

Improved Ackermannian lower bound for the Petri nets reachability problem

ŚŁAWOMIR LASOTA*, University of Warsaw, Poland

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 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 k th level in Grzegorzczuk Hierarchy) in dimension $6k$, and Leroux in dimension $4k + 5$, our simplified construction yields \mathcal{F}_k -hardness already in dimension $3k + 2$.

CCS Concepts: • **Theory of computation** → **Concurrency; Logic and verification; Automata extensions.**

Additional Key Words and Phrases: Petri nets, vector addition systems, reachability problem, vector addition systems with states, counter programs.

ACM Reference Format:

Śławomir Lasota. 2018. Improved Ackermannian lower bound for the Petri nets reachability problem. *Proc. ACM Program. Lang.* 1, CONF, Article 1 (January 2018), 13 pages.

1 INTRODUCTION

Petri nets [Petri 1962] are an established model of concurrency with extensive and diverse applications in various fields, including modelling and analysis of hardware [Burns et al. 2000; Leroux et al. 2015], software [Bouajjani and Emmi 2013; German and Sistla 1992; Kaiser et al. 2014] and database [Bojańczyk et al. 2011] systems, as well as chemical [Angeli et al. 2011], biological [Baldan et al. 2010] and business [Li et al. 2017; van der Aalst 2015] processes (the references on applications are illustrative). The model admits various alternative but essentially equivalent presentations, most notably *vector addition systems* (VAS) [Karp and Miller 1969], and *vector addition systems with states* (VASS) [Greibach 1978, Sec.5], [Hopcroft and Pansiot 1979]. 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 [Schmitz 2016b, 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

*Supported by the NCN grant 2017/27/B/ST6/02093.

Author's address: Śławomir Lasota, University of Warsaw, Poland, Warsaw, sl@mimuw.edu.pl.

2018. 2475-1421/2018/1-ART1 \$15.00
<https://doi.org/>

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 its every step modifies the current 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 [Czerwiński et al. 2019; Czerwinski et al. 2021b,a; Schmitz 2016b], that importance of the Petri nets reachability problem is widespread, as many diverse problems from formal languages [Crespi-Reghizzi and Mandrioli 1977], logic [Colcombet and Manuel 2014; Decker et al. 2014; Demri et al. 2016; Kanovich 1995], concurrent systems [Esparza et al. 2017; Ganty and Majumdar 2012], process calculi [Meyer 2009], linear algebra [Hofman and Lasota 2018] 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 [Schmitz 2016b].

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 70ies [Sacerdote and Tenney 1977], decidability of the problem was established by Mayr [Mayr 1981, 1984] in 1981, whose proof was then simplified by Kosaraju [Kosaraju 1982], and then further refined by Lambert in the 1990s [Lambert 1992]. A different approach, based on inductive invariants, has emerged from a series of papers by Leroux a decade ago [Leroux 2010, 2011, 2012].

Concerning upper complexity bounds, the first such bound has been shown only in 2015 by Leroux and Schmitz [Leroux and Schmitz 2015], consequently improved to the Ackermannian upper bound [Leroux and Schmitz 2019].

Concerning lower complexity bounds, Lipton's landmark exponential space lower bound from 70ies [Lipton 1976] 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 [Czerwiński et al. 2019] (see also [Czerwinski et al. 2021b]): 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 [Czerwinski et al. 2021a]. Finally, a matching Ackermannian lower bound has been announced recently, independently by Czerwiński and Orlikowski [Czerwiński 2021], and by Leroux [Leroux 2021] (the two constructions underlying the proofs seem to be significantly different). These results finally close the long standing complexity gap, and yield ACKERMANN-completeness of the Petri nets reachability problem.

Our contribution. We provide an improvement of the construction of [Czerwiński and Orlikowski 2021], making it conceptually simpler and more direct. 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 nets¹). For formulating the result we refer to the complexity classes \mathcal{F}_α corresponding to the Grzegorzczuk hierarchy of fast-growing functions [Löb and Wainer 1970; Schmitz 2016a], indexed by ordinals $\alpha = 0, 1, 2, \dots, \omega$; 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}_ω is ACKERMANN (class of all decision problems that are solvable in time or space bounded by the Ackermann function, closed under primitive-recursive

¹We 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 [Hopcroft and Pansiot 1979]. Likewise, a VAS corresponding to a VASS of dimension d has dimension $d + 3$.

reductions). Results of [Czerwiński and Orlikowski 2021; Leroux 2021] can be stated in parametric terms as follows: the former one shows \mathcal{F}_k -hardness of the reachability problem for VASS in dimension $6k$, while the latter one shows \mathcal{F}_k -hardness for VASS in dimension $4k + 5$. Our simplified construction yields a better lower bound: \mathcal{F}_k -hardness already in dimension $3k + 2$.

