Improved Ackermannian lower bound for the VASS reachability problem

Sławomir Lasota
University of Warsaw, Poland

Abstract

This draft is a follow-up of the Ackermannian lower bound for the reachability problem in vector addition systems with states (VASS), recently announced by Czerwiński and Orlikowski. Independently, the same result has been announced by Leroux, but with a significantly different proof. We provide a simplification of the former construction, thus improving the lower bound for VASS in fixed dimension: while Czerwiński and Orlikowski prove $\mathcal{F}_k$-hardness in dimension $6k$, and Leroux in dimension $4k + 9$, the simplified construction yields $\mathcal{F}_k$-hardness already in dimension $3k + 2$.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Verification by model checking; Theory of computation → Logic and verification

Keywords and phrases reachability problem, vector addition systems, Petri nets

Digital Object Identifier 10.4230/LIPIcs...

Acknowledgements Thanks to Wojtek Czerwiński and Łukasz Orlikowski for fruitful discussions and suggesting further simplifications of the simplified construction presented in this paper.

1 Introduction

Petri nets [9] are a long established model of concurrency, with extensive applications in various fields. The model admits various alternative but essentially equivalent presentations, most notably *vector addition systems* (VAS) [5], and *vector addition systems with states* (VASS) [3, Sec.5], [4]. In this paper we work with a convenient presentation of VASS as counter programs without zero tests, where the dimension of a VASS corresponds to the number of counters of a program. The central algorithmic problem for this model is reachability: whether from the given initial configuration there exists a sequence of valid execution steps that reaches the given final configuration. Each of the alternative presentations admits its own formulation of the reachability problem, all of them being equivalent due to straightforward polynomial-time translations that preserve reachability, see e.g. Schmitz's survey [11, Section 2.1].

The reachability problem has a long history, as confirmed by numerous decidability proofs. The best known upper bound, shown recently by Leroux and Schmitz [6], is Ackermannian. Furthermore, in fixed dimension k the problem is shown to belong to $\mathcal{F}_{k+4}$ [6], where the complexity class $\mathcal{F}_i$ corresponds to the ith level in Grzegorczyk Hierarchy [10]. Much less has been know, until recently, concerning lower bound: after Lipton's landmark result that the reachability problem requires exponential space [8], a next progress has been reported over 40 years later: a breakthrough construction of [1] shows that the reachability problem is Tower-hard. Finally, the gap between the lower and upper bound has been recently closed by Czerwiński and Orlikowski in [2], and independently by Leroux in [7]. Specifically, the former paper shows $\mathcal{F}_k$-hardness in dimension $6k$, and the latter one in dimension $4k + 9$, any of these results immediately implying Ackermann-hardness for unrestricted dimension.

As a follow-up, in this paper we provide a simplification of the construction of [2] which yields an improved lower bound: $\mathcal{F}_k$-hardness already in dimension $3k + 2$. We also believe that the simplified construction is conceptually easier and more direct.

2 The reachability problem

In this section we define the reachability problem and state the results.

Counter programs. A *counter program* (or simply a *program*) is a sequence of commands, each of which is of one of the following types:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')
zero? x	(zero test: continue if counter x equals 0)

Counters are only allowed to have nonnegative values. We are particularly interested in counter programs *without zero tests*, i.e., ones that use no zero test command. Whenever it is not specified explicitly, counter programs are implicitly assumed to be without zero tests.

▶ **Example 1.** We write $\mathsf{x} \mathrel{+}= m$ (resp. $\mathsf{x} \mathrel{-}= m$) for m consecutive increments (resp. decrements) of x. As an illustration, consider the program with three counters $\mathsf{C} = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$ (on the left), and its more readable presentation using a syntactic sugar **loop** (on the right):

```
1: goto 2 or 6
2: x -= 1
3: y += 1
4: z += 2
5: goto 1 or 1
6: z += 1
```

```
1: loop
2:     x -= 1
3:     y += 1
4:     z += 2
5: z += 1
```

It repeats the block of atomic commands in lines 2–4 some number of times chosen nondeterministically (possibly zero, although not infinite because x is decreasing and hence its initial value bounds the number of iterations) and then halts provided counter x is zero. In the sequel we conveniently use **loop** construct instead of explicit **goto** commands.

We emphasise that counters are only permitted to have nonnegative values. In the program above, that is why the decrement in line 2 works also as a non-zero test.

