Zeno, Hercules and the Hydra: Safety Metric Temporal Logic Is ACKERMANN-complete

RANKO LAZIĆ, University of Warwick
JOËL OUAKNINE and JAMES WORRELL, University of Oxford

Metric temporal logic (MTL) is one of the most prominent specification formalisms for real-time systems. Over infinite timed words, full MTL is undecidable, but satisfiability for a syntactially defined safety fragment, called safety MTL, was proved decidable several years ago. Satisfiability for safety MTL is also known to be equivalent to a fair termination problem for a class of channel machines with insertion errors. However, hitherto its precise computational complexity has remained elusive, with only a non-elementary lower bound.

Via another equivalent problem, namely termination for a class of rational relations, we show that satisfiability for safety MTL is ACKERMANN-complete, i.e., among the easiest non-primitive recursive problems. This is surprising since decidability was originally established using Higman's Lemma, suggesting a much higher non-multiply recursive complexity.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification; F.1.1 [**Computation by Abstract Devices**]: Models of Computation; F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Metric Temporal Logic, Channel Machines, Complexity Hierarchies

ACM Reference Format:
Ranko Lazić, Joël Ouaknine, and James Worrell. 2015. Zeno, Hercules and the Hydra: Safety Metric Temporal Logic Is Ackermann-complete. *ACM Trans. Comput. Logic* V, N, Article A (January YYYY), 27 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Metric temporal logic (MTL) is one of the most popular approaches for extending temporal logic to the real-time setting. MTL extends linear temporal logic by constraining the temporal operators with intervals of real numbers. For example, the formula $\Diamond_{[3,4]}\varphi$ means that φ will hold within 3 to 4 time units in the future. There are two main semantic paradigms for MTL: continuous (state-based) and pointwise (event-based)—cf. [Alur and Henzinger 1992; Henzinger 1998]. In the former, an execution of a system is modelled by a flow which maps each point in time to the state propositions that are true at that moment. In the latter, one records only a countable sequence of events, corresponding to instantaneous changes in the state of the system. In this paper we

Authors' addresses: R. Lazić, Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK; J. Ouaknine and J. Worrell, Department of Computer Science, Parks Road, Oxford OX1 3QD, UK.
An extended abstract of this research was published as [Lazić et al. 2013].

DOI:http://dx.doi.org/10.1145/0000000.0000000

interpret MTL over the pointwise semantics[1] and assume that time is *dense* (arbitrarily many events can happen in a single time unit) but *non-Zeno* (only finitely many events can occur in a single time unit).

Over the past few years, the theory of *well-structured transition systems* has been used to obtain decidability results for MTL. Well-structured transition systems are a general class of infinite-state systems for which certain verification problems, such as reachability and termination, are decidable; see [Finkel and Schnoebelen 2001] for a comprehensive survey. In [Ouaknine and Worrell 2007] satisfiability and model checking for MTL were shown to be decidable by reduction to the reachability problem for a class of well-structured transition systems. Likewise, for a syntactically defined fragment of MTL that expresses safety properties, called *safety MTL*, model checking and satisfiability were shown decidable over infinite timed words by reduction to the termination problem on well-structured transition systems [Ouaknine and Worrell 2006b].

Extracting well-structured systems from MTL formulas relies on Higman's Lemma, which states that over a finite alphabet the subword order is a well-quasi order. Analysis of termination arguments that use Higman's Lemma has been applied to bound the complexity of reachability in lossy channel systems and insertion (or gainy) channel systems: two classes of well-structured systems that arise naturally in the modelling of communication over faulty media. For the reachability and termination problem in lossy channel systems, an upper bound in level $\mathfrak{F}_{\omega^\omega}$ of the fast-growing hierarchy was obtained in [Chambart and Schnoebelen 2008]. (Recall that $\mathfrak{F}_{<\omega}$ comprises the primitive recursive functions, Ackermann's function lies in $\mathfrak{F}_\omega$, while $\mathfrak{F}_{\omega^\omega}$ contains the first non-multiply recursive function.) The same paper also shows that neither problem lies in a lower level of the hierarchy and observes that both lower and upper bounds carry over to MTL satisfiability over *finite* words and to reachability in insertion channel systems, among many other problems.[2] An upper bound in $\mathfrak{F}_{\omega^\omega}$ for safety MTL satisfiability has also been sketched in [Schmitz 2012] using related techniques.

Meanwhile, complexity lower bounds for safety MTL have been obtained utilising a correspondence with the termination problem for insertion channel systems. In [Bouyer et al. 2012] it is shown that termination for insertion channel machines with emptiness tests is primitive recursive, though non-elementary.[3] This result is used to give a non-elementary lower bound in $\mathfrak{F}_3$ for the satisfiability problem for safety MTL. An improved lower bound in $\mathfrak{F}_4$ is given in [Jenkins 2012], again via insertion channel machines, but still leaving a considerable gap with the above-mentioned $\mathfrak{F}_{\omega^\omega}$ upper bound. This gap was highlighted recently in [Karandikar and Schmitz 2013].

The key to determining the precise complexity of satisfiability for safety MTL is to study a refined version of the termination problem for channel machines—namely the *fair termination problem*. Roughly speaking, an infinite computation of an insertion channel machine is *fair* if every message that is written to the channel is eventually consumed—and not continuously preempted by insertion errors. (In the translation between channel machines and MTL, fairness corresponds in a precise sense to the *non-Zenoness* assumption.) We obtain lower and upper complexity bounds for this problem that are *Ackermannian*, i.e., that lie in level $\mathfrak{F}_\omega$ of the fast-growing hierarchy. These

[1]Note that it follows from the thesis work of Henzinger [1991] that MTL satisfiability, even when restricted to the safety fragment, is undecidable over the continuous semantics.

[2]Incidentally, the *model-checking* problem over infinite timed words for safety MTL against timed automata can also be shown to have complexity precisely in $\mathfrak{F}_{\omega^\omega}$, following arguments presented in [Ouaknine and Worrell 2007] together with the results of [Chambart and Schnoebelen 2008]. Also, since safety does not decrease expressiveness over finite words, the same complexity classification applies to safety MTL satisfiability in that setting.

[3]In the presence of insertion errors, read-transitions can always be taken, so the channel is redundant unless there is an extra hypothesis, such as emptiness tests (cf. [Cécé et al. 1996]).

bounds also apply to safety MTL satisfiability, finally closing the above-mentioned complexity gap. More precisely, we establish that both problems are *complete* for ACKERMANN: the class of all decision problems whose complexity is bounded by the Ackermann function, closed under primitive recursive reductions [Schmitz 2013]. In particular, satisfiability for safety MTL over infinite words has lower computational complexity than satisfiability for MTL over finite words.

Unlike [Bouyer et al. 2012], we consider channel machines with a single channel. In [Bouyer et al. 2012], without the hypothesis of fairness, the termination problem was shown to be non-elementary in the number of channels. On the other hand, fair termination is already undecidable if there are two channels. But with a single channel fair termination is non-primitive recursive in the size of the channel alphabet. In common with [Bouyer et al. 2012] we find that termination for insertion channels has a lower complexity than termination for lossy channel systems or reachability for either type of system, neither of which is multiply recursive.

Our technical development is carried out in a slightly more abstract framework than insertion channel systems. Following [Karandikar and Schmitz 2013], we study the termination problem for well-structured transition systems whose states are words over a given alphabet, and whose transition relation is a rational relation that is (downwards) compatible with the subword order. (This is similar to the basic framework of regular model checking [Abdulla et al. 2004b], but with the additional hypothesis of monotonicity.)

To obtain membership of ACKERMANN, we associate a *Hydra battle* with each finite computation of such a system. For our purposes, a Hydra battle is a sequence of 'flat' regular expressions that express assertions about states in the computation. Each regular expression can be seen as arising from its predecessor by a process of truncation (by the sword of Hercules) and regeneration. Our Hydra correspond to the classical tree Hydra of Kirby and Paris [1982] via a natural correspondence between flat regular expressions and trees of height 2.

The basic pattern for proving our lower bound result is a standard one, namely to reduce from the halting problem for Ackermann-bounded Turing machines by simulating their computations. However, in contrast to the common approach in the literature, in which a large function and its inverse are computed weakly before and after the simulation respectively (cf., e.g., [Chambart and Schnoebelen 2008; Schnoebelen 2010; Karandikar and Schmitz 2013]), we bootstrap a counter that can count accurately to an Ackermann bound even in the presence of insertion errors. The bootstrapping involves extending Stockmeyer's yardstick construction, which reaches beyond the elementary functions, to surpass all primitive recursive ones.

2. PRELIMINARIES

2.1. Fast Growing Hierarchy

We define an initial segment of the fast growing hierarchy [Löb and Wainer 1970] of computable functions by following the presentation of Figueira et al. [2011].

For each $k \in \mathbb{N}$, class $\mathfrak{F}_k$ is the closure under substitution and limited recursion of constant, sum and projection functions, and F_n functions for $n \leq k$. The latter are defined so that F_0 is the successor function, and each F_{n+1} is computed by iterating F_n:

$$F_0(x) = x + 1 \qquad\qquad F_{n+1}(x) = (F_n)^{x+1}(x)$$

The following are a few simple observations:

— $\mathfrak{F}_0 = \mathfrak{F}_1$ contains all linear functions, like $\lambda x.x + 3$ or $\lambda x.2x$;
— $\mathfrak{F}_2$ contains all elementary functions, like $\lambda x.2^{2^x}$;

— $\mathfrak{F}_3$ contains all tetration functions, like $\lambda x.\,\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{x}$.

The hierarchy is strict for $k \geq 1$, i.e., $\mathfrak{F}_k \subsetneq \mathfrak{F}_{k+1}$, because $F_{k+1} \notin \mathfrak{F}_k$. Also, for each $k \geq 1$ and $f \in \mathfrak{F}_k$, there exists $p \geq 1$ such that $(F_k)^p$ majorises f, i.e., $f(x_1, \ldots, x_n) < (F_k)^p(\max(x_1, \ldots, x_n))$ for all $x_1, \ldots, x_n$ [Löb and Wainer 1970, Theorem 2.10].

The union $\bigcup_k \mathfrak{F}_k$ is the class of all primitive recursive functions, while F_ω defined by $F_\omega(x) = F_x(x)$ is an Ackermann-like non-primitive recursive function. Defining class $\mathfrak{F}_\omega$ from the functions F_α for $\alpha \leq \omega$ as above, we call Ackermannian the functions that lie in $\mathfrak{F}_\omega \setminus \bigcup_k \mathfrak{F}_k$.

We remark that, following this pattern for successor and limit ordinals, the hierarchy can be continued up to level ω^ω. The union $\bigcup_{\alpha<\omega^\omega} \mathfrak{F}_\alpha$ is the class of all multiply recursive functions, and the non-multiply recursive functions in $\mathfrak{F}_{\omega^\omega}$ have been called 'hyper-Ackermannian'.[4]

The hierarchy gives rise to the following classes of decision problems [Schmitz 2013]. At any level α, their complexity is bounded by F_α composed with any function in $\bigcup_{\beta<\alpha} \mathfrak{F}_\beta$:

$$\mathbf{F}_\alpha = \bigcup \{\mathrm{DT{\small IME}}(F_\alpha(g(n))) \ : \ \exists \beta < \alpha.\, g \in \mathfrak{F}_\beta\}$$

In particular, the class ACKERMANN $= \mathbf{F}_\omega$ consists of all decision problems whose complexity is bounded by $F_\omega(g(n))$ for primitive recursive g, and such a problem is complete if and only if there exist primitive recursive reductions to it from all problems in ACKERMANN.

2.2. Finite Transducers

We work with normalised transducers with ϵ-transitions, whose input and output alphabets are the same. They are tuples of the form $\langle Q, \Sigma, \delta, I, F\rangle$, where Q is a finite set of states, Σ is a finite alphabet, $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Sigma \cup \{\epsilon\}) \times Q$ is a transition relation, and $I, F \subseteq Q$ are sets of initial and final states respectively. We write transitions as $q \xrightarrow{a|a'} q'$, which can be thought of as reading a from the input word (if $a \in \Sigma$) and writing a' to the output word (if $a' \in \Sigma$).

For a transducer $\mathcal{T}$ as above, we say that τ is a *transduction* if and only if it is a path $q_0 \xrightarrow{a_1|a_1'} q_1 \cdots \xrightarrow{a_n|a_n'} q_n$ where q_0 is initial and q_n is final, and we write $\mathrm{In}(\tau)$ and $\mathrm{Out}(\tau)$ for the words $a_1 \ldots a_n$ and $a_1' \ldots a_n'$ respectively. In that case, we also write $q_0 \xrightarrow{\mathcal{T}} q_n$. The relation of $\mathcal{T}$ is then

$$\mathrm{R}(\mathcal{T}) = \{\langle \mathrm{In}(\tau), \mathrm{Out}(\tau)\rangle \ : \ \tau \text{ is a transduction of } \mathcal{T}\}.$$

These transducers recognise exactly the rational relations between Σ^* and Σ^* (cf., e.g., [Sakarovitch 2009, Chapter IV]).

A *computation* of a transducer $\mathcal{T}$ from a word w_1 is a finite or infinite sequence of words $w_1, w_2, \ldots$ such that $w_1 \ \mathrm{R}(\mathcal{T}) \ w_2 \ \mathrm{R}(\mathcal{T}) \cdots$.