2 THE REACHABILITY PROBLEM

In this section we define the reachability problem and explain our contribution. Following [Czerwiński et al. 2019; Czerwinski et al. 2021b,a; Czerwiński and Orlikowski 2021; Leroux 2021], 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 commands, each of which is of one of the following types:

$x \ += \ 1$	(increment counter x)
$x \ -= \ 1$	(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.

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

Example 2.1. We write $x \ += \ m$ (resp. $x \ -= \ m$) as a shorthand for for m consecutive increments (resp. decrements) of x . As an illustration, consider the program with three counters $C = \{x, y, z\}$ (on the left), and its more readable presentation using a syntactic sugar **loop** (on the right):

1: goto 2 or 6	1: loop
2: $x \ -= \ 1$	2: $x \ -= \ 1$
3: $y \ += \ 1$	3: $y \ += \ 1$
4: $z \ += \ 2$	4: $z \ += \ 2$
5: goto 1 or 1	5: $z \ += \ 1$
6: $z \ += \ 1$	

It 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 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 $\text{Val}(C) = \mathbb{N}^C$ denote the set of all valuations of counters. Given an initial valuation of counters, a *run* (or *execution*) 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 subset $R \subseteq \text{VAL}(C)$ 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 $X \subseteq C$ of counters, if it ends with $x = 0$ for all $x \in X$. When $R = \{v\}$ contains a single valuation v , we speak of runs *from* v instead of runs from $\{v\}$. For instance, the program from Example 2.1 has exactly one x -zeroing run from the valuation $x = 10, y = z = 0$, where the final values of counters are $x = 0, y = 10, z = 21$.

By $\mathbf{0}$ we denote the valuation where all counters are 0. Following [Czerwiński et al. 2019; Czerwinski et al. 2021b,a; Czerwiński and Orlikowski 2021; Leroux 2021], 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 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. According to the encoding of VASS as Petri nets, the problem translates further 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. Let $\mathbb{N}_k = \{k, 2k, 3k, \dots\} \subseteq \mathbb{N}$ denote positive integers divisible by k , for $k \geq 1$. We define the complexity classes \mathcal{F}_i corresponding to the i th level in the Grzegorzczak Hierarchy [Schmitz 2016a, Sect. 2.3, 4.1]. The standard family of approximations $A_i : \mathbb{N}_1 \rightarrow \mathbb{N}_1$ of Ackermann function, for $i \in \mathbb{N}_1$, can be defined as follows:

$$A_1(n) = 2n, \quad A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \dots \circ A_i}_n(1).$$

In particular, $A_i(1) = 2$ for all $i \in \mathbb{N}_1$. Using functions A_i , we define the complexity classes \mathcal{F}_i , indexed by $i \in \mathbb{N}_1$, of problems solvable in deterministic time $A_i(p(n))$, where $p : \mathbb{N}_1 \rightarrow \mathbb{N}_1$ ranges over functions computable in deterministic time $A_{i-1}^m(n)$, for some $m \in \mathbb{N}_1$:

$$\mathcal{F}_i = \bigcup_{p \in \mathcal{FF}_{i-1}} \text{DTIME}(A_i(p(n))), \quad \text{where } \mathcal{FF}_i = \bigcup_{m \in \mathbb{N}_1} \text{FDTIME}(A_i^m(n)).$$

Intuitively speaking, the class \mathcal{F}_i contains all problems solvable in time $A_i(n)$, and is closed under reductions computable in time of lower order $A_{i-1}^m(n)$, 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 input size). The classes \mathcal{F}_k are robust with respect to the choice of fast-growing function hierarchy (see [Schmitz 2016a, Sect.4.1]). For $k \geq 3$, instead of deterministic time, one could equivalently take nondeterministic time, or space.

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

THEOREM 2.2. *Let $k \geq 3$. The reachability problem for programs with $3k + 2$ counters is \mathcal{F}_k -hard.*

The result can be compared to \mathcal{F}_k -hardness shown in [Czerwiński and Orlikowski 2021] for $6k$ counters, and in [Leroux 2021] for $4k + 5$ counters. Like the cited results, Theorem 2.2 implies ACKERMANN-hardness for unrestricted number of counters which, together with ACKERMANN upper bound of [Leroux and Schmitz 2019], yields ACKERMANN-completeness of the reachability problem.

Idea of simplification. Czerwiński and Orlikowski [Czerwiński and Orlikowski 2021] use the *ratio technique* introduced previously in [Czerwiński et al. 2019]. Speaking slightly informally, suppose some three counters b, c, d satisfy initially

$$b = B, \quad c > 0, \quad d = b \cdot c, \quad (1)$$

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 , at the price of 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 any auxiliary counters. This novel approach is presented in detail in Section 4.

3 MULTIPLIERS

Following the lines of [Czerwiński et al. 2019; Czerwinski et al. 2021b; Czerwiński and Orlikowski 2021], we rely on a concept of a *multiplier*.