Consider a program with counters C. Let $\textsc{Val}(\mathsf{C}) = \mathbb{N}^{\mathsf{C}}$ denote the set of all valuations of counters. Given an initial valuation of counters, a run of a counter program is a sequence of executions of commands, as expected. A run which has successfully finished we call *complete*; otherwise, the run is either *partial* or *infinite*. Observe that, due to a decrement that would cause a counter to become negative, a partial run may fail to continue because it is blocked from further execution. Moreover, due to nondeterminism of **goto**, the same program may have various runs from the same initial valuation.

When concatenating two programs, we silently assume the appropriate re-numbering of lines referred to by **goto** in the post-composed program.

The reachability problem. Given a set $R \subseteq \textsc{Val}(\mathsf{C})$ of valuations, by a run *from* R we mean any run whose initial valuation belongs to R. A complete run is called X-*zeroing*, for a subset $\mathsf{X} \subseteq \mathsf{C}$ of counters, if it ends with $\mathsf{x} = 0$ for all $\mathsf{x} \in \mathsf{X}$. When $R = \{v\}$ contains a single valuation v, we speak of runs from v instead of runs from $\{v\}$. Let $\mathsf{0}$ denote the valuation where all counters are 0. In this paper we investigate the complexity of the following decision problem (with a partially specified final valuation of counters):

Reachability problem
Input A program $\mathcal{P}$ without zero tests, and a subset X of its counters.
Question Does $\mathcal{P}$ have an X-zeroing run from the zero valuation $\mathsf{0}$?

Fast-growing hierarchy. Let $\mathbb{N}_k \subseteq \mathbb{N}$ denote positive integers divisible by k, for $k \geq 1$. We define the complexity classes $\mathcal{F}_k$ corresponding to the kth level in Grzegorczyk Hierarchy [10, Sect. 2.3, 4.1]. The standard family of approximations $\mathbf{A}_i : \mathbb{N}_1 \to \mathbb{N}_1$ of Ackermann function, for $i \in \mathbb{N}_1$, can be defined as follows:

$$\mathbf{A}_1(n) = 2n, \qquad \mathbf{A}_{i+1}(n) = \underbrace{\mathbf{A}_i \circ \mathbf{A}_i \circ \ldots \circ \mathbf{A}_i}_{n-1}(2).$$

We define the complexity classes $\mathcal{F}_i$, indexed by $i \in \mathbb{N}_1$, of problems solvable in deterministic time $\mathbf{A}_i(p(n))$, where $p : \mathbb{N}_1 \to \mathbb{N}_1$ ranges over functions computable in deterministic time ${\mathbf{A}_{i-1}}^m(n)$, for some $m \in \mathbb{N}_1$:

$$\mathcal{F}_i = \bigcup_{p \in \mathcal{FF}_{i-1}} \text{DTime}(\mathbf{A}_i(p(n))), \qquad \text{where } \mathcal{FF}_i = \bigcup_{m \in \mathbb{N}_1} \text{FDTime}({\mathbf{A}_i}^m(n)).$$

Intuitively speaking, the class $\mathcal{F}_i$ contains problems solvable in time $\mathbf{A}_i(n)$, and is closed under reductions computable in time of lower order ${\mathbf{A}_{i-1}}^m(n)$, for some fixed $m \in \mathbb{N}_1$. In particular, $\mathcal{F}_3 = \text{Tower}$ (problems solvable in a tower of exponentials of time or space, whose height is an elementary function of the input size). The classes $\mathcal{F}_k$ are robust with respect to the choice of fast-growing function hierarchy (see [10, Sect.4.1]). For $k \geq 3$, instead of deterministic time, one could equivalently take nondeterministic time, or space.

Lower bound in fixed dimension. As the main result we prove $\mathcal{F}_k$-hardness for programs with $3k + 2$ counters:

▶ **Theorem 2.** *Let $k \geq 3$. The reachabilty problem for programs with $3k + 2$ counters is $\mathcal{F}_k$-hard.*

The result can be compared to $\mathcal{F}_k$-hardness shown in [2] for $6k$ counters, and in [7] for $4k + 9$ counters. Like the cited results, Theorem 2 implies Ackermann-hardness for unrestricted number of counters which, together with Ackermann upper bound of [6], yields Ackermann-completness of the reachability problem.

Idea of simplification. Czerwiński and Orlikowski [2] use the *ratio technique* introduced previously in [1]. Speaking slightly informally, suppose some three counters $\mathsf{b}, \mathsf{c}, \mathsf{d}$ satisfy initially

$$\mathsf{b} = B, \qquad \mathsf{c} > 0, \qquad \mathsf{d} = \mathsf{b} \cdot \mathsf{c}, \tag{1}$$

for some fixed positive integer $B \in \mathbb{N}$. Furthermore, suppose that the values of c and d may be initially chosen arbitrarily, in a nondeterministic way, as long as they satisfy the latter equality in (1); they are hence unbounded. Under these assumptions, one can correctly simulate unboundedly many zero tests (in fact, the number of simulated zero-tests corresponds to the initial value of c) on counters bounded by B, using some auxiliary counters. The core idea underlying the simplification presented in this paper is, intuitively speaking, to swap the roles of counters b and c: we observe that the above-defined assumption (1) allows us to correctly simulate exactly $B/2$ zero tests (for B even) on unbounded counters (in fact, on counters bounded by the initial value of c), without using any auxiliary counters. This novel approach is presented in detail in Section 4.

