wedge \vdash)$ happens to be applied before $(\leftarrow\vdash)$ or $(\vdash \cdot)$, these two rules can be interchanged. Also, applications of $(\wedge \vdash)$ operating in different φ_i's are independent and can be performed in any order. Thus, we can assume that all applications of $(\wedge \vdash)$ which constructed φ_{a_1} were performed at the very bottom of the derivation.[5] In other words, our sequent got derived from $A \leftarrow (B_1 \cdot \ldots \cdot B_\ell), \varphi_{a_2}, \dots, \varphi_{a_n} \vdash A$. Next, locate the application of $(\leftarrow\vdash)$ which introduced the leftmost $\leftarrow$. Again, this application can be permuted with all other rules' applications, and moved to the bottom of the derivation. Thus, we get $\varphi_{a_2}, \dots, \varphi_{a_n} \vdash B_1 \cdot \ldots \cdot B_\ell$. Finally, we move the $(\vdash \cdot)$ rule, which introduced the succedent, to the bottom of the derivation (again, it is interchangeable with all other rules). Next, we use the induction hypothesis to show that $B_1, \dots, B_\ell$ derive $a_2 \dots a_n$ in the context-free grammar, and then apply the $A \Rightarrow a_1 B_1 \dots B_\ell$ production rule. (For $\ell = 1$ we just use the induction hypothesis and apply $A \Rightarrow a_1 B_1$; $\ell = 0$ is the base case: in this situation there is no $\leftarrow$, and by induction on the derivation we show that $\varphi_{a_2}, \dots, \varphi_{a_n}$ is in fact empty. Application of $A \Rightarrow a_1$ in this case finishes the context-free derivation.) □

Next, let $\psi = \bigvee_{a \in \Sigma \cup Q \cup \{\#\}} \varphi_a$, and by invertibility of $(\vee \vdash)$ and $({}^* \vdash)_\omega$ (Proposition 2) Buszkowski obtains the following two lemmata (notice that we replaced $\mathcal{G}_{\mathfrak{M},x}$ with $\mathcal{G}'_{\mathfrak{M},x}$).

Lemma 6. *$\mathcal{G}'_{\mathfrak{M},x}$ generates all words of length n over $\Sigma \cup Q \cup \{\#\}$ if and only if $\psi^n \vdash S$ is derivable in* $\mathbf{MALC}$.

Proof. By the $(\vee\ \vdash)$ rule and its invertibility (Proposition 2), derivability of $\psi^n \vdash S$ is equivalent to derivability of $\varphi_{a_1}, \dots, \varphi_{a_n} \vdash S$ for arbitrary $a_1, \dots, a_n$. Then apply Lemma 5. □

Lemma 7. *$\mathcal{G}'_{\mathfrak{M},x}$ generates all non-empty words over $\Sigma \cup Q \cup \{\#\}$ (i.e., its language is total) if and only if $\psi^+ \vdash S$ is derivable in* $\mathbf{ACT}_\omega$.

Proof. Immediately from the previous lemma, by $(\cdot\ \vdash)$, $({}^* \vdash)_\omega$, and their invertibility. □

Notice that ψ is constructed from the pair $\langle \mathfrak{M}, x \rangle$ via grammar $\mathcal{G}'_{\mathfrak{M},x}$, so the sequent $\psi^+ \vdash S$ is exactly $\Gamma'_{\mathfrak{M},x} \vdash S$ mentioned in Section II. Lemma 7 yields Buszkowski's result of Π^0_1-hardness of $\mathbf{ACT}_\omega$.

The material presented in this section before this point was indeed previously known. Lemma 5 follows from Gaifman's theorem [1], as shown by Lemma 1 and Corollary 1 in [5]. Lemma 7 is [5, Lemma 5]; Lemma 6 is not explicitly formulated by Buszkowski, but easily follows from his proofs.

We are going further and prove that, if $\mathfrak{M}$ trivially cycles on x, then $\Gamma'_{\mathfrak{M},x} \vdash S$ is provable already in $\mathbf{ACT}$. First we establish a technical fact.

Lemma 8. *The sequent $\psi^+ \vdash U$ is derivable in* $\mathbf{ACT}$.

This lemma states the fact that totality of the language of words generated from U is provable in $\mathbf{ACT}$. Indeed,

[5] This analysis of a cut-free derivation in $\mathbf{MALC}$ is in fact a specific and very simple instance of the *focusing* technique in non-commutative intuitionistic linear logic (see [15] for more details and further references).

production rules for U provide the easiest possible way of generating all non-empty words.