If q and q' are states of a transducer $\mathcal{T}$, we write $\mathcal{T}(q, q')$ for the transducer obtained from $\mathcal{T}$ by making q the only initial state and q' the only final state.

2.3. Composing Transducers

We write $\fatsemi$ for relational composition, as well as for its counterpart in terms of transducers. Recalling a standard definition of the latter operation, given two transducers

[4] Cf. the discussion in Section 1 of the complexity of MTL satisfiability over finite words.

$\mathcal{T}_1 = \langle Q_1, \Sigma, \delta_1, I_1, F_1 \rangle$ and $\mathcal{T}_2 = \langle Q_2, \Sigma, \delta_2, I_2, F_2 \rangle$, the transition relation of their composition $\mathcal{T}_1 \fatsemi \mathcal{T}_2 = \langle Q_1 \times Q_2, \Sigma, \delta, I_1 \times I_2, F_1 \times F_2 \rangle$ is defined so that every output of $\mathcal{T}_1$ must be consumed by an input of $\mathcal{T}_2$:

$$\langle q_1, q_2 \rangle \xrightarrow{a|a'} \langle q_1', q_2' \rangle \text{ iff } \begin{cases} q_1 \xrightarrow{a|\epsilon} q_1' \text{ and } a' = \epsilon \text{ and } q_2 = q_2', \text{ or} \\ q_1 \xrightarrow{a|a''} q_1' \text{ and } q_2 \xrightarrow{a''|a'} q_2' \text{ for some } a'' \in \Sigma, \text{ or} \\ q_1 = q_1' \text{ and } a = \epsilon \text{ and } q_2 \xrightarrow{\epsilon|a'} q_2'. \end{cases}$$

We then have $\mathrm{R}(\mathcal{T}_1 \fatsemi \mathcal{T}_2) = \mathrm{R}(\mathcal{T}_1) \fatsemi \mathrm{R}(\mathcal{T}_2)$.

2.4. Downwards Monotone Transducers

Given an alphabet Σ, we write $\sqsubseteq$ for the subword ordering on Σ^*, i.e., $w \sqsubseteq w'$ if and only if w' can be obtained from w by a number of insertions of letters. The *downward closure* of a subset L of Σ^*, i.e., $\{w \,:\, \exists w'.\, w \sqsubseteq w' \,\wedge\, w' \in L\}$, is denoted by $\downarrow L$.

We say that a relation R on Σ^* is *downwards monotone* if and only if, whenever $w_1 \; R \; w_2$, every replacement of w_1 by a subword w_1' can be matched on the right-hand side of R, i.e.,

$$\forall w_1, w_2, w_1'.\; w_1 \; R \; w_2 \;\wedge\; w_1' \sqsubseteq w_1 \;\Rightarrow\; \exists w_2'.\; w_1' \; R \; w_2' \;\wedge\; w_2' \sqsubseteq w_2.$$

Note that this is the same notion as downward compatibility of R with respect to $\sqsubseteq$ in the theory of well-structured transition systems [Finkel and Schnoebelen 2001].

A transducer $\mathcal{T}$ is said to be *downwards monotone* if and only if $\mathrm{R}(\mathcal{T})$ has the property.

PROPOSITION 2.1. *Composing transducers preserves downward monotonicity.*

We leave open the decidability of whether a given transducer is downwards monotone. We note however that this problem is at least as hard as the regular Post embedding problem ($\mathrm{PEP}^{\mathrm{reg}}$) [Chambart and Schnoebelen 2007], and therefore not multiply recursive [Chambart and Schnoebelen 2008]. Recall that $\mathrm{PEP}^{\mathrm{reg}}$ asks, given two morphisms $f, g : \Sigma^* \to \Gamma^*$ and a regular language $L \subseteq \Sigma^*$, whether $f(x) \sqsubseteq g(x)$ for some $x \in L$. Given an instance of $\mathrm{PEP}^{\mathrm{reg}}$, as above, it is straightforward to compute a transducer $\mathcal{T}$ that recognises the relation

$$\{\langle \%x, f(y) \rangle \,:\, x \in \Sigma^*, y \in L, x \sqsubseteq y\} \,\cup\, \{\langle x, u \rangle \,:\, x \in \Sigma^*, u \in \Gamma^*, u \not\sqsubseteq g(x)\},$$

where $\%$ is a fresh symbol. One can check that $\mathcal{T}$ is downwards monotone if and only if f, g, L is a negative instance of $\mathrm{PEP}^{\mathrm{reg}}$.

2.5. Downward Rational Termination

The principal problem we study is whether a given downwards monotone transducer terminates from a given word:

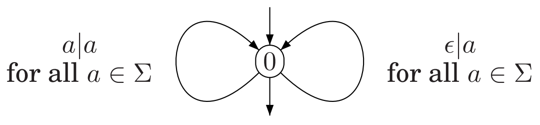
> Given a downwards monotone transducer $\mathcal{T}$ and a word w_1 over its alphabet, is every computation of $\mathcal{T}$ from w_1 finite?

We remark that the standard rational termination problem, i.e., without the assumption of downward monotonicity, is undecidable. Indeed, it is straightforward to compute a transducer that recognises the one-step relation between configurations of a given Turing machine.

Another closely related problem is gainy rational termination (also called increasing rational termination [Karandikar and Schmitz 2013]):

> Given a transducer $\mathcal{T}$ and a word w_1 over its alphabet, is every computation of $\mathcal{T}_{\sqsubseteq}$ from w_1 finite?

Here $\mathcal{T}_{\sqsubseteq} = \sqsubseteq \fatsemi \mathcal{T} \fatsemi \sqsubseteq$, where $\sqsubseteq$ on the right-hand side denotes a transducer whose relation is the subword ordering over the alphabet Σ of $\mathcal{T}$:



Thus, $\mathcal{T}_{\sqsubseteq}$ can be thought of as a 'faulty' version of $\mathcal{T}$ that may gain arbitrary letters in both input and output words, i.e., suffers from 'insertion errors'.

By observing that $\mathcal{T}_{\sqsubseteq}$ is downwards monotone for every transducer $\mathcal{T}$, gainy rational termination reduces to downward rational termination. Conversely, for a downwards monotone transducer $\mathcal{T}$, it is easy to see that $\mathcal{T}$ has an infinite computation from w_1 if and only if the same is true of $\mathcal{T}_{\sqsubseteq}$.

For technical convenience, to establish ACKERMANN-completeness of both problems, we work with downward rational termination for membership (Section 3), and with gainy rational termination for hardness (Section 4).

3. UPPER BOUND

We obtain that downward rational termination is in the complexity class ACKERMANN by proving that, given an instance $\mathcal{T}, w_1$ of the problem, there is a positive integer $N(\mathcal{T}, w_1)$ such that (i) if $\mathcal{T}$ terminates from w_1 then all its computations from w_1 have lengths bounded by $N(\mathcal{T}, w_1)$ and (ii) as a function of the length of $\mathcal{T}$ and w_1, $N(\mathcal{T}, w_1)$ is dominated by F_ω composed with a primitive recursive function.

At the heart of the proof, there is an analysis of computations of $\mathcal{T}$ from w_1 in terms of how frequently they contain words that belong to certain regular languages. A trivial case is when the regular language consists of all words over the alphabet of $\mathcal{T}$, for which the frequency is 1. More interestingly, our central lemma (Lemma 3.6) shows that, assuming that the frequency of the language of a regular expression E in a computation of length N is $1/u$ and that N is sufficiently large in terms of u, either some segment of the computation can be pumped to produce an infinite computation, or E can be refined to some E' whose language's frequency is some smaller $1/u'$. The notion of refinement of the regular expressions is such that only finitely many successive refinements are ever possible, and so if $\mathcal{T}$ terminates from w_1 then repeated applications of the lemma must stop because N is not sufficiently large. Moreover, the refinements of the regular expressions and the decreases in their frequencies observe certain bounds (that depend on $\mathcal{T}$, but not on w_1 or N), which together with the preceding reasoning enables us to obtain a global bound on the lengths of all the computations (provided that $\mathcal{T}$ terminates from w_1).

Before the central lemma, we have two lemmas (Section 3.2) that are about pumpability of computation segments, and its connection with the regular expressions and their refinements. Leading to the main result, we have another two lemmas (Section 3.4), which are concerned with bounding the sequences of regular expressions and frequencies that can arise from repeated applications of the central lemma, and consequences of those bounds for the lengths of computations. However, we first introduce the class of regular expressions used and the notion of refinement, as well as a useful class of auxiliary transducers.

3.1. Flat Regular Expressions (FRE)

A prominent role in the sequel is played by the following subclass of the simple regular expressions of Abdulla et al. [2004a]: we say that a regular expression over an alphabet Σ is *flat* if and only if it is of the form $\Delta_1^* d_1 \Delta_2^* d_2 \cdots \Delta_{K-1}^* d_{K-1} \Delta_K^*$ with $K \geq 1$, $\Delta_1, \ldots, \Delta_K \subseteq \Sigma$ and $d_1, \ldots, d_{K-1} \in \Sigma \cup \{\epsilon\}$.

For such a regular expression E, let:

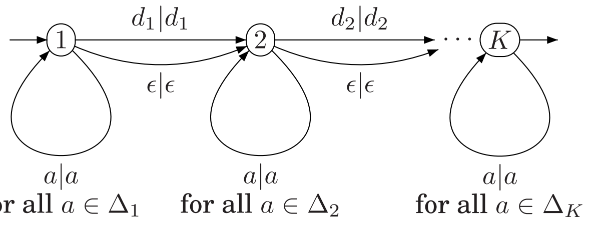
— the *length* of E be K;
— the *height* of E be $\max_{i=1}^{K} |\Delta_i|$.

If $l \in \mathbb{N}^+$, let us say that E is *l-refined* by E' if and only if E' can be obtained from E by replacing some Δ_i^* with an FRE $E^\dagger$ over Δ_i such that:

— the length of $E^\dagger$ is at most l, and
— the height of $E^\dagger$ is strictly less than $|\Delta_i|$, i.e., each set in $E^\dagger$ is strictly contained in Δ_i.

In that case, E' is also an FRE over Σ, of length at most $K + l - 1$. When each set in $E^\dagger$ has size $|\Delta_i| - 1$, we call the refinement *maximal*.

For E still as above, let $\mathcal{I}_E$ denote an identity transducer on the downward closure of $\mathrm{L}(E)$ (the language of E) as follows:



Indeed, $\mathrm{R}(\mathcal{I}_E) = \{\langle w, w\rangle \ : \ w \in {\downarrow}\mathrm{L}(E)\}$, so $\mathcal{I}_E$ is downwards monotone.

3.2. Pumpable Transductions

Since finite sequences of consecutive transductions can be seen as single transductions of composite transducers, it suffices to consider pumpability of transductions instead of considering it for computation segments. The notion we define applies to transductions between words in the language of an FRE $E = \Delta_1^* d_1 \cdots \Delta_{K-1}^* d_{K-1} \Delta_K^*$, and essentially requires that, for all i, while reading the portion of the input word in Δ_i^*, the transduction visits a part of the transducer that is able to consume any word in Δ_i^*. The composition with the identity transducer is a technical tool to ensure that traversing different paths in the state-transition graph still produces words that conform to E.

Definition 3.1. If $\mathcal{T}$ is a downwards monotone transducer and E is an FRE of the form $\Delta_1^* d_1 \cdots \Delta_{K-1}^* d_{K-1} \Delta_K^*$ over an alphabet Σ, and τ is a transduction of composite transducer $\mathcal{T} \fatsemi \mathcal{I}_E$ such that $\mathrm{In}(\tau) \in \mathrm{L}(E)$, let us say that τ is *pumpable* if and only if it can be factored as

$$s_1 \xrightarrow{\tau_1} s_1' \xrightarrow{d_1|e_1} s_2 \xrightarrow{\tau_2} s_2' \xrightarrow{d_2|e_2} \cdots s_K \xrightarrow{\tau_K} s_K',$$

such that, for each $i \in \{1, \ldots, K\}$, $\Delta_i^* \subseteq {\downarrow}\mathrm{dom}(\mathrm{R}((\mathcal{T} \fatsemi \mathcal{I}_E)(s_i, s_i')))$.

We remark that there is an unexpected analogy between this pumpability condition and the θ condition that is central to the proof of Kosaraju [1982] that the reachability problem for vector addition systems is decidable. Indeed, the flat regular expressions are in fact representations of ideals of words over Σ with respect to the subword ordering [Jullien 1969], whereas the structures employed in Kosaraju's θ condition have recently been shown to correspond to ideals of runs of vector addition systems [Leroux and Schmitz 2015].

Our first lemma confirms that pumpable transductions yield infinite computations.

LEMMA 3.2. *If $\mathcal{T}$ is downwards monotone, and $\mathcal{T} \fatsemi \mathcal{I}_E$ has a transduction τ such that* $\mathrm{In}(\tau) \in \mathrm{L}(E)$ *and which is pumpable, then $\mathcal{T}$ has an infinite computation from any word in* $\downarrow\mathrm{L}(E)$.

PROOF. Consider $w \in \downarrow\mathrm{L}(E)$. It suffices to show that $w \; \mathrm{R}(\mathcal{T} \fatsemi \mathcal{I}_E) \; w'$ for some w'.