3 Multipliers

Following the lines of [2], we rely on a concept of a *multiplier*.

Consider a program with counters C, a counter $\mathsf{x} \in \mathsf{C}$ and $R \subseteq \textsc{Val}(\mathsf{C})$. We define the set x-*computed* from R as the set of all valuations of counters at the end of all x-zeroing runs from R. For instance, the program in Example 1 above x-computes, from the set of all valuations satisfying $\mathsf{y} = \mathsf{z} = 0$, the set of all valuations satisfying (trivially) $\mathsf{x} = 0$ and $\mathsf{z} = 2\mathsf{y} + 1$.

Let $\mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathsf{C}$ be some three distinguished counters, and $B \in \mathbb{N}_4$. We define the subset $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}) \subseteq \textsc{Val}(\mathsf{C})$ of all valuations that satisfy (1). A program with counters C that, for some four of its counters $\mathsf{z}, \mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathsf{C}$, z-computes from the zero valuation $\mathsf{0}$ the set $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$, we call B-*multiplier*.

▶ **Example 3.** As a simple example, for every fixed $B \in \mathbb{N}_4$, the following program is a B-multiplier of size $\mathcal{O}(B)$ (counter z is not used at all):

Program $\mathcal{M}_B(\mathsf{b}, \mathsf{c}, \mathsf{d})$:

1: b += B $\quad$ d += B $\quad$ c += 1
2: **loop**
3: $\quad$ d += B $\quad$ c += 1

We prefer to rely on the following family of functions $\mathbf{F}_i : \mathbb{N}_4 \to \mathbb{N}_4$, indexed by $i \in \mathbb{N}_1$, closely related to, but different than, functions $\mathbf{A}_i$ (cf. Claim 4 below):

$$\mathbf{F}_1(n) = 2n, \qquad \mathbf{F}_{i+1} = \widetilde{\mathbf{F}_i},$$

where, for $F : \mathbb{N}_4 \to \mathbb{N}_4$, the successor function $\widetilde{F}$ is defined as

$$\widetilde{F}(n) = \underbrace{F \circ F \circ \ldots \circ F}_{n/4-1}(8).$$

In particular, $\mathbf{F}_i(4) = 8$ for all $i \in \mathbb{N}_1$. By induction on i and n, one easily shows:

▷ Claim 4. $\mathbf{F}_i(4 \cdot n) = 4 \cdot \mathbf{A}_i(n)$, for $i, n \in \mathbb{N}_1$.

As a technical core of the proof of Theorem 2, following the lines of [2], we provide an effective construction of B-multipliers, for $B = \mathbf{F}_k(n)$, of size polynomial in k and n.

▶ **Theorem 5.** *Given $k \geq 2$ and $n \in \mathbb{N}_4$ one can compute, in time polynomial in k and n, an $\mathbf{F}_k(n)$-multiplier with $3k + 2$ counters.*

4 Proof of Theorem 2

Relying on Theorem 5 (to be shown later), we prove in this section Theorem 2.

Maximal iteration. Whenever analysing a single run of a program, we denote by $\overline{\mathsf{x}}$ the initial value of a counter x, and by $\underline{\mathsf{x}}$ the final value thereof. In the sequel we intensively use loops of the following form that, intuitively, flushes the value from counter f to e, decreasing simultaneously the counter d (several command are written in none line to save space):

1: **loop**
2: $\quad$ f −= 1 $\quad$ e += 1 $\quad$ d −= 1

Assuming $\overline{\mathsf{d}} \geq \overline{\mathsf{e}} + \overline{\mathsf{f}}$, we observe that the amount $\overline{\mathsf{d}} - \underline{\mathsf{d}}$ by which d is decreased as an effect of execution (*execution* is used as a synonym to *run*) of the macro may be any value between 0 and $\overline{\mathsf{f}}$, and that the equality $\overline{\mathsf{d}} - \underline{\mathsf{d}} = \overline{\mathsf{e}} + \overline{\mathsf{f}}$ holds if, and only if the loop is *iterated maximally*, i.e., $\overline{\mathsf{e}} = 0$ anf $\underline{\mathsf{f}} = 0$. This simple observation will play a crucial role in the sequel.