Proof. Since for any letter $a \in \Sigma \cup Q \cup \{\#\}$ we have production rules $U \Rightarrow a$ and $U \Rightarrow aU$ in the context-free grammar, formulae U and $U \leftarrow U$ are included in conjunctions φ_a for any a. Thus, by $(\wedge \vdash)$ we have $\varphi_a \vdash U$ and $\varphi_a \vdash U \leftarrow U$, and by $(\vee \vdash)$ we get $\psi \vdash U$ and $\psi \vdash U \leftarrow U$ (recall that $\psi = \bigvee_{a \in \Sigma \cup Q \cup \{\#\}} \varphi_a$).

Now $\psi^+ \vdash U$ is derived as follows:

$$\dfrac{\dfrac{\dfrac{\psi \vdash U}{\Lambda \vdash \psi \to U}\,(\vdash\to) \quad \dfrac{\psi \vdash \psi \quad \dfrac{\psi \vdash U \leftarrow U}{\psi, U \vdash U}\,(\vdash\leftarrow),\ \text{inv.}}{U \vdash \psi \to U}\,(\vdash\to)}{\psi, \psi \to U \vdash \psi \to U}\,(\to\vdash)}{\dfrac{\dfrac{\psi^* \vdash \psi \to U}{\psi, \psi^* \vdash U}\,(\vdash\to),\ \text{inverted}}{\psi^+ \vdash U}\,(\cdot\vdash)}\,(^*\vdash)_{\text{fp}}$$

□

Now we are ready to prove the key lemma.

Lemma 9. *If $\mathfrak{M}$ trivially cycles on input x, then $\psi^+ \vdash S$ is derivable in* **ACT**.

Proof. Suppose that $\mathfrak{M}$ trivially cycles on x. Consider the execution of $\mathfrak{M}$ on input x up to the moment when $\mathfrak{M}$ enters the cycling state q_C (such an execution is unique, because $\mathfrak{M}$ is deterministic). Let the code of the corresponding sequence of configurations be of length n (n is the number of letters in the code, not the number of configurations!). Notice that $n > 1$, since this code includes at least two symbols, namely q_C and the leftmost $\#$.

Now we derive $\psi^+ \vdash S$ using the "long rule" (Lemma 3):

$$\dfrac{\psi \vdash S \quad \psi^2 \vdash S \quad \dots \quad \psi^n \vdash S \quad \psi^n, \psi^+ \vdash S}{\psi^+ \vdash S}$$

All its premises, except the last one, are of the form $\psi^m \vdash S$ and are derivable by Lemma 6. In order to derive the last premise, we first apply $(\vee \vdash)$ as many times as possible to all formulae in ψ^n. Now we have to derive $\varphi_{a_1}, \dots, \varphi_{a_n}, \psi^+ \vdash S$ for any word $a_1 \dots a_n$ over $\Sigma \cup Q \cup \{\#\}$.

Apply cut as follows:

$$\dfrac{\psi^+ \vdash U \quad \varphi_{a_1}, \dots, \varphi_{a_n}, U \vdash S}{\varphi_{a_1}, \dots, \varphi_{a_n}, \psi^+ \vdash S}\ (\text{cut})$$

The left premise is derivable by Lemma 8. For the right premise, consider several cases:

1) $a_1 \neq \#$. Then $S \Rightarrow a_1 U$ is a production rule of the grammar, and φ_{a_1} contains (as a member of the big conjunction) $S \leftarrow U$. Each of the other formulae, $\varphi_{a_2}, \dots, \varphi_{a_n}$, contains $U \leftarrow U$, by the $U \Rightarrow a_i U$ production rule. Thus, our sequent, $\varphi_{a_1}, \dots, \varphi_{a_n}, U \vdash S$, is derived from $S \leftarrow U, U \leftarrow U, \dots, U \leftarrow U, U \vdash S$ (which is derivable in the Lambek calculus) by several applications of $(\wedge \vdash)$.
2) $a_1 a_2 \dots a_n \in \mathbf{T}$. In this case $a_1 = \#$, and the remainder $a_2 \dots a_n$ is generated in $\mathcal{G}'_{\mathfrak{M},x}$ from non-terminal T. By the $S \Rightarrow \#TU$ production rule, we have $S \leftarrow (T \cdot U)$ in φ_{a_1}, and by $(\wedge \vdash)$ we now have to derive the sequent $S \leftarrow (T \cdot U), \varphi_{a_2}, \dots, \varphi_{a_n}, U \vdash S$. By Lemma 5, since $a_2 \dots a_n$ is generated from T, the sequent $\varphi_{a_2}, \dots, \varphi_{a_n} \vdash T$ is derivable. Finally, use cut:
$$\dfrac{\varphi_{a_2}, \dots, \varphi_{a_n} \vdash T \quad S \leftarrow (T \cdot U), T, U \vdash S}{S \leftarrow (T \cdot U), \varphi_{a_2}, \dots, \varphi_{a_n}, U \vdash S}$$
The upper-right sequent is derivable in the Lambek calculus.
3) $a_1 a_2 \dots a_n \in \mathbf{E}$. The same as the previous case, with E instead of T.
4) $a_1 = \#$ and $a_i = q_C$ for some $i \in \{2, 3, \dots n\}$. Essentially the same: use the $S \Rightarrow \#CU$ production rule and notice that $\varphi_{a_2}, \dots, \varphi_{a_n} \vdash C$ is derivable, since $a_2 \dots a_n$ is generated from C.

