Unboundedness problems for machines with reversal-bounded counters

Pascal Baumann[1][0000−0002−9371−0807], Flavio D'Alessandro[2], Moses Ganardi[1][0000−0002−0775−7781], Oscar Ibarra[3], Ian McQuillan[4], Lia Schütze[1][0000−0003−4002−5491], and Georg Zetzsche[1][0000−0002−6421−4388]

[1] Max Planck Institute for Software Systems (MPI-SWS), Germany
[2] Dept. of Mathematics G. Castelnuovo, Sapienza University of Rome, Italy
[3] Dept. of Computer Science, University of California, Santa Barbara, CA, USA
[4] Dept. of Computer Science, University of Saskatchewan, Saskatoon, Canada

Abstract. We consider a general class of decision problems concerning formal languages, called "(one-dimensional) unboundedness predicates", for automata that feature reversal-bounded counters (RBCA). We show that each problem in this class reduces—non-deterministically in polynomial time—to the same problem for just finite automata. We also show an analogous reduction for automata that have access to both a pushdown stack and reversal-bounded counters (PRBCA).

This allows us to answer several open questions: For example, we show that it is coNP-complete to decide whether a given (P)RBCA language L is bounded, meaning whether there exist words $w_1, \ldots, w_n$ with $L \subseteq w_1^* \cdots w_n^*$. For PRBCA, even decidability was open. Our methods also show that there is no language of a (P)RBCA of intermediate growth. This means, the number of words of each length grows either polynomially or exponentially. Part of our proof is likely of independent interest: We show that one can translate an RBCA into a machine with $\mathbb{Z}$-counters in logarithmic space, while preserving the accepted language.

Keywords: Formal languages · Decidability · Complexity · Counter automata · Reversal-bounded · Pushdown · Boundedness · Unboundedness

1 Introduction

A classic idea in the theory of formal languages is the concept of boundedness of a language. A language L over an alphabet Σ is called *bounded* if there exists a number $n \in \mathbb{N}$ and words $w_1, \ldots, w_n \in \Sigma^*$ such that $L \subseteq w_1^* \cdots w_n^*$. What makes boundedness important is that a rich variety of algorithmic problems become decidable for bounded languages. For example, when Ginsburg and Spanier [26] introduced boundedness in 1964, they already showed that given two context-free languages, one of them bounded, one can decide inclusion [26, Theorem 6.3]. This is because if $L \subseteq w_1^* \cdots w_n^*$ for a context-free language, then the set $\{(x_1, \ldots, x_n) \in \mathbb{N}^n \mid w_1^{x_1} \cdots w_n^{x_n} \in L\}$ is effectively semilinear, which permits expressing inclusion in Presburger arithmetic. Here, boundedness is a

crucial assumption: Hopcroft has shown that if $L_0 \subseteq \Sigma^*$ is context-free, then the problem of deciding $L_0 \subseteq L$ for a given context-free language L is decidable if and only if L_0 is bounded [36, Theorem 3.3].

The idea of translating questions about bounded languages into Presburger arithmetic has been applied in several other contexts. For example, Esparza, Ganty, and Majumdar [20] have shown that many classes of infinite-state systems are *perfect modulo bounded languages*, meaning that the bounded languages form a subclass that is amenable to many algorithmic problems. As another example, the subword ordering has a decidable first-order theory on bounded context-free languages [46], whereas on languages Σ^*, even the existential theory is undecidable [34]. This, in turn, implies that initial limit Datalog is decidable for the subword ordering on bounded context-free languages [6]. Finally, bounded context-free languages can be closely approximated by regular ones [16].

This raises the question of how one can decide whether a given language is bounded. For context-free languages this problem is decidable [26, Theorem 5.2(a)] in polynomial time [25, Theorem 19].

Boundedness for RBCA. Despite the importance of boundedness, it had been open for many years [8, 17][5] whether boundedness is decidable for one of the most well-studied types of infinite-state systems: *reversal-bounded (multi-)counter automata* (RBCA). These are machines with counters that can be incremented, decremented, and even tested for zero. However, in order to achieve decidability of basic questions, there is a bound on the number of times each counter can *reverse*, that is, switch between *incrementing* and *decrementing phases*. They were first studied in the 1970s [2, 37] and have received a lot of attention since [7–11, 13, 18, 23, 29, 33, 34, 40–42, 59]. The desirable properties mentioned above for bounded context-free languages also apply to bounded RBCA. Furthermore, any bounded language accepted by an RBCA (even one augmented with a stack) can be effectively determinized [39] (see also [8, 10]), opening up even more avenues to algorithmic analysis. This makes it surprising that decidability of boundedness remained open for many years.

Decidability of boundedness for RBCA was settled in [15], which proves boundedness decidable even for the larger class of vector addition systems with states (VASS), with acceptance by configuration. However, the results from [15] leave several aspects unclarified, which we investigate here:

Q1: What is the complexity of deciding boundedness for RBCA? The algorithm in [15] employs the KLMST decomposition for VASS [44, 47, 49, 51, 55], which is well-known to incur Ackermannian complexity [50].

Q2: Is boundedness decidable for *pushdown RBCA* (PRBCA) [37]? These are automata which, in addition to reversal-bounded counters, feature a stack. They can model recursive programs with numeric data types [33]. Whether boundedness is decidable was stated as open in [17, 18].

Q3: Are there languages of RBCA of intermediate growth? As far as we know, this is a long-standing open question in itself [38]. The *growth* of a language

---
[5] Note that [8] is about Parikh automata, which are equivalent to RBCA.

$L \subseteq \Sigma^*$ is the *counting function* $g_L \colon \mathbb{N} \to \mathbb{N}$, where $g_L(n)$ is the number of words of length n in L. This concept is closely tied to boundedness: For regular and context-free languages, it is known that a language has polynomial growth if and only if it is bounded (and it has exponential growth otherwise). A language is said to have *intermediate growth* if it has neither polynomial nor exponential growth.

Contribution I: We prove versions of one of the main results in [15], one for RBCA and one for PRBCA. Specifically, the paper [15] not only shows that boundedness is decidable for VASS, but it introduces a general class of *unboundedness predicates* for formal languages. It is then shown in [15] that any unboundedness predicate is decidable for VASS if and only if it is decidable for regular languages. Our first two main results are:

MR1: Deciding any unboundedness predicate for RBCA reduces in NP to deciding the same predicate for regular languages.

MR2: Deciding any unboundedness predicate for PRBCA reduces in NP to deciding the same predicate for context-free languages.

However, it should be noted that our results only apply to those unboundedness predicates from [15] that are *one-dimensional.* Fortunately, these are enough for our applications. These results allow us to settle questions (Q1)–(Q3) above and derive the exact complexity of several other problems. It follows that boundedness for both RBCA and PRBCA is coNP-complete, thus answering (Q1) and (Q2). Furthermore, the proof shows that if boundedness of a PRBCA does not hold, then its language has exponential growth. This implies that there are no RBCA languages of intermediate growth (thus settling (Q3)), and even that the same holds for PRBCA. In particular, deciding polynomial growth of (P)RBCA is coNP-complete and deciding exponential growth of (P)RBCA is NP-complete. We can also derive from our result that deciding whether a (P)RBCA language is infinite is NP-complete (but this also follows easily from [33], see Section 2). Finally, our results imply that it is PSPACE-complete to decide if an RBCA language $L \subseteq \Sigma^*$ is *factor universal*, meaning it contains every word of Σ^* as a factor (i.e. as an infix). Whether this problem is decidable for RBCA was also left as an open problem in [17, 18] (under the name *infix density*).

We prove our results (MR1) and (MR2) by first translating (P)RBCA into models that have $\mathbb{Z}$-counters instead of reversal-bounded counters. A $\mathbb{Z}$-*counter* is one that can be incremented and decremented, but cannot be tested for zero. Moreover, it can assume negative values. With these counters, acceptance is defined by reaching a configuration where all counters are zero (in particular, the acceptance condition permits a single zero-test on each counter). Here, finite automata with $\mathbb{Z}$-counters are called $\mathbb{Z}$-*VASS* [30]. $\mathbb{Z}$-counters are also known as *blind counters* [27] and it is a standard fact that RBCA are equivalent (in terms of accepted languages) to $\mathbb{Z}$-VASS [27, Theorem 2].

Despite the equivalence between RBCA and $\mathbb{Z}$-VASS being so well-known, there was apparently no known translation from RBCA to $\mathbb{Z}$-VASS in polynomial

Problem	$\mathbb{Z}$-VASS/RBCA	$\mathbb{Z}$-grammars/PRBCA
Boundedness	coNP-complete	coNP-complete
Finiteness	coNP-complete	coNP-complete
Factor universality	PSPACE-complete	undecidable

Table 1. Complexity results. The completeness statements are meant with respect to deterministic logspace reductions.

time. Here, the difficulty stems from simulating zero-tests (which can occur an unbounded number of times in an RBCA): To simulate these, the $\mathbb{Z}$-VASS needs to keep track of which counter has completed which incrementing/decrementing phase, using only polynomially many control states. It is also not obvious how to employ the $\mathbb{Z}$-counters for this, as they are only checked in the end.

Contribution II: As the first step of showing (MR1), we show that

MR3: RBCA can be translated (preserving the language) into $\mathbb{Z}$-VASS in logarithmic space.

This also implies that translations to and from another equivalent model, Parikh automata [42], are possible in polynomial time: It was recently shown that Parikh automata (which have received much attention in recent years [5, 7–9, 13, 22]) can be translated in polynomial time into $\mathbb{Z}$-VASS [31]. Together with our new result, this implies that one can translate among RBCA, $\mathbb{Z}$-VASS, and Parikh automata in polynomial time. Furthermore, our result yields a logspace translation of PRBCA into $\mathbb{Z}$-grammars, an extension of context-free grammars with $\mathbb{Z}$-counters. The latter is the first step for (MR2).

2 Main Results: Unboundedness and (P)RBCA

Reversal-bounded counter automata and pushdowns. A *pushdown automaton with k counters* is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, T, F)$ where Q is a finite set of states, Σ is an input alphabet, Γ is a stack alphabet, $q_0 \in Q$ is an initial state, T is a finite set of transitions $(p, w, \mathrm{op}, q) \in Q \times \Sigma^* \times \mathrm{Op} \times Q$, and $F \subseteq Q$ is a set of final states. Here Op is defined as

$$\mathrm{Op} = \{\mathsf{inc}_i, \mathsf{dec}_i, \mathsf{zero}_i, \mathsf{nz}_i \mid 1 \leq i \leq k\} \cup \Gamma \cup \bar{\Gamma} \cup \{\varepsilon\},$$

containing counter and stack operations. Here $\bar{\Gamma} = \{\bar{\gamma} \mid \gamma \in \Gamma\}$ is a disjoint copy of Γ. A *configuration* is a tuple $(p, \alpha, \boldsymbol{v}) \in Q \times \Gamma^* \times \mathbb{N}^k$. We write $(p, \alpha, \boldsymbol{u}) \xrightarrow{w} (p', \alpha', \boldsymbol{u}')$ if there is a $(p', w, \mathrm{op}, p') \in T$ such that one of the following holds:

- op $= \mathsf{inc}_i$, $\boldsymbol{u}' = \boldsymbol{u} + \boldsymbol{e}_i$, and $\alpha' = \alpha$ where $\boldsymbol{e}_i \in \mathbb{N}^k$ is the i-th unit vector,
- op $= \mathsf{dec}_i$, $\boldsymbol{u}' = \boldsymbol{u} - \boldsymbol{e}_i$, and $\alpha' = \alpha$,
- op $= \mathsf{zero}_i$, $\boldsymbol{u}[i] = 0$, $\boldsymbol{u}' = \boldsymbol{u}$, and $\alpha' = \alpha$

- op $= \mathsf{nz}_i$, $\boldsymbol{u}[i] \neq 0$, $\boldsymbol{u}' = \boldsymbol{u}$, and $\alpha' = \alpha$,
- op $= \gamma \in \varGamma$, $\boldsymbol{u}' = \boldsymbol{u}$, and $\alpha' = \alpha\gamma$,
- op $= \bar{\gamma} \in \bar{\varGamma}$, $\boldsymbol{u}' = \boldsymbol{u}$, and $\alpha'\gamma = \alpha$,
- op $= \varepsilon$, $\boldsymbol{u}' = \boldsymbol{u}$, and $\alpha' = \alpha$.

We extend this notation to longer runs in the natural way.

A (k, r)-*PRBCA* (pushdown reversal-bounded counter automaton) $(\mathcal{A}, r)$ consists of a pushdown automaton with k counters $\mathcal{A}$ and a number $r \in \mathbb{N}$, encoded in unary. A counter c_i *reverses* if the last (non-test) operation affecting it was inc_i and the next operation is dec_i, or vice versa. A run is r-*reversal bounded* if every counter reverses at most r times. The *language* of $(\mathcal{A}, r)$ is

$$L(\mathcal{A}, r) = \{w \in \varSigma^* \mid \exists q \in F, \ r\text{-reversal bounded run } (q_0, \varepsilon, \boldsymbol{0}) \xrightarrow{w} (q, \varepsilon, \boldsymbol{0})\}.$$

A (k, r)-*RBCA* (reversal-bounded counter automaton) is a (k, r)-PRBCA where $\mathcal{A}$ only uses counter operations. We denote by RBCA and PRBCA the class of (P)RBCA languages.

Notice that we impose the reversal bound *externally* (following [33]) whereas in alternative definitions found in the literature the automaton has to ensure *internally* that the number of reversals on every (accepting) run does not exceed r, e.g. [37]. Clearly, our definition subsumes the latter one; in particular, Theorem 1 also holds for (P)RBCAs with an internally checked reversal bound.