Bounded number of zero tests. Let $\mathcal{P}$ be a counter program with counters C that uses zero tests only on two counters $\mathsf{x}, \mathsf{y} \in \mathsf{C}$ (the construction easily extends to programs with an arbitrary number of zero-tested counters). We add to $\mathcal{P}$ three counters $\mathsf{b}, \mathsf{c}, \mathsf{d}$ (let $\mathsf{C}^* = \mathsf{C} \cup \{\mathsf{b}, \mathsf{c}, \mathsf{d}\}$), and transform $\mathcal{P}$ into a program $\mathcal{P}^*$ *without zero tests* that, assuming the initial valuation of counters belongs to $\textsc{Ratio}(2m, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}^*)$, for some $m \in \mathbb{N}$, simulates correctly m zero tests (jointly) on counters x, y, as long as their sum is bounded by the initial value of c.

The transformation proceeds in two steps. First, we accompany every increment (decrement) on x with a decrement (increment) of c:

command	replaced by	
x += 1	x += 1	c −= 1
x −= 1	x −= 1	c += 1

Likewise we do for y. In the resulting program $\bar{\mathcal{P}}$ counters x, y are, intuitively speaking, put on a shared 'budget' c. This clearly enforces $\mathsf{x} + \mathsf{y}$ to not exceed the initial value of c, and the sum $s = \mathsf{c} + \mathsf{x} + \mathsf{y}$ to remain invariant. Second, we replace in $\bar{\mathcal{P}}$ all **zero?** x commands by the following macros, thus obtaining $\mathcal{P}^*$:

Zero? x**:**

1: **loop**
2: y −= 1 x += 1 d −= 1
3: **loop**
4: c −= 1 y += 1 d −= 1
5: **loop**
6: c += 1 y −= 1 d −= 1
7: **loop**
8: y += 1 x −= 1 d −= 1
9: b −= 2

Likewise we replace all **zero?** y commands by an analogous macro Zero? y, where x and y are swapped. Both macros preserve the sum $s = \mathsf{c} + \mathsf{x} + \mathsf{y}$.

An execution of Zero? x is *maximally iterated* if all four loop are so. Observe that every such execution forcedly satisfies $\overline{\mathsf{x}} = \underline{\mathsf{x}} = 0$. The idea behind Zero? x is to flush from y to a zero-tested counter x and back, but also flush from c to y and back, in an appropriately nested way that guarantees that the amount $\overline{\mathsf{d}} = \underline{\mathsf{d}}$ by which d is decreased equals $2s$ exactly in maximally iterated executions:

▷ Claim 6. Consider an execution of Zero? macro, assuming $\overline{\mathsf{d}} \geq 2s$. Then $0 \leq \overline{\mathsf{d}} - \underline{\mathsf{d}} \leq 2s$. Furthermore, the equality $\overline{\mathsf{d}} - \underline{\mathsf{d}} = 2s$ holds if, and only if the execution is maximally iterated.

Proof. Consider an execution of Zero?, assuming $\overline{\mathsf{d}} \geq 2s$, and let $\dot{\mathsf{y}}$ denote the value of y at the exit from the first loop. The amount by which d is decreased in the two loops in lines 1–2 and 7–8 is at most

$$\Delta_1 = 2(\overline{\mathsf{y}} - \dot{\mathsf{y}}) + \overline{\mathsf{x}}.$$

Furthermore, the amount by which d is decreased in the two loops in lines 3–6 is at most

$$\Delta_2 = 2\overline{\mathsf{c}} + \dot{\mathsf{y}}.$$

The sum $\Delta_1 + \Delta_2$ clearly satisfies $\Delta_1 + \Delta_2 \leq 2s = 2(\overline{\mathsf{c}} + \overline{\mathsf{x}} + \overline{\mathsf{y}})$. It equals $2s$ if, and only if $\Delta_1 = 2\overline{\mathsf{y}}$ an $\Delta_2 = 2\overline{\mathsf{c}}$, i.e., exactly when all four loops are maximally iterated. ◀

In consequence, as b is decreased by 2, if the invariant $\mathsf{d} = \mathsf{b} \cdot s$ is preserved by an execution of Zero?$\,\mathsf{x}$ then the zero test is forcedly *correct*: $\overline{\mathsf{x}} = \underline{\mathsf{x}} = 0$. Furthermore notice that once the invariant is violated, i.e., $\mathsf{d} > \mathsf{b} \cdot s$, by the first part of Claim 6 the invariant can not be recovered later. These observations lead to the following correctness claim:

▶ **Lemma 7.** *The following conditions are equivalent:*
- *$\mathcal{P}$ has a complete run from $\mathsf{0}$ that does exactly m zero-tests.*
- *$\mathcal{P}^*$ has a d-zeroing run from* Ratio$(2m, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}^*)$.

