Improved Ackermannian Lower Bound for the Petri Nets Reachability Problem

Sławomir Lasota
University of Warsaw, Poland

Abstract

Petri nets, equivalently presentable as vector addition systems with states, are an established model of concurrency with widespread applications. The reachability problem, where we ask whether from a given initial configuration there exists a sequence of valid execution steps reaching a given final configuration, is the central algorithmic problem for this model. The complexity of the problem has remained, until recently, one of the hardest open questions in verification of concurrent systems. A first upper bound has been provided only in 2015 by Leroux and Schmitz, then refined by the same authors to non-primitive recursive Ackermannian upper bound in 2019. The exponential space lower bound, shown by Lipton already in 1976, remained the only known for over 40 years until a breakthrough non-elementary lower bound by Czerwiński, Lasota, Lazic, Leroux and Mazowiecki in 2019. Finally, a matching Ackermannian lower bound announced this year by Czerwiński and Orlikowski, and independently by Leroux, established the complexity of the problem.

Our primary contribution is an improvement of the former construction, making it conceptually simpler and more direct. On the way we improve the lower bound for vector addition systems with states in fixed dimension (or, equivalently, Petri nets with fixed number of places): while Czerwiński and Orlikowski prove $\mathcal{F}_k$-hardness (hardness for kth level in Grzegorczyk Hierarchy) in dimension $6k$, our 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 Petri nets, reachability problem, vector addition systems

Digital Object Identifier 10.4230/LIPIcs.STACS.2022.46

Related Version Preliminary arXiv report:
Full Version: https://arxiv.org/abs/2105.08551

Funding *Sławomir Lasota*: Supported by the ERC project 'Lipa' within the EU Horizon 2020 research and innovation programme (No. 683080).

1 Introduction

Petri nets [41] are an established model of concurrency with extensive and diverse applications in various fields, including modelling and analysis of hardware [6, 27], software [18, 5, 22] and database [4] systems, as well as chemical [1], biological [2] and business [45, 35] processes (the references on applications are illustrative). The model admits various alternative but essentially equivalent presentations, most notably *vector addition systems* (VAS) [24], and *vector addition systems with states* (VASS) [19, Sec.5], [21]. The central algorithmic question for this model is the *reachability problem* that asks whether from a given initial configuration there exists a sequence of valid execution steps reaching a 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 [44, Section 2.1]. For instance, in terms of VAS, the problem is stated as follows: given a finite set T of integer vectors in d-dimensional space and two d-dimensional vectors $\mathbf{v}$ and $\mathbf{w}$ of nonnegative integers, does there exist a walk from $\mathbf{v}$ to $\mathbf{w}$ such that it stays within the nonnegative orthant, and every step modifies the current

39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022).
Editors: Petra Berenbrink and Benjamin Monmege; Article No. 46; pp. 46:1–46:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

position by adding some vector from T? The model of VASS is a natural extension of VAS with finite control, where $\mathbf{v}$ is additionally equipped with an initial control state, $\mathbf{w}$ with a final one, and each vector in T is additionally equipped with a source-target pair of control states.

We recall, following [44, 9, 10, 11], that importance of the Petri nets reachability problem is widespread, as many diverse problems from formal languages [8], logic [23, 14, 13, 7], concurrent systems [17, 16], process calculi [40], linear algebra [20] and other areas (the references are again illustrative) are known to admit reductions from the VASS reachability problem; for more such problems and a wider discussion, we refer to [44].

Brief history of the problem. The complexity of the Petri nets reachability problem has remained unsettled over the past half century. Concerning the decidability status, after an incomplete proof by Sacerdote and Tenney in 1970s [42], decidability of the problem was established by Mayr [38, 39] in 1981, whose proof was then simplified by Kosaraju [25], and then further refined by Lambert in the 1990s [26]. A different approach, based on inductive invariants, has emerged from a series of papers by Leroux a decade ago [28, 29, 30].

Concerning upper complexity bounds, the first such bound has been shown only in 2015 by Leroux and Schmitz [33], consequently improved to the Ackermannian upper bound [34].

Concerning lower complexity bounds, Lipton's landmark exponential space lower bound from 70ies [36] has remained the state of the art for over 40 years until a breakthrough non-elementary lower bound by Czerwiński, Lasota, Lazic, Leroux and Mazowiecki in 2019 [9] (see also [10]): hardness of the reachability problem for the class Tower of all decision problems that are solvable in time or space bounded by a tower of exponentials whose height is an elementary function of input size. A further refinement of Tower-hardness, in terms of fine-grained complexity classes closed under polynomial-time reductions, has been reported by Czerwiński, Lasota and Orlikowski [11]. Finally, a matching Ackermannian lower bound has been announced recently, independently by Czerwiński and Orlikowski [12], and by Leroux [31] (the two constructions underlying the proofs seem to be significantly different). These results finally close the long standing complexity gap, and yield Ackermann-completness of the Petri nets reachability problem. The techniques used in [12] and [31] substantially differ.