Finally, we notice that this case analysis is exhaustive. Indeed, any word of at least two letters, which starts with $\#$ and does not belong to $\mathbf{T}$ or $\mathbf{E}$, is a prefix of a correct protocol of execution of $\mathfrak{M}$ on input x. Such an execution is unique, and we know that it includes q_C as one of the first n symbols of the configuration sequence code. □

Notice that the P non-terminal did not become completely useless in the presence of C: while for long words we use C instead of P, shorter prefixes of the configuration sequence, which appear in derivations of $\psi^m \vdash S$, $m < n$, are still generated using P. For long prefixes of the infinite run of $\mathfrak{M}$ on x, however, the rules with C provide a uniform, purely inductive way of proving $\psi^m \vdash S$, which is encodable in **ACT**.

VI. Undecidability of ACT

Let us fix the encoding of pairs $\langle \mathfrak{M}, x \rangle$ as sequents of action logic as described in the previous two sections: $\Gamma'_{\mathfrak{M},x} = \psi^+$, where ψ is obtained from $\mathcal{G}'_{\mathfrak{M},x}$. Let $\mathcal{L}$ be an arbitrary theory in the language of **ACT** and $\mathbf{ACT}_\omega$.

Definition 5. By $\mathcal{K}(\mathcal{L})$ we denote the class of pairs $\langle \mathfrak{M}, x \rangle$, for which the sequent $\Gamma'_{\mathfrak{M},x} \vdash S$ is derivable in $\mathcal{L}$ ("pairs of a Turing machine and its input, for which $\mathcal{L}$ can prove non-halting").

(Formally speaking, one can identify $\mathcal{L}$ with the set of its theorems, so "derivable in $\mathcal{L}$" actually means "belongs to $\mathcal{L}$.")

Let $\mathcal{H}$ be the class of all pairs where $\mathfrak{M}$ halts on x, $\overline{\mathcal{H}}$ be its complement (the class of all pairs where $\mathfrak{M}$ does not halt on x), and $\mathcal{C}$ be the class of all pairs where $\mathfrak{M}$ trivially cycles on x. When talking about (un)decidability of such classes, we suppose that $\mathfrak{M}$ is encoded as a binary word in a standard way.

Using these notations, we can summarize results of the previous section in the following way:

Lemma 7: $\mathcal{K}(\mathbf{ACT}_\omega) = \overline{\mathcal{H}}$;
Lemma 9: $\mathcal{K}(\mathbf{ACT}) \supseteq \mathcal{C}$.

Moreover, for any $\mathcal{L}$ between $\mathbf{ACT}$ and $\mathbf{ACT}_\omega$ we have

$$\mathcal{C} \subseteq \mathcal{K}(\mathbf{ACT}) \subseteq \mathcal{K}(\mathcal{L}) \subseteq \mathcal{K}(\mathbf{ACT}_\omega) = \overline{\mathcal{H}}.$$

Now we use a folklore fact that $\mathcal{C}$ and $\mathcal{H}$ are recursively inseparable:

Proposition 10. *There exists no decidable class $\mathcal{K}$ which includes $\mathcal{C}$ ($\mathcal{C} \subseteq \mathcal{K}$) and is disjoint with $\mathcal{H}$ ($\mathcal{K} \subseteq \overline{\mathcal{H}}$).*

Proof. Diagonalization. Suppose there exists a decidable class $\mathcal{K}$ which separates $\mathcal{C}$ and $\mathcal{H}$. Fix an encoding of Turing machines. By decidability of $\mathcal{K}$, the following program can be implemented as a Turing machine:

given $y \in \{0,1\}^*$,
  if y is a code of a Turing machine $\mathfrak{M}$ and $\langle \mathfrak{M}, y\rangle \in \mathcal{K}$,
    then halt;
  if y is a code of a Turing machine $\mathfrak{M}$ and $\langle \mathfrak{M}, y\rangle \notin \mathcal{K}$,
    then enter a cycling state;
  if y is not a code of a Turing machine,
    then halt.