A d-dimensional $\mathbb{Z}$-*VASS* ($\mathbb{Z}$-*vector addition system with states*) is a tuple $\mathcal{V} = (Q, \varSigma, q_0, T, F)$, where Q is a finite set of states, $\varSigma$ is an alphabet, $q_0 \in Q$ is an initial state, T is a finite set of transitions $(p, w, \boldsymbol{v}, p') \in Q \times \varSigma^* \times \mathbb{Z}^d \times Q$, and $F \subseteq Q$ is a set of final states. A *configuration* of a $\mathbb{Z}$-VASS is a tuple $(p, \boldsymbol{v}) \in Q \times \mathbb{Z}^d$. We write $(p, \boldsymbol{u}) \xrightarrow{w} (p', \boldsymbol{u}')$ if there is a transition $(p, w, \boldsymbol{v}, p')$ such that $\boldsymbol{u}' = \boldsymbol{u} + \boldsymbol{v}$. We extend this notation to longer runs in the natural way. The *language* of the $\mathbb{Z}$-VASS is defined as

$$L(\mathcal{V}) = \{w \in \varSigma^* \mid \exists q \in F : \ (q_0, \boldsymbol{0}) \xrightarrow{w} (q, \boldsymbol{0})\}.$$

A *(d-dimensional) $\mathbb{Z}$-grammar* is a tuple $G = (N, \varSigma, S, P)$ with disjoint finite sets N and $\varSigma$ of nonterminal and terminal symbols, a start nonterminal $S \in N$, and a finite set of productions P of the form $(A, u, \boldsymbol{v}) \in N \times (N \cup \varSigma)^* \times \mathbb{Z}^d$. We also write $(A \to u, \boldsymbol{v})$ instead of $(A, u, \boldsymbol{v})$. We call $\boldsymbol{v}$ the *(counter) effect* of the production $(A \to u, \boldsymbol{v})$. For words $x, y \in (N \cup \varSigma)^*$, we write $x \Rightarrow^{\boldsymbol{v}} y$ if there is a production $(A \to u, \boldsymbol{v})$ such that $x = rAs$ and $y = rus$. Moreover, we write $x \overset{*}{\Rightarrow}{}^{\boldsymbol{v}} y$ if there are words $x_1, \ldots, x_n \in (N \cup \varSigma)^*$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{Z}^d$ with $x \Rightarrow^{\boldsymbol{v}_1} x_1 \Rightarrow^{\boldsymbol{v}_2} \cdots \Rightarrow^{\boldsymbol{v}_n} x_n = y$ and $\boldsymbol{v} = \boldsymbol{v}_1 + \cdots + \boldsymbol{v}_n$. We use the notation $\Rightarrow$ if the counter effects do not matter: We have $x \Rightarrow y$ if there exists $\boldsymbol{v}$ such that $x \Rightarrow_{\boldsymbol{v}} y$; and similarly for $\overset{*}{\Rightarrow}$. If derivations are restricted to a subset $Q \subseteq P$ of productions, we write $\Rightarrow_Q$ (resp. $\overset{*}{\Rightarrow}_Q$).

The *language* of the $\mathbb{Z}$-grammar G is the set of all words $w \in \varSigma^*$ such that $S \overset{*}{\Rightarrow}{}^{\boldsymbol{0}} w$. In other words, if there exists a derivation $S \overset{*}{\Rightarrow} w$ where the effects of all occurring productions sum to the zero vector $\boldsymbol{0}$. $\mathbb{Z}$-grammars of dimension d are also known as *valence grammars over $\mathbb{Z}^d$* [21].

For our purposes it suffices to assume a unary encoding of the $\mathbb{Z}^d$-vectors (effects) occurring in $\mathbb{Z}$-VASS and $\mathbb{Z}$-grammars. However, this is not a restriction: Counter updates with n-bit binary encoded numbers can be easily simulated with unary encodings at the expense of dn many fresh counters (see Appendix A.1).

Conversion results. The following is our first main theorem:

Theorem 1. *RBCA can be converted into $\mathbb{Z}$-VASS in logarithmic space. PRBCA can be converted into $\mathbb{Z}$-grammars in logarithmic space.*

By *convert*, we mean a translation that preserves the accepted (resp. generated) language. There are several machine models that are equivalent (in terms of accepted languages) to RBCA. With Theorem 1, we provide the last missing translation:

Corollary 1. *The following models can be converted into each other in logarithmic space: (i) RBCA, (ii) $\mathbb{Z}$-VASS, (iii) Parikh automata with $\exists$PA acceptance, and (iv) Parikh automata with semilinear acceptance.*

Roughly speaking, a Parikh automaton is a machine with counters that can only be incremented. Then, a run is accepting if the final counter values belong to some semilinear set. Parikh automata were introduced by Klaedtke and Rueß [42], where the acceptance condition is specified using a semilinear representation (with base and period vectors), yielding (iv) above. As done, e.g., in [34], one could also specify it using an existential Presburger formula (briefly $\exists$PA), yielding the model in (iii) above. Theorem 1 proves (i)$\Rightarrow$(ii), whereas (ii)$\Rightarrow$(i) is easy (a clever and very efficient translation is given in [41, Theorem 4.5]). Moreover, (ii)$\Rightarrow$(iii) and (ii)$\Rightarrow$(iv) are clear as well. For (iii)$\Rightarrow$(ii), one can proceed as in [31, Prop. V.1], and (iv)$\Rightarrow$(ii) is also simple.

Unboundedness predicates. We shall use Theorem 1 to prove our second main theorem, which involves unboundedness predicates as introduced in [15]. In [15], unboundedness predicates can be one-dimensional or multi-dimensional, but in this work, we only consider one-dimensional unboundedness predicates.

Let Σ be an alphabet. A *(language) predicate* is a set of languages over Σ. If $\mathfrak{p}$ is a predicate and $L \subseteq \Sigma^*$ is a language, then we write $\mathfrak{p}(L)$ to denote that $\mathfrak{p}$ holds for the language L (i.e. $L \in \mathfrak{p}$). A predicate $\mathfrak{p}$ is called a *(one-dimensional) unboundedness predicate* if the following conditions are met for all $K, L \subseteq \Sigma^*$:

(U1) If $\mathfrak{p}(K)$ and $K \subseteq L$, then $\mathfrak{p}(L)$.   (U3) If $\mathfrak{p}(K \cdot L)$, then $\mathfrak{p}(K)$ or $\mathfrak{p}(L)$.
(U2) If $\mathfrak{p}(K \cup L)$, then $\mathfrak{p}(K)$ or $\mathfrak{p}(L)$.   (U4) $\mathfrak{p}(L)$ if and only if $\mathfrak{p}(F(L))$.

Here $F(L) = \{v \in \Sigma^* \mid \exists u, w \in \Sigma^*: uvw \in L\}$ is the set of *factors* of L (sometimes also called *infixes*). In particular, the last condition says that $\mathfrak{p}$ only depends on the set of factors occurring in a language.

For an unboundedness predicate $\mathfrak{p}$ and a class $\mathcal{C}$ of finitely represented languages (such as automata or grammars), let $\mathfrak{p}(\mathcal{C})$ denote the problem of deciding $\mathfrak{p}$ for a given language L from $\mathcal{C}$. Formally, $\mathfrak{p}(\mathcal{C})$ is the following decision problem:

Given A language L from $\mathcal{C}$.
Question Does $\mathfrak{p}(L)$ hold?

For example, $\mathfrak{p}(\mathsf{RBCA})$ is the problem of deciding $\mathfrak{p}$ for reversal-bounded multi-counter automata and $\mathfrak{p}(\mathsf{NFA})$ is the problem of deciding $\mathfrak{p}$ for NFAs. We mention that the axioms (U1)–(U4) are slightly stronger than the axioms used in [15], but the resulting set of decision problems is the same with either definition (since in [15], one always decides whether $\mathfrak{p}(F(L))$ holds). Thus, the statement of Theorem 2 is unaffected by which definition is used. See Appendix A.2 for details.

The following examples of (one-dimensional) unboundedness predicates for languages $L \subseteq \Sigma^*$ have already been established in [15]. We mention them here to give an intuition for the range of applications of our results:

Not being bounded Let $\mathfrak{p}_{\mathsf{notb}}(L)$ if and only if L is *not* a bounded language.
Non-emptiness Let $\mathfrak{p}_{\neq\emptyset}(L)$ if and only if $L \neq \emptyset$.
Infinity Let $\mathfrak{p}_\infty(L)$ if and only if L is infinite.
Factor-universality Let $\mathfrak{p}_{\mathsf{funi}}(L)$ if and only if $\Sigma^* \subseteq F(L)$.

It is not difficult to prove that these are unboundedness predicates, but proofs can be found in [15]. The following is our second main theorem:

Theorem 2. *Let $\mathfrak{p}$ be a one-dimensional unboundedness predicate. There is an* NP *reduction from* $\mathfrak{p}(\mathsf{PRBCA})$ *to* $\mathfrak{p}(\mathsf{PDA})$. *Moreover, there is an* NP *reduction from* $\mathfrak{p}(\mathsf{RBCA})$ *to* $\mathfrak{p}(\mathsf{NFA})$.

Here, an NP *reduction* from problem $A \subseteq \Sigma^*$ to $B \subseteq \Sigma^*$ is a non-deterministic polynomial-time Turing machine such that for every input word $w \in \Sigma^*$, we have $w \in A$ iff there exists a run of the Turing machine producing a word in B.

Let us now see some applications of Theorem 2, see also Table 1. The following completeness results are all meant w.r.t. deterministic logspace reductions.

Corollary 2. *Boundedness for* PRBCA *and for* RBCA *is* coNP-*complete.*

For Corollary 2, we argue that deciding *non-boundedness* is NP-complete. To this end, we apply Theorem 2 to the predicate $\mathfrak{p}_{\mathsf{notb}}$ and obtain an NP upper bound, because boundedness for context-free languages is decidable in polynomial time [25]. The NP lower bound follows easily from NP-hardness of the non-emptiness problem for RBCA [29, Theorem 3] and thus PRBCA.

Corollary 3. *Finiteness for* PRBCA *and for* RBCA *is* coNP-*complete.*

We show Corollary 3 by proving that checking *infinity* is NP-complete. The upper bound follows from Theorem 2 via the predicate $\mathfrak{p}_\infty$. As above, NP-hardness is inherited from the non-emptiness problem for RBCA and PRBCA.

The results in Corollary 3 are, however, not new. They follow directly from the fact that for a given PRBCA (or RBCA), one can construct in polynomial time a formula in existential Presburger arithmetic ($\exists$PA) for its Parikh image, as shown in [37] for RBCA and in [33] for PRBCA. It is a standard result about $\exists$PA that for each formula φ, there exists a bound B such that (i) B is at most

exponential in the size of φ and (ii) φ defines an infinite set if and only if φ is satisfied for some vector with some entry above B. For example, this can be deduced from [54]. Therefore, one can easily construct a second $\exists PA$ formula φ' such that φ defines an infinite set if and only if φ' is satisfiable.

Corollary 4. *Factor universality for* RBCA *is* PSPACE-*complete.*

Whether factor universality is decidable for RBCA was left as an open problem in [17, 18] (there under the term *infix density*). Corollary 4 follows from Theorem 2 using $\mathfrak{p}_{\mathsf{funi}}$, because factor universality for NFAs is PSPACE-complete: To decide if $\Sigma^* \subseteq F(R)$, for a regular language R, we can just compute an automaton for $F(R)$ and check inclusion in PSPACE. For the lower bound, one can reduce the PSPACE-complete universality problem for NFAs, since for $R \subseteq \Sigma^*$, the language $(R\#)^* \subseteq (\Sigma \cup \{\#\})^*$ is factor universal if only if $R = \Sigma^*$. Note that factor universality is known to be undecidable already for one-counter languages [18], and thus in particular for PRBCA. However, it is decidable for pushdown automata with a bounded number of reversals of the stack [18].

Beyond pushdowns. Theorem 2 raises the question of whether for any class $\mathcal{M}$ of machines, one can reduce any unboundedness predicates for $\mathcal{M}$ extended with reversal-bounded counters to the same predicate for just $\mathcal{M}$. This is not the case: For example, consider second-order pushdown automata, short 2-PDA. If we extend these by adding reversal-bounded counters, then we obtain 2-PRBCA. Then, the infinity problem is decidable for 2-PDA [35] (see [3, 4, 14, 32, 53, 57] for stronger results). However, the class of 2-PRBCA does not even have decidable emptiness, let alone decidable infinity. This is shown in [58, Proposition 7] (see [43, Theorem 4] for an alternative proof). Thus, infinity for 2-PRBCA cannot be reduced to infinity for 2-PDA.

Growth. Finally, we employ the methods of the proof of Theorem 2 to show a dichotomy of the growth behavior of languages accepted by RBCA. For an alphabet Σ, we denote by $\Sigma^{\leq m}$ the set of all words over Σ of length at most m. We say that a language $L \subseteq \Sigma^*$ has *polynomial growth*[6] if there is a polynomial $p(x)$ such that $|L \cap \Sigma^{\leq m}| \leq p(m)$ for all $m \geq 0$. Languages of polynomial growth are also called *sparse* or *poly-slender*. We say that L has *exponential growth* if there is a real number $r > 1$ such hat $|L \cap \Sigma^{\leq m}| \geq r^m$ for infinitely many m. Since a language of the form $w_1^* \cdots w_n^*$ clearly has polynomial growth, it is well-known that bounded languages have polynomial growth. We show that (a) within the PRBCA languages (and in particular within the RBCA languages), the converse is true as well and (b) all other languages have exponential growth (in contrast to some models, such as 2-PDA [28], where this dichotomy does not hold):

Theorem 3. *Let L be a language accepted by a PRBCA. Then L has polynomial growth if and only if L is bounded. If L is not bounded, it has exponential growth.*

---

[6] In [25], polynomial and exponential growth are defined with Σ^m in place of $\Sigma^{\leq m}$, but this leads to equivalent notions, see Appendix D

3   Translating reversal-bounded counters into $\mathbb{Z}$-counters

Reducing the number of reversals to one. In this section we prove Theorem 1, the conversion from RBCA to $\mathbb{Z}$-VASS. In [29, Lemma 1], it is claimed that given a (k, r)-RBCA, one can construct in time polynomial in k and r a $(k\lceil (r+1)/2\rceil, 1)$-RBCA that accepts the same language. The reference [2] that they provide does include such a construction [2, proof of Theorem 5]. The construction in [2] is only a rough sketch and makes no claims about complexity, but by our reading of the construction, it keeps track of the reversals of each counter in the state, which would result in an exponential blow-up.