Our contribution. We provide an improvement of the construction of [12]. As our main contribution, we make the construction conceptually simpler and more direct (the idea of improvement is discussed at the end of Section 2, and the central ingredient of our construction is presented in Section 4). Moreover, on the way we improve the parametric lower bound with respect to the dimension of vector addition systems with states (or, equivalently, the number of places of Petri nets[1]). For formulating the result we refer to the complexity classes $\mathcal{F}_\alpha$ corresponding to the Grzegorczyk hierarchy of fast-growing functions [37, 43], indexed by ordinals $\alpha = 0, 1, 2, \ldots, \omega$; for instance, the class $\mathcal{F}_3$ is Tower (class of all decision problems that are solvable in time or space bounded by a tower of exponentials, closed under elementary reductions) and $\mathcal{F}_\omega$ is Ackermann (class of all decision problems that are solvable in time or space bounded by the Ackermann function, closed under primitive-recursive reductions). Results of [12, 31] can be stated in parametric terms as follows: the former shows $\mathcal{F}_k$-hardness of the reachability problem for VASS in dimension $6k$, while the latter one shows $\mathcal{F}_k$-hardness

[1] We remark that a Petri net corresponding to a VASS of dimension d has $d + 3$ places, due to 3 extra places for encoding the control states of VASS [21]. Likewise, a VAS corresponding to a VASS of dimension d has dimension $d + 3$.

for VASS in dimension $4k+5$. Our simplified construction yields a better lower bound: $\mathcal{F}_k$-hardness already in dimension $3k+2$. This improvement is a step towards establishing the tight dimension-parametric complexity of the problem, as the best known upper bound is $\mathcal{F}_k$-membership in dimension $k-4$ [34]. As a next step, an improvement of the construction of [31] to dimension $2k+4$ has been recently reported in [32].

2 The reachability problem

In this section we define the reachability problem and explain our contribution. Following [9, 10, 11, 12, 31], 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.

Counter programs. A *counter program* (or simply a *program*) is a sequence of (line-numbered) 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'	(nondeterministically 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 we use zero tests in the sequel, it is always in view of faithfully simulating them by programs without zero tests.

Convention: In the sequel, unless specified explicitly, counter programs are implicitly assumed to be without zero tests.

▶ **Example 1.** We write $\mathsf{x} \mathrel{+}= m$ (resp. $\mathsf{x} \mathrel{-}= m$) as a shorthand for 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
```

The program repeats the block of 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 increments z. In the sequel we conveniently use **loop** construct instead of explicit **goto** commands. (A dummy command is implicitly added after a **loop** in case it appears at the very end of a program.)

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. By $\mathbb{N}^{\mathsf{C}}$ we denote the set of all valuations of counters. Given an initial valuation of counters, a *run* (or *execution*) of a counter program is a finite sequence of executions of commands, as expected. A run which has successfully finished we call *complete*; otherwise, the run is *partial.* 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**, a program may have various runs from the same initial valuation.

Two programs $\mathcal{P}, \mathcal{Q}$ may be *composed* by concatenating them, written $\mathcal{P}\ \mathcal{Q}$. We silently assume the appropriate re-numbering of lines referred to by **goto** command in $\mathcal{Q}$.

The reachability problem. Given a subset $R \subseteq \mathbb{N}^{\mathsf{C}}$ 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 $\mathsf{X} = \{\mathsf{x}\}$ and/or $R = \{r\}$ are a singleton we write simply "x-zeroing" and/or "from r". For instance, the program from Example 1 has exactly one x-zeroing run from the valuation $\mathsf{x} = 10$, $\mathsf{y} = \mathsf{z} = 0$, where the final values of counters are $\mathsf{x} = 0$, $\mathsf{y} = 10$, $\mathsf{z} = 21$.

By $\mathbf{0}$ we denote the valuation where all counters are 0. Following [9, 10, 11, 12, 31], we investigate the complexity of the following variant of the reachability 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 $\mathbf{0}$?

Since counter programs without zero tests can be seen as presentations of VASS, the above decision problem translates to a variant of the reachability problem for the latter model, where all components of the initial vector are 0, and the specified components of the final vector are required to be 0. This variant polynomially reduces to the classical one where all components of the final vector are fully specified (say, required to be 0), and the reduction preserves dimension. According to the encoding of VASS as Petri nets, the problem translates to the *submarking reachability* problem for the latter model, where all places (except for those encoding the control states) are initially empty, and the specified places are required to be finally empty. Finally, the submarking reachability problem is polynomially equivalent to a variant where the final content of all places is fully specified.

Fast-growing hierarchy. For a positive integer k, let $\mathbb{N}_k = \{k, 2k, 3k, \ldots\} \subseteq \mathbb{N}$ denote positive multiplicities of k. We define the complexity classes $\mathcal{F}_i$ corresponding to the ith level in the Grzegorczyk Hierarchy [43, 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:

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

In particular, $\mathbf{A}_2(n) = 2^n$ and $\mathbf{A}_i(1) = 2$ for all $i \in \mathbb{N}_1$. Using functions $\mathbf{A}_i$, 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 $m \in \mathbb{N}_1$:

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

Intuitively speaking, the class $\mathcal{F}_i$ contains all 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, $\mathcal{F}_3 =$ Tower (problems solvable in a tower of exponentials of time or space, whose height is an elementary function of input size). The classes $\mathcal{F}_k$ are robust with respect to the choice of fast-growing function hierarchy (see [43, Sect.4.1]). For $k \geq 3$, instead of deterministic time, one could equivalently take nondeterministic time, or space.

Dimension-parametric lower bound. As the main result we prove $\mathcal{F}_k$-hardness for programs with the fixed number $3k+2$ of counters:

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

The proof is in Section 6. The result can be compared to $\mathcal{F}_k$-hardness shown in [12] for $6k$ counters, and in [31] for $4k+5$ counters. Like the cited results, Theorem 2 implies Ackermann-hardness for unrestricted number of counters which, together with Ackermann upper bound of [34], yields Ackermann-completness of the reachability problem.

Idea of simplification. Czerwiński and Orlikowski [12] use the *ratio technique* introduced previously in [9]. 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 initial values of c and d may be arbitrary, in a nondeterministic way, as long as they satisfy the latter equality in (1); they are hence unbounded. Under these assumptions, the ratio technique of [9] allows one to correctly simulate unboundedly many zero tests (in fact, the number of simulated zero tests corresponds to the initial value of c which may be arbitrarily large) on counters bounded by B, at the price of using some auxiliary counters.

As our technical contribution, we improve and simplify the ratio technique. The core idea underlying our simplification 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 which may be arbitrarily large), without any auxiliary counters. This novel approach is presented in detail in Section 4.

3 Multipliers

Following the lines of [9, 10, 12], we rely on a concept of *multiplier*.

Sets computed by programs. Consider a program $\mathcal{P}$ with counters C, a set of counters $\mathsf{X} \subseteq \mathsf{C}$ and $R \subseteq \mathbb{N}^{\mathsf{C}}$. We define the set X-*computed by $\mathcal{P}$ from R* as the set of all valuations of counters at the end of all X-zeroing (and hence forcedly complete) runs of $\mathcal{P}$ from R. Formally, denoting by $\textsc{Runs}_{\mathcal{P}}(R, \mathsf{X})$ the set of all X-zeroing runs of $\mathcal{P}$ from R, and by $\textsc{fin}(\pi)$ the final counter valuation of a complete run π of $\mathcal{P}$, the set X-computed by $\mathcal{P}$ from R is

$$\textsc{Comp}_{\mathcal{P}}(R, \mathsf{X}) \;=\; \{\textsc{fin}(\pi) \mid \pi \in \textsc{Runs}_{\mathcal{P}}(R, \mathsf{X})\}.$$

We omit X when it is irrelevant. As before, when $\mathsf{X} = \{\mathsf{x}\}$ and/or $R = \{r\}$ are a singleton we write simply 'x-computed' and/or 'from r'.

▶ **Example 3.** The program in Example 1 above, x-computes from the set of all valuations satisfying $\mathsf{y} = \mathsf{z} = 0$ (no constraint for x), the set of all valuations satisfying $\mathsf{x} = 0$ (trivially) and $\mathsf{z} = 2\mathsf{y} + 1$.

Likewise, for a fixed integer $m \in \mathbb{N}$ and a program $\mathcal{P}$ with zero tests, we define the set X-computed by $\mathcal{P}$ from R *using m zero tests*, by restricting the above definition to runs $\pi \in \textsc{Runs}_{\mathcal{P}}(R, \mathsf{X})$ that do exactly m zero tests. This finer variant of the definition will be used in the next section.

Multipliers. 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 $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}) \subseteq \mathbb{N}^{\mathsf{C}}$, called informally the *ratio of B*, consisting of all valuations that satisfy the three conditions (1) and assign 0 to all other counters $\mathsf{x} \in \mathsf{C} \setminus \{\mathsf{b}, \mathsf{c}, \mathsf{d}\}$.

▶ **Definition 4.** *A program $\mathcal{M}$ (with no zero tests) with counters C that z-computes from the zero valuation $\mathbf{0}$ the set $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$, for some four of its counters $\mathsf{z}, \mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathsf{C}$, we call B-**multiplier**. In formula: $\textsc{Comp}_{\mathcal{M}}(\mathbf{0}, \mathsf{z}) = \textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$.*

▶ **Example 5.** As a simple example, for every fixed $B \in \mathbb{N}_4$, the following program is a B-multiplier of size $\mathcal{O}(B)$ (several commands are written in one line to save space). Counter z is not used at all.

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