Denote this Turing machine by $\mathfrak{M}_\delta$ and let y_δ be its code. Consider $\langle \mathfrak{M}_\delta, y_\delta\rangle$.

Case 1: $\langle \mathfrak{M}_\delta, y_\delta\rangle \in \mathcal{K}$. On one hand, $\mathfrak{M}_\delta$, by its definition, halts on y_δ. On the other hand, it does not, since $\mathcal{K} \subseteq \overline{\mathcal{H}}$. Contradiction.

Case 2: $\langle \mathfrak{M}_\delta, y_\delta\rangle \notin \mathcal{K}$. On one hand, $\mathfrak{M}_\delta$, by its definition, trivially cycles on y_δ, *i.e.,* $\langle \mathfrak{M}_\delta, y_\delta\rangle \in \mathcal{C}$. On the other hand, $\mathcal{C} \subseteq \mathcal{K}$, therefore $\langle \mathfrak{M}_\delta, y_\delta\rangle \notin \mathcal{C}$. Contradiction. □

In particular, $\mathcal{K}(\mathcal{L})$ is undecidable for any $\mathcal{L}$ such that $\mathbf{ACT} \subseteq \mathcal{L} \subseteq \mathbf{ACT}_\omega$. This finishes the proof of Theorem 3 and, in particular, establishes undecidability of $\mathbf{ACT}$ (Theorem 2).

VII. Conclusion

We have proved that $\mathbf{ACT}$ is algorithmically undecidable. Some related questions, however, still require further investigation. First, the exact complexity class for $\mathbf{ACT}$ should be settled. We conjecture Σ^0_1-completeness: thus, undecidabilities of $\mathbf{ACT}_\omega$ and $\mathbf{ACT}$ are of different nature. A more technical question is an exact characterization of $\mathcal{K}(\mathbf{ACT})$: which sorts of "regular" halting, besides trivial cycling, can be proved in $\mathbf{ACT}$ (this depends, however, on the construction of $\mathcal{G}'_{\mathfrak{M},x}$). Second, Buszkowski proves Π^0_1-hardness not only for $\mathbf{ACT}_\omega$ itself, but also for its fragments with only one additive connective (either $\vee$ or $\wedge$, but not both). It looks plausible that our undecidability proof for $\mathbf{ACT}$ would also work in these fragments. The fragment without $\wedge$ (only $\vee$) is particularly interesting, since it is the logic of action algebras originally introduced by Pratt. Finally, the old question of constructing a good (analytic) Gentzen-style calculus for $\mathbf{ACT}$ is still open. One possible approach would be to take non-well-founded cut-free derivations for $\mathbf{ACT}_\omega$ [8] and consider the subclass of derivations corresponding to $\mathbf{ACT}$ (cf. [4] for arithmetics).

Acknowledgment

Though not mentioned explicitly in this paper, the encoding of cyclic executions of Turing machines in $\mathbf{ACT}$ is essentially connected to circular proofs in the version of $\mathbf{ACT}_\omega$ with non-well-founded derivations (cf. [8]). The author is grateful to Daniyar Shamkanov and Anupam Das for sharing ideas on circular proofs. The author would also like to thank Fedor Pakhomov, Max Kanovich, Andre Scedrov, and Stanislav Speranski for fruitful discussions, and the anonymous reviewers for their helpful comments. Being a Young Russian Mathematics award winner, the author thanks its jury and sponsors for this high honour.

Financial Support

This article was prepared within the framework of the HSE University Basic Research Program and funded by the Russian Academic Excellence Project '5-100.'

References

[1] Y. Bar-Hillel, C. Gaifman, E. Shamir. On the categorial and phrase-structure grammars. *Bulletin of the Research Council of Israel* 9F (1960), 1–16.

[2] P. Brunet, D. Pous. A formal exploration of nominal Kleene algebra. In: *MFCS 2016,* LIPIcs vol. 58, Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2016, pp. 22:1–22:13.

[3] P. Brunet, D. Pous, G. Struth. On decidability of concurrent Kleene algebra. In: *CONCUR 2017,* LIPIcs vol. 85, Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2017, pp. 28:1–28:15.