Consider a program \mathcal{P} with counters C , a counter $x \in C$ and $R \subseteq \text{VAL}(C)$. 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 runs of \mathcal{P} from R . Formally, denoting by $\text{RUNS}_{\mathcal{P}}(R, x)$ the set of all x -zeroing runs of \mathcal{P} from R , and by $\text{FIN}(\pi)$ the final counter valuation of a complete run π of \mathcal{P} , the set x -computed by \mathcal{P} from R is

$$\{\text{FIN}(\pi) \mid \pi \in \text{RUNS}_{\mathcal{P}}(R, x)\}.$$

For instance, the program in Example 2.1 above x -computes, from the set of all valuations satisfying $y = z = 0$ (no constraint for x), the set of all valuations satisfying $x = 0$ (trivially) and $z = 2y + 1$.

Definition 3.1. Let $b, c, d \in C$ be some three distinguished counters, and $B \in \mathbb{N}_4$. We define the subset $\text{RATIO}(B, b, c, d, C) \subseteq \text{VAL}(C)$ of all valuations that satisfy the three conditions (1), and moreover assign 0 to all other counters $x \in C \setminus \{b, c, d\}$.

A program (with no zero-tests) with counters C that z -computes from the zero valuation $\mathbf{0}$ the set $\text{RATIO}(B, b, c, d, C)$, for some four of its counters $z, b, c, d \in C$, we call B -multiplier.

Example 3.2. As a simple example, for every fixed $B \in \mathbb{N}_4$, the following program is a B -multiplier of size $O(B)$ (counter z is not used at all):

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

```
1: b += B   d += B   c += 1
2: loop
3:   d += B   c += 1
```

For technical convenience we prefer to rely on the following family of functions $F_i : \mathbb{N}_4 \rightarrow \mathbb{N}_4$, indexed by $i \in \mathbb{N}_1$, closely related to functions A_i (cf. Claim 3.3 below):

$$F_1(n) = 2n, \quad F_{i+1} = \underbrace{F_i \circ F_i \circ \dots \circ F_i}_{n/4}(4). \quad (2)$$

In particular, $F_i(4) = 8$ for all $i \in \mathbb{N}_1$. By induction on i one easily shows that F_i is a linear re-scaling of A_i :

CLAIM 3.3. $F_i(4 \cdot n) = 4 \cdot A_i(n)$, for $i, n \in \mathbb{N}_1$.

PROOF. As $F_1(n) = A_1(n) = 2n$, the claim holds for $i = 1$. Assuming the claim for $i \in \mathbb{N}_1$, by n -fold application thereof we derive

$$F_{i+1}(4 \cdot n) = \underbrace{F_i \circ \dots \circ F_i}_n(4) = 4 \cdot \underbrace{A_i \circ \dots \circ A_i}_n(1) = 4 \cdot A_{i+1}(n),$$

as required. \square

As a technical core of the proof of Theorem 2.2, combining our simplification with the lines of [Czerwiński and Orlikowski 2021], we provide an effective construction of B -multipliers with $3k + 2$ counters, where $B = F_k(n)$, of size polynomial in k and n .

THEOREM 3.4. *Given $k \in \mathbb{N}_1$ and $n \in \mathbb{N}_4$ one can compute, in time polynomial in k and n , an $F_k(n)$ -multiplier with $3k + 2$ counters.*

4 PROOF OF THEOREM 2.2

Relying on Theorem 3.4 (to be shown later), we prove in this section Theorem 2.2.

Maximal iteration. Whenever analysing a single run of a program, we denote by \bar{x} the initial value of a counter x , and by \underline{x} the final value thereof. 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 (several command are written in none line to save space):

```
1: loop
2:   f -= 1   e += 1   d -= 1
```

Assuming $\bar{d} \geq \bar{e} + \bar{f}$, we observe that the amount $\bar{d} - \underline{d}$ by which d is decreased as an effect of execution (we use the word *execution* as a synonym to *run*) of the macro may be any value between 0 and \bar{f} , and that the equality $\bar{d} - \underline{d} = \bar{e} + \bar{f}$ holds if, and only if the loop is *iterated maximally*, i.e., $\bar{e} = 0$ and $\underline{f} = 0$. 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 , and assume that \mathcal{P} uses zero tests only on two its counters $x, y \in C$ (the construction easily extends to programs with an arbitrary number of zero-tested counters). We add to \mathcal{P} three fresh counters b, c, d (let $C^* = C \cup \{b, c, d\}$), and transform \mathcal{P} into a program \mathcal{P}^* *without zero tests* that, assuming the initial valuation of counters belongs to $\text{RATIO}(2m, b, c, d, C^*)$, 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 , and likewise we do for y :

command	replaced by	command	replaced by
$x += 1$	$x += 1 \quad c -= 1$	$y += 1$	$y += 1 \quad c -= 1$
$x -= 1$	$x -= 1 \quad c += 1$	$y -= 1$	$y -= 1 \quad c += 1$

In the resulting program $\bar{\mathcal{P}}$ counters x, y are, intuitively speaking, put on a shared 'budget' c . Assuming x and y are initially 0, this clearly enforces $x + y$ to not exceed the initial value of c , and the sum $s = c + x + y$ to remain invariant.