```
b += B    d += B    c += 1
loop
    d += B    c += 1
```

Directly from the definition we derive the following fundamental property of multipliers, to be used in the proofs in Sections 5 and 6:

▷ Claim 6. Let $\mathcal{M}$ be a B-multiplier with counters C as in Definition 4, let $\mathcal{P}$ be a counter program with counters $\mathsf{C} \setminus \{\mathsf{z}\}$, and let $\mathsf{Y} \subseteq \mathsf{C}$. Then the set Y-computed by $\mathcal{P}$ from $\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$ is equal to the set $(\{\mathsf{z}\} \cup \mathsf{Y})$-computed by the composed program $\mathcal{M}\ \mathcal{P}$ from $\mathbf{0}$:

$$\textsc{Comp}_{\mathcal{P}}(\textsc{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}), \mathsf{Y}) \;=\; \textsc{Comp}_{\mathcal{M}\,\mathcal{P}}(\mathbf{0}, \{\mathsf{z}\} \cup \mathsf{Y}).$$

Proof. The claim is a special case of the following general composition rule: for two programs $\mathcal{P}$ and $\mathcal{Q}$, if $\textsc{Comp}_{\mathcal{P}}(A, \mathsf{X}) = B$ and $\mathcal{Q}$ does not use counters X, then $\textsc{Comp}_{\mathcal{P}\,\mathcal{Q}}(A, \mathsf{X} \cup \mathsf{Y}) = \textsc{Comp}_{\mathcal{Q}}(B, \mathsf{Y})$. Indeed, under the above assumptions $(\mathsf{X} \cup \mathsf{Y})$-zeroing runs of $\mathcal{P}\ \mathcal{Q}$ from A are in mutual correspondence with Y-zeroing runs of $\mathcal{Q}$ from B. ◁

Computing multipliers. For technical convenience 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 functions $\mathbf{A}_i$ (cf. Claim 7 below):

$$\mathbf{F}_1(n) = 2n, \qquad \mathbf{F}_{i+1} = \widetilde{\mathbf{F}_i} \quad \text{where} \quad \widetilde{F}(n) = \underbrace{F \circ F \circ \ldots \circ F}_{n/4}(4). \tag{2}$$

By induction on i one easily shows that $\mathbf{F}_i$ is a linear re-scaling of $\mathbf{A}_i$:

▷ Claim 7. $\mathbf{F}_i(4 \cdot n) = 4 \cdot \mathbf{A}_i(n)$, for $i, n \in \mathbb{N}_1$.

Proof. As $\mathbf{F}_1(n) = 2n$ and $\mathbf{A}_1(n) = 2n$, the claim holds for $i = 1$. Assuming the claim for $i \in \mathbb{N}_1$, by n-fold application thereof we derive the required equality for $i + 1$:

$$\mathbf{F}_{i+1}(4 \cdot n) \;=\; \underbrace{\mathbf{F}_i \circ \ldots \circ \mathbf{F}_i}_{n}(4) \;=\; 4 \cdot \underbrace{\mathbf{A}_i \circ \ldots \circ \mathbf{A}_i}_{n}(1) \;=\; 4 \cdot \mathbf{A}_{i+1}(n). \qquad ◁$$

As a technical core of the proof of Theorem 2, combining our simplification with the lines of [12], we provide an effective construction of B-multipliers with $3k+2$ counters, where $B = \mathbf{F}_k(n)$, of size polynomial in k and n.

▶ **Theorem 8.** *Given $k \in \mathbb{N}_1$ 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.*

The proof is in Section 5.

4 Bounded number of zero tests

In this section we provide a novel construction that enables simulating a bounded number m of zero tests (cf. Lemma 12) at the cost of introducing additional 3 counters initialised to the ratio of $B = 2(m+1)$. This construction is a core ingredient of the proofs of Theorems 2 and 8.

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.

Maximal iteration. In the sequel we intensively use loops of the following form that, intuitively, flush the value from counter f to e, decreasing simultaneously counter d (and possibly execute some further commands):