Let $s_1 \xrightarrow{\tau_1} s'_1 \xrightarrow{d_1|e_1} \cdots s_K \xrightarrow{\tau_K} s'_K$ be a factoring of τ which demonstrates its pumpability, where $E = \Delta_1^* d_1 \cdots \Delta_{K-1}^* d_{K-1} \Delta_K^*$. Let us also write w as $w_1 d'_1 w_2 d'_2 \cdots w_K$, where $w_i \in \Delta_i^*$ for all $i \in \{1, \ldots, K\}$, and $d'_i \in \{d_i, \epsilon\}$ for all $i \in \{1, \ldots, K-1\}$. Now, for each i, we have that

$$w_i \in \downarrow\mathrm{dom}(\mathrm{R}((\mathcal{T} \fatsemi \mathcal{I}_E)(s_i, s'_i))),$$

so there exists a transduction τ'_i of $(\mathcal{T} \fatsemi \mathcal{I}_E)(s_i, s'_i)$ such that $w_i \sqsubseteq \mathrm{In}(\tau'_i)$. Hence

$$\tau' = s_1 \xrightarrow{\tau'_1} s'_1 \xrightarrow{d_1|e_1} \cdots s_K \xrightarrow{\tau'_K} s'_K,$$

is a transduction of $\mathcal{T} \fatsemi \mathcal{I}_E$ from a superword of w. But $\mathcal{T} \fatsemi \mathcal{I}_E$ is downwards monotone (recall Proposition 2.1), so it must also have a transduction from w, which completes the proof. □

The following is a 'pumping lemma': roughly, if a transduction from E to E is such that its input word is not in the language of any 'short' refinement of E, then it is pumpable. Here 'short' amounts to a bound which is the product of the length of E and the size of the transducer's state space.

LEMMA 3.3. *Suppose that:*

— *$\mathcal{T}$ is a downwards monotone transducer with set of states Q and alphabet Σ;*
— *E is an FRE over Σ, of length K;*
— *τ is a transduction of $\mathcal{T} \fatsemi \mathcal{I}_E$ such that* $\mathrm{In}(\tau) \in \mathrm{L}(E)$.

Then either τ is pumpable, or $\mathrm{In}(\tau) \in \mathrm{L}(E')$ *for some $K|Q|$-refinement E' of E.*

In the proof that follows, we show that pumpability is equivalent to the transduction visiting, for each Δ^* in E, a strongly connected component of the state-transition graph that, for every word in Δ^*, is able to consume some superword. The next definition plays a key role.

Definition 3.4. For a strongly connected component C of a transducer $\mathcal{T}$ over Σ, and a subset Δ of Σ, let us say that C is Δ-*total* if and only if, for each $a \in \Delta$, there is an input of a in C (i.e., a transition labelled by $\langle a|a' \rangle$ for some a', whose source and target states are in C).

PROOF. Let $E = \Delta_1^* d_1 \cdots \Delta_{K-1}^* d_{K-1} \Delta_K^*$. Since $\mathrm{In}(\tau) \in \mathrm{L}(E)$, we can factor τ as $s_1 \xrightarrow{\tau_1} s'_1 \xrightarrow{d_1|e_1} \cdots s_K \xrightarrow{\tau_K} s'_K$, where $\mathrm{In}(\tau_i) \in \Delta_i^*$ for all i.

Assume that τ is not pumpable. Then, for some i, Δ_i^* is not contained in the downward closure of the domain of transducer $(\mathcal{T} \fatsemi \mathcal{I}_E)(s_i, s'_i)$. The key observation is:

($*$) No state in τ_i can belong to a Δ_i-total strongly connected component of $\mathcal{T} \fatsemi \mathcal{I}_E$.

Indeed, we would otherwise have that τ_i is of the form $s_i \xrightarrow{\tau'_i} s''_i \xrightarrow{\tau''_i} s'_i$, where s''_i belongs to a Δ_i-total strongly connected component, which would mean that

$$\Delta_i^* \subseteq \downarrow\mathrm{dom}(\mathrm{R}((\mathcal{T} \fatsemi \mathcal{I}_E)(s''_i, s''_i))) \subseteq \downarrow\mathrm{dom}(\mathrm{R}((\mathcal{T} \fatsemi \mathcal{I}_E)(s_i, s'_i))).$$

Now, from ($*$) and since $\mathrm{In}(\tau_i) \in \Delta_i^*$, we have that $\mathrm{In}(\tau_i)$ must belong to the language of some FRE $E^\dagger = (\Delta_1^\dagger)^* d_1^\dagger \cdots (\Delta_{K^\dagger-1}^\dagger)^* d_{K^\dagger-1}^\dagger (\Delta_{K^\dagger}^\dagger)^*$ over Δ_i such that:

— $K^\dagger$ is the number of strongly connected components that are visited by τ_i;
— for each $j \in \{1, \dots, K^\dagger\}$, $\Delta_j^\dagger$ is a strict subset of Δ_i.

Hence, letting E' be obtained from E by replacing Δ_i^* with $E^\dagger$, we have that

$$\mathrm{In}(\tau) = \mathrm{In}(\tau_1) d_1 \cdots \mathrm{In}(\tau_i) d_i \cdots \mathrm{In}(\tau_K) \in \mathrm{L}(E')$$

and that E' is a $K^\dagger$-refinement of E. It remains to observe that $K^\dagger$ is at most $K|Q|$, the number of states of $\mathcal{T} \fatsemi \mathcal{I}_E$. □

3.3. Sword of Hercules

Our central lemma, assuming that γ is a computation of length N from w_1 of a downwards monotone $\mathcal{T}$ which terminates from w_1, can be applied repeatedly to γ to yield some sequence $\langle E_0, u_0 \rangle, \langle E_1, u_1 \rangle, \dots$ of pairs of FREs and positive integers, as long as N is sufficiently large. For each h, there are at least $\lfloor N/u_h \rfloor$ occurrences in γ of words from the language of E_h. Moreover, each E_{h+1} refines E_h, and the length of E_{h+1} as well as u_{h+1} are bounded by elementary functions of: the number of states of $\mathcal{T}$, the length and height of E_h, and u_h.

Recalling the notion of refinement, each application of the lemma can be thought of as a strike of Hercules on the FRE E_h, after which the latter has a Hydra-like response: although some component of the form Δ_h^* is removed from E_h, it is replaced in E_{h+1} by some FRE $E_h^\dagger$. The height of $E_h^\dagger$, however, must be strictly smaller than the size of Δ_h, but the bound on its length grows with every strike.

Definition 3.5. For $\alpha \in (0, 1]$, let us say that a regular expression E is *α-frequent* in a sequence of words $w_1, \dots, w_N$ if and only if there exists $J \subseteq \{1, \dots, N\}$ of size $\lfloor N\alpha \rfloor$ such that $w_j \in \mathrm{L}(E)$ for all $j \in J$.

LEMMA 3.6. *Suppose that $\gamma = w_1, \dots, w_N$ is a computation of a downwards monotone transducer $\mathcal{T}$ with set of states Q and alphabet Σ, and that $\mathcal{T}$ terminates from w_1. If an FRE E over Σ and $u \in \mathbb{N}^+$ are such that $N \geq 16u^2$ and E is $1/u$-frequent in γ, then there exists a $K|Q|^{4u}$-refinement E' of E which is $1/u'$-frequent in γ, where K is the length of E, H is the height of E, and $u' = 16u^2 K (H+1)^{2K|Q|^{4u}}$.*

PROOF. Let $J \subseteq \{1, \dots, N\}$ of size $\lfloor N/u \rfloor$ be such that $w_j \in \mathrm{L}(E)$ for all $j \in J$. Thinking of N as 'large' and u as 'small', we first observe that J contains many disjoint pairs of indices such that the differences within them are small and equal. Consider the set J as a sequence $j_1 < j_2 < \cdots$. The sequence of adjacent differences $j_2 - j_1, j_4 - j_3, \dots$ has length $\lfloor |J|/2 \rfloor = \lfloor N/2u \rfloor$ and its sum is at most N. Hence, at least $\lfloor N/4u \rfloor$ of the adjacent differences are not greater than $4u$. Let $D \leq 4u$ be a value that occurs most often in that subsequence of at least $\lfloor N/4u \rfloor$ adjacent differences, which means that it occurs at least $\lfloor \lfloor N/4u \rfloor / 4u \rfloor = \lfloor N/16u^2 \rfloor$ times. We therefore have a subset $J^\dagger$ of J of size $\lfloor N/16u^2 \rfloor$, such that $j + D \in J$ for all $j \in J^\dagger$.

Let $\mathcal{T}^D$ be the transducer obtained by D times iterating $\mathcal{T}$:

$$\mathcal{T}^D = \underbrace{\mathcal{T} \fatsemi \cdots \fatsemi \mathcal{T}}_{D}.$$

Consider $j \in J^\dagger$. Since $w_j \; R(\mathcal{T}^D) \; w_{j+D}$ and $w_j, w_{j+D} \in \mathrm{L}(E)$, the composite transducer $\mathcal{T}^D \fatsemi \mathcal{I}_E$ has a transduction τ_j such that $\mathrm{In}(\tau_j) = w_j \in \mathrm{L}(E)$. Also, $\mathcal{T}^D$ is downwards monotone because $\mathcal{T}$ is (cf. Proposition 2.1), and it terminates from w_j as otherwise $\mathcal{T}$ would not terminate from w_1. Therefore, τ_j is not pumpable by Lemma 3.2, so Lemma 3.3 gives us that w_j belongs to the language of some $K|Q|^D$-refinement E'_j of E, which we can assume is maximal.

To complete the proof, let E' be a regular expression which occurs most often in the sequence of regular expressions E'_j as j enumerates $J^\dagger$. Since $D \le 4u$, we have that E' is indeed a $K|Q|^{4u}$-refinement of E. Also, the cardinality of the set of all E'_j is at most $K(H+1)^{2K|Q|^D}$ and $|J^\dagger| = \lfloor N/16u^2 \rfloor$, so we can let J' be a subset of $J^\dagger$ of size $\lfloor N/u' \rfloor$ such that $E' = E'_j$ for all $j \in J'$. It remains to recall that, for each $j \in J'$, $w_j \in \mathrm{L}(E'_j) = \mathrm{L}(E')$. □

3.4. Slaying the Hydra

The next two lemmas show that every sequence of pairs of FREs and positive integers that can arise from repeated applications of Lemma 3.6 is finite, i.e., Hercules always defeats the Hydra eventually, and that if $N \ge 16u^2$ for every u in such a sequence and $\mathcal{T}$ terminates from w_1, then $\mathcal{T}$ cannot have a computation from w_1 of length N. Moreover, from the single-step bounds in Lemma 3.6, we establish a bound for each pair in terms of $|Q|$, $|\Sigma|$ and the distance from the initial pair $\langle \Sigma^*, 1\rangle$, where Q and Σ are the state space and the alphabet of $\mathcal{T}$.

We first define a directed graph which contains every sequence that Lemma 3.6 can yield. To show that every path that starts from $\langle \Sigma^*, 1\rangle$ is finite, we also introduce a measure on FREs E over Σ in terms of $|\Sigma|$-tuples of natural numbers. The latter records, for each $s \in \{1, \ldots, |\Sigma|\}$, how many sets of size s occur in E.

We say that a sequence $y_0, y_1, \ldots$ of tuples in some $\mathbb{N}^k$ is *bad* if and only if there do not exist $i < j$ such that $y_i \le y_j$, where $\le$ is the pointwise ordering. We recall that, by Dickson's Lemma, $\le$ is a well-quasi ordering on $\mathbb{N}^k$, i.e., there is no infinite bad sequence. Hence, the finiteness of every path from $\langle \Sigma^*, 1\rangle$ follows once we show that every corresponding sequence of measures in $\mathbb{N}^{|\Sigma|}$ is bad.

Definition 3.7. Given a set of states Q and an alphabet Σ, let $\Upsilon_{Q,\Sigma}$ be the graph:

— the vertices are pairs $\langle E, u\rangle$ where E is an FRE over Σ and $u \in \mathbb{N}^+$;
— there is an edge from $\langle E, u\rangle$ to $\langle E', u'\rangle$ if and only if E' is a $K|Q|^{4u}$-refinement of E and $u' = 16u^2K(H+1)^{2K|Q|^{4u}}$, where K is the length of E and H is the height of E.

Definition 3.8. For $E = \Delta_1^* d_1 \cdots \Delta_{K-1}^* d_{K-1} \Delta_K^*$ an FRE over Σ and $s \in \{0, \ldots, |\Sigma|\}$, let $Y_s(E) = |\{i \,:\, i \in \{1, \ldots, K\} \text{ and } |\Delta_i| = s\}|$.

LEMMA 3.9. *Suppose that Q is a set of states, Σ is an alphabet, and $\langle E_0, u_0\rangle \to \langle E_1, u_1\rangle \to \ldots$ is a path from $\langle \Sigma^*, 1\rangle$ in $\Upsilon_{Q,\Sigma}$. Then*

$$\langle Y_1(E_0), \ldots, Y_{|\Sigma|}(E_0)\rangle, \langle Y_1(E_1), \ldots, Y_{|\Sigma|}(E_1)\rangle, \ldots$$

is a bad sequence, and letting $f(u) = 16u^{3+2u^{1+4u}}$, we have $\sum_{s=0}^{|\Sigma|} Y_s(E_h), u_h < f^{h+\max(|Q|,|\Sigma|)}(2)$ for all h.