Instead, we proceed as follows. Consider a (k, r)-RBCA with counters $c_1, \ldots, c_k$. Without loss of generality, assume $r = 2m - 1$. We will construct an equivalent $(2k(r+1), 1)$-RBCA. Looking at the behavior of a single counter c_i, we can decompose every r-reversal bounded run into subruns without reversals. We call these subruns *phases* and number them from 1 to at most $2m$. The odd (even) numbered phases are *positive* (*negative*), where c_i is only incremented (decremented). We replace c_i by m one-reversal counters $c_{i,1}, \ldots, c_{i,m}$, where $c_{i,j}$ records the increments on c_i during the positive phase $2j - 1$.

However, our machine needs to keep track of which counters are in which phase, in order to know which of the counters $c_{i,j}$ it currently has to use. We achieve this as follows: For each of the k counters c_i, we also have an additional set of $2m = r + 1$ "phase counters" $p_{i,1}, \ldots, p_{i,2m}$ to store which phase we are in. This gives $km + k(r+1) \le 2k(r+1)$ counters in total. We encode that counter c_i is in phase j by setting $p_{i,j}$ to 1 and setting $p_{i,j'}$ to 0 for each $j' \ne j$. Since we only ever increase the phase, the phase counters are one-reversal as well.

Using non-zero-tests, at any point, the automaton can nondeterministically guess and verify the current phase of each counter. This allows it to pick the correct counter $c_{i,j}$ for each instruction. When counter c_i is in a positive phase $2j - 1$, then increments and decrements on c_i are simulated as follows:

increment increment $c_{i,j}$
decrement go into the next (negative) phase $2j$; then non-deterministically pick some $\ell \in [1, j]$ and decrement $c_{i,\ell}$. We cannot simply decrement $c_{i,j}$ as we might have switched to phase j while c_i had a non-zero value and hence it is possible that c_i could be decremented further than just $c_{i,j}$ allows.

When counter c_i is in a negative phase $2j$, then we simulate increments and decrements as follows:

increment go into the next phase $2j + 1$ (unless $j = m$; then the machine blocks) and increment $c_{i,j+1}$.
decrement non-deterministically pick some $\ell \in [1, j]$ and decrement $c_{i,\ell}$.

Finally, to simulate a zero-test on c_i, we test all counters $c_{i,1}, \ldots, c_{i,m}$ for zero, while for the simulation of a non-zero-test on c_i we non-deterministically pick one of the counters $c_{i,1}, \ldots, c_{i,m}$ to test for non-zero.

Correctness can be easily verified by the following properties. If at some point c_i is in phase $2j - 1$ or $2j$ then (i) $\sum_{\ell=1}^{j} c_{i,\ell} = c_i$, (ii) the counters $c_{i,1}, \ldots, c_{i,j}$

have made at most one reversal, and (iii) the counters $c_{i,j+1}, \ldots, c_{i,m}$ have not been touched (in particular, they are zero). Furthermore, if c_i is in a positive phase $2j - 1$ then $c_{i,j}$ has made no reversal yet.

Note that this construction replaces every transition of the original system with $\mathcal{O}(r)$ new transitions (and states). Our construction therefore yields only a linear blowup in the size of the system (constant if r is fixed). See Appendix B for the details of the construction.

From 1-reversal to $\mathbb{Z}$-counters. We now turn the $(k, 1)$-RBCA into a $\mathbb{Z}$-VASS. The difference between a 1-reversal-bounded counter and a $\mathbb{Z}$-counter is that (i) a non-negative counter should block if it is decremented on counter value 0, and (ii) a 1-reversal-bounded counter allows (non-)zero-tests. Observe that all zero-tests occur before the first increment or after the last decrement. All non-zero-tests occur between the first increment and the last decrement.

If the number k of counters is bounded, then the following simple solution works. The $\mathbb{Z}$-VASS stores the information which of the counters has not been incremented yet and which counters will not be incremented again in the future. This information suffices to simulate the counters faithfully (in terms of the properties (i) and (ii) above) and increases the state space by a factor of $2^k \cdot 2^k$. The latter information needs to be guessed (by the automaton) and is verified by means that all counters are zero in the end.

In the general case we introduce a variant of $\mathbb{Z}$-VASS that can guess polynomially many bits in the beginning and read them throughout the run. A d-dimensional $\mathbb{Z}$-*VASS with guessing* ($\mathbb{Z}$-*VASSG*) has almost the same format as a d-dimensional $\mathbb{Z}$-VASS, except that each transition additionally carries a propositional formula over some finite set of variables X. A word $w \in \Sigma^*$ is accepted by the $\mathbb{Z}$-VASSG if there exists an assignment $\nu \colon X \to \{0, 1\}$ and an accepting run $(q_0, \mathbf{0}) \xrightarrow{w} (q, \mathbf{0})$ for some $q \in F$ such that all formulas appearing throughout the run are satisfied by ν.

We have to eliminate zero- and non-zero-tests of the $(k, 1)$-RBCA. Whether a (non-)zero-test is successful depends on which phase a counter is currently in (and whether in the end, every counter is zero; but we assume that our acceptance condition ensures this). Each counter goes through at most 4 phases:

1. before the first increment,
2. the "increment phase",
3. the "decrement phase", and
4. after the last decrement.

Hence, every run can be decomposed into $4k$ (possibly empty) segments, in which no counter changes its phase. The idea is to guess the phase of each counter in each segment. Hence, we have propositional variables $p_{i,j,\ell}$ for $i \in [1, 4k]$, $j \in [1, k]$, and $\ell \in [1, 4]$. Then $p_{i,j,\ell}$ is true iff in segment i, counter j is in phase ℓ. We will have to check that the assignment is *admissible* for each counter, meaning that the sequence of phases for each counter adheres to the order described above.

We modify the machine as follows. In its state, it keeps a number $i \in [1, 4k]$ which holds the current segment. At the beginning of the run, the machine checks that the assignment ν is admissible using a propositional formula: It checks that

(i) for each segment i and each counter j there exists exactly one phase ℓ so that $\mathtt{p}_{i,j,\ell}$ is true, and (ii) the order of phases above is obeyed. Then, for every operation on a counter, the machine checks that the operation is consistent with the current segment. Moreover, if the current operation warrants a change of the segment, then the segment counter i is incremented. For example, if a counter in phase 1 is incremented, it switches to phase 2 and the segment counter is incremented; or, if a counter in phase 3 is tested for zero, it switches to phase 4 and the segment counter is incremented.

With these modifications, we can zero-test by checking variables corresponding to the current segment: A zero-test can only succeed in phase 1 and 4. Similarly, for a non-zero-test, we can check if the counter is in phase 2 or 3.

Turning a $\mathbb{Z}$-VASSG into a $\mathbb{Z}$-VASS. To handle the general case mentioned above, we need to show how to convert $\mathbb{Z}$-VASSG into ordinary $\mathbb{Z}$-VASS. In a preparatory step, we ensure that each formula is a literal. A transition labeled by a formula φ is replaced by a series-parallel graph: After bringing φ in negation normal form by pushing negations inwards, we can replace conjunctions by a series composition and disjunctions by a parallel composition (non-determinism).

The $\mathbb{Z}$-VASS works as follows. In addition to the original counters of the $\mathbb{Z}$-VASSG, it has for each variable $\mathtt{x} \in X$ two additional counters: $\mathtt{x}^+$ and $\mathtt{x}^-$. Here, $\mathtt{x}^+$ ($\mathtt{x}^-$) counts how many times $\mathtt{x}$ is read with a positive (negative) assignment. By making sure that either $\mathtt{x}^+ = 0$ or $\mathtt{x}^- = 0$ in the end, we guarantee that we always read the same value of $\mathtt{x}$.

Thus, in order to check a literal, our $\mathbb{Z}$-VASS increments the corresponding counter. In the end, before reaching a final state, it goes through each variable $\mathtt{x} \in X$ and either enters a loop decrementing $\mathtt{x}^+$ or a loop decrementing $\mathtt{x}^-$. Then, it can reach the zero vector only if all variable checks had been consistent.

From PRBCA to $\mathbb{Z}$-grammars. It remains to convert in logspace an (r, k)-PRBCA into an equivalent $\mathbb{Z}$-grammar. Just as for converting an RBCA into a $\mathbb{Z}$-VASS, one can convert a PRBCA into an equivalent $\mathbb{Z}$-*PVASS* (pushdown vector addition system with $\mathbb{Z}$-counters). Afterwards, one applies the classical transformation from pushdown automata to context-free grammars (a.k.a. triple construction), cf. [1, Lemma 2.26]: We introduce for every state pair (p, q) a nonterminal $X_{p,q}$, deriving all words which are read between p to q (starting and ending with empty stacks). For example, we introduce productions $X_{p,q} \to a X_{p',q'} b$ for all push transitions (p, a, γ, p') and pop transitions $(q', b, \bar{\gamma}, q)$. The counter effects of transitions in the $\mathbb{Z}$-PVASS (vectors in $\mathbb{Z}^k$) are translated into effects of the productions, e.g. the effect of the production $X_{p,q} \to a X_{p',q'} b$ above is the sum of the effects of the corresponding push- and pop-transition.

4   Deciding unboundedness predicates

Proof overview. In this section, we prove Theorem 2. Let us begin with a sketch. Our task is to take a PRBCA $\mathcal{A}$ and non-deterministically compute a

PDA $\mathcal{A}'$ so that $L(\mathcal{A})$ satisfies $\mathfrak{p}$ if and only if some of the outcomes for $\mathcal{A}'$ satisfy $\mathfrak{p}$. It will be clear from the construction that if the input was an RBCA, then the resulting PDA will be an NFA. Using Theorem 1 we will phrase the main part of the reduction in terms of $\mathbb{Z}$-grammars, meaning we take a $\mathbb{Z}$-grammar G as input and non-deterministically compute context-free grammars G'.

The idea of the reduction is to identify a set of productions in G that, in some appropriate sense, can be canceled (regarding the integer counter values) by a collection of other productions. Then, G' is obtained by only using a set of productions that can be canceled. Moreover, these productions are used regardless of what counter updates they perform. Then, to show the correctness, we argue in two directions: First, we show that any word derivable by G' occurs as a factor of $L(G)$. Essentially, this is because each production used in G' can be canceled by adding more productions in G, thus yielding a complete derivation of G. Thus, we have that $L(G') \subseteq F(L(G))$, which by the axioms of unboundedness predicates means that $\mathfrak{p}(L(G'))$ implies $\mathfrak{p}(L(G))$. Second, we show that $L(G)$ is a finite union of products (i.e. concatenations) $P_i = L_1 \cdot L_2 \cdots L_k$ such that each L_i is either finite or included in $L(G')$ for some G' among all non-deterministic outcomes. Again, by the axioms of unboundedness predicates, this means that if $\mathfrak{p}(L(G))$, then $\mathfrak{p}(L(G'))$ must hold for some G'.

Unboundedness predicates and finite languages. Before we start with the proof, let us observe that we may assume that our unboundedness predicate is only satisfied for infinite sets. First, suppose $\mathfrak{p}$ is satisfied for $\{\varepsilon\}$. This implies that $\mathfrak{p} = \mathfrak{p}_{\neq\emptyset}$ and hence we can just decide whether $\mathfrak{p}(L)$ by deciding whether $L \neq \emptyset$, which can be done in NP [33]. From now on, suppose that $\mathfrak{p}$ is not satisfied for $\{\varepsilon\}$. Consider the alphabet $\Sigma_1 := \{a \in \Sigma \mid \mathfrak{p}(\{a\})\}$. Now observe that if $K \subseteq \Sigma^*$ is finite, then by the axioms of unboundedness predicates, we have $\mathfrak{p}(K)$ if and only if some letter from Σ_1 appears in K. Thus, if $L \subseteq (\Sigma \backslash \Sigma_1)^*$, then $\mathfrak{p}(L)$ can only hold if L is infinite. This motivates the following definition. Given a language $L \subseteq \Sigma^*$, we define

$$L_0 = L \cap (\Sigma \setminus \Sigma_1)^*, \quad L_1 = L \cap \Sigma^* \Sigma_1 \Sigma^*.$$

Then, $\mathfrak{p}(L)$ if and only if $\mathfrak{p}(L_0)$ or $\mathfrak{p}(L_1)$. Moreover, $\mathfrak{p}(L_1)$ is equivalent to $L_1 \neq \emptyset$.

Therefore, our reduction proceeds as follows. We construct (P)RBCA for L_0 and for L_1. This can be done in logspace, because intersections with regular languages can be done with a simple product construction. Then, we check in NP whether $L_1 \neq \emptyset$. If yes, then we return "unbounded". If no, we regard $\mathfrak{p}$ as an unboundedness predicate on languages over $\Sigma \setminus \Sigma_1$ with the additional property that $\mathfrak{p}$ is only satisfied for infinite languages. Thus, it suffices to prove Theorem 2 in the case that $\mathfrak{p}$ is only satisfied for infinite sets.