```
1: loop
2:     f −= 1    e += 1    d −= 1    ...
```
(3)

Assuming $\overline{\mathsf{d}} \geq \overline{\mathsf{f}}$, we observe that the amount $\overline{\mathsf{d}} - \underline{\mathsf{d}}$ by which d is decreased as an effect of execution (we use the word *execution* as a synonym to *complete run*) of the above loop may be any value between 0 and $\overline{\mathsf{f}}$. Furthermore, assuming $\overline{\mathsf{d}} \geq \overline{\mathsf{e}} + \overline{\mathsf{f}}$, the equality $\overline{\mathsf{d}} - \underline{\mathsf{d}} = \overline{\mathsf{e}} + \overline{\mathsf{f}}$ holds if and only if

$$\overline{\mathsf{e}} = 0 = \underline{\mathsf{f}}. \tag{4}$$

This simple observation will play a crucial role in the sequel, and deserves a definition:

▶ **Definition 9.** *Whenever an execution of a loop of the form* (3) *satisfies the two equalities* (4) *we call this execution* **maximally iterated**.

The construction. Let $\mathcal{P}$ be a counter program with counters C, and assume that $\mathcal{P}$ uses zero tests only on two its 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 fresh 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 its initial valuation of counters belongs to $\textsc{Ratio}(2(m+1), \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}^*)$ 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 (cf. Lemma 12).

The transformation proceeds in three steps. First, we accompany every increment (decrement) on x with a decrement (increment) of c, and likewise we do for y:

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

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

In the resulting program $\overline{\mathcal{P}}$ counters x, y are, intuitively speaking, put on a shared 'budget' c. Assuming x and y initially 0, 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.

As the second step, we replace in $\overline{\mathcal{P}}$ every **zero?** x command by the following macro:

<u>Zero? x:</u>

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

Likewise we replace every **zero?** y command by an analogous macro Zero? y obtained from Zero? x by swapping x and y. This yields the program $\widehat{\mathcal{P}}$ without zero tests. We note that each of the two Zero? macros preserves the sum $s = \mathsf{c}+\mathsf{x}+\mathsf{y}$, and decrements the counter b by 2. Furthermore, each of the two Zero? macros decrements d by at most $2s$ (cf. Claim 10 below), and hence the macros preserve the inequality $d \geq \mathsf{b} \cdot s$ (recall that $\mathsf{d} = \mathsf{b} \cdot s$ holds initially).

As the final step we adjoint at the end of $\widehat{\mathcal{P}}$ the following program Set-c-to-zero, thus obtaining the transformed program $\mathcal{P}^*$:

<u>Set-c-to-zero:</u>