PROOF. To show that the sequence in $\mathbb{N}^{|\Sigma|}$ is bad, consider two indices $i < j$. By the definition of the graph $\Upsilon_{Q,\Sigma}$, for each $h \in \{i, \ldots, j-1\}$, we have that E_{h+1} can be obtained from E_h by replacing some Δ_h^* with an FRE $E_h^\dagger$ whose height is strictly less than $|\Delta_h|$. Hence:

$$\begin{aligned} Y_{|\Delta_h|}(E_{h+1}) &= Y_{|\Delta_h|}(E_h) - 1 \\ Y_s(E_{h+1}) &= Y_s(E_h) \text{ for all } s \in \{|\Delta_h|+1, \ldots, |\Sigma|\}, \end{aligned}$$

so letting $S_{i,j}$ be the largest such $|\Delta_h|$, we conclude that $Y_{S_{i,j}}(E_j) < Y_{S_{i,j}}(E_i)$. Thus, as required,

$$\langle Y_1(E_i), \ldots, Y_{|\Sigma|}(E_i)\rangle \not\le \langle Y_1(E_j), \ldots, Y_{|\Sigma|}(E_j)\rangle.$$

We prove $\sum_{s=0}^{|\Sigma|} Y_s(E_h), u_h < f^{h+\max(|Q|,|\Sigma|)}(2)$ by induction on h.

We have $E_0 = \Sigma^*$ so $\sum_{s=0}^{|\Sigma|} Y_s(E_0) = 1$, and also $u_0 = 1$. Observing that $f^{\max(|Q|,|\Sigma|)}(2) \geq 2$, the base case is done.

Consider an edge in the path $\langle E_h, u_h\rangle \to \langle E_{h+1}, u_{h+1}\rangle$. Writing U for $f^{h+\max(|Q|,|\Sigma|)}(2)$, assume that $\sum_{s=0}^{|\Sigma|} Y_s(E_h)$ and u_h are strictly less than U. Since the lengths of E_h and $E_h^\dagger$ are, respectively, equal to $\sum_{s=0}^{|\Sigma|} Y_s(E_h)$ and at most $\left(\sum_{s=0}^{|\Sigma|} Y_s(E_h)\right) |Q|^{4u_h}$, and since $U > |Q|$, we have:

$$\sum_{s=0}^{|\Sigma|} Y_s(E_{h+1}) \leq \sum_{s=0}^{|\Sigma|} Y_s(E_h) + \left(\sum_{s=0}^{|\Sigma|} Y_s(E_h)\right) |Q|^{4u_h} - 1 < \\ U\left(1 + |Q|^{4U}\right) < U^{1+4U} < f(U).$$

Similarly, since also $U > |\Sigma|$, we have:

$$u_{h+1} \leq 16u_h^2 \left(\sum_{s=0}^{|\Sigma|} Y_s(E_h)\right) (|\Sigma|+1)^{2\left(\sum_{s=0}^{|\Sigma|} Y_s(E_h)\right)|Q|^{4u_h}} < \\ 16U^3(|\Sigma|+1)^{2U|Q|^{4U}} < 16U^3U^{2U^{1+4U}} = f(U).$$

But $f(U) = f^{(h+1)+\max(|Q|,|\Sigma|)}(2)$, so the induction is complete. □

LEMMA 3.10. *Suppose that:*

— *$\mathcal{T}$ is a downwards monotone transducer with set of states Q and alphabet Σ;*
— *$\mathcal{T}$ terminates from w_1;*
— *$N \geq 16u^2$ for all vertices $\langle E, u\rangle$ that are reachable from $\langle \Sigma^*, 1\rangle$ in $\Upsilon_{Q,\Sigma}$.*

Then $\mathcal{T}$ does not have a computation from w_1 of length N.

PROOF. For a contradiction, assume γ is a computation of $\mathcal{T}$ from w_1 of length N. Trivially, Σ^* is 1-frequent in γ. By applying Lemma 3.6 repeatedly, we obtain an infinite path $\langle E_0, u_0\rangle \to \langle E_1, u_1\rangle \to \ldots$ from $\langle \Sigma^*, 1\rangle$ in $\Upsilon_{Q,\Sigma}$, which is nonsense by Lemma 3.9 because of Dickson's Lemma. □

3.5. Main Result

Given the preceding lemmas, it remains to do two things. The first is to show that, in every graph $\Upsilon_{Q,\Sigma}$, the positive integers in all vertices that are reachable from $\langle \Sigma^*, 1\rangle$ are bounded by an Ackermann-like function of $|Q|$ and $|\Sigma|$. Although the vertices and edges of $\Upsilon_{Q,\Sigma}$ can be encoded using the classical Hydra trees of Kirby and Paris [1982], we do not require the full generality of the latter, but are able to obtain an Ackermann bound using Lemma 3.9 and recent results of Figueira et al. [2011] on lengths of bad sequences of tuples of natural numbers.

Writing $N(|Q|, |\Sigma|)$ for the obtained bound, it then remains to argue that a computation of $\mathcal{T}$ from w_1 can be non-deterministically guessed and checked in Ackermann time or space, but that can be done by a straightforward non-deterministic algorithm that explores the state-transition graph of the iterated transducer $\mathcal{T}^{N(|Q|,|\Sigma|)-1}$ on the fly.

THEOREM 3.11. *Termination for a downwards monotone transducer $\mathcal{T}$ with set of states Q and alphabet Σ, from a word w_1 over Σ, is in* ACKERMANN. *For fixed $|\Sigma|$, the problem is in $\mathbf{F}_{|\Sigma|+3}$.*

PROOF. From Lemma 3.9, for every path $\langle E_0, u_0\rangle \to \langle E_1, u_1\rangle \to \ldots$ from vertex $\langle \Sigma^*, 1\rangle$ in $\Upsilon_{Q,\Sigma}$, the sequence

$$\langle Y_1(E_0), \ldots, Y_{|\Sigma|}(E_0)\rangle, \langle Y_1(E_1), \ldots, Y_{|\Sigma|}(E_1)\rangle, \ldots$$

in $\mathbb{N}^{|\Sigma|}$ is bad, and for all h, $\max(Y_1(E_h), \ldots, Y_{|\Sigma|}(E_h)) < f^{h+\max(|Q|,|\Sigma|)}(2)$, i.e., in the terminology of Figueira et al. [2011], the sequence is $\max(|Q|, |\Sigma|)$-controlled by the function $g(h) = f^h(2)$. Since f is in class $\mathfrak{F}_2$ of the fast growing hierarchy, we have that g belongs to $\mathfrak{F}_3$. Also, g is monotone and satisfies $g(h) \geq \max(1, h)$ for all h, and we can assume that $|\Sigma| \geq 1$. Hence, [Figueira et al. 2011, Proposition 5.2] applies and gives us a function $M_s(t)$ such that M_s is in $\mathfrak{F}_{s+2}$ for each $s \geq 1$, and the length of $\langle Y_1(E_0), \ldots, Y_{|\Sigma|}(E_0)\rangle$, $\langle Y_1(E_1), \ldots, Y_{|\Sigma|}(E_1)\rangle$, ... is at most $M_{|\Sigma|}(\max(|Q|, |\Sigma|))$.

Since the distance of each $\langle E, u\rangle$ that is reachable from $\langle \Sigma^*, 1\rangle$ in $\Upsilon_{Q,\Sigma}$ is at most $M_{|\Sigma|}(\max(|Q|, |\Sigma|)) - 1$, we have by Lemma 3.9 that $N(|Q|, |\Sigma|) \geq 16u^2$, where

$$N(k, s) = 16(g(M_s(\max(k, s)) - 1 + \max(k, s)))^2.$$

Therefore, by Lemma 3.10, $\mathcal{T}$ terminates from w_1 if and only if it does not have a computation from w_1 of length $N(|Q|, |\Sigma|)$.

We conclude that termination of $\mathcal{T}$ from w_1 is decidable by guessing and checking an $N(|Q|, |\Sigma|)$-long computation of $\mathcal{T}$ from w_1, which is equivalent to guessing and checking a transduction of the iterated transducer $\mathcal{T}^{N(|Q|,|\Sigma|)-1}$ from w_1. From the definition of composition, states of $\mathcal{T}^{N(|Q|,|\Sigma|)-1}$ are $(N(|Q|, |\Sigma|) - 1)$-tuples of states of $\mathcal{T}$, and its transitions are

$$\langle q_1, \ldots, q_{N(|Q|,|\Sigma|)-1}\rangle \xrightarrow{a|a'} \langle q'_1, \ldots, q'_{N(|Q|,|\Sigma|)-1}\rangle$$

such that, for some $1 \leq i \leq i' \leq N(|Q|, |\Sigma|) - 1$, we have:

— $q_j = q'_j$ for all $1 \leq j < i$;
— there exist transitions

$$q_i \xrightarrow{a_i|a'_i} q'_i, \ldots, q_{i'} \xrightarrow{a_{i'}|a'_{i'}} q'_{i'}$$

with $a'_j = a_{j+1} \in \Sigma$ for all $i \leq j < i'$;
— $q_j = q'_j$ for all $i' < j \leq N(|Q|, |\Sigma|) - 1$;
— $a = a_i$, and if $1 < i$ then $a = \epsilon$;
— $a'_{i'} = a'$, and if $i' < N(|Q|, |\Sigma|) - 1$ then $a' = \epsilon$.

Observe also that it suffices to consider transductions of $\mathcal{T}^{N(|Q|,|\Sigma|)-1}$ whose segments between successive reads from w_1 are of length at most $|Q|^{N(|Q|,|\Sigma|)-1}$, because otherwise they would contain redundant loops. Since the guessing and checking can be performed without computing the entire iterated transducer and without remembering an entire transduction, but by storing at most one transition of $\mathcal{T}^{N(|Q|,|\Sigma|)-1}$ at a time together with a fixed number of pointers to $\mathcal{T}$ and w_1, and a counter up to $|Q|^{N(|Q|,|\Sigma|)-1}$, we have that space

$$O(N(|Q|, |\Sigma|) \times (\log |Q| + \log |\Sigma|) + \log |w_1|)$$

is sufficient for a non-deterministic algorithm.

Recalling that $M_{|\Sigma|}$ is in $\mathfrak{F}_{|\Sigma|+2}$ and that g is in $\mathfrak{F}_3 \subseteq \mathfrak{F}_{|\Sigma|+2}$, we have that $N(|Q|, |\Sigma|)$ as a function of $|Q|$ is also in $\mathfrak{F}_{|\Sigma|+2}$. Therefore, as a function of the combined size of $\mathcal{T}$ and w_1, the non-deterministic space bound is in $\mathfrak{F}_{|\Sigma|+2}$ when $|\Sigma|$ is fixed, and in general dominated by F_ω composed with a primitive recursive function. The respective memberships of $\mathbf{F}_{|\Sigma|+3}$ and ACKERMANN follow by the robustness of those classes with respect to changes in the model of computation [Schmitz 2013, Section 4.2.1]. □

We remark that, from the bound obtained in the proof, the downward rational termination problem for fixed $\mathcal{T}$ is decidable non-deterministically in space logarithmic in $|w_1|$.

4. LOWER BOUND

We use the following variant of the fast growing functions F_k, which give rise to the *Ackermann hierarchy* (cf. e.g. [Friedman 2001]):

$$A_1(x) = 2x \qquad\qquad A_{k+1}(x) = (A_k)^x(1), \text{ for } k \geq 1.$$

For example, A_2 is exactly exponentiation of 2, and A_3 is exactly tetration of 2.

One can check that, for all $k, p \geq 1$, there exists $x_{k,p} \geq 0$ such that, for all $x \geq x_{k,p}$, we have $A_k(x) > (F_{k-1})^p(x)$; hence $A_k \notin \mathfrak{F}_{k-1}$ if $k \geq 2$ by [Löb and Wainer 1970, Theorem 2.10]. Conversely, $A_k(x) \leq F_k(x)$ for all $k \geq 1$ and $x \geq 0$, so $A_k \in \mathfrak{F}_k$.

Moreover, each of the complexity classes $\mathbf{F}_k$ is equal [Schmitz 2013, Theorem 4.1] to

$$\bigcup_{k' < k,\, g \in \mathfrak{F}_{k'}} \mathrm{DT{\small IME}}(A_k(g(n))).$$

To obtain our lower bound result, we provide a construction of 'dependent counter programs' D_1, D_2, ... such that each D_{k+1} is computable from D_k in logarithmic space. For every k, D_k consists of routines for basic counter operations (increment, decrement, zero test, maximum test), and is dependent in the sense that it may call as subroutines the operations of another (non-dependent) counter program, where the latter remains to be specified and composed with D_k. Moreover, D_k is closely related to the A_k function above: provided C is a counter program that reliably implements a counter bounded by N (in the sense that transducers that correspond to its routines compute correctly, even if insertion errors are possible), then the composition $D_k[C]$ reliably implements a counter bounded by $A_k(N)$. Given a Turing machine of size K, we then use $D_K[C]$ with C reliable up to K to build a transducer that reliably simulates $A_K(K)$ steps of the machine (in the presence of insertion errors), and diverges if and only if the machine halts.