Second, we replace in $\bar{\mathcal{P}}$ every **zero?** x command by the following macro **ZERO?** x , and likewise we replace every **zero?** y command by an analogous macro **ZERO?** y , where x and y are swapped:

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

```

ZERO? y:

```

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

```

This yields the transformed program \mathcal{P}^* . We note that the macros preserve the sum $s = c + x + y$.

An execution of ZERO? x (resp. ZERO? y) is called *maximally iterated* if all four loops are so. Observe that every such execution is forcedly *correct*, i.e. satisfies:

$$\bar{x} = \underline{x} = 0, \quad \bar{y} = \underline{y}$$

(resp. $\bar{y} = y = 0, \bar{x} = \underline{x}$). 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 $\bar{d} = \underline{d}$ by which d is decreased equals $2s$ exactly in maximally iterated executions:

CLAIM 4.1. *Consider an execution of ZERO? macro, 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 execution is maximally iterated.*

PROOF. Consider an execution of ZERO?, assuming $\bar{d} \geq 2s$, and let \dot{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

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

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. \square

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 4.1 the invariant can not be recovered later. These observations lead to the following correctness claim:

LEMMA 4.2. *The following conditions are equivalent:*

- \mathcal{P} has a complete run from 0 that does exactly m zero-tests.
- \mathcal{P}^* has a d-zeroing run from $\text{RATIO}(2m, b, c, d, C^*)$.

PROOF. One direction is immediate: 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 $\text{RATIO}(2m, b, c, d, C^*)$, 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 all loops in ZERO? macros.

For the converse direction, consider a d-zeroing run π of \mathcal{P}^* from $\text{RATIO}(2m, b, c, d, C^*)$. The initial counter valuation satisfies the equality $d = b \cdot s$. Each execution of ZERO? decreases b by 2, and d by at most $2s$ (by the first part of Claim 4.1). Therefore, since b and d are not affected

elsewhere and $d = 0$ finally, each execution of **ZERO?** forcedly decreases d by *exactly* $2s$. By the second part of Claim 4.1 we derive:

CLAIM 4.3. *Each execution of **ZERO?** in π is correct.*

Furthermore, as d dorcedly decreases m times by $2s$, we deduce:

CLAIM 4.4. *There are exactly m executions of **ZERO?** in π .*

Due to Claims 4.3 and 4.4, once we project away from π the counters b, c, d , we obtain a complete run of \mathcal{P} from $\mathbf{0}$ that does exactly m zero-tests, as required. \square

Proof of Theorem 2.2. Fix $k \geq 3$. The proof proceeds by a polynomial-time reduction from the following \mathcal{F}_k -hard problem:

\mathcal{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 $\mathcal{F}_k(n)/2$ zero tests?

The problem is \mathcal{F}_k -hard, since any program 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 $m = \mathcal{F}_k(n)/2$ zero tests if, and only if, \mathcal{P}' has a $\{d, z\}$ -zeroing run (for some $d, z \in C$).

First, using Theorem 3.4 we compute a $2m$ -multiplier \mathcal{M} with $3k + 2$ counters C that z -computes from $\mathbf{0}$ the set $\text{RATIO}(2m, b, c, d, C)$, for some $z, b, c, d \in C$. 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:    $x \leftarrow x - 1$ 
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 4.5. *The following conditions are equivalent:*

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

The second and the third point are equivalent due to Lemma 4.2, 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 $C' = C - \{b, c, d, z\}$ in place of x, y . The latter equivalence in Claim 4.5 remains true, as z -zeroing runs of \mathcal{M} from $\mathbf{0}$ are necessarily C' -zeroing too. \square

5 PROOF OF THEOREM 3.4

The proof proceeds by combining the concept of amplifier lifting of [Czerwiński and Orlikowski 2021] with the program transformation of the previous section.

Amplifiers. Let $F : \mathbb{N}_4 \rightarrow \mathbb{N}_4$ 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* $b, c, d \in C$ and three *output counters* $b', c', d' \in C$. The program is called *F-amplifier* if for every $B \in \mathbb{N}_4$, it d-computes from $\text{RATIO}(B, b, c, d, C)$ the set $\text{RATIO}(F(B), b', c', d', C)$. We note that no condition is imposed on d-zeroing runs from counter valuations not belonging to any set $\text{RATIO}(B, b, c, d, C)$. As an example, consider the following program \mathcal{L}_ℓ , for $\ell \in \mathbb{N}_1$, with input counters b, c, d and output counters b', c', d' :

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