```
loop
    c −= 1    d −= 2
Zero? c
```

The macro Zero? c is obtained from Zero? x by swapping x and c. We note that an execution of Set-c-to-zero may decrease the sum $\mathsf{c}+\mathsf{x}+\mathsf{y}$ (but Zero? c preserves it).

Correctness. Recall that $\widehat{\mathcal{P}}$ preserves the sum $\mathsf{c}+\mathsf{x}+\mathsf{y}$; we denote by s the value of this sum. An execution of Zero? x is called *maximally iterated* if all four loops are so. Observe that every such execution is forcedly *correct*, i.e. satisfies:

$$\overline{\mathsf{x}} = \underline{\mathsf{x}} = 0, \qquad \overline{\mathsf{y}} = \underline{\mathsf{y}}, \qquad \overline{\mathsf{c}} = \underline{\mathsf{c}}. \tag{5}$$

(Likewise in case of Zero? y and Zero? c.) 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 10. Consider an execution of Zero? x (resp. Zero? y) 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? x, 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

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

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

In consequence, as b is decreased by 2, if the invariant $d = b \cdot s$ is preserved by an execution of ZERO? x (resp. ZERO? y) then the zero test is forcedly correct. Furthermore notice that once the invariant is violated, i.e., $d > b \cdot s$, due to the first part of Claim 10 the invariant can not be recovered later. These observations lead to the correctness claim stated in Lemma 12.

In the proof of Lemma 12 we will also need the following corollary of Claim 10, where s denotes, as before, the sum $c + x + y$ at the start of SET-c-TO-ZERO(c):

▷ Claim 11. Consider an execution of SET-c-TO-ZERO(c), assuming $\bar{d} \geq 2s$. Then $0 \leq \bar{d} - \underline{d} \leq 2s$. Furthermore, the equality $\bar{d} - \underline{d} = 2s$ holds if and only if the ZERO? c macro is maximally iterated.

Proof. Consider an execution of SET-c-TO-ZERO(c) and denote by $\dot{s}$ the value of $c + x + y$ just before entering ZERO? c. Thus d decrease by $2(s - \dot{s})$ before entering ZERO? c. Moreover, due to Claim 10, the macro ZERO? c decreases d by at most $2\dot{s}$, and furthermore the macro decreases d by exactly $2\dot{s}$ if and only if it is maximally iterated. These observations imply the claim. ◁

Recall that $C^* = C \cup \{b, c, d\}$. We define the C^*-*extension* of a counter valuation $v \in \mathbb{N}^C$ as the extension of v where b, c and d are all set to 0. The C^*-extension of a set $R \subseteq \mathbb{N}^C$ is defined as the set of C^*-extensions of all valuations in R.

▶ **Lemma 12.** *The following sets are equal (as subsets of $\mathbb{N}^{C^*}$):*
- *the C^*-extension of the set computed by $\mathcal{P}$ from* **0** *using m zero tests.*
- *the set d-computed by $\mathcal{P}^*$ from* RATIO$(2(m+1), b, c, d, C^*)$.

Proof. For the inclusion of the former set in the latter, we show that for each complete run π of $\mathcal{P}$ from **0** that does m zero tests on x, y, there is a corresponding d-zeroing run of $\mathcal{P}^*$ from RATIO$(2(m+1), b, c, d, C^*)$, for any initial value $\bar{c}$ at least as large as the maximal value of the sum $x + y$ along π. The run iterates maximally ZERO? x and ZERO? y macros, decrements c to 0 in line 2 in SET-c-TO-ZERO(c), and then iterates maximally ZERO? c. Thus the final counter valuation of the run is the C^*-extension of the final counter valuation of π.

For the converse direction, consider a d-zeroing run π of $\mathcal{P}^*$ from RATIO$(2(m+1), b, c, d, C^*)$. The initial counter valuation satisfies the equalities $b = 2(m+1)$ and $d = 2(m+1) \cdot s$. Each execution of ZERO? x or ZERO? y or SET-c-TO-ZERO(c) decreases b by 2, and d by at most $2s$ (by the first part of Claims 10 and Claim 11). Therefore, since b and d are not affected elsewhere and $\underline{d} = 0$ finally, we deduce:

▷ Claim 13. Each execution of ZERO? x, ZERO? y or ZERO? c in π decreases d by *exactly* $2s$.

▷ Claim 14. There are exactly m executions of ZERO? x or ZERO? y in π.

▷ Claim 15. Finally, $\underline{b} = 0$.

By Claim 13 and the second part of Claim 10 we derive:

▷ Claim 16. Each execution of ZERO? x in π is correct, i.e. satisfies the equalities (5). Likewise for ZERO? y.

Analogously, by Claim 13 and the second part of Claim 11 we derive:

▷ Claim 17. Finally, $\underline{\mathsf{c}} = 0$.

Due to Claims 14 and 16, once we project away from π the counters $\mathsf{b}, \mathsf{c}, \mathsf{d}$, we obtain a complete run of $\mathcal{P}$ from $\mathbf{0}$ that does exactly m zero tests, as required. Finally, due to Claims 15 and 17, the C^*-extension of the final counter valuation of the obtained run is exactly the final counter valuation of π. ◀

5 Computing a large multiplier (Proof of Theorem 8)

The proof proceeds by combining the concept of amplifier lifting of [12] with the program transformation of Section 4.

Amplifiers. Let $F : \mathbb{N}_4 \to \mathbb{N}_4$ be a monotone function satisfying $F(n) \geq n$ for $n \in \mathbb{N}_4$. Informally speaking, an F-*amplifier* is a program without zero tests that computes the ratio of $F(B)$ from the ratio of B, for every $B \in \mathbb{N}_4$.

▶ **Definition 18.** *Consider a program $\mathcal{P}$ with counters C without zero tests and distinguished 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 called* 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:         $\mathsf{c} \mathrel{-}= 1 \quad \mathsf{c}' \mathrel{+}= 1 \quad \mathsf{d} \mathrel{-}= 1 \quad \mathsf{d}' \mathrel{+}= \ell$
4:     **loop**
5:         $\mathsf{c}' \mathrel{-}= 1 \quad \mathsf{c} \mathrel{+}= 1 \quad \mathsf{d} \mathrel{-}= 1 \quad \mathsf{d}' \mathrel{+}= \ell$
6:     $\mathsf{b} \mathrel{-}= 2 \quad \mathsf{b}' \mathrel{+}= 2\ell$
7: **loop**
8:     $\mathsf{c} \mathrel{-}= 1 \quad \mathsf{c}' \mathrel{+}= 1 \quad \mathsf{d} \mathrel{-}= 2 \quad \mathsf{d}' \mathrel{+}= 2\ell$
9: $\mathsf{b} \mathrel{-}= 2 \quad \mathsf{b}' \mathrel{+}= 2\ell$

▷ Claim 19. 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 $(0, 0, 0, \ell \cdot B, c, \ell \cdot d)$. Indeed, as $\mathsf{d} = 0$ finally, each of the two inner loops in lines 2–5, as well as the last loop in lines 7–8, is forcedly maximally iterated. ◁