Proof. One direction is immediate: for each complete run π of $\mathcal{P}$ from $\mathsf{0}$ that does m zero-tests on x, y, there is a corresponding d-zeroing run of $\mathcal{P}^*$ from from Ratio$(2m, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}^*)$, for any initial value $\overline{\mathsf{c}}$ at least as large as the maximal value of the sum $\mathsf{x} + \mathsf{y}$ along π. The run iterates maximally all loops in Zero? macros.

For the converse direction, consider a d-zeroing run π of $\mathcal{P}^*$ from Ratio$(2m, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}^*)$. The initial counter valuation satisfies the equality $\mathsf{d} = \mathsf{b} \cdot s$. Each execution of Zero? decreases b by 2, and d by at most $2s$ (by the first part of Claim 6). Therefore, since b and d are not affected elsewhere and $\mathsf{d} = 0$ finally, each execution of Zero? forcedly decreases d by *exactly* $2s$. By the second part of Claim 6 we derive:

▷ Claim 8. Each execution of Zero? in π is correct.

Furthermore, forcedly $\mathsf{b} = 0$ finally as well, which implies:

▷ Claim 9. There are exactly m executions of Zero? in π.

Due to Claims 8 and 9, once we remove from $\mathcal{P}^*$ the counters $\mathsf{b}, \mathsf{c}, \mathsf{d}$ we obtain a complete run of $\mathcal{P}$ from $\mathsf{0}$ that does exactly m zero-tests, as required. ◀

Proof of Theorem 2. Fix $k \in \mathbb{N}_1$. The proof proceeds by a polynomial-time reduction from the following $\mathcal{F}_k$-hard problem:

$\mathbf{F}_k$-bounded halting problem
Input A program $\mathcal{P}$ of size n (w.l.o.g. assume $n \in \mathbb{N}_4$) with 2 zero-tested counters.
Question Does $\mathcal{P}$ have a complete run that does at most $\mathbf{F}_k(n)/2$ zero tests?

The problem is $\mathcal{F}_k$-hard, since any program with with arbitrary many zero-tested counters can be simulated by a program with 2 such counters, where simulation of each command of the former one requires a zero test of the latter one, which translates a bound on time of computation to the same bound on the number of zero tests.

Given $\mathcal{P}$ as above with two counters x, y, we transform it to a counter program $\mathcal{P}'$ with $3k + 2$ counters C but without zero tests, such that $\mathcal{P}$ has a complete run that does at most $m = \mathbf{F}_k(n)/2$ zero tests if, and only if, $\mathcal{P}'$ has a $\{\mathsf{d}, \mathsf{z}\}$-zeroing run (for some $\mathsf{d}, \mathsf{z} \in \mathsf{C}$).

First, using Theorem 5 we compute a $2m$-multiplier $\mathcal{M}$ with $3k + 2$ counters C that z-computes from $\mathsf{0}$ the set Ratio$(2m, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$, for some $\mathsf{z}, \mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathsf{C}$. Second, we post-compose $\mathcal{P}$ with a simple program $\mathcal{L}$ that first decrements x, and then zero tests it, nondeterministically many times both:

1: **loop**
2: x −= 1
3: **loop**
4: **zero?** x

Thus $\mathcal{P}$ has a complete run that does *at most* m zero tests if, and only if the composed program $\mathcal{P}\ \mathcal{L}$ has a complete run that does *exactly* m zero tests. Finally, we compose $\mathcal{M}$ with the transformed program $(\mathcal{P}\ \mathcal{L})^*$, and get the required equivalence:

▷ Claim 10. The following conditions are equivalent:
- $\mathcal{P}$ has a complete run from $\mathsf{0}$ that does at most m zero tests if;
- $\mathcal{P}\ \mathcal{L}$ has a complete run from $\mathsf{0}$ that does exactly m zero tests;
- $(\mathcal{P}\ \mathcal{L})^*$ has a d-zeroing run from $\textsc{Ratio}(2m, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}^*)$;
- $\mathcal{P}' = \mathcal{M}\ (\mathcal{P}\ \mathcal{L})^*$ has a $\{\mathsf{z}, \mathsf{d}\}$-zeroing run from $\mathsf{0}$.

The second and the third point are equivalent due to Lemma 7, while the equivalence of the third and the last point follows by the defining property of multipliers.