Pumps and cancelation. In order to define our notion of cancelable productions, we need some terminology. We will need to argue about derivation trees for $\mathbb{Z}$-grammars. For any alphabet Γ and $d \in \mathbb{N}$, let $\mathcal{T}_{\Gamma,d}$ be the set of all finite trees where every node is labeled by both (i) a letter from Γ and (ii) a

vector from $\mathbb{Z}^d$. Suppose $G = (N, \Sigma, P, S)$ is a d-dimensional $\mathbb{Z}$-grammar. For a production $p = (A \rightarrow u, \boldsymbol{v})$, we write $\varphi(p) := \boldsymbol{v}$ for its associated counter effect. To each derivation in G, we associate a derivation tree from $\mathcal{T}_{N \cup \Sigma, d}$ as for context-free grammars. The only difference is that whenever we apply a production $(A \rightarrow u, \boldsymbol{v})$, then the node corresponding to the rewritten A is also labeled with $\boldsymbol{v}$. As in context-free grammars, the leaf nodes carry terminal letters; their vector label is just $\boldsymbol{0} \in \mathbb{Z}^d$.

We extend the map φ to both vectors in $\mathbb{N}^P$ and to derivation trees. If $\boldsymbol{u} \in \mathbb{N}^P$, then $\varphi(\boldsymbol{u}) = \sum_{p \in P} \varphi(p) \cdot \boldsymbol{u}[p]$. Similarly, if τ is a derivation tree, then $\varphi(\tau) \in \mathbb{Z}^d$ is the sum of all labels from $\mathbb{Z}^d$. A derivation tree τ for a derivation $A \overset{*}{\Rightarrow} u$ is called *complete* if $A = S$, $u \in \Sigma^*$ and $\varphi(\tau) = \boldsymbol{0}$. In other words, τ derives a terminal word and the total counter effect of the derivation is zero. For such a complete derivation, we also write $\mathsf{yield}(\tau)$ for the word u. A derivation tree τ is called a *pump* if it is the derivation tree of a derivation of the form $A \overset{*}{\Rightarrow} uAv$ for some $u, v \in \Sigma^*$ and $A \in N$. A subset $M \subseteq N$ of the non-terminals is called *realizable* if there exists a complete derivation of G that contains all non-terminals in M and no non-terminals outside of M.

A production p in P is called M-*cancelable* if there exist pumps $\tau_1, \ldots, \tau_k$ (for some $k \in \mathbb{N}$) such that (i) p occurs in some τ_i and (ii) $\varphi(\tau_1) + \cdots + \varphi(\tau_k) = \boldsymbol{0}$, i.e. the total counter effect of $\tau_1, \ldots, \tau_k$ is zero and (iii) all productions in $\tau_1, \ldots, \tau_k$ only use non-terminals from M. We say that a subset $Q \subseteq P$ is M-*cancelable* if all productions in Q are M-cancelable.

The reduction. Using the notions of M-cancelable productions, we are ready to describe how the context-free grammars are constructed. Suppose that M is realizable, that $Q \subseteq P$ is M-cancelable, and that $A \in M$. Consider the language

$$L_{A,Q} = \{u, v \in \Sigma^* \mid \exists \text{ derivation } A \overset{*}{\Rightarrow}_Q uAv\}.$$

Thus $L_{A,Q}$ consists of all words u and v appearing in derivations (whose counter values are not necessarily zero) of the form $A \overset{*}{\Rightarrow} uAv$, if we only use M-cancelable productions. The $L_{A,Q}$ will be the languages $L(G')$ mentioned above.

It is an easy observation that we can, given G and a subset $Q \subseteq P$, construct a context-free grammar for $L_{A,Q}$:

Lemma 1. *Given a $\mathbb{Z}$-grammar G, a non-terminal A, and a subset $Q \subseteq P$, we can construct in logspace a context-free grammar for $L_{A,Q}$. Moreover, if G is left-linear, then the construction yields an NFA for $L_{A,Q}$.*

We provide details in Appendix C. Now, our reduction works as follows:

1. Guess a subset $M \subseteq N$ and an $A \in M$; verify that M is realizable.
2. Guess a subset $Q \subseteq P$; verify that Q is M-cancelable.
3. Compute a context-free grammar for $L_{A,Q}$.

Here, we need to show that steps 1 and 2 can be done in NP:

Lemma 2. *Given a subset $M \subseteq N$, we can check in NP whether M is realizable. Moreover, given $M \subseteq N$ and $p \in P$, we can check in NP if p is M-cancelable.*

Both can be done using the fact that for a given context-free grammar, one can construct a Parikh-equivalent existential Presburger formula [56] and the fact that satisfiability of existential Presburger formulas is in NP. See Appendix C for details. This completes the description of our reduction. Therefore, it remains to show correctness of the reduction. In other words, to prove:

Proposition 1. *We have $\mathfrak{p}(L(G))$ if and only if $\mathfrak{p}(L_{A,Q})$ for some subset $Q \subseteq P$ such that there is a realizable $M \subseteq N$ with $A \in M$ and Q being M-cancelable.*

Proposition 1 will be shown in two lemmas:

Lemma 3. *If M is realizable and Q is M-cancelable, then $L_{A,Q} \subseteq F(L(G))$ for every $A \in M$.*

Lemma 4. *$L(G)$ is included in a finite union of sets of the form $K_1 \cdot K_2 \cdots K_m$, where each K_i is either finite or a set $L_{A,Q}$, where Q is M-cancelable for some realizable $M \subseteq N$, and $A \in M$.*

Let us see why Proposition 1 follows from Lemmas 3 and 4.

Proof (Proposition 1). We begin with the "if" direction. Thus, suppose $\mathfrak{p}(L_{A,Q})$ for A and Q as described. Then by Lemma 3 and the first and fourth axioms of unboundedness predicates, this implies $\mathfrak{p}(L(G))$.

For the "only if" direction, suppose $\mathfrak{p}(L(G))$. By the first axiom of unboundedness predicates, $\mathfrak{p}$ must hold for the finite union provided by Lemma 4. By the second axiom, this implies that $\mathfrak{p}(K_1 \cdots K_m)$ for a finite product $K_1 \cdots K_m$ as in Lemma 4. Moreover, by the third axiom, this implies that $\mathfrak{p}(K_i)$ for some $i \in \{1, \ldots, m\}$. If K_i is finite, then by assumption, $\mathfrak{p}(K_i)$ does not hold. Therefore, we must have $\mathfrak{p}(K_i)$ for some $K_i = L_{A,Q}$, as required. $\qquad\square$

Flows. It remains to prove Lemmas 3 and 4. We begin with Lemma 3 and for this we need some more terminology. Let Σ be an alphabet. By $\Psi \colon \Sigma^* \to \mathbb{N}^\Sigma$, we denote the *Parikh map*, which is defined as $\Psi(w)(a) = |w|_a$ for $w \in \Sigma^*$ and $a \in \Sigma$. In other words, $\Psi(w)(a)$ is the number of occurrences of a in $w \in \Sigma^*$. If $\Gamma \subseteq \Sigma$ is a subset, then $\pi_\Gamma \colon \Sigma^* \to \Gamma^*$ is the homomorphism with $\pi_\Gamma(a) = \varepsilon$ for $a \in \Sigma \setminus \Gamma$ and $\pi_\Gamma(a) = a$ for $a \in \Gamma$. We also call π_Γ the *projection to Γ*.

Suppose we have a $\mathbb{Z}$-grammar $G = (N, \Sigma, P, S)$ with non-terminals N and productions P. For a derivation tree τ, we write $\Psi(\tau)$ for the vector in $\mathbb{N}^P$ that counts how many times each production appears in τ. We introduce a map ∂, which counts how many non-terminals each production consumes and produces. Formally, $\partial \colon \mathbb{N}^P \to \mathbb{Z}^N$ is the monoid homomorphism that sends the production $p = A \to w$ to the vector $\partial(p) = -A + \Psi(\pi_N(w))$. Here, $-A \in \mathbb{Z}^N$ denotes the vector with -1 at the position of A and 0 everywhere else. A vector $\boldsymbol{u} \in \mathbb{N}^P$ is a *flow* if $\partial(\boldsymbol{u}) = \boldsymbol{0}$. Observe that a derivation tree τ is a pump if and only if $\Psi(\tau)$ is a flow. In this case, we also call the vector $\boldsymbol{u} \in \mathbb{N}^P$ with $\boldsymbol{u} = \Psi(\tau)$ a *pump*.

The following lemma will provide an easy way to construct derivations. It is a well-known result by Esparza [19, Theorem 3.1], and has since been exploited in several results on context-free grammars. Our formulation is slightly weaker than Esparza's. However, it is enough for our purposes and admits a simple proof, which is inspired by a proof of Kufleitner [45].

Lemma 5. *Let $\boldsymbol{f} \in \mathbb{N}^P$. Then $\boldsymbol{f}$ is a flow if and only if it is a sum of pumps.*

Proof. The "if" direction is trivial, because every pump is clearly a flow. Conversely, suppose $\boldsymbol{f} \in \mathbb{N}^P$ is a flow. We can clearly write $\boldsymbol{f} = \Psi(\tau_1) + \cdots + \Psi(\tau_n)$, where $\tau_1, \ldots, \tau_n$ are derivation trees: We can just view each production in $\boldsymbol{f}$ as its own derivation tree. Now suppose that we have $\boldsymbol{f} = \Psi(\tau_1) + \cdots + \Psi(\tau_n)$ so that n is minimal. We claim that then, each τ_i is a pump, proving the lemma.

Suppose not, then without loss of generality, τ_1 is not a pump. Since τ_1 is a derivation, this means $\Psi(\tau_1)$ cannot be a flow and thus there must be a non-terminal A with $\partial(\tau_1)(A) \neq 0$.

Let us first assume that $\partial(\tau_1)(A) > 0$. This means there is a non-terminal A occurring at a leaf of τ_1 such that A is not the start symbol of τ_1. Since $\boldsymbol{f} = \Psi(\tau_1) + \cdots + \Psi(\tau_n)$ is a flow, we must have $\partial(\Psi(\tau_2) + \cdots + \Psi(\tau_n))(A) < 0$. This, in turn, is only possible if some τ_j has A as its start symbol. We can therefore merge τ_1 and τ_j by replacing τ_1's A-labelled leaf by the new subtree τ_j. We obtain a new collection of $n - 1$ trees whose Parikh image is $\boldsymbol{f}$, in contradiction to the choice of n. If $\partial(\tau_1)(A) < 0$, then there must be a τ_j with $\partial(\tau_j)(A) > 0$ and thus we can insert τ_1 below τ_j, reaching a similar contradiction.   □

Constructing derivations. Using flows, we can now prove Lemma 3.

Proof. Suppose there is a derivation $\tau \colon A \stackrel{*}{\Rightarrow}_Q uAv$ with $A \in M$ and $u, v \in \Sigma^*$. We have to show that both u and v occur in some word $w \in L(G)$. Furthermore, if G is in Chomsky normal form, we can choose w such that $|w|$ is linear in $|u|$ and $|v|$. Our goal is to construct a derivation of G in which we find u and v as factors. We could obtain a derivation tree by inserting τ into some derivation tree for G (at some occurrence of A), but this might yield non-zero counter values. Therefore, we will use the fact that Q is M-cancelable to find other pumps that can be inserted as well in order to bring the counter back to zero.

Since $M \subseteq N$ is realizable, there exists a complete derivation τ_0 that derives some word $w_0 \in L(G)$ and uses precisely the non-terminals in M. Since $Q \subseteq P$ is M-cancelable, we know that for each production $p \in Q$, there exist pumps $\tau_1, \ldots, \tau_k$ such that (i) p occurs in some τ_i, (ii) $\varphi(\tau_1) + \cdots + \varphi(\tau_k) = \boldsymbol{0}$ and (iii) all productions in $\tau_1, \ldots, \tau_k$ only use non-terminals in M. This allows us to define $\boldsymbol{f}_p := \Psi(\tau_1) + \cdots + \Psi(\tau_k)$. Observe that $\boldsymbol{f}_p$ contains only productions with non-terminals from M, we have $\boldsymbol{f}_p[p] > 0$, and $\varphi(\boldsymbol{f}_p) = 0$. We can use the flows $\boldsymbol{f}_p$ to find the desired canceling pumps. Since by Lemma 5, every flow can be decomposed into a sum of pumps, it suffices to construct a particular flow. Specifically, we look for a flow $\boldsymbol{f}_\tau \in \mathbb{N}^P$ such that:

1. any production p with $\boldsymbol{f}_\tau[p] > 0$ uses only non-terminals from M, and

2. $\varphi(\boldsymbol{f}_\tau + \Psi(\tau)) = 0$.

The first condition ensures that all the resulting pumps can be inserted into τ_0. The second condition ensures that the resulting total counter values will be zero. We claim that with

$$\boldsymbol{f}_\tau = \left(\sum_{p \in Q} \Psi(\tau)[p] \cdot \boldsymbol{f}_p\right) - \Psi(\tau), \tag{1}$$

we achieve these conditions. First, observe that $\boldsymbol{f}_\tau \in \mathbb{N}^P$: We have

$$\boldsymbol{f}_\tau[q] \quad \geq \quad \Psi(\tau)[q] \cdot \boldsymbol{f}_q[q] - \Psi(\tau)[q] \quad = \quad \Psi(\tau)[q] \cdot (\boldsymbol{f}_q[q] - 1)$$

which is at least zero as $\boldsymbol{f}_q[q]$ must be non-zero by definition. Second, note that $\boldsymbol{f}_\tau$ is indeed a flow, because it is a $\mathbb{Z}$-linear combination of flows. Moreover, all productions appearing in $\boldsymbol{f}_\tau$ also appear in $\boldsymbol{f}_p$ for some $p \in Q$ or in τ, meaning that all non-terminals must belong to M. Finally, the total counter effect of $\boldsymbol{f}_\tau + \Psi(\tau)$ is zero as $\boldsymbol{f}_\tau + \Psi(\tau) = \sum_{p \in Q} \Psi(\tau)[p] \cdot \boldsymbol{f}_p$ is a sum of flows each with total counter effect zero.