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 (2) of functions $\mathbf{F}_i$; in particular $\mathbf{F}_1 = L_2$. 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 $\mathcal{P}$ to a program $\widetilde{\mathcal{P}}$, also without zero tests, such that assuming that $\mathcal{P}$ is an F-amplifier for some function $F : \mathbb{N}_4 \to \mathbb{N}_4$, the program $\widetilde{\mathcal{P}}$ 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 $\mathsf{C}^* = \mathsf{C} \cup \{\mathsf{b}, \mathsf{c}, \mathsf{d}\}$. We let input counters of $\widetilde{\mathcal{P}}$ be $\mathsf{b}, \mathsf{c}, \mathsf{d}$, and its output counters be $\mathsf{b}_2, \mathsf{c}_2, \mathsf{d}_2$.

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 4-multiplier $\mathcal{M} = \mathcal{M}_4(\mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1)$ of Example 5, both without zero tests. For defining the program $\widetilde{\mathcal{P}}$ we apply the transformation of Section 4 (with counters d_1 and d_2 in place of x and y) to the following program $\mathcal{Q}$ built using $\mathcal{P}$, $\mathcal{L}$ and $\mathcal{M}$:

Program $\mathcal{Q}$:

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

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

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

Formally, $\widetilde{\mathcal{P}} = \mathcal{Q}^*$. Intuitively speaking, the program $\mathcal{Q}$ directly implements the computation of $\widetilde{F}$ according to the definition: with $2\ell + 1$ zero tests it computes, from $\mathbf{0}$, the ratio of $F^{\ell+1}(4)$. Note that counters of $\mathcal{Q}$ are C while counters of $\widetilde{\mathcal{P}}$ are C^*. Lemma 20 states the crucial amplifier-lifting property of the program transformation $\mathcal{P} \mapsto \widetilde{\mathcal{P}}$.

▶ **Lemma 20.** *If $\mathcal{P}$ is an F-amplifier, then $\widetilde{\mathcal{P}}$ is an $\widetilde{F}$-amplifier.*

Proof. Let $\mathcal{P}$ be an F-amplifier. Thus for every $B \in \mathbb{N}_4$, $\text{COMP}_{\mathcal{P}}(\text{RATIO}(B, \mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1, \mathsf{C}), \mathsf{d}_1) = \text{RATIO}(F(B), \mathsf{b}_2, \mathsf{c}_2, \mathsf{d}_2, \mathsf{C})$. The program $\mathcal{L}$, being an identity-amplifier, d_2-computes from $\text{RATIO}(B, \mathsf{b}_2, \mathsf{c}_2, \mathsf{d}_2, \mathsf{C})$ the set $\text{RATIO}(B, \mathsf{b}_1, \mathsf{c}_1, \mathsf{d}_1, \mathsf{C})$. Let $B = 4(\ell + 1) \in \mathbb{N}_4$ for an arbitrary $\ell \in \mathbb{N}$. As $\mathcal{P}$ is an F-amplifier and $\mathcal{L}$ is an identity-amplifier, we deduce:

▷ Claim 21. $\mathcal{Q}$ computes from $\mathbf{0}$ using $2\ell + 1$ zero tests the set $\text{RATIO}(F^{\ell+1}(4), \mathsf{b}_2, \mathsf{c}_2, \mathsf{d}_2, \mathsf{C})$.

As $\widetilde{\mathcal{P}} = \mathcal{Q}^*$, by Lemma 12 we deduce:

▷ Claim 22. $\text{COMP}_{\widetilde{\mathcal{P}}}(\text{RATIO}(4(\ell + 1), \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}^*), \mathsf{d}) = \text{RATIO}(F^{\ell+1}(4), \mathsf{b}_2, \mathsf{c}_2, \mathsf{d}_2, \mathsf{C})$.

As $B \in \mathbb{N}_4$ was chosen arbitrarily and $\widetilde{F}(B) = F^{\ell+1}(4)$, the last claim says that $\widetilde{\mathcal{P}}$ is an $\widetilde{F}$-amplifier. ◀

▶ Remark 23. The program $\mathcal{P}$ appears twice in the body of $\widetilde{\mathcal{P}}$. This doubling can be easily avoided by re-structuring the loop using explicit **goto** commands. In this way, the size of $\widetilde{\mathcal{P}}$ becomes larger than the size of $\mathcal{P}$ only by a constant.

Proof of Theorem 8. We rely on Lemma 20. 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 transformation $\mathcal{P} \mapsto \widetilde{\mathcal{P}}$ described above, starting from the $\mathbf{F}_1$-amplifier

$\mathcal{L}_2$ of Claim 19. The construction is linear in k due to Remark 23. Let $\mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathsf{C}$ be input counters of $\mathcal{A}_k$. Relying on Claim 6 in Section 3, the $\mathbf{F}_k(n)$-multiplier is obtained by pre-composing $\mathcal{A}_k$ with an n-multiplier (e.g. $\mathcal{M}_n(\mathsf{b}, \mathsf{c}, \mathsf{d})$ from Section 2) that outputs the set $\textsc{Ratio}(n, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$. The whole construction is thus linear in n.

Finally we observe that the counter b is bounded by n and hence can be eliminated: we encode its values in control locations, by cloning the program into $n+1$ copies, where ith (for $i = 0, \ldots, n$) copy corresponds to the value $\mathsf{b} = i$. The resulting program has $3k+2$ counters. ◀

6 Hardness of the reachability problem (Proof of Theorem 2)

Relying on Lemma 12 and Theorem 8, we prove in this section Theorem 2. Fix $k \geq 3$. 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 from $\mathbf{0}$ that does at most $(\mathbf{F}_k(n) - 1)/2$ zero tests?

▷ Claim 24. The above problem is $\mathcal{F}_k$-hard.

Proof. Indeed, the standard $\mathcal{F}_k$-hard halting problem (does a program $\mathcal{P}$ with *arbitrarily many* zero-tested counters $\mathsf{x}_1, \ldots, \mathsf{x}_\ell$ have a complete run that does at most $(\mathbf{F}_k(n) - 1)/2$ *steps*?) reduces polynomially to the above one using the standard simulation of arbitrarily many zero-tested counters by 2 such counters $\mathsf{y}_1, \mathsf{y}_2$. The simulation stores the values of all counters $\mathsf{x}_1, \ldots, \mathsf{x}_\ell$ on one of $\mathsf{y}_1, \mathsf{y}_2$ (e.g., using Gödel encoding), and the simulation of each command involves flushing the value of that counter to the other, followed by the zero test. Thus a bound on time of computation is translated 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 from $\mathbf{0}$ that does at most $m = (\mathbf{F}_k(n) - 1)/2$ zero tests if and only if $\mathcal{P}'$ has a $\{\mathsf{d}, \mathsf{z}\}$-zeroing run from $\mathbf{0}$ (for some $\mathsf{d}, \mathsf{z} \in \mathsf{C}$).

First, we post-compose $\mathcal{P}$ with a simple program $\mathcal{L}$ that first decrements x nondeterministically many times, and then zero tests it nondeterministically many times:

```
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. We will apply the transformation of Section 4 to the composed program $\mathcal{P}\ \mathcal{L}$. Let $\mathsf{b}, \mathsf{c}, \mathsf{d}$ be the three counters added in the course of the transformation.

Second, using Theorem 8 we compute a $2(m+1)$-multiplier $\mathcal{M}$ (recall that $2(m+1) = \mathbf{F}_k(n)$) with $3k+2$ counters C that z-computes from $\mathbf{0}$ the set $\textsc{Ratio}(2(m+1), \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$, for some counter z different than x, y. Thus $\mathsf{z}, \mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathsf{C}$.

Finally, we define $\mathcal{P}'$ as a composition of $\mathcal{M}$ with the transformed program $(\mathcal{P}\ \mathcal{L})^*$, and get the required equivalence:

▷ Claim 25. The following conditions are equivalent:

- $\mathcal{P}$ has a complete run from $\mathbf{0}$ that does at most m zero tests;
- $\mathcal{P}\ \mathcal{L}$ has a complete run from $\mathbf{0}$ that does exactly m zero tests;
- $(\mathcal{P}\ \mathcal{L})^*$ has a d-zeroing run from $\textsc{Ratio}(2(m+1), \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$;
- $\mathcal{P}' = \mathcal{M}\ (\mathcal{P}\ \mathcal{L})^*$ has a $\{\mathsf{z}, \mathsf{d}\}$-zeroing run from $\mathbf{0}$.

The second and the third point are equivalent due to Lemma 12, while the equivalence of the third and the last point follows by Claim 6 in Section 3.

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. The latter equivalence in Claim 25 remains true, as z-zeroing runs of $\mathcal{M}$ from $\mathbf{0}$ are necessarily C'-zeroing too, by the definition of multipliers.

This completes the proof of Theorem 2.

7 Final remarks

Primarily, we propose a conceptual simplification of the Ackermann-hardness construction of [12].

As a secondary achievement, we improve the dimension-parametric lower bound for the VASS (Petri nets) reachability problem: compared to $\mathcal{F}_k$-hardness in dimension $6k$ [12] and $4k+5$ [31], respectively, we obtain $\mathcal{F}_k$-hardness already in dimension $3k+2$. (We believe that by combining with the insights of [12] one can further optimise our construction and lower the dimension by a small constant.) The dimension $4k+5$ of [31] has been recently further improved to $2k+4$ [32], thus beating ours.

The best known upper bound places the VASS reachability problem in dimension $k-4$ is in $\mathcal{F}_k$ [34]. Establishing exact parametric complexity of the problem, i.e., closing the gap between dimensions $k-4$ and $2k+4$, arises therefore as an intriguing open problem.

Finally we remind that except for dimension 1 and 2, where the reachability problem seems to be well understood [3, 15], we know no additional complexity bounds for small fixed dimensions k except for the lower bound derived from dimension 2, and the generic $\mathcal{F}_{k+4}$ upper bound of [34].

References

1 David Angeli, Patrick De Leenheer, and Eduardo D. Sontag. Persistence results for chemical reaction networks with time-dependent kinetics and no global conservation laws. *SIAM Journal of Applied Mathematics*, 71(1):128–146, 2011.