[4] W. Buchholz. Notation systems for infinitary derivations. *Archive for Mathematical Logic* 30 (1991), 277–296.

[5] W. Buszkowski. On action logic: equational theories of action algebras. *Journal of Logic and Computation* 17 (2007), 199–217.

[6] W. Buszkowski. Some open problems in substructural logics. Invited talk, Kick-off meeting for project TICAMORE. Vienna, 14–16 March, 2017. https://ticamore.logic.at/legacy/program/ticamore-kick-off-vienna-2017-Buszkowski.pdf

[7] J. H. Conway. Regular algebra and finite machines. Chapman and Hall, London, 1971.

[8] A. Das, D. Pous. Non-wellfounded proof theory for (Kleene+action) (algebras+lattices). In: *CSL 2018*, LIPIcs vol. 119, Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2018, pp. 19:1–19:18.

[9] D.-Z. Du, K.-I. Ko. Problem solving in automata, languages, and complexity. John Wiley & Sons, New York, 2001.

[10] M. J. Gabbay, V. Ciancia. Freshness and name-restriction in sets of traces with names. In: *FoSSaCS 2011,* LNCS vol. 6604, Springer, 2011, pp. 365–380

[11] S. Greibach. A new normal-form theorem for context-free phrase structure grammars. *Journal of the ACM* 12 (1965), 42–52.

[12] T. Hoare, B. Möller, G. Struth, I. Wehrman. Concurrent Kleene algebra and its foundations. *Journal of Logic and Algebraic Programming* 80 (2011), 266–296.

[13] P. Jipsen. From semirings to residuated Kleene algebras. *Studia Logica* 76 (2004), 291–303.

[14] M. Kanovich, S. Kuznetsov, V. Nigam, A. Scedrov. Subexponentials in non-commutative linear logic. *Mathematical Structures in Computer Science,* FirstView (2018), https://doi.org/10.1017/S0960129518000117

[15] M. Kanovich, S. Kuznetsov, V. Nigam, A. Scedrov. A logical framework with commutative and non-commutative subexponentials. In: *IJCAR 2018*, LNAI vol. 10900, Springer, 2018, pp. 228–245.

[16] D. Kozen. On Kleene algebras and closed semirings. In: *MFCS 1990,* LNCS vol. 452, Springer, 1990, pp. 26–47.

[17] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation* 110 (1994), 366–390.

[18] D. Kozen. On action algebras. In: *Logic and Information Flow*, MIT Press, 1994, pp. 78–88.

[19] D. Kozen, F. Smith. Kleene algebra with tests: completeness and decidability. In: *CSL'96,* LNCS vol. 1258, Springer, 1996, pp. 244–259.

[20] D. Kozen. Automata and computability. Springer-Verlag, New York, 1997.
[21] D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic* 1 (2000), 60–76.
[22] D. Kozen, K. Mamouras, A. Silva. Completeness and incompleteness in nominal Kleene algebra. *Journal of Logical and Algebraic Methods in Programming* 91 (2017), 17–32.
[23] S. C. Kleene. Representation of events in nerve nets and finite automata. In: *Automata Studies,* Princeton University Press, 1956, pp. 3–41.
[24] D. Krob. Complete systems of $\mathscr{B}$-rational identities. *Theoretical Computer Science* 89 (1991), 207–343.
[25] S. Kuznetsov. *-continuity vs. induction: divide and conquer. In: *Advances in Modal Logic* vol. 12, College Publications, London, 2018, pp. 493–510.
[26] J. Lambek. The mathematics of sentence structure. *The American Mathematical Monthly* 65 (1958), 154–170.
[27] P. Lincoln, J. Mitchell, A. Scedrov, N. Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic* 56 (1992), 239–311.
[28] H. Ono. Semantics for substructural logics. In: *Substructural Logics*, Studies in Logic and Computation 2, Clarendon Press, Oxford, 1993, pp. 259–292.
[29] E. Palka. An infinitary sequent system for the equational theory of *-continuous action lattices. *Fundamenta Informaticae* 78.2 (2007), 295–309.
[30] V. Pratt. Action logic and pure induction. In: *JELIA 1990: Logics in AI*, LNCS (LNAI) vol. 478, Springer, 1991, pp. 97–120.
[31] V. N. Redko. On defining relations for the algebra of regular events (in Russian). *Ukrainskiĭ Matematicheskiĭ Zhurnal* 16 (1964), 120–126.
[32] M. Sipser. Introduction to the theory of computation, 3rd ed. Cengage Learning, 2012.