Now, since $\boldsymbol{f}_\tau$ is a flow, Lemma 5 tells us that there are pumps $\tau'_1, \ldots, \tau'_m$ such that $\boldsymbol{f}_\tau = \Psi(\tau'_1) + \cdots + \Psi(\tau'_m)$. Therefore, inserting τ and $\tau'_1, \ldots, \tau'_m$ into τ_0 must yield a derivation of a word that has both u and v as factors and also has counter value

$$\underbrace{\varphi(\tau_0)}_{=\mathbf{0}} + \underbrace{\varphi(\tau) + \varphi(\tau'_1) + \cdots \varphi(\tau'_m)}_{=\varphi(\tau)+\varphi(\boldsymbol{f}_\tau)=\mathbf{0}} = \mathbf{0}.$$

Thus, we have a complete derivation of G. Hence $L_{A,Q} \subseteq F(L(G))$.   □

Decomposition into finite union. It remains to prove Lemma 4. For the decomposition, we show that there exists a finite set D_0 of complete derivations such that all complete derivations of G can be obtained from some derivation in D_0 and then inserting pumps that produce words in $L_{A,Q}$, for some appropriate A and Q. Here, it is key that the set D_0 of "base derivations" is finite. Showing this for context-free grammars would just require a simple "unpumping" argument based on the pigeonhole principle as in Parikh's theorem [52]. However, in the case of $\mathbb{Z}$-grammars, where D_0 should only contain derivations that have counter value zero, this is not obvious. To achieve this, we employ a well-quasi ordering on (labeled) trees. Recall that a *quasi ordering* is a reflexive and transitive ordering. For a quasi ordering $(X, \leq)$ and a subset $Y \subseteq X$, we write $Y \uparrow$ for the set $\{x \in X \mid \exists y \in Y : y \leq x\}$. We say that $(X, \leq)$ is a *well-quasi ordering* (WQO) if every non-empty subset $Y \subseteq X$ has a finite subset $Y_0 \subseteq Y$ such that $Y \subseteq Y_0 \uparrow$.

We define an ordering on all trees in $\mathcal{T}_{N \cup \Sigma, d}$. A tree s is a *subtree* of t if there exists a node x in t such that s consists of all nodes of t that are descendants of x. If $\tau_1, \ldots, \tau_n$ are trees, then we denote by $r[\tau_1, \ldots, \tau_n]$ the tree with a root node r and the subtrees $\tau_1, \ldots, \tau_n$ directly under the root. Now let $\tau = (A, \boldsymbol{u})[\tau_1, \ldots, \tau_n]$

and $\tau' = (B, \boldsymbol{v})[\sigma_1, \ldots, \sigma_m]$ be trees in $\mathcal{T}_{N \cup \Sigma, d}$. We define the ordering $\preceq$ as follows. If $n = 0$ (i.e. τ consists of only one node), then we have $\tau \preceq \tau'$ if and only if $A = B$ and $m = 0$. If $n \geq 1$, then we define inductively:

$$\tau \preceq \tau' \quad \Longleftrightarrow \quad A = B \text{ and } \exists \text{ subtree } \tau'' = (A, \boldsymbol{u}')[\tau'_1, \ldots, \tau'_n] \text{ of } \tau'$$
$$\text{with } \tau_i \preceq \tau'_i \text{ for } i = 1, \ldots, n$$

Based on $\preceq$, we define as slight refinement: We write $\tau \sqsubseteq \tau'$ if and only if $\tau \preceq \tau'$ and the set of non-terminals appearing in τ is the same as in τ'.

Lemma 6. $(\mathcal{T}_{N \cup \Sigma, d}, \sqsubseteq)$ *is a WQO.*

Proof. In [48, Lemma 3.3], it was shown that $\preceq$ is a WQO. Then $\sqsubseteq$ is the product of equality on a finite set, which is a WQO, and the WQO $\preceq$.   □

Lemma 6 allows us to decompose $L(G)$ into a finite union: For each complete derivation τ of G, we define

$$L_\tau(G) = \{w \in \Sigma^* \mid \exists \text{ complete derivation } \tau' \text{ with } \tau \sqsubseteq \tau' \text{ and } \mathsf{yield}(\tau') = w\}.$$

Lemma 7. *There exists a finite set $D_0 \subseteq \mathcal{T}_{N \cup \Sigma, d}$ of complete derivations of G such that $L(G) = \bigcup_{\tau \in D_0} L_\tau(G)$.*

Proof. Since $(\mathcal{T}_{N \cup \Sigma, d}, \sqsubseteq)$ is a WQO, the set $D \subseteq \mathcal{T}_{N \cup T, d}$ of all complete derivations of G has a finite subset D_0 with $D \subseteq D_0 \uparrow$. This implies the lemma.   □

Decomposition into finite product. In light of Lemma 7, it remains to be shown that for each tree τ, we can find a product $K_1 \cdot K_2 \cdots K_m$ of languages such that $L_\tau(G) \subseteq K_1 \cdot K_2 \cdots K_m$ and each K_i is either finite or is of the form $L_{A,Q}$. We construct the overapproximation of $L_\tau(G)$ inductively as follows. Let $M \subseteq N$ and $Q \subseteq P$ be subsets of the non-terminals and the productions, respectively. If τ has one node, labeled by $a \in \Sigma$, then we set $\mathrm{App}_Q(\tau) := \{a\}$. Moreover, if $\tau = (A, \boldsymbol{u})[\tau_1, \ldots, \tau_n]$ for $A \in N$ and trees $\tau_1, \ldots, \tau_n$, then we set

$$\mathrm{App}_Q(\tau) := L_{A,Q} \cdot \mathrm{App}_Q(\tau_1) \cdot \mathrm{App}_Q(\tau_2) \cdots \mathrm{App}_Q(\tau_n) \cdot L_{A,Q}.$$

Finally, we set $\mathrm{App}(\tau) := \mathrm{App}_Q(\tau)$, where $Q \subseteq P$ is the set of all M-cancelable productions, where M is the set of all non-terminals appearing in τ. Now clearly, each $\mathrm{App}(\tau)$ is a finite product $K_1 \cdot K_2 \cdots K_m$ as desired: This follows by induction on the size of τ. Thus, to prove Lemma 4, the following suffices:

Lemma 8. *For every complete derivation tree τ of G, we have $L_\tau(G) \subseteq \mathrm{App}(\tau)$.*

Proof. Suppose $w \in L_\tau(G)$ is derived using a complete derivation tree τ' with $\tau \sqsubseteq \tau'$. Then, the set of non-terminals appearing in τ must be the same as in τ'; we denote it by M. Let $Q \subseteq P$ be the set of all M-cancelable productions. Moreover, since $\tau \preceq \tau'$, we can observe that there exist pumps $\tau_1, \ldots, \tau_n$ with root non-terminals $A_1, \ldots, A_n$ and nodes $x_1, \ldots, x_n$ in τ such that τ' can be obtained from τ by replacing each node x_i by the pump τ_i.

Since both τ and τ' are complete derivations of G, each must have counter effect $\mathbf{0}$. Thus, $\varphi(\tau_1) + \cdots + \varphi(\tau_n) = \varphi(\tau') - \varphi(\tau) = \mathbf{0}$. Hence, the pumps $\tau_1, \ldots, \tau_n$ witness that the productions appearing in $\tau_1, \ldots, \tau_n$ are M-cancelable. Thus, the derivation corresponding to τ_i uses only productions in Q and thus τ_i corresponds to $A_i \overset{*}{\Rightarrow}_Q u_i A v_i$ for some u_i, v_i and we have $u_i, v_i \in L_{A,Q}$. □

5   Growth

In this section, we prove Theorem 3. Since clearly, a bounded language has polynomial growth, it remains to be shown that if L is accepted by a PRBCA and L is not bounded, then it has exponential growth. For two languages $L_1, L_2 \subseteq \Sigma^*$, we write $L_1 \hookrightarrow_{\text{lin}} L_2$ if there exists a constant $c \in \mathbb{N}$ such that for every word $w_1 \in L_1$, there exists $w_2 \in L_2$ with $|w_2| \leq c \cdot |w_1|$ and w_1 is a factor of w_2. It is not difficult to observe that for two languages $L_1, L_2 \subseteq \Sigma^*$, if $L_1 \hookrightarrow_{\text{lin}} L_2$ and L_1 has exponential growth, then so does L_2.

In order to show Theorem 3, we need an adapted version of Lemma 3. A $\mathbb{Z}$-grammar is in *Chomsky normal form* if all productions are of the form $(A \to BC, \boldsymbol{v})$ or $(A \to a, \boldsymbol{v})$ with $A, B, C \in N$, $a \in \Sigma$, and $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}^k$. In other words, the context-free grammar obtained by forgetting all counter vectors is in Chomsky normal form. Fernau and Stiebe [21, Proposition 5.12] have shown that every $\mathbb{Z}$-grammar has an equivalent $\mathbb{Z}$-grammar in Chomsky normal form.

Lemma 9. *If $G = (N, \Sigma, P, S)$ is a $\mathbb{Z}$-grammar in Chomsky normal form, $M \subseteq N$ is realizable, $Q \subseteq P$ is M-cancelable, and $A \in M$, then $L_{A,Q} \hookrightarrow_{\text{lin}} L(G)$.*

This is shown essentially the same way as Lemma 3. Let us now show that if a language L accepted by a PRBCA is not bounded, then it must have exponential growth. We have seen above that as a PRBCA language, L is generated by some $\mathbb{Z}$-grammar. As shown by Fernau and Stiebe [21, Proposition 5.12], this implies that $L = L(G)$ for some $\mathbb{Z}$-grammar G in Chomsky normal form. Since L is not bounded, Lemma 4 yields A and Q such that $L_{A,Q}$ is not a bounded language. It is well-known that any context-free languages that is not bounded has exponential growth (this fact has apparently been independently discovered at least six times, see [25] for references). Thus, $L_{A,Q}$ has exponential growth. By Lemma 9, we have $L_{A,Q} \hookrightarrow_{\text{lin}} L$ and thus L has exponential growth.

References

[1] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling. 1: Parsing.* Prentice-Hall, 1972. ISBN: 0139145567. URL: https://www.worldcat.org/oclc/310805937.

[2] Brenda S Baker and Ronald V Book. "Reversal-bounded multipushdown machines". In: *Journal of Computer and System Sciences* 8.3 (1974), pp. 315–332. DOI: 10.1016/S0022-0000(74)80027-9.

[3] David Barozzini, Lorenzo Clemente, Thomas Colcombet, and Pawel Parys. "Cost Automata, Safe Schemes, and Downward Closures". In: *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*. Ed. by Artur Czumaj, Anuj Dawar, and Emanuela Merelli. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 109:1–109:18. DOI: 10.4230/LIPIcs.ICALP.2020.109.

[4] David Barozzini, Pawel Parys, and Jan Wroblewski. "Unboundedness for Recursion Schemes: A Simpler Type System". In: *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*. Ed. by Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 112:1–112:19. DOI: 10.4230/LIPIcs.ICALP.2022.112.

[5] Alin Bostan, Arnaud Carayol, Florent Koechlin, and Cyril Nicaud. "Weakly-Unambiguous Parikh Automata and Their Link to Holonomic Series". In: *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 114:1–114:16. DOI: 10.4230/LIPIcs.ICALP.2020.114.

[6] Toby Cathcart Burn, Luke Ong, Steven J. Ramsay, and Dominik Wagner. "Initial Limit Datalog: a New Extensible Class of Decidable Constrained Horn Clauses". In: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470527.

[7] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. "Affine Parikh automata". In: *RAIRO Theor. Informatics Appl.* 46.4 (2012), pp. 511–545. DOI: 10.1051/ita/2012013.

[8] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. "Bounded Parikh Automata". In: *Int. J. Found. Comput. Sci.* 23.8 (2012), pp. 1691–1710. DOI: 10.1142/S0129054112400709.

[9] Michaël Cadilhac, Andreas Krebs, and Pierre McKenzie. "The Algebraic Theory of Parikh Automata". In: *Theory Comput. Syst.* 62.5 (2018), pp. 1241–1268. DOI: 10.1007/s00224-017-9817-2.

[10] Arturo Carpi, Flavio D'Alessandro, Oscar H Ibarra, and Ian McQuillan. "Relationships between bounded languages, counter machines, finite-index grammars, ambiguity, and commutative regularity". In: *Theoretical Computer Science* 862 (2021), pp. 97–118. DOI: 10.1016/j.tcs.2020.10.006.

[11]   Tat-hung Chan. "Pushdown Automata with Reversal-Bounded Counters". In: *J. Comput. Syst. Sci.* 37.3 (1988), pp. 269–291. DOI: 10.1016/0022-0000(88)90008-6.

[12]   Laura Ciobanu and Susan Hermiller. "Conjugacy growth series and languages in groups". In: *Transactions of the American Mathematical Society* 366.5 (2014), pp. 2803–2825.

[13]   Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. "Regular Separability of Parikh Automata". In: *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland.* Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 117:1–117:13. DOI: 10.4230/LIPIcs.ICALP.2017.117.

[14]   Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. "The Diagonal Problem for Higher-Order Recursion Schemes is Decidable". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016.* Ed. by Martin Grohe, Eric Koskinen, and Natarajan Shankar. ACM, 2016, pp. 96–105. DOI: 10.1145/2933575.2934527.