To facilitate formulating the construction of the routines that make up the dependent counter programs D_k, we introduce a few notions and notations for programming with transducers. We begin with the most basic and general (Section 4.1): forming sequential routines from transducers and non-deterministic jumps. Such a routine is itself compiled to a transducer, whose alphabet includes the routine's line labels. We then introduce a mechanism for calls to and returns from subroutines (Section 4.2), albeit with bounded stack heights which suffice for our purpose. The final and most intricate construct is a star operator (Section 4.3). The latter is a program transformer, and is able to produce programs that reliably implement counters up to successively larger bounds.

Along the way, we specialise from routines with arbitrary alphabets to those that aim to implement the four operations on bounded counters: increment, decrement, zero test and maximum test. The initial generality, however, is not excessive: our later constructions build transducers that implement the counter operations by sequences of transductions over carefully defined alphabets. The initial and final words in those computations are appropriate encodings of counter values. The latter can be thought of as sophisticated representations of natural numbers with large upper bounds, that enable the four operations to be performed correctly by small transducers in the presence of insertion errors.

Before reading the technical material that follows, the reader may want to glance forwards at the definitions of:

— the program $\mathit{Enum}(N)$ (Section 4.1) that implements a counter up to N using an alphabet of size N, so that each value is encoded by a single letter;

— the dependent program Double (Section 4.2) that implements a counter up to $2N$ using a single binary digit, assuming a library of routines that implements an N-bounded counter, for any N.

4.1. Basic Counter Programs

For a finite set of labels Λ and a finite alphabet Σ which are disjoint, a Λ-*labelled* Σ-*transducing routine* is a sequence of labelled commands

$$l_1 : c_1, \ldots, l_K : c_K,$$

where $l_1, \ldots, l_K \in \Lambda$ are pairwise distinct, for each $i < K$ we have that

— either c_i is a transducer with alphabet Σ
— or c_i is of the form $\mathsf{goto}\ G$ for a non-empty subset G of $\{l_1, \ldots, l_K\}$,

and c_K is return.

Given a Λ-labelled Σ-transducing routine P as above, let $\mathcal{T}(P)$ be a transducer with alphabet $\Lambda \cup \Sigma$ and the following properties:

— if the input word is of the form $l_i w$ with c_i a transducer and $w \in \Sigma^*$, the possible output words are all $l_{i+1} w'$ where w' is an output of c_i from input w;
— if the input word is of the form $l_i w$ with $c_i = \mathsf{goto}\ G$ and $w \in \Sigma^*$, the possible output words are $l_{i'} w$ where $l_{i'} \in G$;
— for all other inputs, there are no outputs.

We omit the straightforward details of $\mathcal{T}(P)$, but note that it is computable from P in logarithmic space.

Recall the notion of a computation of a finite transducer in Section 2.2.

Definition 4.1. For P still as above, and a relation R on Σ^*, we say that P *reliably computes* R if and only if

$$\{\langle w, w' \rangle \in (\Sigma^*)^2 \ : \ \mathcal{T}(P)_{\sqsubseteq} \text{ has a computation from } l_1 w \text{ to } l_K w'\} = R_{\sqsubseteq}$$

and all computations of $\mathcal{T}(P)_{\sqsubseteq}$ are finite, where $\mathcal{T}(P)_{\sqsubseteq}$ is the gainy version of the transducer of P (cf. Section 2.5) and $R_{\sqsubseteq} = \sqsubseteq ⨾ R ⨾ \sqsubseteq$ is the gainy version of the relation R.

Note that the requirements for computing reliably are strong. In particular, if started from $l_1 w$ where w is not a subword of a valid input for R, then P is required to stop before reaching return. When w is a valid input (or a subword), P must be able to produce (upon successful termination) all corresponding outputs of R; however, P may also have other computations that stop before reaching return, for example because a wrong guess was made along the way, or because insertion errors have been detected that would make the output invalid.

We now formalise when a collection of routines is regarded to simulate reliably the four basic operations on a bounded counter, using words as encodings of counter values. By a Λ-*labelled* Σ-*transducing counter program* C, we mean a collection of Λ-labelled Σ-transducing routines $C.\mathit{inc}$, $C.\mathit{dec}$, $C.\mathit{iszero}$ and $C.\mathit{ismax}$. We then say that C is N-*reliable* if and only if there exist words

$$\nu(0), \nu(1), \ldots, \nu(N-1) \in \Sigma^*$$

such that:

— for all distinct n and n', we have $\nu(n) \not\sqsubseteq \nu(n')$;
— $C.\mathit{inc}$ reliably computes $\{\langle \nu(n), \nu(n+1) \rangle \ : \ n < N-1\}$;
— $C.\mathit{dec}$ reliably computes $\{\langle \nu(n), \nu(n-1) \rangle \ : \ n > 0\}$;
— $C.\mathit{iszero}$ reliably computes $\{\langle \nu(0), \nu(0) \rangle\}$;

— $C.\mathit{ismax}$ reliably computes $\{\langle \nu(N-1), \nu(N-1)\rangle\}$.

Observe that, when C is N-reliable, the gainy transducers of its routines behave as expected: for example, running $\mathcal{T}(C.\mathit{iszero})_{\sqsubseteq}$ checks that the initial word is a subword of $\nu(0)$ (i.e., $\mathcal{T}(C.\mathit{iszero})_{\sqsubseteq}$ stops otherwise) and produces a superword w_0 of $\nu(0)$; running $\mathcal{T}(C.\mathit{inc})_{\sqsubseteq}$ from w_0 checks that w_0 in fact equals $\nu(0)$ and produces a superword w_1 of $\nu(1)$; similarly, for any $n < N$, running $\mathcal{T}(C.\mathit{inc})_{\sqsubseteq}$ from w_1 a further $n-1$ times checks that all intermediate words are correct and produces a superword w_n of $\nu(n)$; running $\mathcal{T}(C.\mathit{ismax})_{\sqsubseteq}$ from w_n checks that w_n equals $\nu(N-1)$ (in which case n is maximal) and produces a superword w'_n of $\nu(n)$; etc.

For a positive integer N, we define $\mathit{Enum}(N)$ to be a counter program with alphabet $\{a_0, a_1, \ldots, a_{N-1}\}$ as follows:

— $\mathit{Enum}(N).\mathit{inc}$ and $\mathit{Enum}(N).\mathit{dec}$ are two-line routines. Their first commands are transducers that input only one-letter words of the form a_n, where $n < N-1$ or $n > 0$ respectively, and output a_{n+1} or a_{n-1} respectively. Their second commands are return.
— Similarly, $\mathit{Enum}(N).\mathit{iszero}$ and $\mathit{Enum}(N).\mathit{ismax}$ are two-line routines with transducers that input only one-letter words a_0 or a_{N-1} respectively, and output them unchanged.

Skipping the straightforward details, we note that $\mathit{Enum}(N)$ is computable in space logarithmic in N.

LEMMA 4.2. *For every N, the counter program $\mathit{Enum}(N)$ is N-reliable.*

PROOF. The encodings of counter values are the one-letter words a_0, a_1, …, a_{N-1}, which are certainly incomparable with respect to the subword ordering.

Consider the routine $\mathit{Enum}(N).\mathit{inc}$, and let l_1 and l_2 be the labels of its two lines. By the definition of $\mathcal{T}(\mathit{Enum}(N).\mathit{inc})$, its only transductions are from words of the form $l_1 a_n$ to words of the form $l_2 a_{n+1}$. Hence, the only non-trivial computations of its gainy version $\mathcal{T}(\mathit{Enum}(N).\mathit{inc})_{\sqsubseteq}$ are of the form w, w', where $w \sqsubseteq l_1 a_n$ and $l_2 a_{n+1} \sqsubseteq w'$. We conclude that $\mathit{Enum}(N).\mathit{inc}$ reliably computes $\{\langle a_n, a_{n+1}\rangle : n < N-1\}$.

The arguments for the remaining routines of $\mathit{Enum}(N)$ are analogous. □

4.2. Dependent Counter Programs

Let Λ-*labelled* Σ-*transducing dependent routines* be defined by generalising the notion of routine to allow commands of the form call op, where

$$op \in \{\mathit{inc}, \mathit{dec}, \mathit{iszero}, \mathit{ismax}\}.$$

To complete such a dependent routine P, it needs to be composed with a Λ'-labelled Σ'-transducing counter program C. Any subroutine invocation call op in P then performs the counter operation $C.op$.

For such P and C, their composition is defined as a routine denoted $P[C]$ in Figure 1. Its alphabet is the union of Λ, Σ, Λ', Σ', and two fresh symbols: \$ and $\#$. For readability, only some line labels of $P[C]$ are displayed, but as standard for routines, they are all assumed to be disjoint from $P[C]$'s alphabet. We note that $P[C]$ is computable from P and C in logarithmic space.

The basic idea behind the definition of $P[C]$ is simple: configurations of P and C are both kept on the tape (i.e., in the word that $P[C]$ transduces repeatedly), separated by a special symbol; all transducers in the code of $P[C]$ check that a separator occurs at most once, hence catching errors that insert spurious special symbols. The counter-encoding output by each call of a subroutine of C is used as input for any next such call. Moreover, the initial tape when $P[C]$ is called may, in addition to an input for P, contain

```
            transduce $w\#w'$ to $lw\#w'$ where $w \in \Sigma^*,\ w' \in \Sigma'^*,\ l$ is the first label in $P$
$loop$ :    goto $\{trans^{\$}, trans^{\#}, goto^{\$}, goto^{\#}, call^{\#}, return^{\$}, return^{\#}\}$
$trans^{\$}$ : transduce $lw\$l'w'$ to $lw\$l'_{>}w'_{>}$ where the command at $l$ in $P$ is call $op$,
            $w \in \Sigma^*,$ the command at $l'$ in $C.op$ is a transducer $\mathcal{T}$,
            $l'_{>}$ is the next label, $w'\ R(\mathcal{T})\ w'_{>}$;   goto $\{loop\}$
$trans^{\#}$ : transduce $lw\#w'$ to $l_{>}w_{>}\#w'$ where the command at $l$ in $P$ is a transducer $\mathcal{T}$,
            $l_{>}$ is the next label, $w\ R(\mathcal{T})\ w_{>},\ w' \in \Sigma'^*$;   goto $\{loop\}$
$goto^{\$}$ : transduce $lw\$l'w'$ to $lw\$l'_{>}w'$ where the command at $l$ in $P$ is call $op$,
            $w \in \Sigma^*,$ the command at $l'$ in $C.op$ is goto $G$,
            $l'_{>} \in G,\ w' \in \Sigma'^*$;   goto $\{loop\}$
$goto^{\#}$ : transduce $lw\#w'$ to $l_{>}w\#w'$ where the command at $l$ in $P$ is goto $G$,
            $l_{>} \in G,\ w \in \Sigma^*,\ w' \in \Sigma'^*$;   goto $\{loop\}$
$call^{\#}$ : transduce $lw\#w'$ to $lw\$l'w'$ where the command at $l$ in $P$ is call $op$,
            $w \in \Sigma^*,\ l'$ is the first label in $C.op,\ w' \in \Sigma'^*$;   goto $\{loop\}$
$return^{\$}$ : transduce $lw\$l'w'$ to $l_{>}w\#w'$ where the command at $l$ in $P$ is call $op$,
            $l_{>}$ is the next label, $w \in \Sigma^*,$ the command at $l'$ in $C.op$ is return,
            $w' \in \Sigma'^*$;   goto $\{loop\}$
$return^{\#}$ : transduce $lw\#w'$ to $w\#w'$ where the command at $l$ in $P$ is return,
            $w \in \Sigma^*,\ w' \in \Sigma'^*$;   return
```

Fig. 1. Composing dependent routines with counter programs

a counter encoding which is to be used as input for the first call of a subroutine of C. The latter feature enables one or more dependent routines that use a common counter program to be performed in succession and possibly multiple times, while operating on the same counter.

The transducers in the loop of $P[C]$ consume words that conform to one of the following two disjoint regular expressions:

$\Lambda\Sigma^*\$\Lambda'\Sigma'^*$. Such a tape contains configurations of both P and C. The next step in the composition is a step of C. Moreover, the current line of P contains a call to the subroutine of C that is currently being executed.

$\Lambda\Sigma^*\#\Sigma'^*$. Here, the tape contains a configuration of P and the latest output of a subroutine of C. The next step in the composition is a step of P.

The type of the current tape, as well as the kind of operation that the current line of P or C (as appropriate) specifies, are guessed non-deterministically in line labelled $loop$. If such a guess is incorrect, the subsequent verification fails and $P[C]$ stops without terminating successfully. Otherwise, $P[C]$ updates the tape by performing the required operation and repeats the loop, unless the operation was P's return in which case $P[C]$ also returns.

Definition 4.3. Extending the notion of dependence to counter programs, for a dependent counter program D and a counter program C, we write $D[C]$ for the counter program whose routines $D[C].op$ are defined as $(D.op)[C]$.

Definition 4.4. For a dependent counter program D and a function $f : \mathbb{N}^+ \to \mathbb{N}^+$, we say that D is *f-reliable* if and only if, for all N-reliable counter programs C, $D[C]$ is $f(N)$-reliable.

```
 $l_1$ : goto $\{l_2, l_5\}$
 $l_2$ : transduce 0 to 0, or transduce 1 to 1
 $l_3$ : call $inc$; goto $\{l_{11}\}$
 $l_5$ : call $ismax$; transduce 0 to 1
 $l_7$ : goto $\{l_8, l_{10}\}$
 $l_8$ : call $dec$; goto $\{l_7\}$
$l_{10}$ : call $iszero$
$l_{11}$ : return
```

Fig. 2. Incrementing a double dependent counter

We now define a dependent counter program which is $\lambda x.2x$-reliable. When composed with an N-reliable counter program C, the former keeps track by a single 0 or 1 whether its value n is in the interval $[0, N)$ or $[N, 2N)$ (respectively), and employs C to maintain the value n or $n - N$ (respectively).