The program $\mathcal{P}'$ has $3k+4$ counters ($3k+2$ counters of $\mathcal{M}$ plus x, y) but, since $k \geq 3$, this number can be decreased back to $3k+2$, by re-using some of $3k-2$ counters from $\mathsf{C}' = \mathsf{C} - \{\mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{z}\}$ in place of x, y. Claim 10 remains true, as z-zeroing runs of $\mathcal{M}$ from $\mathsf{0}$ are necessarily C'-zeroing too. ◀

5 Proof of Theorem 5

The proof proceeds by combining the concept of amplifiers and amplifier lifting of [2] with the program transformation of the previous section.

Amplifiers. Let $F : \mathbb{N}_4 \to \mathbb{N}_4$ be a monotone function satisfying $f(n) \geq n$ for $n \in \mathbb{N}_4$. Consider a program $\mathcal{P}$ with counters C and distinguished six counters: three *input counters* $\mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathsf{C}$ and three *output counters* $\mathsf{b}', \mathsf{c}', \mathsf{d}' \in \mathsf{C}$. The program is an *$F$-amplifier* if for every $B \in \mathbb{N}_4$, it d-computes from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$ the set $\textsc{Ratio}(F(B), \mathsf{b}', \mathsf{c}', \mathsf{d}', \mathsf{C})$. We note that no condition is imposed on d-zeroing runs from counter valuations not belonging to any set $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$. As an example, consider the following program $\mathcal{L}_\ell$, for $\ell \in \mathbb{N}_1$, with input counters $\mathsf{b}, \mathsf{c}, \mathsf{d}$ and output counters $\mathsf{b}', \mathsf{c}', \mathsf{d}'$:

Program $\mathcal{L}_\ell(\mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{b}', \mathsf{c}', \mathsf{d}')$:

1: **loop**
2: **loop**
3: c −= 1 c' += 1 d −= 1 d' += ℓ
4: **loop**
5: c += 1 c' −= 1 d −= 1 d' += ℓ
6: b −= 2 b' += 2ℓ
7: **loop**
8: c −= 1 c' += 1 d −= 2 d' += 2ℓ
9: b −= 2 b' += 2ℓ

▷ Claim 11. The above program is an L_ℓ-amplifier, where $L_\ell : \mathbb{N}_4 \to \mathbb{N}_4 : x \mapsto \ell \cdot x$.

Proof sketch. Writing counter valuations as vectors $(\mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{b}', \mathsf{c}', \mathsf{d}')$, one shows that the program d-computes, from the set containing just one counter valuation $(B, c, d, 0, 0, 0)$, the set containing one counter valuation $(0, 0, 0, \ell \cdot B, c, \ell \cdot d)$. ◀

Putting $\ell = 1$ we get an identity-amplifier $\mathcal{L}_1(\mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{b}', \mathsf{c}', \mathsf{d}')$.

Amplifier lifting. Recall the definition of functions $\mathbf{F}_1 = L_2$ and $\mathbf{F}_{i+1} = \widetilde{\mathbf{F}}$. Let $\mathcal{P}$ be a program with counters C, without zero tests, with distinguished input counters $\mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1 \in \mathsf{C}$ and output counters $\mathsf{b}_2, \mathsf{c}_2, \mathsf{d}_2 \in \mathsf{C}$. We describe a transformation of the program $\mathcal{P}$ without zero tests to a program $\widetilde{\mathcal{P}}$ without zero tests such that assuming that $\mathcal{P}$ is an F-amplifier, $\widetilde{\mathcal{P}}$ is an $\widetilde{F}$-amplifier. The program $\widetilde{\mathcal{P}}$ uses, except for the counters of $\mathcal{P}$, three fresh counters $\mathsf{b}, \mathsf{c}, \mathsf{d}$. Thus counters of $\widetilde{\mathcal{P}}$ are $\widetilde{\mathsf{C}} = \mathsf{C} \cup \{\mathsf{b}, \mathsf{c}, \mathsf{d}\}$. We let input counters of $\widetilde{\mathcal{P}}$ be $\mathsf{b}, \mathsf{c}, \mathsf{d}$, and its output counters be $\mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1$.

In the transformation we use the identity-amplifier $\mathcal{L} = \mathcal{L}_1(\mathsf{b}_2, \mathsf{c}_2, \mathsf{d}_2, \mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1)$ with input counters $\mathsf{b}_2, \mathsf{c}_2, \mathsf{d}_2$ and output counters $\mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1$, and the 8-multiplier $\mathcal{M} = \mathcal{M}_8(\mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1)$ of Example 3. We also use the ZERO? macros with d_1 and d_2 in place of x and y.