[15]   Wojciech Czerwinski, Piotr Hofman, and Georg Zetzsche. "Unboundedness Problems for Languages of Vector Addition Systems". In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic.* Ed. by Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 119:1–119:15. DOI: 10.4230/LIPIcs.ICALP.2018.119.

[16]   Flavio D'Alessandro and Benedetto Intrigila. "On the commutative equivalence of bounded context-free and regular languages: The semi-linear case". In: *Theor. Comput. Sci.* 572 (2015), pp. 1–24. DOI: 10.1016/j.tcs.2015.01.008. URL: https://doi.org/10.1016/j.tcs.2015.01.008.

[17]   Joey Eremondi, Oscar H. Ibarra, and Ian McQuillan. "On the Density of Context-Free and Counter Languages". In: *Developments in Language Theory - 19th International Conference, DLT 2015, Liverpool, UK, July 27-30, 2015, Proceedings.* Ed. by Igor Potapov. Vol. 9168. Lecture Notes in Computer Science. Springer, 2015, pp. 228–239. DOI: 10.1007/978-3-319-21500-6_18.

[18]   Joey Eremondi, Oscar H. Ibarra, and Ian McQuillan. "On the Density of Context-Free and Counter Languages". In: *Int. J. Found. Comput. Sci.* 29.2 (2018), pp. 233–250. DOI: 10.1142/S0129054118400051.

[19]   Javier Esparza. "Petri nets, commutative context-free grammars, and basic parallel processes". In: *Fundamenta Informaticae* 31.1 (1997), pp. 13–25.

[20]   Javier Esparza, Pierre Ganty, and Rupak Majumdar. "A Perfect Model for Bounded Verification". In: *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012.* IEEE Computer Society, 2012, pp. 285–294. DOI: 10.1109/LICS.2012.39.

[21]   Henning Fernau and Ralf Stiebe. "Sequential grammars and automata with valences". In: *Theor. Comput. Sci.* 276.1-2 (2002), pp. 377–405. DOI: [10. 1016/S0304-3975(01)00282-1](10.1016/S0304-3975(01)00282-1).

[22]   Emmanuel Filiot, Shibashis Guha, and Nicolas Mazzocchi. "Two-Way Parikh Automata". In: *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India.* Vol. 150. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 40:1–40:14. DOI: [10.4230/LIPIcs. FSTTCS.2019.40](10.4230/LIPIcs.FSTTCS.2019.40).

[23]   Alain Finkel and Arnaud Sangnier. "Reversal-Bounded Counter Machines Revisited". In: *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings.* Ed. by Edward Ochmanski and Jerzy Tyszkiewicz. Vol. 5162. Lecture Notes in Computer Science. Springer, 2008, pp. 323–334. DOI: [10.1007/978-3-540-85238-4_26](10.1007/978-3-540-85238-4_26).

[24]   Moses Ganardi. "Language recognition in the sliding window model". PhD thesis. University of Siegen, Germany, 2019. URL: https://dspace.ub.uni-siegen.de/handle/ubsi/1523.

[25]   Pawel Gawrychowski, Dalia Krieger, Narad Rampersad, and Jeffrey O. Shallit. "Finding the Growth Rate of a Regular or Context-Free Language in Polynomial Time". In: *Int. J. Found. Comput. Sci.* 21.4 (2010), pp. 597–618. DOI: [10.1142/S0129054110007441](10.1142/S0129054110007441).

[26]   Seymour Ginsburg and Edwin H Spanier. "Bounded ALGOL-like languages". In: *Transactions of the American Mathematical Society* 113.2 (1964), pp. 333–368.

[27]   Sheila A. Greibach. "Remarks on Blind and Partially Blind One-Way Multicounter Machines". In: *Theor. Comput. Sci.* 7 (1978), pp. 311–324. DOI: [10.1016/0304-3975(78)90020-8](10.1016/0304-3975(78)90020-8).

[28]   Rostislav Grigorchuk and A. Machì. "An example of an indexed language of intermediate growth". In: *Theoretical computer science* 215.1-2 (1999), pp. 325–327.

[29]   Eitan M. Gurari and Oscar H. Ibarra. "The complexity of decision problems for finite-turn multicounter machines". In: *Journal of Computer and System Sciences* 22.2 (1981), pp. 220–229. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(81)90028-3.

[30]   Christoph Haase and Simon Halfon. "Integer Vector Addition Systems with States". In: *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings.* Ed. by Joël Ouaknine, Igor Potapov, and James Worrell. Vol. 8762. Lecture Notes in Computer Science. Springer, 2014, pp. 112–124. DOI: [10.1007/978-3-319-11439-2_9](10.1007/978-3-319-11439-2_9).

[31]   Christoph Haase and Georg Zetzsche. "Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests". In: *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019.* IEEE, 2019, pp. 1–14. DOI:

10.1109/LICS.2019.8785850. URL: https://doi.org/10.1109/LICS.2019.8785850.

[32]   Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. "Unboundedness and downward closures of higher-order pushdown automata". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. ACM, 2016, pp. 151–163. DOI: 10.1145/2837614.2837627.

[33]   Matthew Hague and Anthony Widjaja Lin. "Model Checking Recursive Programs with Numeric Data Types". In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 743–759. DOI: 10.1007/978-3-642-22110-1_60.

[34]   Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. "Decidability, complexity, and expressiveness of first-order logic over the subword ordering". In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 2017, pp. 1–12. DOI: 10.1109/LICS.2017.8005141.

[35]   Takeshi Hayashi. "On Derivation Trees of Indexed Grammars—An Extension of the uvwxy-Theorem". In: *Publications of the Research Institute for Mathematical Sciences* 9.1 (1973), pp. 61–92. DOI: 10.2977/prims/1195192738.

[36]   John E. Hopcroft. "On the equivalence and containment problems for context-free languages". In: *Mathematical systems theory* 3.2 (1969), pp. 119–124.

[37]   Oscar H. Ibarra. "Reversal-Bounded Multicounter Machines and Their Decision Problems". In: *J. ACM* 25.1 (1978), pp. 116–133. DOI: 10.1145/322047.322058.

[38]   Oscar H. Ibarra and Bala Ravikumar. "On Sparseness, Ambiguity and other Decision Problems for Acceptors and Transducers". In: *STACS 86, 3rd Annual Symposium on Theoretical Aspects of Computer Science, Orsay, France, January 16-18, 1986, Proceedings*. Ed. by Burkhard Monien and Guy Vidal-Naquet. Vol. 210. Lecture Notes in Computer Science. Springer, 1986, pp. 171–179. DOI: 10.1007/3-540-16078-7_74.

[39]   Oscar H. Ibarra and Shinnosuke Seki. "Characterizations of Bounded semilinear Languages by One-Way and Two-Way Deterministic Machines". In: *Int. J. Found. Comput. Sci.* 23.6 (2012), pp. 1291–1306. DOI: 10.1142/S0129054112400539.

[40]   Oscar H. Ibarra, Jianwen Su, Zhe Dang, Tevfik Bultan, and Richard A. Kemmerer. "Counter Machines and Verification Problems". In: *Theor. Comput. Sci.* 289.1 (2002), pp. 165–189. DOI: 10.1016/S0304-3975(01)00268-7.

[41]   Matthias Jantzen and Alexy Kurganskyy. "Refining the hierarchy of blind multicounter languages and twist-closed trios". In: *Inf. Comput.* 185.2 (2003), pp. 159–181. DOI: 10.1016/S0890-5401(03)00087-7.

[42] Felix Klaedtke and Harald Rueß. "Monadic Second-Order Logics with Cardinalities". In: *Proceedings of ICALP 2003*. Ed. by Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger. Berlin, Heidelberg: Springer, 2003, pp. 681–696.

[43] Naoki Kobayashi. "Inclusion between the frontier language of a non-deterministic recursive program scheme and the Dyck language is undecidable". In: *Theor. Comput. Sci.* 777 (2019), pp. 409–416. DOI: 10.1016/j.tcs.2018.09.035.

[44] S. Rao Kosaraju. "Decidability of Reachability in Vector Addition Systems (Preliminary Version)". In: *STOC 1982, May 5-7, 1982, San Francisco, California, USA*. 1982, pp. 267–281.

[45] Manfred Kufleitner. *Yet another proof of Parikh's Theorem*. Oct. 6, 2022. arXiv: 2210.02925.

[46] Dietrich Kuske and Georg Zetzsche. "Languages Ordered by the Subword Order". In: *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*. Ed. by Mikolaj Bojanczyk and Alex Simpson. Vol. 11425. Lecture Notes in Computer Science. Springer, 2019, pp. 348–364. DOI: 10.1007/978-3-030-17127-8_20.

[47] Jean-Luc Lambert. "A Structure to Decide Reachability in Petri Nets". In: *Theor. Comput. Sci.* 99.1 (1992), pp. 79–104.

[48] Jérôme Leroux, M. Praveen, Philippe Schnoebelen, and Grégoire Sutre. "On Functions Weakly Computable by Pushdown Petri Nets and Related Systems". In: *CoRR* abs/1904.04090 (2019). arXiv: 1904.04090.

[49] Jérôme Leroux and Sylvain Schmitz. "Demystifying Reachability in Vector Addition Systems". In: *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. IEEE Computer Society, 2015, pp. 56–67. DOI: 10.1109/LICS.2015.16.

[50] Jérôme Leroux and Sylvain Schmitz. "Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension". In: *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, 2019, pp. 1–13. DOI: 10.1109/LICS.2019.8785796.

[51] Ernst W. Mayr. "An Algorithm for the General Petri Net Reachability Problem". In: *STOC 1981, May 11-13, 1981, Milwaukee, Wisconsin, USA*. 1981, pp. 238–246.

[52] Rohit J Parikh. "On context-free languages". In: *Journal of the ACM (JACM)* 13.4 (1966), pp. 570–581.

[53] Pawel Parys. "The Complexity of the Diagonal Problem for Recursion Schemes". In: *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*. Ed. by Satya V. Lokam and R. Ramanujam. Vol. 93. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 45:1–45:14. DOI: 10.4230/LIPIcs.FSTTCS.2017.45.

[54]   Loic Pottier. "Minimal Solutions of Linear Diophantine Systems: Bounds and Algorithms". In: *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings.* Ed. by Ronald V. Book. Vol. 488. Lecture Notes in Computer Science. Springer, 1991, pp. 162–173. DOI: 10.1007/3-540-53904-2_94.

[55]   George S. Sacerdote and Richard L. Tenney. "The decidability of the reachability problem for vector addition systems (preliminary version)". In: *STOC 1977.* ACM. 1977, pp. 61–76.

[56]   Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. "On the Complexity of Equational Horn Clauses". In: *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings.* Ed. by Robert Nieuwenhuis. Vol. 3632. Lecture Notes in Computer Science. Springer, 2005, pp. 337–352. DOI: 10.1007/11532231_25.

[57]   Georg Zetzsche. "An Approach to Computing Downward Closures". In: *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II.* Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Vol. 9135. Lecture Notes in Computer Science. Springer, 2015, pp. 440–451. DOI: 10.1007/978-3-662-47666-6_35.

[58]   Georg Zetzsche. "An approach to computing downward closures". In: *CoRR* abs/1503.01068 (2015). arXiv: 1503.01068.

[59]   Georg Zetzsche. "The Complexity of Downward Closure Comparisons". In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy.* Ed. by Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi. Vol. 55. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 123:1–123:14. DOI: 10.4230/LIPIcs.ICALP.2016.123.

A   Additional material on Section 2

A.1   Counter updates

Here we argue that for d-dimensional $\mathbb{Z}$-VASS we can without loss of generality assume unary encoded counter updates. In particular, we show that counter updates comprised of n-bit binary encoded numbers can be simulated by ones comprised of unary encoded numbers. Moreover, we also mention how to extend this result to the case of $\mathbb{Z}$-grammars, which uses a very similar construction. The general idea is to replace each counter by n many counters and to replace each number in a counter update by its binary representation.

Let $\mathcal{V} = (Q, \Sigma, q_0, T, F)$ be a d-dimensional $\mathbb{Z}$-VASS with binary encoded counter updates using at most n bit per number. We construct a dn-dimensional $\mathbb{Z}$-VASS $\mathcal{V}' = (Q, \Sigma, q_0, T', F)$ with unary encoded counter updates as follows:

- For every transition $(p, w, (v_1, \ldots, v_d), p') \in T$ add the transition $(p, w, (\mathsf{bin}(v_1), \ldots, \mathsf{bin}(v_d)), p')$ to T', where $\mathsf{bin} : \mathbb{Z} \to \{-1, 0, 1\}$ maps an n-bit number to its binary representation, while keeping the sign.
- For every $i \in [1, d]$, every $j \in [1, n-1]$ and every state $p \in Q$ add the loops $(p, \varepsilon, (u_1, \ldots, u_{dn}), p)$ and $(p, \varepsilon, (v_1, \ldots, v_{dn}), p)$ to T', where $u_{(i-1)n+j} = -2$, $u_{(i-1)n+j+1} = 1$, $v_{(i-1)n+j} = 2$, $v_{(i-1)n+j+1} = -1$, and $u_k = v_k = 0$ for every $k \notin \{(i-1)n+j, (i-1)n+j+1\}$.

Here the first set of transitions applies the binary representations as counter effects, while the loops of the second set ensure the correct relationship among each set of n counters simulating a single counter of $\mathcal{V}$. In particular, the loops make sure that each such counter represents one of the n bits, with each successive bit being twice as significant as the previous one.

Let us now briefly talk about the construction for $\mathbb{Z}$-grammars. The idea is the same, but instead of replacing transitions, we replace productions. Then instead of the loops, we add for each non-terminal A productions that replace A by itself. In both cases the counter updates are exactly the same as in the $\mathbb{Z}$-VASS construction, and the proof is very similar as well.