Let $Double$ be a dependent counter program with alphabet $\{0, 1\}$ as follows:

— $Double.inc$ is shown in Figure 2, with some lines merged for readability. In the first line, a guess is made whether the increment preserves the leading binary digit or not. In the former case, the routine just increments the auxiliary counter (i.e., calls inc); but in the latter, the auxiliary counter is checked for maximality, then the digit is changed from 0 to 1, and finally the auxiliary counter is repeatedly decremented until zero (lines l_7–l_{10}). $Double.dec$ is defined analogously.

— $Double.iszero$ and $Double.ismax$ have three lines each: check that the digit is 0 (respectively, 1), check that the auxiliary counter is zero (respectively, maximal), return.

LEMMA 4.5. *The dependent counter program* $Double$ *is* $\lambda x.2x$*-reliable.*

PROOF. Suppose C is an N-reliable counter program which uses $\nu(0), \ldots, \nu(N-1)$ as encodings of counter values. We claim that words

$$0\#\nu(0), \ldots, 0\#\nu(N-1), 1\#\nu(0), \ldots, 1\#\nu(N-1)$$

work as encodings of $0, \ldots, 2N-1$. Firstly, they are incomparable with respect to the subword ordering, since $\nu(0), \ldots, \nu(N-1)$ are and since $\#$ is not in the alphabet of C.

To show that $Double[C].inc$ reliably computes

$$\begin{aligned} R \;=\; & \{\langle 0\#\nu(n), 0\#\nu(n+1)\rangle \;:\; n < N-1\} \;\cup \\ & \{\langle 0\#\nu(N-1), 1\#\nu(0)\rangle\} \;\cup \\ & \{\langle 1\#\nu(n), 1\#\nu(n+1)\rangle \;:\; n < N-1\}, \end{aligned}$$

the non-trivial direction is to establish that, for every computation γ of the gainy transducer $\mathcal{T}((Double.inc)[C])_{\sqsubseteq}$ from a word of the form $l^{\dagger}w$ to a word of the form $l^{\ddagger}w'$, where $l^{\dagger}$ and $l^{\ddagger}$ are the first and last labels (respectively) in the routine $(Double.inc)[C]$, and where w and w' are words over its alphabet, we have $w \; R_{\sqsubseteq} \; w'$.

Consider such γ, w and w'. By the definitions of composition, and of the transducer of a routine, removing leading labels and top-level jump-steps from γ yields a sequence of words ζ which is from w to w', and in which all intermediate words are of the form either $\Lambda\Sigma^*\$\Lambda'\Sigma'^*$ or $\Lambda\Sigma^*\#\Sigma'^*$, where Λ, Σ and Λ', Σ' are the labels-alphabet pairs of $Double.inc$ and C (respectively). By the definitions of composition again, and of

$Double.inc$, and by the N-reliability of C, the most involved case is ζ being of the form

$$\begin{gathered} w \sqsubseteq u_0\#u'_0,\ l_1u_1\#u'_1,\ l_5u_2\#u'_2,\ l_5u_3\$l'_1u'_3,\ \ldots,\ l_5u_4\$l'_2u'_4,\ l_6u_5\#u'_5,\ l_7u_6\#u'_6, \\ \vdots \\ l_8u_{7+5i}\#u'_{7+5i},\ l_8u_{8+5i}\$l'_3u'_{8+5i},\ \ldots,\ l_8u_{9+5i}\$l'_4u'_{9+5i}, \\ l_9u_{10+5i}\#u'_{10+5i},\ l_7u_{11+5i}\#u'_{11+5i}, \\ \vdots \\ l_{10}u_{7+5k}\#u'_{7+5k},\ l_{10}u_{8+5k}\$l'_5u'_{8+5k},\ \ldots,\ l_{10}u_{9+5k}\$l'_6u'_{9+5k}, \\ l_{10}u_{10+5k}\#u'_{10+5k},\ u_{11+5k}\#u'_{11+5k} \sqsubseteq w', \end{gathered}$$

where:

— $l_1, \ldots, l_{11}$ are the labels of $Double.inc$ as in Figure 2;
— $u_0 \sqsubseteq \cdots \sqsubseteq u_5 \sqsubseteq 0$ and $1 \sqsubseteq u_6 \sqsubseteq \cdots \sqsubseteq u_{11+5k}$;
— l'_1 and l'_2 are the first and last labels (respectively) in $C.ismax$, and we have $u'_0 \sqsubseteq \cdots \sqsubseteq u'_3 \sqsubseteq \nu(N-1)$ and $\nu(N-1) \sqsubseteq u'_4 \sqsubseteq \cdots \sqsubseteq u'_7$;
— l'_3 and l'_4 are the first and last labels (respectively) in $C.dec$, and for each $0 \leq i < k$, we have $u'_{7+5i} \sqsubseteq u'_{8+5i} \sqsubseteq \nu(n_i)$ and $\nu(n_i - 1) \sqsubseteq u'_{9+5i} \sqsubseteq \cdots \sqsubseteq u'_{7+5(i+1)}$ for some $n_i > 0$;
— l'_5 and l'_6 are the first and last labels (respectively) in $C.iszero$, and we have $u'_{7+5k} \sqsubseteq \cdots \sqsubseteq u'_{8+5k} \sqsubseteq \nu(0)$ and $\nu(0) \sqsubseteq u'_{9+5k} \sqsubseteq \cdots \sqsubseteq u'_{11+5k}$.

It follows that $k = N - 1$ and $n_i = N - 1 - i$ for all i. We also conclude that $w \sqsubseteq 0\#\nu(N-1)$ and $1\#\nu(0) \sqsubseteq w'$, so indeed $w\ R_{\sqsubseteq}\ w'$. The latter is also inferred in the remaining cases for ζ, by obtaining $w \sqsubseteq b\#\nu(n)$ and $b\#\nu(n+1) \sqsubseteq w'$ for some $b \in \{0,1\}$ and $n < N - 1$.

We also need to establish that all computations of $\mathcal{T}((Double.inc)[C])_{\sqsubseteq}$ are finite. As in the preceding argument, in any such computation, intermediate insertion errors possibly occur only within the two local tape segments, but are managed by the transducers in the code of $Double.inc$ and the N-reliability of C. In particular, the loop in lines l_7–l_{10} can cycle at most $N - 1$ times.

Showing that $Double[C].dec$ reliably computes the appropriate relation is analogous, and the arguments for $Double[C].iszero$ and $Double[C].ismax$ are straightforward. □

An example corollary of Lemmas 4.2 and 4.5 is that

$$\underbrace{Double[\cdots Double}_{N}[Enum(1)]\cdots]$$

is a 2^N-reliable counter program. It works with encodings of counter values that are essentially N-digit binary numbers.

Let us additionally remark that an alternative definition of a double dependent counter, where the least significant binary digit is kept on the local tape (instead of the most significant as above) and, in case that digit is 0, the routine for increment just replaces it by 1, would not give us $(\lambda x.2x)$-reliability. That is because such increments would terminate successfully regardless of the auxiliary counter's local tape, i.e., without checking that the latter contributes to an encoding of a counter value.

We end our introduction of dependent counter programs by observing that, since dependent counter programs interact with the auxiliary counter exclusively by calling its subroutines, which of the latter's encodings are used is determined by the counter values. Formally:

LEMMA 4.6. *For any f-reliable Σ-transducing dependent counter program D, there exist unique functions*

$$\begin{aligned} N \in \mathbb{N}^+, n \in \{0, \ldots, f(N) - 1\} &\mapsto \mu(N, n) \in \Sigma^+ \\ N \in \mathbb{N}^+, n \in \{0, \ldots, f(N) - 1\} &\mapsto g(N, n) \in \{0, \ldots, N - 1\} \end{aligned}$$

such that, for any N-reliable Σ'-transducing counter program C and any $n \in \{0, \ldots, f(N) - 1\}$, the encoding of counter value n for $D[C]$ is $\mu(N, n) \# \nu(g(N, n))$, where $\nu(0), \ldots, \nu(N-1) \in \Sigma'^*$ are the encodings for C.

For example, for the $(\lambda x.2x)$-reliable $\{0, 1\}$-transducing dependent counter program $Double$, we have that functions

$$\mu(N, n) = \lfloor n/N \rfloor \qquad\qquad g(N, n) = n \bmod N$$

have the properties stated in Lemma 4.6.

4.3. A Star Operator

As an intermediate stage to the central construction of our lower bound proof for downward rational termination, suppose D is an f-reliable Λ-labelled Σ-transducing dependent counter program and N is a positive integer, and consider the counter program

$$\underbrace{D[\cdots D}_{N}[Enum(1)] \cdots]$$

which is the N-fold composition of D with the 1-reliable Λ'-labelled Σ'-transducing counter program $Enum(1)$ (recall that Σ' hence consists of the single letter a_0). For each $n \in \{1, \ldots, N\}$, let us write $\$_n$ and $\#_n$ for the separating symbols introduced by the n^{th} composition (counting from the outermost one).

Let μ and g be the functions as in Lemma 4.6 for D. We have that the N-fold composition of D with $Enum(1)$ is $f^N(1)$-reliable, and that the encoding of any counter value $k \in \{0, \ldots, f^N(1) - 1\}$ is

$$\mu(f^{N-1}(1), k) \,\#_1\, \mu(f^{N-2}(1), g(f^{N-1}(1), k)) \,\#_2 \\ \mu(f^{N-3}(1), g(f^{N-2}(1), g(f^{N-1}(1), k)) \,\#_3\, \cdots \,\#_N\, a_0.$$

By observing that control in the N-fold composition does not depend on the line labels that are introduced by the composition construct (cf. the trivial control structure in Figure 1), and that such labels do not appear in the counter value encodings, simplifying $\underbrace{D[\cdots D}_{N}[Enum(1)] \cdots]$ by omitting them yields the routines defined in Figure 3, which we denote by $D^N.op_1$, where

$$op_1 \in \{inc, dec, iszero, ismax\}.$$

The counter program D^N thus defined therefore also is $f^N(1)$-reliable and has the same counter value encodings as the N-fold composition.

The transducers in the loop of $D^N.op_1$ consume words that conform to one of the following two disjoint regular expressions:

$\Lambda\Sigma^*\$_1 \cdots \Lambda\Sigma^*\$_N\Lambda'\Sigma'^*$. Such a tape contains a stack of $N-1$ calls of subroutines of D and a call of a subroutine of the counter program $Enum(1)$, followed by a configuration of the latter. The next step is a step of $Enum(1)$.

$\Lambda\Sigma^*\$_1 \cdots \Lambda\Sigma^*\$_{n-1}\Lambda\Sigma^*\#_n\Sigma^*\#_{n+1} \cdots \Sigma^*\#_N\Sigma'^*$ *where* $1 \leq n \leq N$. After a stack of $n-1$ calls of subroutines of D, the configuration of the currently active subroutine is

```
        transduce $w_1\#_1 \cdots w_N\#_N w'$ to $l w_1\#_1 \cdots w_N\#_N w'$ where
        $l$ is the first label in $D.op_1,\ w_1, \ldots, w_N \in \Sigma^*,\ w' \in \Sigma'^*$
 loop : goto $\{step^{\$}, step^{\#}\}$
step$ : transduce $l_1 w_1 \$_1 \cdots l_N w_N \$_N l' w'$
        by performing on $w'$ the command at $l'$ in $Enum(1).op_{N+1}$
        (which may be a transducer, goto, return), where
        the command at $l_i$ in $D.op_i$ is call $op_{i+1}$ for all $1 \le i \le N$,
        $w_1, \ldots, w_N \in \Sigma^*,\ w' \in \Sigma'^*$
        goto $\{loop\}$
step# : transduce $l_1 w_1 \$_1 \cdots l_{n-1} w_{n-1} \$_{n-1} l_n w_n \#_n w_{n+1} \#_{n+1} \cdots w_N \#_N w'$
        by performing on $w_n$ the command at $l_n$ in $D.op_n$
        (which may be a transducer, goto, call, return), where
        $1 \le n \le N$, the command at $l_i$ in $D.op_i$ is call $op_{i+1}$ for all $1 \le i \le n-1$,
        $w_1, \ldots, w_N \in \Sigma^*,\ w' \in \Sigma'^*$
        if $n > 1$ or the command was not a return, then goto $\{loop\}$, else return
```

Fig. 3. Simplifying the N-fold composition of a dependent counter program

followed by outputs of the latest subroutine calls at all deeper levels, ending with the latest output from the counter program $Enum(1)$.

Given an f-reliable Λ-labelled Σ-transducing dependent counter program D, our key construction is of a dependent counter program D^* which is $(\lambda N.f^N(1))$-reliable. In other words, instead of the family of separate counter programs D^1, D^2, ... defined in Figure 3, we obtain a single dependent counter program, which behaves like D^N whenever it is composed with an N-reliable auxiliary counter.