```

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 5.1. *The above program is an L_ℓ -amplifier, where $L_\ell : \mathbb{N}_4 \rightarrow \mathbb{N}_4 : x \mapsto \ell \cdot x$.*

PROOF SKETCH. Writing counter valuations as vectors (b, c, d, b', c', 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 $d = 0$ 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. \square

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

Amplifier lifting. Recall the definition (2) of functions F_i ; in particular $F_1 = L_2$. For a function $F : \mathbb{N}_4 \rightarrow \mathbb{N}_4$, we define the successor function $\tilde{F} : \mathbb{N}_4 \rightarrow \mathbb{N}_4$ as

$$\tilde{F}(n) = \underbrace{F \circ F \circ \dots \circ F}_{n/4-1}(8). \quad (3)$$

Relying on the equality $F_i(4) = 8$, for $i \in \mathbb{N}_1$, we note:

CLAIM 5.2. $F_{i+1} = \tilde{F}_i$, for $i \in \mathbb{N}_1$.

Let \mathcal{P} be a program with counters C , without zero tests, with distinguished input counters $b_1, c_1, d_1 \in C$ and output counters $b_2, c_2, d_2 \in C$. We describe a transformation of the program \mathcal{P} to a program $\tilde{\mathcal{P}}$, also without zero tests, such that assuming that \mathcal{P} is an F -amplifier for some function $F : \mathbb{N}_4 \rightarrow \mathbb{N}_4$, the program $\tilde{\mathcal{P}}$ is an \tilde{F} -amplifier. The program $\tilde{\mathcal{P}}$ uses, except for the counters of \mathcal{P} , three fresh counters b, c, d . Thus counters of $\tilde{\mathcal{P}}$ are $\tilde{C} = C \cup \{b, c, d\}$. We let input counters of $\tilde{\mathcal{P}}$ be b, c, d , and its output counters be b_1, c_1, d_1 .

In the transformation we use the identity-amplifier $\mathcal{L} = \mathcal{L}_1(b_2, c_2, d_2, b_1, c_1, d_1)$ with input counters b_2, c_2, d_2 and output counters b_1, c_1, d_1 , and the 8-multiplier $\mathcal{M} = \mathcal{M}_8(b_1, c_1, d_1)$ of Example 3.2. 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 $\overline{\mathcal{P}}$, 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 $\overline{\mathcal{L}}$, and the multiplier \mathcal{M} to $\overline{\mathcal{M}}$. Using $\overline{\mathcal{P}}$, $\overline{\mathcal{L}}$, $\overline{\mathcal{M}}$ and a macro SET-C-TO-ZERO as building blocks, we define a counter program $\widetilde{\mathcal{P}}$ as follows:

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

```

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

```

SET-C-TO-ZERO:

```

1: loop
2:    $c \ -= \ 1$ 
3:    $d \ -= \ 4$ 
4: ZERO?  $c$ 
5: ZERO?  $c$ 

```

Without the macro SET-C-TO-ZERO in line 7, the program $\widetilde{\mathcal{P}}$ would be exactly the result of applying the transformation $Q \mapsto Q^*$ of Section 4 to the following program Q (with counters d_1 and d_2 in place of x and y):

```

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

```

Note that $\overline{\mathcal{M}}$ initialises counters C to $\text{RATIO}(8, b_1, c_1, d_1, C)$. 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 decreases d by at most $4(\overline{c} + \overline{d}_1 + \overline{d}_2)$.

As in Section 4, we observe that the amount $s = c + d_1 + d_2$ is preserved by ZERO? macros and hence stays invariant in each run of $\widetilde{\mathcal{P}}$ before entering to the SET-C-TO-ZERO macro. Writing s to denote this amount before entering to the SET-C-TO-ZERO macro, we derive the following property of SET-C-TO-ZERO from Claim 4.1 (note however that SET-C-TO-ZERO does not forcedly preserve $c + d_1 + d_2$):

CLAIM 5.3. *Consider an arbitrary execution of SET-C-TO-ZERO(c) and assume $\overline{d} \geq 4s$. Then $0 \leq \overline{d} - \underline{d} \leq 4s$. Furthermore, the equality $\overline{d} - \underline{d} = 4s$ holds if, and only if both ZERO? macros are maximally iterated.*

Lemma 5.4 states the crucial amplifier-lifting property of the transformation $\mathcal{P} \mapsto \widetilde{\mathcal{P}}$.