To prove the construction correct in the $\mathbb{Z}$-VASS case we show by induction on run length that there is a run $(q, u_1, \ldots, u_n) \xrightarrow{w} (p, v_1, \ldots, v_n)$ of $\mathcal{V}$ if and only if there is a run $(q, u'_1, \ldots, u'_{dn}) \xrightarrow{w} (p, v'_1, \ldots, v'_{dn})$ of $\mathcal{V}'$ with $u_i = \sum_{j=1}^{n} 2^{j-1} u'_{(i-1)n+j}$ and $v_i = \sum_{j=1}^{n} 2^{j-1} v'_{(i-1)n+j}$. The base case is immediate from the starting configuration in both $\mathbb{Z}$-VASS.

For the inductive case let us firstly assume that there is a run $(q, u_1, \ldots, u_n)$ $\xrightarrow{w} (p, v_1, \ldots, v_n) \xrightarrow{w'} (r, x_1, \ldots, x_n)$ of $\mathcal{V}$ such that the second part is due to a single transition $(p, w', \boldsymbol{x} - \boldsymbol{v}, r) \in T$. Then by induction hypothesis there is a run $(q, u'_1, \ldots, u'_{dn}) \xrightarrow{w} (p, v'_1, \ldots, v'_{dn})$ of $\mathcal{V}'$ with $u_i = \sum_{j=1}^{n} 2^{j-1} u'_{(i-1)n+j}$ and $v_i = \sum_{j=1}^{n} 2^{j-1} v'_{(i-1)n+j}$. Furthermore we have $(p, w', (\mathsf{bin}(x_1 - v_1), \ldots, \mathsf{bin}(x_d - v_d)), r) \in T'$ by construction. Appending this transition to the run of $\mathcal{V}'$ ending in $(p, \boldsymbol{v}')$ leads to the configuration $(r, y'_1, \ldots, y'_{dn})$, where $y'_{(i-1)n+j} = v'_{(i-1)n+j} + \mathsf{bin}(x_i - v_i)_j$ for $i \in [1, d]$ and $j \in [1, n]$. Here $\mathsf{bin}(-)_j$ refers to the jth digit in

the binary representation, which corresponds to the $(j-1)$th power of 2. This gives us the following relationship between $\boldsymbol{x}$ and $\boldsymbol{y}'$, as required:

$$\sum_{j=1}^{n} 2^{j-1} y'_{(i-1)n+j} = \sum_{j=1}^{n} 2^{j-1} v'_{(i-1)n+j} + 2^{j-1}\mathsf{bin}(x_i - v_i)_j$$

$$= (x_i - v_i) + \sum_{j=1}^{n} 2^{j-1} v'_{(i-1)n+j} = (x_i - v_i) + v_i = x_i.$$

Secondly, let us now assume that there is a run $(q, u'_1, \ldots, u'_{dn}) \xrightarrow{w} (p, v'_1, \ldots, v'_{dn}) \xrightarrow{w'} (r, y'_1, \ldots, y'_{dn})$ of $\mathcal{V}'$ such that the second part is due to a single transition $t' = (p, w', \boldsymbol{y}' - \boldsymbol{v}', r) \in T'$. Then by induction hypothesis there is a run $(q, u_1, \ldots, u_n) \xrightarrow{w} (p, v_1, \ldots, v_n)$ of $\mathcal{V}$ with $u_i = \sum_{j=1}^{n} 2^{j-1} u'_{(i-1)n+j}$ and $v_i = \sum_{j=1}^{n} 2^{j-1} v'_{(i-1)n+j}$. Now we need to distinguish two cases:

- If t' was added to T' based on a transition $t \in T$, then we have $t = (p, w', (\sum_{j=1}^{n} 2^{j-1}(y'_j - v'_j), \ldots, \sum_{j=1}^{n} 2^{j-1}(y'_{(d-1)n+j} - v'_{(d-1)n+j})), r)$ by construction. Appending this transition to the run of $\mathcal{V}$ ending in $(p, \boldsymbol{v})$ leads to the configuration $(r, x_1, \ldots, x_d)$, where $x_i = v_i + \sum_{j=1}^{n} 2^{j-1}(y'_{(i-1)n+j} - v'_{(i-1)n+j})$ for $i \in [1, d]$. This gives us the following relationship between $\boldsymbol{x}$ and $\boldsymbol{y}'$, as required:

$$x_i = v_i + \sum_{j=1}^{n} 2^{j-1}(y'_{(i-1)n+j} - v'_{(i-1)n+j})$$

$$= (v_i - v_i) + \sum_{j=1}^{n} 2^{j-1} y'_{(i-1)n+j} = \sum_{j=1}^{n} 2^{j-1} y'_{(i-1)n+j}.$$

- If t' is one of the loops in T' that was no based on a transition of T, then we have $w' = \varepsilon$, $p = r$, and the counter effect $\boldsymbol{y}' - \boldsymbol{v}'$ changes the value $v'_{(i-1)n+j}$ by 2, and the value $v'_{(i-1)n+j+1}$ by 1, for some $i \in [1, d]$, $j \in [1, n-1]$. Since all other values are not changed by this effect, it only affects the sum $\sum_{j=1}^{n} 2^{j-1} v'_{(i-1)n+j}$. Since the change by 2 and the change by 1 always have opposite signs, this sum is also not affected. Therefore the run of $\mathcal{V}$ ending in $(p, \boldsymbol{v})$ is already as required.

For language equivalence between $\mathcal{V}$ and $\mathcal{V}'$, we still need to show that if $\mathcal{V}$ reaches $\boldsymbol{0}$ then $\mathcal{V}'$ can simulate this by reaching exactly $\boldsymbol{0}$ as well. Formally we show that there is a run $(q_0, \boldsymbol{0}) \xrightarrow{w} (p, \boldsymbol{0})$ of $\mathcal{V}$ if and only if there is a run $(q_0, \boldsymbol{0}) \xrightarrow{w} (p, \boldsymbol{0})$ of $\mathcal{V}'$.

The "only if"-direction follows directly from the statement we just proved. For the "if"-direction we get that there is a run $(q_0, \boldsymbol{0}) \xrightarrow{w} (p, \boldsymbol{v}')$ of $\mathcal{V}'$ with $\sum_{j=1}^{n} 2^{j-1} v'_{(i-1)n+j} = 0$ for every $i \in [1, d]$. Assume that for some $i \in [1, d]$ there is a maximal $j \in [1, n]$ such that $v'_{(i-1)n+j} \neq 0$. Then $j > 1$, because the above sum must evaluate to 0, so there must be a smaller $j' \in [1, n]$ with $v'_{(i-1)n+j'} \neq 0$

as well. Consider the loop $\ell = (p, \varepsilon, (u_1, \ldots, u_{dn}), p) \in T'$ where $u_{(i-1)n+j-1} = -2$, $u_{(i-1)n+j} = 1$ if $v_{(i-1)n+j} < 0$, or $u_{(i-1)n+j-1} = 2$, $u_{(i-1)n+j} = -1$ if $v_{(i-1)n+j} < 0$. Appending ℓ $|v_{(i-1)n+j}|$ many times to the run ending in $(p, \boldsymbol{v'})$ will change the value $v_{(i-1)n+j}$ to 0, and otherwise only changes values $v'_{(i-1)n+j''}$ with $j'' \in [1, n]$ and $j'' < j$. However, this run extension does not alter the sum $\sum_{j=1}^{n} 2^{j-1} v'_{(i-1)n+j}$. Meaning if we do this iteratively for every j and i, we eventually arrive at $\boldsymbol{0}$.

A.2   Unboundedness predicates

We comment here on the relationship between the notion of one-dimensional unboundedness predicates in [15] and the notion used here. The set of axioms in [15] is as follows, for any languages $K, L \subseteq \Sigma^*$:

(O1) If $\mathfrak{p}(K)$ and $K \subseteq L$, then $\mathfrak{p}(L)$.
(O2) If $\mathfrak{p}(K \cup L)$, then $\mathfrak{p}(K)$ or $\mathfrak{p}(L)$.
(O3) If $\mathfrak{p}(F(K \cdot L))$, then $\mathfrak{p}(F(K))$ or $\mathfrak{p}(F(L))$.

Moreover, in [15], the resulting decision problem is defined as:

Input Language $L \subseteq \Sigma^*$.
Question Does $\mathfrak{p}(F(L))$ hold?

Note that in this definition, one decides whether $\mathfrak{p}(F(L))$, in contrast to the definition in Section 2, where we decide whether $\mathfrak{p}(L)$.

1. The definition (U1)–(U4) is slightly stronger. Observe that a predicate $\mathfrak{p}$ that satisfies (U1)–(U4) also satisfies (O1)–(O3). However, there are some (pathological) examples of predicates that satisfy (O1)–(O3), but not (U1)–(U4). For example, the predicate $\mathfrak{p}_{\neg a}$ with

$$\mathfrak{p}_{\neg a}(L) \iff L \text{ is non-empty and } L \neq \{a\}$$

   satisfies (O1)–(O3), but not (U4). The latter is because $F(\{a\}) = \{\varepsilon, a\}$ satisfies $\mathfrak{p}_{\neg a}$, but $\{a\}$ itself does not.
2. Although (U1)–(U4) are slightly stronger, they yield the same set of decision problems. Indeed, suppose $\mathfrak{p}$ is a predicate that satisfies (O1)–(O3). Define the predicate $\mathfrak{p}'$ with $\mathfrak{p}'(L) \iff \mathfrak{p}(F(L))$. Then, $\mathfrak{p}'$ satisfies (U1)–(U4). Moreover, the decision problem induced by $\mathfrak{p}$ and $\mathfrak{p}'$ is the same: In each case, we decide whether $\mathfrak{p}(F(L))$.
3. Thus, the set of decision problems induced by the axiom sets (O1)–(O3) and (U1)–(U4) is the same, and hence Theorem 2 is independent of which set of axioms is used.

The reason the axioms (O1)–(O3) were used in [15] is that if one also introduces multi-dimensional unboundedness predicates, the axioms (O1)–(O3) fit more nicely with the general case. The slight disadvantage of the definition in [15] is that one has to decide $\mathfrak{p}(F(L))$ instead of $\mathfrak{p}(L)$, which is why here, we opted for (U1)–(U4).

B   Additional material on Section 3

Translating transitions of (k, r)-RBCA to $(k', 1)$-RBCA. We provide an explicit construction for the four transition types inc_i, dec_i, zero_i, and nz_i.

Let us first consider a transition $(p, a, \mathsf{inc}_i, q)$ (Fig. 1). We start with a sequence of tests (marked in blue) for $\mathsf{p}_{i,j}$ to identify which phase we are currently in. If $\mathsf{p}_{i,j}$ is non-zero for an odd j, we know we are in an positive phase and simply increment the corresponding program counter $\mathsf{c}_{i,\frac{j+1}{2}}$. Otherwise we are in a negative phase. Thus we need to switch to an positive phase, decrementing $\mathsf{p}_{i,j}$ and incrementing $\mathsf{p}_{i,j+1}$, before incrementing the corresponding program counter $\mathsf{c}_{i,\frac{j}{2}+1}$.

Observe that the state f can never be reached in an r-reversal bounded run. As the phase counters $\mathsf{p}_{i,1}$ to $\mathsf{p}_{i,2m-1}$ have all tested negative, we must be in phase $r = 2m$. As this is an even number, it must be descending, and incrementing would mean adding another reversal, breaking the bound.
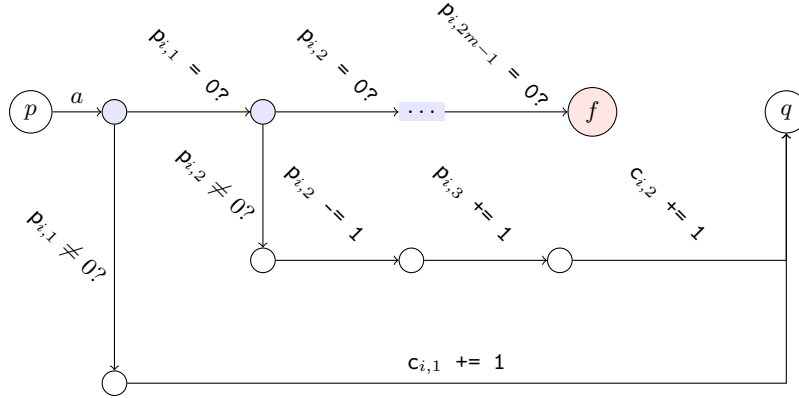

Fig. 1. Transition structure to replace an incrementing transition $(p, a, \mathsf{inc}_i, q)$.

Next, let us consider a transition $(p, a, \mathsf{dec}_i, q)$ (Fig. 2). Similarly to the previous construction, we again first identify the current phase in the state sequence marked in blue. This time, we change the phase counter if we are currently in a positive (odd) phase and move to a negative phase. Once the phase counter is updated, we change to the states marked in green, which allow us to non-deterministically decrement one of the program counters (recall that if we are in phase j, we choose one of the counters $\mathsf{c}_{i,1}, \ldots, \mathsf{c}_{i,\lceil \frac{j}{2} \rceil}$ to decrement).