We define D^* in Figure 4, which shows any dependent routine $D^*.op_1$. The latter differs from the routines $D^N.op_1$ (cf. Figure 3) as follows:

— The alphabet of $D^*.op_1$ is the union of Λ, Σ, Λ', Σ', the two fresh symbols \$ and #, and their hatted versions $\widehat{\$}$ and $\widehat{\#}$. Thus, the number of separating symbols is 4, in contrast to $2N$ in the alphabet of $D^N.op_1$.
— Initially, and at the start of each cycle through the main loop, the auxiliary counter is used to check that the total number of the separators \$ and # is Max, and that their hatted versions do not occur, where $Max - 1$ is the maximum value of that counter. An implementation (omitted for readability) is:
  (1) check that the auxiliary counter is zero;
  (2) check that the hatted symbols do not occur, and hat the first occurence of \$ or #;
  (3) increment the auxiliary counter until maximal, at each iteration moving the hat from the unique hatted separator to the next unhatted one;
  (4) remove the unique hat, and decrement the auxiliary counter until zero.

We observe that D^* is computable from D in logarithmic space.

LEMMA 4.7. *For every $f(x)$-reliable dependent counter program D, we have that D^* is $f^x(1)$-reliable.*

PROOF. Suppose C is an N-reliable counter program. As observed above, we have that D^N, the counter program defined in Figure 3, is $f^N(1)$-reliable, and that its en-

```
           check that the occurrence count of $\$, \#$ is $Max$
           and that the occurrence count of $\widehat{\$}, \widehat{\#}$ is $0$
           transduce $w_1\# \cdots w_m\#w'$ to $lw_1\# \cdots w_m\#w'$ where
           $m \geq 1,\ l$ is the first label in $D.op_1,\ w_1, \ldots, w_m \in \Sigma^*,\ w' \in \Sigma'^*$
  $loop$ : check that the occurrence count of $\$, \#$ is $Max$
           and that the occurrence count of $\widehat{\$}, \widehat{\#}$ is $0$
           goto $\{step^{\$}, step^{\#}\}$
$step^{\$}$ : transduce $l_1w_1\$ \cdots l_mw_m\$l'w'$
           by performing on $w'$ the command at $l'$ in $Enum(1).op_{N+1}$
           (which may be a transducer, goto, return), where
           $m \geq 1,$ the command at $l_i$ in $D.op_i$ is call $op_{i+1}$ for all $1 \leq i \leq m,$
           $w_1, \ldots, w_m \in \Sigma^*,\ w' \in \Sigma'^*$
           goto $\{loop\}$
$step^{\#}$ : transduce $l_1w_1\$ \cdots l_{n-1}w_{n-1}\$l_nw_n\#w_{n+1}\# \cdots w_m\#w'$
           by performing on $w_n$ the command at $l_n$ in $D.op_n$
           (which may be a transducer, goto, call, return), where
           $1 \leq n \leq m,$ the command at $l_i$ in $D.op_i$ is call $op_{i+1}$ for all $1 \leq i \leq n-1,$
           $w_1, \ldots, w_m \in \Sigma^*,\ w' \in \Sigma'^*$
           if $n > 1$ or the command was not a return, then goto $\{loop\}$, else return
```

Fig. 4. Defining the star operator on dependent counter programs

coding of any counter value $k \in \{0, \ldots, f^N(1) - 1\}$ is

$$\begin{aligned}\kappa(N, n) \ = \ & \mu(f^{N-1}(1), k) \,\#_1\, \mu(f^{N-2}(1), g(f^{N-1}(1), k)) \,\#_2 \\ & \mu(f^{N-3}(1), g(f^{N-2}(1), g(f^{N-1}(1), k)) \,\#_3\, \cdots \,\#_N\, a_0,\end{aligned}$$

where μ and g are the functions as in Lemma 4.6 for D.

It suffices to establish that $D^*[C]$ is also $f^N(1)$-reliable, with counter-value encodings $\mu^*(N, k)\#'\nu(g^*(N, k))$, where:

— the local-tape part

$$\begin{aligned}\mu^*(N, k) \ = \ & \mu(f^{N-1}(1), k) \,\#\, \mu(f^{N-2}(1), g(f^{N-1}(1), k)) \,\# \\ & \mu(f^{N-3}(1), g(f^{N-2}(1), g(f^{N-1}(1), k)) \,\#\, \cdots \,\#\, a_0\end{aligned}$$

is obtained from the encoding $\kappa(N, k)$ by erasing the separators' indices,
— the fresh symbol $\#'$ is introduced by the composition of D^* and C,
— the auxiliary-counter value $g^*(N, k)$ is 0, and
— the function ν gives the encodings of C.

Let us call a computation of $\mathcal{T}(D^N.op_1)_{\sqsubseteq}$ or of $\mathcal{T}((D^*.op_1)[C])_{\sqsubseteq}$, where

$$op_1 \in \{inc, dec, iszero, ismax\},$$

a *phase* if and only if:

— the line label of its initial configuration is either the first one or $loop$,
— its intermediate configurations do not have line label $loop$, and
— the line label of its final configuration is either $loop$ or the last one (whose command is return), or the computation is infinite.

The claim follows from the following observations:

(1) All phases of $\mathcal{T}(D^N.op_1)_\sqsubseteq$ are finite by definition, and the same is true for $\mathcal{T}((D^*.op_1)[C])_\sqsubseteq$ since C is N-reliable.
(2) For every phase of $\mathcal{T}(D^N.op_1)_\sqsubseteq$, the computation obtained by erasing the separators' indices can be simulated by a phase of $\mathcal{T}((D^*.op_1)[C])_\sqsubseteq$. In the latter, the initial and final values of the auxiliary counter are zero, and the checks involving it succeed since the phase of $\mathcal{T}(D^N.op_1)_\sqsubseteq$ contains transductions which verify that the total number of separators is N.
(3) For every phase of $\mathcal{T}((D^*.op_1)[C])_\sqsubseteq$, there are three cases:
  (a) If the total number of separators never exceeds N, then $\mathcal{T}(D^N.op_1)_\sqsubseteq$ can simulate a computation obtained from the phase by removing the counting checks, removing the auxiliary tape, and attaching indices 1 to N to the separators.
  (b) If the total number of separators does exceed N and the line label of the final configuration is $loop$, then the next counting check will fail, so $\mathcal{T}((D^*.op_1)[C])_\sqsubseteq$ cannot have a next phase.
  (c) If the total number of separators does exceed N and the phase ends by $\mathcal{T}((D^*.op_1)[C])_\sqsubseteq$ terminating successfully, then $\mathcal{T}(D^N.op_1)_\sqsubseteq$ can simulate a computation obtained from the phase by removing the counting check, removing the auxiliary tape, attaching indices from 1 to N to the separators that enabled the counting check to succeed, and removing other separators (spuriously inserted). □

We remark that the checking of occurence counts at every cycle through the main loops in D^* is necessary. Otherwise, insertion errors could cause infinite computations by spurious $\Lambda\Sigma^*\$$ segments that make the stack of subroutine calls grow unboundedly.

As example corollaries of Lemmas 4.2, 4.5 and 4.7, $Double^*$ is 2^x-reliable, and so $Double^*[Enum(N)]$ is 2^N-reliable. Like the counter program

$$\underbrace{Double[\cdots Double}_{N}[Enum(1)]\cdots]$$

considered earlier, the latter works with encodings of counter values that are essentially N-digit binary numbers.

Carrying on, $(Double^*)^*$ is $\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{x}$-reliable, and so $(Double^*)^*[Enum(N)]$ is $\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{N}$-reliable. The latter counter program is essentially an implementation of Stockmeyer's yardstick construction. It provides a tetrationally large reliable counter by working with a sequence of N reliable counters, each of which is used to count digits in the binary encoding of the next.

4.4. Simulating Ackermann-Bounded Turing Machines

Given a deterministic Turing machine $\mathcal{M}$, let $Test(\mathcal{M})$ be a dependent routine as follows, which is computable from $\mathcal{M}$ in logarithmic space:

(1) Use the auxiliary counter to insert N blank symbols onto the tape, where $N - 1$ is the maximal value of that counter.
(2) Use the auxiliary counter to simulate N steps of $\mathcal{M}$, working with symbols on the tape.
(3) If $\mathcal{M}$ has not halted within the N steps, stop.
(4) Otherwise, use the auxiliary counter to check that there are still only N symbols on the tape, stopping in case there are more.
(5) Diverge, i.e., enter an infinite loop.

LEMMA 4.8. *For every deterministic Turing machine $\mathcal{M}$ and N-reliable counter program C, we have that $\mathcal{M}$ halts within time N if and only if $\mathcal{T}(\mathit{Test}(\mathcal{M})[C])_{\sqsubseteq}$ has an infinite computation from the initial line label.*

PROOF. The 'only if' direction is trivial, and does not need to involve insertion errors. For the 'if' direction, observe that since C is N-reliable, $\mathit{Test}(\mathcal{M})[C]$ can diverge only by reaching the artificial infinite loop, in which case it has verified that $\mathcal{M}$ had halted within time N and had not been affected by insertion errors. □

We are now equipped to establish that, indeed, downward rational termination is ACKERMANN-hard:

THEOREM 4.9. *Given a deterministic Turing machine $\mathcal{M}$ of size K, we have that a transducer $\mathcal{T}(\mathcal{M})$ and a word w_1, over an alphabet of linear size, are computable in elementary time, such that $\mathcal{M}$ halts within time $A_K(K)$ if and only if $\mathcal{T}(\mathcal{M})_{\sqsubseteq}$ does not terminate from w_1.*

PROOF. Let D_K be the dependent counter program $(\cdots(\mathit{Double}\overbrace{{}^*)^*\cdots)^*}^{K-1}$. By Lemmas 4.2, 4.5 and 4.7, D_K is $A_K(x)$-reliable and $D_K[\mathit{Enum}(K)]$ is $A_K(K)$-reliable. Moreover, since the star operator is computable in logarithmic space, and since each of the $K-1$ applications enlarges the alphabet by an additive constant, we have that the counter program $D_K[\mathit{Enum}(K)]$ is computable in time elementary in K and has an alphabet of size linear in K. It remains to apply Lemma 4.8 to $\mathcal{M}$ and $D_K[\mathit{Enum}(K)]$. We thus obtain a routine

$$\mathit{Test}(\mathcal{M})[(\cdots(\mathit{Double}\overbrace{{}^*)^*\cdots)^*}^{K-1}[\mathit{Enum}(K)]]$$

whose transducer, let us call it $\mathcal{T}(\mathcal{M})$, has the required properties. □

5. SAFETY MTL SATISFIABILITY

We now show that the satisfiability problem for the safety fragment of MTL is inter-reducible with the termination problem for gainy transducers (equivalently, for downwards monotone transducers, cf. Section 2.5), thus improving the best known upper and lower bounds for the former. We rely here on results in the literature concerning *insertion channel machines (ICMs)* [Cécé et al. 1996], a model that is very closely related to gainy transducers.

5.1. Syntax of MTL

The formulas (negation normal) of MTL are built over a set of atomic events Σ using monotone Boolean connectives and time-constrained versions of the *next* operator $\bigcirc$, the *until* operator $\mathcal{U}$, and the *dual until* operator $\widetilde{\mathcal{U}}$:

$$\varphi ::= \top \mid \bot \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid a \mid \bigcirc_I \varphi \mid \varphi_1 \,\mathcal{U}_I\, \varphi_2 \mid \varphi_1 \,\widetilde{\mathcal{U}}_I\, \varphi_2,$$

where $a \in \Sigma$ and $I \subseteq \mathbb{R}_{\geq 0}$ is an interval with endpoints in $\mathbb{N} \cup \{\infty\}$.[5]

The primitive operators of MTL can be used to express further temporal operators, including the *constrained eventually* operator $\Diamond_I \varphi \equiv \top \,\mathcal{U}_I\, \varphi$ and the *constrained always* operator $\Box_I \varphi \equiv \bot \,\widetilde{\mathcal{U}}_I\, \varphi$, as well as further Boolean connectives by translation to negation normal form.

[5]Instead of atomic events that determine letters in timed words, atomic propositions could be employed. The associated complexity considerations do not affect the main results in this work.

5.2. Semantics of MTL

A *timed word* over alphabet Σ is a pair $\rho = \langle \sigma, \tau \rangle$, where σ is an infinite word over Σ and τ is an infinite sequence of non-negative reals that is strictly increasing and unbounded. The unboundedness assumption on τ is called *non-Zenoness* since it entails that each bounded time interval contains only finitely many events.