LEMMA 5.4. *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 4.2. Consider all d -zeroing runs of $\widetilde{\mathcal{P}}$ from $\text{RATIO}(B, b, c, d, \widetilde{C})$. All initial counter valuations satisfy the equality $d = b \cdot s$, and the equality is preserved by $\overline{\mathcal{M}}$. Further on, each execution of ZERO? decreases b by 2, and d by at most $2s$ (by the first part of Claim 4.1). Likewise, the execution of SET-C-TO-ZERO decreases b by 4 and d by at most $4s$ (by the first part of Claim 5.3). Therefore, since b and d are not affected elsewhere and $\underline{d} = 0$ finally, we have the following properties of d -zeroing runs of $\widetilde{\mathcal{P}}$ from $\text{RATIO}(B, b, c, d, \widetilde{C})$:

CLAIM 5.5. *Finally, $\underline{b} = 0$.*

CLAIM 5.6. *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 $\text{RATIO}(B, b, c, d, \tilde{C})$, the program $\tilde{\mathcal{P}}$ d -computes exactly the set $\text{RATIO}(\tilde{F}(B), b, c, d, \tilde{C})$. By Claim 5.6 we deduce:

CLAIM 5.7. *Each d -zeroing run from $\text{RATIO}(B, b, c, d, \tilde{C})$ contains exactly 2ℓ executions of ZERO? macro, all of them correct.*

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

As \mathcal{M} is an 8-multiplier, $R_0 = \text{RATIO}(8, b_1, c_1, d_1, C)$. We show:

CLAIM 5.8. $R_{2\ell} = \text{RATIO}(F^\ell(8), b_1, c_1, d_1, C)$.

If $\ell = 0$, $R_{2\ell} = R_0$ proves the claim. Otherwise, suppose $\ell > 0$. By Claim 5.7, each execution of $\tilde{\mathcal{P}}$ 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 $\tilde{\mathcal{P}}$ w.r.t. runs of \mathcal{P} , we get $R_1 = \text{RATIO}(F(8), b_2, c_2, d_2, C)$. By Claim 5.7 again, each execution of $\tilde{\mathcal{L}}$ is d_2 -zeroing; therefore $R_2 = \text{RATIO}(F(8), b_1, c_1, d_1, C)$ similarly as before. By an ℓ -fold repetition of this argument we derive Claim 5.8.

By the second part of Claim 5.3, the macro SET-C-TO-ZERO does not change the value of d_1 in d -zeroing runs from $\text{RATIO}(B, b, c, d, \tilde{C})$, and ends with $\underline{c} = d_2 = 0$. By Claim 5.5 we have $\underline{b} = 0$. As SET-C-TO-ZERO affects no other counters, by Claim 5.8 we deduce that $\tilde{\mathcal{P}}$ d -computes from $\text{RATIO}(B, b, c, d, \tilde{C})$ the set $\text{RATIO}(F^\ell(8), b_1, c_1, d_1, \tilde{C})$. As $F^\ell(8) = \tilde{F}(B)$, $\tilde{\mathcal{P}}$ is an \tilde{F} -amplifier. \square

Proof of Theorem 3.4. We rely on Claim 5.2 and Lemma 5.4. Given $k \in \mathbb{N}_1$ and $n \in \mathbb{N}_4$ we compute, in time linear in k , the F_k -amplifier \mathcal{A}_k with $3k + 3$ counters C , by $(k - 1)$ -fold application of the amplifier lifting transformation $\mathcal{P} \mapsto \tilde{\mathcal{P}}$ described above, starting from the F_1 -amplifier \mathcal{L}_2 of Claim 5.1. The $F_k(n)$ -multiplier is obtained by pre-composing \mathcal{A} with an n -multiplier (e.g. $\mathcal{M}_n(b, c, d)$ from Section 2) outputting the set $\text{RATIO}(n, b, c, d, C)$ in input counters $b, c, d \in C$ of \mathcal{A}_k .

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 i th (for $i = 0, \dots, n$) copy corresponds to the value $b = i$. The resulting program has $3k + 2$ counters. \square

6 FINAL REMARKS

We propose a simplification of the construction of [Czerwiński and Orlikowski 2021], thus improving the dimension-parametric lower bound for the VASS (Petri nets) reachability problem: compared to \mathcal{F}_k -hardness in dimension $6k$ [Czerwiński and Orlikowski 2021] and $4k + 5$ [Leroux 2021], respectively, we obtain \mathcal{F}_k -hardness already in dimension $3k + 2$. We believe that by combining with the insights of [Czerwiński and Orlikowski 2021] one can further optimise our construction and lower the dimension by a small constant.

On the other hand, the best know upper bound states that the VASS reachability problem in dimension $k - 4$ is in \mathcal{F}_k [Leroux and Schmitz 2019]. Establishing exact parametric complexity of the problem, i.e., closing the gap between dimensions $k - 4$ and $3k + 2$, 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 [Blondin et al. 2015; Englert et al. 2016], 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 [Leroux and Schmitz 2019].