2 Paolo Baldan, Nicoletta Cocco, Andrea Marin, and Marta Simeoni. Petri nets for modelling metabolic pathways: a survey. *Natural Computing*, 9(4):955–989, 2010. doi:10.1007/s11047-010-9180-6.

3 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSPACE-complete. In *Proc. LICS*, pages 32–43, 2015.

4 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403.

5 Ahmed Bouajjani and Michael Emmi. Analysis of recursively parallel programs. *ACM Trans. Program. Lang. Syst.*, 35(3):10:1–10:49, 2013. doi:10.1145/2518188.

6 Frank P. Burns, Albert Koelmans, and Alexandre Yakovlev. WCET analysis of superscalar processors using simulation with coloured Petri nets. *Real-Time Systems*, 18(2/3):275–288, 2000. doi:10.1023/A:1008101416758.

7 Thomas Colcombet and Amaldev Manuel. Generalized data automata and fixpoint logic. In *FSTTCS*, volume 29 of *LIPIcs*, pages 267–278. Schloss Dagstuhl, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.267.

8 Stefano Crespi-Reghizzi and Dino Mandrioli. Petri nets and Szilard languages. *Information and Control*, 33(2):177–192, 1977. doi:10.1016/S0019-9958(77)90558-7.

9 Wojciech Czerwiński, Sławomir 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.

10 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. *J. ACM*, 68(1):7:1–7:28, 2021. doi:10.1145/3422822.

11 Wojciech Czerwinski, Slawomir Lasota, and Łukasz Orlikowski. Improved lower bounds for reachability in vector addition systems. In *Proc. ICALP*, volume 198 of *LIPIcs*, pages 128:1–128:15, 2021.

12 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *Proc. FOCS 2021*, 2021. To appear.

13 Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. Ordered navigation on multi-attributed data words. In *Proc. CONCUR*, volume 8704 of *LNCS*, pages 497–511. Springer, 2014. doi:10.1007/978-3-662-44584-6_34.

14 Stéphane Demri, Diego Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:1)2016.

15 Matthias Englert, Ranko Lazić, and Patrick Totzke. Reachability in two-dimensional unary vector addition systems with states is NL-complete. In *Proc. LICS*, pages 477–484. ACM, 2016. doi:10.1145/2933575.2933577.

16 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Inf.*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

17 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, 2012. doi:10.1145/2160910.2160915.

18 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.

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

20 Piotr Hofman and Sławomir Lasota. Linear equations with ordered data. In *Proc. CONCUR*, volume 118 of *LIPIcs*, pages 24:1–24:17. Schloss Dagstuhl, 2018. doi:10.4230/LIPIcs.CONCUR.2018.24.

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

22 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. A widening approach to multithreaded program verification. *ACM Trans. Program. Lang. Syst.*, 36(4):14:1–14:29, 2014. doi:10.1145/2629608.

23 Max I. Kanovich. Petri nets, Horn programs, linear logic and vector games. *Ann. Pure Appl. Logic*, 75(1–2):107–135, 1995. doi:10.1016/0168-0072(94)00060-G.

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

25 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proc. STOC*, pages 267–281. ACM, 1982. doi:10.1145/800070.802201.

26 Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

27 Hélène Leroux, David Andreu, and Karen Godary-Dejean. Handling exceptions in Petri net-based digital architecture: From formalism to implementation on FPGAs. *IEEE Trans. Industrial Informatics*, 11(4):897–906, 2015.

28 Jérôme Leroux. The general vector addition system reachability problem by Presburger inductive invariants. *Logical Methods in Computer Science*, 6(3), 2010. doi:10.2168/LMCS-6(3:22)2010.

29 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL*, pages 307–316. ACM, 2011. doi:10.1145/1926385.1926421.

30 Jérôme Leroux. Vector addition systems reachability problem (A simpler solution). In *Turing-100*, volume 10 of *EPiC Series in Computing*, pages 214–228. EasyChair, 2012. URL: http://www.easychair.org/publications/paper/106497, doi:10.29007/bnx2.

31 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *Proc. FOCS 2021*, 2021. To appear.

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

33 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *Proc. LICS*, pages 56–67, 2015.

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

35 Yuliang Li, Alin Deutsch, and Victor Vianu. VERIFAS: A practical verifier for artifact systems. *PVLDB*, 11(3):283–296, 2017. doi:10.14778/3157794.3157798.

36 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.

37 Martin H. Löb and Stanley S. Wainer. Hierarchies of number-theoretic functions. I. *Archiv für mathematische Logik und Grundlagenforschung*, 13(1–2):39–51, 1970. doi:10.1007/BF01967649.

38 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *STOC*, pages 238–246. ACM, 1981. doi:10.1145/800076.802477.

39 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984. doi:10.1137/0213029.

40 Roland Meyer. A theory of structural stationarity in the *pi*-calculus. *Acta Inf.*, 46(2):87–137, 2009. doi:10.1007/s00236-009-0091-x.

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

42 George S. Sacerdote and Richard L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). In *STOC*, pages 61–76. ACM, 1977. doi:10.1145/800105.803396.

43 Sylvain Schmitz. Complexity hierarchies beyond elementary. *TOCT*, 8(1):3:1–3:36, 2016.

44 Sylvain Schmitz. The complexity of reachability in vector addition systems. *SIGLOG News*, 3(1):4–21, 2016.

45 Wil M. P. van der Aalst. Business process management as the “killer app” for Petri nets. *Software and System Modeling*, 14(2):685–691, 2015. doi:10.1007/s10270-014-0424-2.