Finally, we consider the zero- (Fig. 3) and non-zero-tests (Fig. 4). We test a counter c_i for zero by making sure *all* partial counters $\mathsf{c}_{i,j}$ are zero: we test a counter c_i for non-zero by testing if *any* partial counter $\mathsf{c}_{i,j}$ is non-zero.
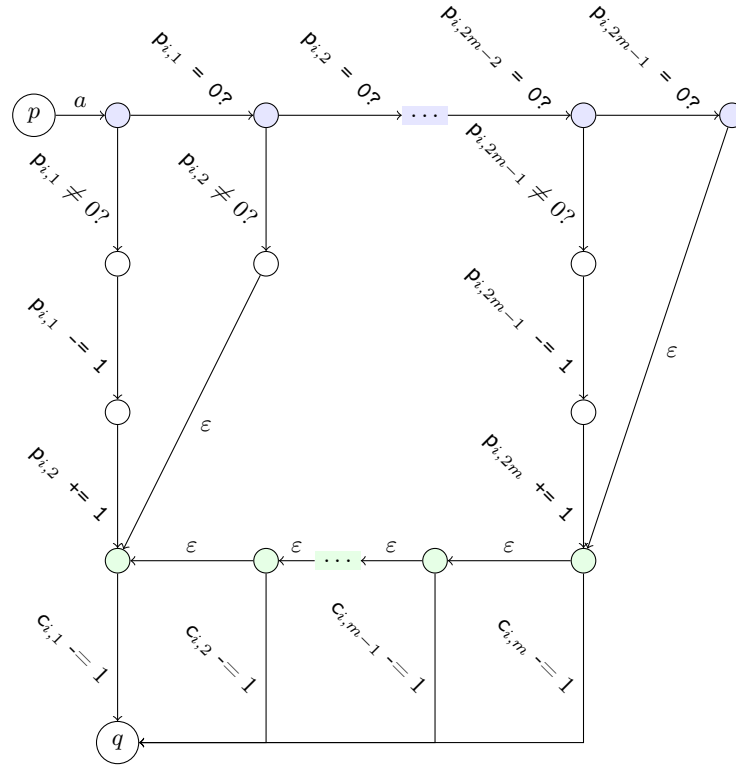
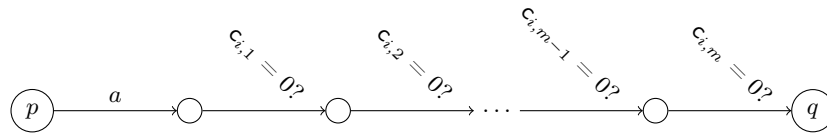**Fig. 2.** Transition structure to replace a decrementing transition $(p, a, \mathsf{dec}_i, q)$.

Fig. 3. Transition structure to replace a zero-test $(p, a, \mathsf{zero}_i, q)$.
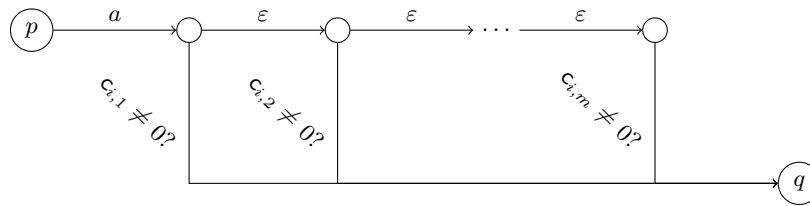

Fig. 4. Transition structure to replace a non-zero-test (p, a, nz_i, q).

C   Additional material on Section 4

Let us first observe that one can, given a subset Q, compute in logspace a context-free grammar for $L_{A,Q}$. To compute this language, we construct two grammars (one for the u part of $A \overset{*}{\Rightarrow}_Q uAv$, one for the v part). Then $L_{A,Q}$ is the union of those two grammars. We present the construction of the u-grammar here, the other can be constructed symmetrically.

Intuitively, for each production in Q, we will attempt to guess which of its resulting non-terminals will produce the subtree with the dangling A. All non-terminals to the left will be evaluated normally, everything to the right (including terminals) will be discarded, and the selected non-terminal will continue guessing which of *its* children will be responsible for producing A.

Formally, we construct a grammar as follows:

1. The set of non-terminals is duplicated, $N := M \cup \{\hat{X} \mid X \in M\}$
2. All the original productions are kept, and additionally, we introduce the following productions: If $X \to w$ is a production from Q with at least one non-terminal in w, then for all ways uYv to split w along a non-terminal, we introduce the production $\hat{X} \to u\hat{Y}$.
3. Introduce a production $\hat{A} \to \varepsilon$.
4. The start symbol is $\hat{A}$.

Observe that any marked non-terminal $\hat{X}$ can only ever occur as the rightmost symbol in any (partial) derivation. It can only be removed by the production $\hat{A} \to \varepsilon$. Therefore, a word u is produced by this new grammar if and only if there is a derivation $A \overset{*}{\Rightarrow}_Q uAv$ in Q.

We also remark the following fact about the structure of the new grammar:

Remark 1. If all productions of the original grammar are left-linear (or all are right-linear) then we also get left-linear (respectively right-linear) grammars for the languages $L_{A,Q}$. We can therefore construct NFAs for $L_{A,Q}$, which yields better complexity results for $\mathbb{Z}$-VASS.

Lemma 2. *Given a subset $M \subseteq N$, we can check in* NP *whether M is realizable. Moreover, given $M \subseteq N$ and $p \in P$, we can check in* NP *if p is M-cancelable.*

Proof. We begin by showing that we can decide realizability of a set $M \subseteq N$ in NP. In [56], it was shown that one can compute in polynomial time an existential Presburger formula ψ such that a vector $\boldsymbol{u} \in \mathbb{N}^P$ satisfies ψ if and only if $\boldsymbol{u}$ corresponds to a terminal derivation of a context-free grammar. We augment this formula with constraints that state that for all $A \in M$ some production $A \to w$ occurs, and no productions $A \to w$ with $A \notin M$ occur; as well as a constraint that the total counter effect should be zero, then check for satisfiability.

In order to check M-realizability of a production p, we first construct a second grammar G'. The grammar G' simulates pumps of G. More precisely, G' simulates some pump $A_1 \overset{*}{\Rightarrow} u_1 A_1 v_1$, then some pump $A_2 \overset{*}{\Rightarrow} u_2 A_2 v_2$, etc, and stops after simulating finitely many pumps. However, instead of producing

the terminals that these pumps generate, it uses P as its set of terminals and produces the letter $p' \in P$ whenever it applies the production p'. Then, since P is the terminal alphabet of G', we have for every $\boldsymbol{u} \in \mathbb{N}^P$:

$$\boldsymbol{u} \in \Psi(L(G')) \quad \Longleftrightarrow \quad \exists k \in \mathbb{N}, \text{ pumps } \tau_1, \ldots, \tau_k \colon \boldsymbol{u} = \Psi(\tau_1) + \cdots + \Psi(\tau_k)$$

Such a grammar G' can clearly be computed in polynomial time. Given the grammar G', we apply again the result of [56] to compute an existential Presburger formula ψ that defines $\Psi(L(G'))$. With this, we observe that p is M-realizable if and only if there exists some $\boldsymbol{u} \in \Psi(L(G'))$ such that

1. $\boldsymbol{u}(p) > 0$,
2. $\varphi(\boldsymbol{u}) = \boldsymbol{0}$, and
3. all productions occurring in $\boldsymbol{u}$ belong to M.

Since with ψ, the existence of such a $\boldsymbol{u}$ can easily expressed in another existential Presburger formula ψ', it remains to check satisfiability of ψ', which can be done in NP. $\qquad \square$

D   Additional material on Section 5

The relation $\hookrightarrow_{\mathrm{lin}}$ and growth Let us show here the following fact used in the main text:

Lemma 10. *If $L_1 \hookrightarrow_{\mathrm{lin}} L_2$ and L_1 has exponential growth, then L_2 has exponential growth as well.*

Proof. We first claim that there exists $c > 0$ with $|L_1 \cap \Sigma^{\leq m}| \leq cm(m+1) \cdot |L_2 \cap \Sigma^{\leq cm}|$. Since $L_1 \hookrightarrow_{\mathrm{lin}} L_2$, there exists a $c > 0$ such that for every word $w_1 \in L_1$, there exists a word $w_2 \in L_2$ such that $|w_2| \leq c \cdot |w_1|$ and w_1 is a factor of w_2. This allows us to construct an injection

$$f_m \colon L_1 \cap \Sigma^{\leq m} \to (L_2 \cap \Sigma^{\leq cm}) \times \{1, \ldots, cm\} \times \{0, \ldots, m\},$$

which sends each word $w_1 \in L_1 \cap \Sigma^{\leq m}$ to $(w_2, r, |w_1|)$, where $w_2 \in L_2 \cap \Sigma^{\leq cm}$ is the word existing due to $L_1 \hookrightarrow_{\mathrm{lin}} L_2$ and $r \in \{1, \ldots, cm\}$ is a position in w_2 at which w_1 appears. Then f_m must clearly be injective. Since the set on the right-hand side has cardinality $cm(m+1) \cdot |L_2 \cap \Sigma^{\leq cm}|$, this proves the claim.

We can now deduce that L_2 has exponential growth: Since L_1 has exponential growth, there is some real $r > 1$ such that $|L_1 \cap \Sigma^{\leq m}| \geq r^m$ for infinitely many m. By our claim, this implies $|L_2 \cap \Sigma^{\leq cm}| \geq \frac{r^m}{cm(m+1)}$ for infinitely many m. Since $\frac{r^m}{cm(m+1)} \geq (\sqrt[2c]{r})^{cm}$ for almost all m, we have in particular $|L_2 \cap \Sigma^{\leq cm}| \geq (\sqrt[2c]{r})^{cm}$ for infinitely many m. Thus, L_2 has exponential growth. $\qquad \square$

Proof of Lemma 9

Proof. We proceed as in the proof of Lemma 3. However, we need to show that each word u,v can be found as a factor of some word $w \in L(G)$ that is at most linear in $|u|$ or $|v|$, respectively. We show this for u, the case of v is analogous. First, choose $\ell \in \mathbb{N}$ so that for every non-terminal $B \in M$, there is a word $x \in \Sigma^*$ with $B \overset{*}{\Rightarrow}_Q x$ and $|x| \leq \ell$. Then since $A \overset{*}{\Rightarrow}_Q uAv$, we can also find a derivation $A \overset{*}{\Rightarrow}_Q uAv'$, where $|v'| \leq \ell \cdot |N| \cdot (|u| + 1)$. This is because in the derivation $A \overset{*}{\Rightarrow}_Q uAv$, the path from A to the A-leaf has at most $|u|$ nodes that are *left-branching* (meaning: branch to the left). By cutting out pumps, we may assume that between any two left-branching nodes (and above the first, and below the last), there are at most $|N|$ nodes. Thus, we arrive at a derivation tree for $A \overset{*}{\Rightarrow}_Q uAv''$ with at most $|N| \cdot (|u| + 1)$ right-branching nodes. Then, replacing each subtree under a right-branching node by a tree deriving a word of length $\leq \ell$ yields the derivation $A \overset{*}{\Rightarrow}_Q uAv'$.

Now we apply the proof of Lemma 3 to the derivation $A \overset{*}{\Rightarrow}_Q uAv'$. Observe that the total number of productions used in $\tau_0, \tau_1', \ldots, \tau_m'$ and τ is bounded by $O(|u| + |v'|) = O(|u|)$: First, note that τ_0 and the flows $\boldsymbol{f}_p$ are independent of u and v'. The total number of productions in $\boldsymbol{f}_\tau = \Psi(\tau_1') + \cdots + \Psi(\tau_m')$ is bounded linearly in the number of productions of τ by (1). The latter in turn is bounded by $|u| + |v'|$ if G is in Chomsky normal form. $\qquad\square$

Notions of polynomial and exponential growth In [25], polynomial growth for a language $L \subseteq \Sigma^*$ is defined as the existence of a polynomial $p(x)$ such that $|L \cap \Sigma^m| \leq p(m)$ for every $m \geq 0$. Moreover, exponential growth is defined as the existence of a real $r > 1$ such that $|L \cap \Sigma^m| \geq r^m$ for infinitely many $m \geq 0$. This differs slightly from our definition, which is also sometimes called *cumulative growth* [12]. One can show that in fact both definitions are equivalent with respect to polynomial and exponential growth, see e.g. [24, Lemma 2.3]. In the following we provide a proof for completeness.

First, if there is a polynomial $p(x)$ with $|L \cap \Sigma^m| \leq p(m)$ for all m, then we also have $|L \cap \Sigma^{\leq m}| \leq \sum_{i=0}^m |L \cap \Sigma^i| \leq \sum_{i=0}^m p(i) \leq (m+1) \cdot p(m)$, which is also a polynomial bound. (Note that we may clearly assume that $x \mapsto p(x)$ is a monotone function). Thus, L has polynomial growth by our definition. The converse is trivial.

Second, suppose there is a real $r > 1$ such that $|L \cap \Sigma^{\leq m}| \geq r^m$ for infinitely many m. We claim that then $|L \cap \Sigma^m| \geq \sqrt{r}^m$ for infinitely many m. Towards a contradiction, suppose that there is a $k \geq 0$ such that for all $m \geq k$, we have $|L \cap \Sigma^m| < \sqrt{r}^m$. This implies

$$r^m \leq |L \cap \Sigma^{\leq m}| = \sum_{i=0}^m |L \cap \Sigma^i| < M + \sum_{i=0}^m \sqrt{r}^i = M + \frac{\sqrt{r}^{m+1} - 1}{\sqrt{r} - 1} \qquad (2)$$

for all $m \geq k$, where M is the constant $\sum_{i=0}^{k} |L \cap \Sigma^i| - \sqrt{r}^i$. The last equality follows from the induction

$$\sqrt{r}^0 = 1 = \frac{\sqrt{r} - 1}{\sqrt{r} - 1} \quad \text{and} \quad \sum_{i=0}^{m+1} \sqrt{r}^i \stackrel{\text{IH}}{=} \frac{\sqrt{r}^{m+1} - 1}{\sqrt{r} - 1} + \sqrt{r}^{m+1} = \frac{\sqrt{r}^{m+2} - 1}{\sqrt{r} - 1}.$$

However, Eq. (2) is clearly violated for large enough m. Again, the converse is obvious: If $|L \cap \Sigma^m| \geq r^m$, then clearly also $|L \cap \Sigma^{\leq m}| \geq r^m$.