ACKNOWLEDGMENTS

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

REFERENCES

- David Angeli, Patrick De Leenheer, and Eduardo D. Sontag. 2011. Persistence Results for Chemical Reaction Networks with Time-Dependent Kinetics and No Global Conservation Laws. *SIAM Journal of Applied Mathematics* 71, 1 (2011), 128–146.
- Paolo Baldan, Nicoletta Cocco, Andrea Marin, and Marta Simeoni. 2010. Petri nets for modelling metabolic pathways: a survey. *Natural Computing* 9, 4 (2010), 955–989. <https://doi.org/10.1007/s11047-010-9180-6>
- Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. 2015. Reachability in Two-Dimensional Vector Addition Systems with States Is PSPACE-Complete. In *Proc. LICS*. 32–43.
- Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. 2011. Two-variable logic on data words. *ACM Trans. Comput. Log.* 12, 4 (2011), 27:1–27:26. <http://doi.acm.org/10.1145/1970398.1970403>
- Ahmed Bouajjani and Michael Emmi. 2013. Analysis of Recursively Parallel Programs. *ACM Trans. Program. Lang. Syst.* 35, 3 (2013), 10:1–10:49. <http://doi.acm.org/10.1145/2518188>
- Frank P. Burns, Albert Koelmans, and Alexandre Yakovlev. 2000. WCET Analysis of Superscalar Processors Using Simulation With Coloured Petri Nets. *Real-Time Systems* 18, 2/3 (2000), 275–288. <https://doi.org/10.1023/A:1008101416758>
- Thomas Colcombet and Amaldev Manuel. 2014. Generalized Data Automata and Fixpoint Logic. In *FSTTCS (LIPIcs, Vol. 29)*. Schloss Dagstuhl, 267–278. <https://doi.org/10.4230/LIPIcs.FSTTCS.2014.267>
- Stefano Crespi-Reghizzi and Dino Mandrioli. 1977. Petri Nets and Szilard Languages. *Information and Control* 33, 2 (1977), 177–192. [https://doi.org/10.1016/S0019-9958\(77\)90558-7](https://doi.org/10.1016/S0019-9958(77)90558-7)
- Wojciech Czerwinski, Sławomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. 2019. The reachability problem for Petri nets is not elementary. In *Proc. STOC 2019*, Moses Charikar and Edith Cohen (Eds.). ACM, 24–33.
- Wojciech Czerwinski, Sławomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. 2021b. The Reachability Problem for Petri Nets Is Not Elementary. *J. ACM* 68, 1 (2021), 7:1–7:28. <https://doi.org/10.1145/3422822>
- Wojciech Czerwinski, Sławomir Lasota, and Łukasz Orlikowski. 2021a. Improved lower bounds for reachability in vector addition systems. In *Proc. ICALP*. To appear.
- Wojciech Czerwinski and Łukasz Orlikowski. 2021. Reachability in Vector Addition Systems is Ackermann-complete.
- Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. 2014. Ordered Navigation on Multi-attributed Data Words. In *Proc. CONCUR (LNCS, Vol. 8704)*. Springer, 497–511. https://doi.org/10.1007/978-3-662-44584-6_34
- Stéphane Demri, Diego Figueira, and M. Praveen. 2016. Reasoning about Data Repetitions with Counter Systems. *Logical Methods in Computer Science* 12, 3 (2016). [https://doi.org/10.2168/LMCS-12\(3:1\)2016](https://doi.org/10.2168/LMCS-12(3:1)2016)
- Matthias Englert, Ranko Lazic, and Patrick Totzke. 2016. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In *Proc. LICS*. ACM, 477–484. <http://doi.acm.org/10.1145/2933575.2933577>
- Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2017. Verification of population protocols. *Acta Inf.* 54, 2 (2017), 191–215. <https://doi.org/10.1007/s00236-016-0272-3>
- Pierre Ganty and Rupak Majumdar. 2012. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.* 34, 1 (2012), 6:1–6:48. <http://doi.acm.org/10.1145/2160910.2160915>
- Steven M. German and A. Prasad Sistla. 1992. Reasoning about Systems with Many Processes. *J. ACM* 39, 3 (1992), 675–735. <http://doi.acm.org/10.1145/146637.146681>
- Sheila A. Greibach. 1978. Remarks on Blind and Partially Blind One-Way Multicounter Machines. *Theor. Comput. Sci.* 7 (1978), 311–324. [https://doi.org/10.1016/0304-3975\(78\)90020-8](https://doi.org/10.1016/0304-3975(78)90020-8)
- Piotr Hofman and Sławomir Lasota. 2018. Linear Equations with Ordered Data. In *Proc. CONCUR (LIPIcs, Vol. 118)*. Schloss Dagstuhl, 24:1–24:17. <https://doi.org/10.4230/LIPIcs.CONCUR.2018.24>
- John E. Hopcroft and Jean-Jacques Pansiot. 1979. On the Reachability Problem for 5-Dimensional Vector Addition Systems. *Theor. Comput. Sci.* 8 (1979), 135–159. [https://doi.org/10.1016/0304-3975\(79\)90041-0](https://doi.org/10.1016/0304-3975(79)90041-0)
- Alexander Kaiser, Daniel Kroening, and Thomas Wahl. 2014. A Widening Approach to Multithreaded Program Verification. *ACM Trans. Program. Lang. Syst.* 36, 4 (2014), 14:1–14:29. <http://doi.acm.org/10.1145/2629608>