We start by transforming $\mathcal{P}$ to $\bar{\mathcal{P}}$, by accompanying every increment (decrement) of d_1 or d_2 with decrement (increment) of c (similarly as in the previous section). Likewise, we transform the identity-amplifier $\mathcal{L}$ to $\bar{\mathcal{L}}$, and the multiplier $\mathcal{M}$ to $\bar{\mathcal{M}}$. Using $\bar{\mathcal{P}}$, $\bar{\mathcal{L}}$ and $\bar{\mathcal{M}}$ as building blocks, we define a counter program $\widetilde{\mathcal{P}}$ as follows:

Program $\widetilde{\mathcal{P}}$:

1: $\bar{\mathcal{M}}$
2: **loop**
3: $\quad\bar{\mathcal{P}}$
4: $\quad$ZERO? d_1
5: $\quad\bar{\mathcal{L}}$
6: $\quad$ZERO? d_2
7: SET-C-TO-ZERO

SET-C-TO-ZERO**:**

1: **loop**
2: $\quad\mathsf{c} \mathrel{-}= 1$
3: $\quad\mathsf{d} \mathrel{-}= 4$
4: ZERO? c
5: ZERO? c

Without the macro SET-C-TO-ZERO, the program $\widetilde{\mathcal{P}}$ would be exactly the result of applying the transformation of the previous section to the following program:

1: $\mathcal{M}$
2: **loop**
3: $\quad\mathcal{P}$
4: $\quad$**zero?** d_1
5: $\quad\mathcal{L}$
6: $\quad$**zero?** d_2

Note that $\bar{\mathcal{M}}$ initialises counters C to RATIO$(8, \mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1, \mathsf{C})$. Intuitively speaking, the purpose of the macro SET-C-TO-ZERO is to set c to 0. It uses a macro ZERO? c obtained from ZERO? d_1 by swapping d_1 and c. The doubled use of ZERO? c guarantees that the macro SET-C-TO-ZERO decreases b by exactly 4, and hence decreases d by at most $4(\overline{\mathsf{c}} + \overline{\mathsf{d}}_1 + \overline{\mathsf{d}}_2)$.

As in Section 4, we observe that the amount $s = \mathsf{c} + \mathsf{d}_1 + \mathsf{d}_2$ is preserved by ZERO? macros and hence stays invariant in each run of $\widetilde{\mathcal{P}}$ before entering the SET-C-TO-ZERO macro. We derive the following property of SET-C-TO-ZERO from Claim 6 (note however that SET-C-TO-ZERO does not forcedly preserve $\mathsf{c} + \mathsf{d}_1 + \mathsf{d}_2$):

▷ Claim 12. Consider an arbitrary execution of SET-C-TO-ZERO(c) and assume $\overline{\mathsf{d}} \geq 4s$. Then $0 \leq \overline{\mathsf{d}} - \underline{\mathsf{d}} \leq 4s$. Furthermore, the equality $\overline{\mathsf{d}} - \underline{\mathsf{d}} = 4s$ holds if, and only if both ZERO? macros are maximally iterated.

▶ **Lemma 13.** *If $\mathcal{P}$ is an F-amplifier, then $\widetilde{\mathcal{P}}$ is an $\widetilde{F}$-amplifier.*

Proof. Let $\mathcal{P}$ be an F-amplifier and let $B = 4(\ell+1) \in \mathbb{N}_4$, where $\ell \in \mathbb{N}$.

We argue similarly as in the proof of Lemma 7. Consider all d-zeroing runs of $\widetilde{\mathcal{P}}$ from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$. All initial counter valuations satisfy the equality $\mathsf{d} = \mathsf{b} \cdot s$, and the equality is preserved by $\bar{\mathcal{M}}$. Further on, each execution of Zero? decreases b by 2, and d by at most $2s$ (by the first part of Claim 6). Likewise, the execution of Set-c-to-zero decreases b by 4 and d by at most $4s$ (by the first part of Claim 12). Therefore, since b and d are not affected elsewhere and $\underline{\mathsf{d}} = 0$ finally, we have the following properties of d-zeroing runs of $\widetilde{\mathcal{P}}$ from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$:

▷ Claim 14. Finally, $\underline{\mathsf{b}} = 0$.

▷ Claim 15. Each execution of Zero? forcedly decreases d by *exactly* $2s$, and the execution of Set-c-to-zero forcedly decreases d by *exactly* $4s$.

We aim at proving that from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$, the program $\widetilde{\mathcal{P}}$ d-computes exactly the set $\textsc{Ratio}(\widetilde{F}(B), \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$. By Claim 15 we deduce:

▷ Claim 16. Each d-zeroing run from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$ contains exactly 2ℓ executions of Zero? macro, all of them correct.