Given a timed word $\rho = \langle \sigma, \tau \rangle$ and an MTL formula φ, the satisfaction relation $\langle \rho, i \rangle \models \varphi$ (read 'ρ satisfies φ at position i') is defined as follows:

— $\langle \rho, i \rangle \models a$ iff $\sigma_i = a$;
— $\langle \rho, i \rangle \models \varphi_1 \wedge \varphi_2$ iff $\langle \rho, i \rangle \models \varphi_1$ and $\langle \rho, i \rangle \models \varphi_2$;
— $\langle \rho, i \rangle \models \varphi_1 \vee \varphi_2$ iff $\langle \rho, i \rangle \models \varphi_1$ or $\langle \rho, i \rangle \models \varphi_2$;
— $\langle \rho, i \rangle \models \bigcirc_I \varphi$ iff $\tau_{i+1} - \tau_i \in I$ and $\langle \rho, i+1 \rangle \models \varphi$;
— $\langle \rho, i \rangle \models \varphi_1 \, \mathcal{U}_I \, \varphi_2$ iff there exists $j \geq i$ such that $\langle \rho, j \rangle \models \varphi_2$, $\tau_j - \tau_i \in I$, and $\langle \rho, k \rangle \models \varphi_1$ for all k with $i \leq k < j$;
— $\langle \rho, i \rangle \models \varphi_1 \, \widetilde{\mathcal{U}}_I \, \varphi_2$ iff for all $j \geq i$ such that $\tau_j - \tau_i \in I$, either $\langle \rho, j \rangle \models \varphi_2$ or there exists k with $i \leq k < j$ and $\langle \rho, k \rangle \models \varphi_1$.

We say that ρ satisfies φ if $\langle \rho, 0 \rangle \models \varphi$.

5.3. Safety MTL

The *satisfiability problem* for MTL asks whether a given formula is satisfied by some timed word. This problem was shown undecidable in [Ouaknine and Worrell 2006a], motivating the introduction of the sub-logic safety MTL in [Ouaknine and Worrell 2006b]. Safety MTL is the fragment of MTL obtained by requiring that the interval I in each until operator $\mathcal{U}_I$ have finite length. Thus safety MTL allows bounded eventualities, such as $\Diamond_{(0,1)} a$, but not unbounded eventualities, such as $\Diamond_{(0,\infty)} a$. Here notice that, thanks to the non-Zenoness property, if a timed word ρ fails to satisfy $\Diamond_{(0,1)} a$ then there is some finite prefix ρ' of ρ such that all infinite extensions of ρ' also fail to satisfy $\Diamond_{(0,1)} a$. Such a property holds in general for safety MTL formulas; in the terminology of [Henzinger 1992] a formula of safety MTL defines a *safety property relative to the divergence of time*.

The satisfiability problem for safety MTL was shown to be decidable in [Ouaknine and Worrell 2006b] by an argument involving Higman's Lemma. It was later observed that this argument yields an upper bound in level $\mathfrak{F}_{\omega^\omega}$ of the fast-growing hierarchy [Schmitz 2012]. A non-elementary lower bound (in $\mathfrak{F}_3$) is given in [Bouyer et al. 2012] and an improved lower bound in $\mathfrak{F}_4$ is given in [Jenkins 2012].

5.4. Insertion Channel Machines

A *channel machine* consists of a finite-state automaton acting on an unbounded channel, or queue. In an *insertion channel machine (ICM)* the queue is subject to insertion errors, that is, extra letters can non-deterministically be inserted during a computation. Formally, a channel machine is a tuple $\mathcal{C} = \langle S, M, \Delta \rangle$, where S is a finite set of *control states*, M is a finite set of *messages*, and $\Delta \subseteq S \times \Sigma \times S$ is the transition relation over label set $\Sigma = \{m!, m? : m \in M\}$. A transition labelled $m!$ writes message m to the tail of the channel, and a transition labelled $m?$ reads message m from the head of the channel. We assume that M contains a special 'pointer' symbol $\triangleleft$, which is used below to define a notion of fairness.

We define an operational semantics for insertion channel machines as follows. A *global state* of $\mathcal{C}$ is a pair $\gamma = \langle s, x \rangle$, where $s \in S$ is the control state and $x \in M^*$ represents the contents of the channel. The rules in Δ induce a Σ-labelled transition relation on the set of global states thus: $\langle s, m!, t \rangle \in \Delta$ yields a transition $\langle s, x \rangle \xrightarrow{m!} \langle t, xm \rangle$ that writes m to the tail of the channel, and $\langle s, m?, t \rangle \in \Delta$ yields transitions

$\langle s, mx\rangle \xrightarrow{m?} \langle t, x\rangle$ and $\langle s, x\rangle \xrightarrow{m?} \langle t, x\rangle$. The last transition represents a type of insertion error: the ‘read’ transition is taken, although no letter is consumed from the head of the channel.

A *computation* of an ICM is an infinite sequence of transitions between global states:

$$\langle s_0, x_0\rangle \xrightarrow{\alpha_0} \langle s_1, x_1\rangle \xrightarrow{\alpha_1} \langle s_2.x_2\rangle \xrightarrow{\alpha_2} \langle s_3, x_3\rangle \xrightarrow{\alpha_3} \ldots$$

Such a computation is said to be *fair* if there are infinitely many consecutive pairs of transitions of the form:

$$\langle s_i, \triangleleft x\rangle \xrightarrow{\triangleleft ?} \langle s_{i+1}, x\rangle \xrightarrow{\triangleleft !} \langle s_{i+2}, x\triangleleft\rangle \qquad (1)$$

and there are no other transitions that write or read $\triangleleft$. Since each read transition (1) is error-free, in a fair computation, $\triangleleft$ is not subject to insertion errors and is read from the channel and immediately written back to the channel infinitely often. Intuitively each segment of a computation between consecutive reads of $\triangleleft$ represents a *cycle* of the channel. The *fair termination problem* for ICMs asks whether a given ICM has no infinite fair computation.

5.5. Four Reductions

Theorems 3.11 and 4.9 entail that the satisfiability problem for safety MTL is ACKERMANN-complete through four reductions:

$$\begin{tikzcd}
\boxed{\text{gainy rational termination}} \arrow[r, bend left, "\text{(ii)}"] & \boxed{\text{ICM fair termination}} \arrow[l, bend left, "\text{(i)}"] \arrow[r, bend left, "\text{(iii)}"] & \boxed{\text{safety MTL satisfiability}} \arrow[l, bend left, "\text{(iv)}"]
\end{tikzcd}$$

The reductions (i) and (ii) are almost immediate and require only logarithmic space. The idea is that reading and writing in an ICM correspond to input and output in a transducer, with insertion errors in a channel machine corresponding to gaininess of the corresponding transducer. Given a transducer $\mathcal{T}$ on alphabet Σ, one can construct an ICM $\mathcal{C}$ on channel alphabet $\Sigma \cup \{\triangleleft\}$ such that $w\ \mathrm{R}(\mathcal{T})\ w'$ if and only if $\mathcal{C}$ has a computation from $\langle s, \triangleleft w\rangle$ to $\langle s, \triangleleft w'\rangle$ for some fixed control state s. This can be achieved by mapping each transition $s \xrightarrow{a|b} t$ of $\mathcal{T}$ to transitions $s \xrightarrow{a?} s' \xrightarrow{b!} t$ in $\mathcal{C}$, where s' is a new ‘intermediate’ state. The translation in the reverse direction, from channel machines to transducers, is based on the same idea but with the caveat that the transducer must keep track of the control state of the ICM at the beginning and end of each cycle (which can easily be done, e.g., by augmenting the alphabet of the transducer). Under this correspondence, since each cycle of the channel machine $\mathcal{C}$ corresponds to a single complete transduction of $\mathcal{T}$, a computation of $\mathcal{C}$ yields an infinite computation of the transducer just in case it is fair.

The reduction (iii) is again straightforward and can be done in logarithmic space. The key idea is that the MTL formula $\Box_{[0,\infty)}(m! \to \Diamond_{\{1\}} m?)$ can be used to capture the behaviour of an unbounded channel. The formula says that each write-event is followed in exactly one time unit by a corresponding read-event. Since arbitrarily many of these events can occur within unit-length intervals, we can model an unbounded channel. Moreover we capture the behaviour of a fair channel because of the non-Zenoness assumption on timed words. Finally, in the absence of past operators, it is not possible to specify in MTL that a read-event be preceeded by a matching write-event. Thus we model a channel with insertion errors. We refer to [Bouyer et al. 2012] for full details of this reduction.

The most complex reduction is (iv): it is doubly exponential, and its details are available in [Jenkins 2012, Proposition 5.27], which builds on a translation from MTL to channel machines in [Bouyer et al. 2007].

ACKNOWLEDGMENTS

We thank Sylvain Schmitz, Patrick Totzke and the anonymous referees for helpful comments.

REFERENCES

Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. 2004a. Using Forward Reachability Analysis for Verification of Lossy Channel Systems. *Formal Meth. Sys. Des.* 25, 1 (2004), 39–65.

Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. 2004b. A Survey of Regular Model Checking. In *CONCUR (LNCS)*, Vol. 3170. Springer, Berlin, Heidelberg, 35–48.

R. Alur and T. A. Henzinger. 1992. Back to the Future: Towards a Theory of Timed Regular Languages. In *FOCS*. IEEE Comput. Soc., Los Alamitos, CA, 177–186.

Patricia Bouyer, Nicolas Markey, Joël Ouaknine, Philippe Schnoebelen, and James Worrell. 2012. On termination and invariance for faulty channel machines. *Formal Asp. Comput.* 24, 4–6 (2012), 595–607.

Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. 2007. The Cost of Punctuality. In *LICS*. IEEE Comput. Soc., Los Alamitos, CA, 109–120.

Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. 1996. Unreliable Channels are Easier to Verify than Perfect Channels. *Inf. Comput.* 124, 1 (1996), 20–31.

Pierre Chambart and Philippe Schnoebelen. 2007. Post Embedding Problem Is Not Primitive Recursive, with Applications to Channel Systems. In *FSTTCS (LNCS)*, Vol. 4855. Springer, Berlin, Heidelberg, 265–276.

Pierre Chambart and Philippe Schnoebelen. 2008. The Ordinal Recursive Complexity of Lossy Channel Systems. In *LICS*. IEEE Comput. Soc., Los Alamitos, CA, 205–216.

Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. 2011. Ackermannian and Primitive-Recursive Bounds with Dickson's Lemma. In *LICS*. IEEE Comput. Soc., Los Alamitos, CA, 269–278. Ext. ver. in CoRR, vol. abs/1007.2989.

Alain Finkel and Philippe Schnoebelen. 2001. Well-structured transition systems everywhere! *Theor. Comput. Sci.* 256, 1–2 (2001), 63–92.

H.M. Friedman. 2001. Long finite sequences. *J. Combin. Theor., Ser. A* 95, 1 (2001), 102–144.

T. A. Henzinger. 1991. *The Temporal Specification and Verification of Real-Time Systems*. Ph.D. Dissertation. Stanford University. Tech. rep. STAN-CS-91-1380.

T. A. Henzinger. 1992. Sooner is Safer Than Later. *Inf. Process. Lett.* 43, 3 (1992), 135–141.

T. A. Henzinger. 1998. It's About Time: Real-Time Logics Reviewed. In *CONCUR (LNCS)*, Vol. 1466. Springer, Berlin, Heidelberg, 439–454.

Mark Jenkins. 2012. *Synthesis and Alternating Automata over Real Time*. D.Phil. thesis. Oxford University.

Pierre Jullien. 1969. *Contribution à l'étude des types d'ordres dispersés*. Thèse de doctorat. Université de Marseille.

Prateek Karandikar and Sylvain Schmitz. 2013. The Parametric Ordinal-Recursive Complexity of Post Embedding Problems. In *FoSSaCS (LNCS)*, Vol. 7794. Springer, Berlin, Heidelberg, 273–288.

Laurie Kirby and Jeff Paris. 1982. Accessible independence results for Peano arithmetic. *Bull. London Math. Soc.* 14, 4 (1982), 285–293.

S. Rao Kosaraju. 1982. Decidability of Reachability in Vector Addition Systems (Preliminary Version). In *STOC*. ACM, New York, NY, 267–281.

Ranko Lazić, Joël Ouaknine, and James Worrell. 2013. Zeno, Hercules and the Hydra: Downward Rational Termination Is Ackermannian. In *MFCS (LNCS)*, Vol. 8087. Springer, Berlin, Heidelberg, 643–654.

Jérôme Leroux and Sylvain Schmitz. 2015. Reachability in vector addition systems demystified. In *LICS*. IEEE Comput. Soc., Los Alamitos, CA, 56–67. Ext. ver. in CoRR, vol. abs/1503.00745.

M.H. Löb and S.S. Wainer. 1970. Hierarchies of number-theoretic functions. I. *Arch. Math. Log.* 13, 1 (1970), 39–51.

Joël Ouaknine and James Worrell. 2006a. On Metric Temporal Logic and Faulty Turing Machines. In *FoSSaCS (LNCS)*, Vol. 3921. Springer, Berlin, Heidelberg, 217–230.

Joël Ouaknine and James Worrell. 2006b. Safety Metric Temporal Logic is Fully Decidable. In *TACAS (LNCS)*, Vol. 3920. Springer, Berlin, Heidelberg, 411–425.

Joël Ouaknine and James Worrell. 2007. On the Decidability of Metric Temporal Logic Over Finite Words. *Log. Meth. Comput. Sci.* 3, 1, Article 8 (2007), 27 pages.

Jacques Sakarovitch. 2009. *Elements of Automata Theory*. Camb. Univ. Press, UK. I–XXIV, 1–758 pages.

Sylvain Schmitz. 2012. Scientific Report - ESF Short Visit to Oxford University. (2012).

Sylvain Schmitz. 2013. *Complexity hierarchies beyond elementary*. Technical Report abs/1312.5686. CoRR.

Philippe Schnoebelen. 2010. Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets. In *MFCS (LNCS)*, Vol. 6281. Springer, Berlin, Heidelberg, 616–628.

Received Month Year; revised Month Year; accepted Month Year