- Max I. Kanovich. 1995. Petri Nets, Horn Programs, Linear Logic and Vector Games. *Ann. Pure Appl. Logic* 75, 1–2 (1995), 107–135. [https://doi.org/10.1016/0168-0072\(94\)00060-G](https://doi.org/10.1016/0168-0072(94)00060-G)
- Richard M. Karp and Raymond E. Miller. 1969. Parallel Program Schemata. *J. Comput. Syst. Sci.* 3, 2 (1969), 147–195. [https://doi.org/10.1016/S0022-0000\(69\)80011-5](https://doi.org/10.1016/S0022-0000(69)80011-5)
- S. Rao Kosaraju. 1982. Decidability of Reachability in Vector Addition Systems (Preliminary Version). In *Proc. STOC*. ACM, 267–281. <http://doi.acm.org/10.1145/800070.802201>
- Jean-Luc Lambert. 1992. A Structure to Decide Reachability in Petri Nets. *Theor. Comput. Sci.* 99, 1 (1992), 79–104. [https://doi.org/10.1016/0304-3975\(92\)90173-D](https://doi.org/10.1016/0304-3975(92)90173-D)
- Hélène Leroux, David Andreu, and Karen Godary-Dejean. 2015. Handling Exceptions in Petri Net-Based Digital Architecture: From Formalism to Implementation on FPGAs. *IEEE Trans. Industrial Informatics* 11, 4 (2015), 897–906.
- Jérôme Leroux. 2010. The General Vector Addition System Reachability Problem by Presburger Inductive Invariants. *Logical Methods in Computer Science* 6, 3 (2010). [https://doi.org/10.2168/LMCS-6\(3:22\)2010](https://doi.org/10.2168/LMCS-6(3:22)2010)
- Jérôme Leroux. 2011. Vector addition system reachability problem: a short self-contained proof. In *POPL*. ACM, 307–316. <http://doi.acm.org/10.1145/1926385.1926421>
- Jérôme Leroux. 2012. Vector Addition Systems Reachability Problem (A Simpler Solution). In *Turing-100 (EPIc Series in Computing, Vol. 10)*. EasyChair, 214–228. <http://www.easychair.org/publications/paper/106497>
- Jérôme Leroux. 2021. The Reachability Problem for Petri Nets is Not Primitive Recursive.
- Jérôme Leroux and Sylvain Schmitz. 2015. Demystifying Reachability in Vector Addition Systems. In *Proc. LICS*. 56–67.
- Jérôme Leroux and Sylvain Schmitz. 2019. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *Proc. LICS*. IEEE, 1–13.
- Yuliang Li, Alin Deutsch, and Victor Vianu. 2017. VERIFAS: A Practical Verifier for Artifact Systems. *PVLDB* 11, 3 (2017), 283–296. <http://www.vldb.org/pvldb/vol11/p283-li.pdf>
- Richard J. Lipton. 1976. *The reachability problem requires exponential space*. Technical Report 62. Yale University. <http://cpsc.yale.edu/sites/default/files/files/tr63.pdf>
- Martin H. Löb and Stanley S. Wainer. 1970. Hierarchies of number-theoretic functions. I. *Archiv für mathematische Logik und Grundlagenforschung* 13, 1–2 (1970), 39–51. <https://doi.org/10.1007/BF01967649>
- Ernst W. Mayr. 1981. An Algorithm for the General Petri Net Reachability Problem. In *STOC*. ACM, 238–246. <http://doi.acm.org/10.1145/800076.802477>
- Ernst W. Mayr. 1984. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comput.* 13, 3 (1984), 441–460. <https://doi.org/10.1137/0213029>
- Roland Meyer. 2009. A theory of structural stationarity in the π -Calculus. *Acta Inf.* 46, 2 (2009), 87–137. <https://doi.org/10.1007/s00236-009-0091-x>
- Carl Adam Petri. 1962. *Kommunikation mit Automaten*. Ph.D. Dissertation. Universität Hamburg. <http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/>
- George S. Sacerdote and Richard L. Tenney. 1977. The Decidability of the Reachability Problem for Vector Addition Systems (Preliminary Version). In *STOC*. ACM, 61–76. <http://doi.acm.org/10.1145/800105.803396>
- Sylvain Schmitz. 2016a. Complexity Hierarchies beyond Elementary. *TOCT* 8, 1 (2016), 3:1–3:36.
- Sylvain Schmitz. 2016b. The complexity of reachability in vector addition systems. *SIGLOG News* 3, 1 (2016), 4–21.
- Wil M. P. van der Aalst. 2015. Business process management as the “Killer App” for Petri nets. *Software and System Modeling* 14, 2 (2015), 685–691. <https://doi.org/10.1007/s10270-014-0424-2>