In further analysis of all d-zeroing runs of $\widetilde{\mathcal{P}}$ from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$ we ignore the (values of the) added counters $\mathsf{b}, \mathsf{c}, \mathsf{d}$, and thus restrict to counters C. Let $R_0 \subseteq \textsc{Val}(\mathsf{C})$ be the set of all valuations of counters in C at the exit from $\bar{\mathcal{M}}$ in all d-zeroing runs from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$. Similarly, let $R_i \subseteq \textsc{Val}(\mathsf{C})$ be the set of all valuations of counters in C after i executions of Zero? in all d-zeroing runs from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$.

As $\mathcal{M}$ is an 8-multiplier, $R_0 = \textsc{Ratio}(8, \mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1, \mathsf{C})$. We show:

▷ Claim 17. $R_{2\ell} = \textsc{Ratio}(F^\ell(8), \mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1, \mathsf{C})$.

If $\ell = 0$, $R_{2\ell} = R_0$ proves the claim. Otherwise, suppose $\ell > 0$. By Claim 16, each execution of $\bar{\mathcal{P}}$ is d_1-zeroing; therefore, as $\mathcal{P}$ is an F-amplifier and the value of c may be large enough to not restrict runs of $\bar{\mathcal{P}}$ w.r.t. runs of $\mathcal{P}$, we get $R_1 = \textsc{Ratio}(F(8), \mathsf{b}_2, \mathsf{c}_2, \mathsf{d}_2, \mathsf{C})$. By Claim 16 again, each execution of $\bar{\mathcal{L}}$ is d_2-zeroing; therefore $R_2 = \textsc{Ratio}(F(8), \mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1, \mathsf{C})$ similarly as before. By an ℓ-fold repetition of this argument we derive Claim 17.

By the second part of Claim 12, the macro Set-c-to-zero does not change the value of d_1 d-zeroing runs from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$, and ends with $\underline{\mathsf{c}} = \mathsf{d}_2 = 0$. By Claim 14 we have $\underline{\mathsf{b}} = 0$. As Set-c-to-zero affects no other counters, by Claim 17 we deduce that $\widetilde{\mathcal{P}}$ d-computes from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \widetilde{\mathsf{C}})$ the set $\textsc{Ratio}(F^\ell(8), \mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1, \widetilde{\mathsf{C}})$. As $F^\ell(8) = \widetilde{F}(B)$, $\widetilde{\mathcal{P}}$ is an $\widetilde{F}$-amplifier. ◀

Proof of Theorem 5. Given $k \in \mathbb{N}_1$ and $n \in \mathbb{N}_4$ we compute, in time linear in k, the $\mathbf{F}_k$-amplifier $\mathcal{A}_k$ with $3k+3$ counters C, by $(k-1)$-fold application of the amplifier lifting construction described above, starting from the $\mathbf{F}_1$-amplifier $\mathcal{L}_2$ of Claim 11. The $\mathbf{F}_k(n)$-multiplier is obtained by pre-composing $\mathcal{A}$ with an n-multiplier (e.g. $\mathcal{M}_n(\mathsf{b}, \mathsf{c}, \mathsf{d})$ from Section 2) outputting the set $\textsc{Ratio}(n, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$ in input counters $\mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathsf{C}$ of $\mathcal{A}_k$.

Finally we observe that counter b is bounded by n and hence can be eliminated: we encode its values in control locations, by taking as locations the cross-product with $\{0, \ldots, n\}$. ◀

References

1 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In Moses Charikar and Edith Cohen, editors, *Proc. STOC 2019*, pages 24–33. ACM, 2019.

2 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete, 2021. arXiv:2104.13866.

3 Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.*, 7:311–324, 1978. URL: https://doi.org/10.1016/0304-3975(78)90020-8.

4 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979. URL: https://doi.org/10.1016/0304-3975(79)90041-0.

5 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969. URL: https://doi.org/10.1016/S0022-0000(69)80011-5.

6 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Proc. LICS 2019*, pages 1–13. IEEE, 2019.

7 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive, 2021. arXiv:2104.12695.

8 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976. URL: http://cpsc.yale.edu/sites/default/files/files/tr63.pdf.

9 Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962. URL: http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/.

10 Sylvain Schmitz. Complexity hierarchies beyond elementary. *TOCT*, 8(1):3:1–3:36, 2016. URL: http://doi.acm.org/10.1145/2858784.

11 Sylvain Schmitz. The complexity of reachability in vector addition systems. *SIGLOG News*, 3(1):4–21, 2016. URL: http://doi.acm.org/10.1145/2893582.2893585.