EasyChair Preprint

№ 8619

Lower Bounds for the Reachability Problem in Fixed Dimensional VASSes

Wojciech Czerwiński and Łukasz Orlikowski

August 8, 2022

Lower Bounds for the Reachability Problem in Fixed Dimensional VASSes

WOJCIECH CZERWIŃSKI, University of Warsaw, Poland
ŁUKASZ ORLIKOWSKI, University of Warsaw, Poland

We study the complexity of the reachability problem for Vector Addition Systems with States (VASSes) in fixed dimensions. We provide four lower bounds improving the currently known state-of-the-art: 1) NP-hardness for unary flat 4-VASSes (VASSes in dimension 4), 2) PSpace-hardness for unary 5-VASSes, 3) ExpSpace-hardness for binary 6-VASSes and 4) Tower-hardness for unary 8-VASSes.

Additional Key Words and Phrases: vector addition systems, reachability problem, lower bounds, Petri nets

ACM Reference Format:
Wojciech Czerwiński and Łukasz Orlikowski. 2022. Lower Bounds for the Reachability Problem in Fixed Dimensional VASSes. 1, 1 (August 2022), 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Vector Addition Systems (VASes) together with essentially equivalent Petri nets and Vector Addition Systems with States (VASSes) are fundamental models of computation with many application in practice and theory. The central algorithmic problem concerning VASSes is the reachability problem asking whether in a given VASS there exists a run from one given configuration to another. The long research history of this problem dates back to seventies when Lipton has proven ExpSpace-hardness of the reachability problem [18]. Decidability of the problem was shown a few years later by Mayr in [19], where he presented a very involved algorithm. After a few decades of research recently the complexity of the problem was settled to be Ackermann-complete. The upper bound was shown by Leroux and Schmitz in [16] three years ago. Last year Ackermann-hardness was independently proven by Leroux [14] and by Czerwiński and Orlikowski [6].

Despite settling the computational complexity of the reachability problem in VASSes a lot of questions about VASSes remain to be solved. Even the reachability problem is not fully understood and the most clear evidence for that is the existence of big complexity gaps for the problem in small fixed dimensions. The prominent example here is the dimension three with complexity gap between PSpace-hardness (inherited from dimension two [2]) and super-Tower (concretely speaking $\mathcal{F}_7$, namely the 7-th level of the Grzegorczyk hierarchy [16]). The reachability problem was already extensively studied for fixed dimensions. For dimension one (i.e. for 1-VASSes) for binary encoding of numbers occurring in transitions it was shown to be NP-complete in [12]. For unary encoded 1-VASSes it is easy to see that the reachability problem is NL-complete. For 2-VASSes the problem is known to be PSpace-complete in the case of binary encoding [2] and moreover NL-complete in the case of unary encoding [10], both results are described as well in the joint full version [1]. However, beyond dimension two the situation is much less clear.

In [5] several cases of the reachability problem for fixed dimensional VASSes were considered. In particular a subclass of flat VASSes was investigated, namely VASSes without nested loops in the state structure. This class was introduced in [17] and has a bunch of nice properties. In particular the reachability relation is semilinear and the reachability problem can be easily shown to be in NP, even in the case of binary encoding. In [5] it was shown that the reachability

Authors' addresses: Wojciech Czerwiński, wczerwin@mimuw.edu.pl, University of Warsaw, Poland; Łukasz Orlikowski, lo418363@students.mimuw.edu.pl, University of Warsaw, Poland.

2022. Manuscript submitted to ACM

problem is NP-hard already for a fixed dimension and unary encoding, namely for unary 7-VASSes, but the status of the problem for lower dimensions remained unsettled.

The first ExpSpace-hardness result for fixed dimension follows from [4], where it was shown that the problem is h-ExpSpace-hard for unary $(h + 13)$-VASSes, thus ExpSpace-hard for unary 14-VASSes. Recent Ackermann-hardness results delivered also Tower-hardness (and in particular ExpSpace-hardness) results in fixed dimensions. Notice that Tower-hardness for binary d-VASSes implies Tower-hardness for unary d-VASSes as the Tower complexity class is closed under exponential blowup of running time. Thus we may not emphasise encoding when talking about Tower-hardness. The dimension in which the problem is Tower-hard was step by step decreased from 21 in the initial version of [14] (see also the arxiv version [15]) and 18 in [6] through dimension 17 in third version of [15], 11 in the recent Lasota's work [13] to a currently best value of 10 in last version of [15]. We further decrease the dimension and show that the reachability problem is already Tower-hard for 8-VASSes.

Our contribution. We believe it is important to pursue the search for exact complexities for fixed dimensional VASSes. First of all low dimensional VASSes are very natural computation models and currently known techniques used to provide hardness results are very likely not to work in some small dimensions. Secondly, it is easier to invent a sophisticated technique working in a simpler setting. Therefore it is quite possible that the search for exact complexity bounds for the reachability problem in low dimensions will result in finding new techniques useful in much broader generality. Thirdly, despite very high pessimistic complexity of the reachability problem it still can be solved in practise in some cases [3, 9]. Therefore it is not only a theoretical, but may also be of practical interest to understand for which VASS subclasses the reachability problem have relatively low complexity and avoiding which obstacles may lead to efficient algorithms. One obvious way to pursue this idea is to understand better low dimensional VASSes.

Our main results are the four lower bound theorems, which improve the previously mentioned lower bounds. Additionally we introduce a novel technique of proving lower bounds inspired by the multiplication triples technique introduced in [4] and used also in [6, 13]. We call it the quadratic pairs technique and use it to decrease the dimension of VASSes in certain hardness results. Concretely speaking we apply this approach to prove Theorems 1.2 and 1.3.

Beside that our main conceptual contribution is to compose already known techniques in a subtle way in order to get lower bounds, which are 1) substantially stronger than currently known, and 2) shown by some not very involved constructions. We would like to emphasise that our constructions are rather simple, but we see it as an advantage rather than a disadvantage.

As a first contribution we provide a simple construction which decreases the dimension in which the reachability problem is NP-hard for unary, flat VASSes, namely we decrease the dimension from 7 in [5] to a dimension 4.

Theorem 1.1. *The reachability problem for unary, flat* 4*-VASSes is* NP*-hard.*

We need only one dimension more to show PSpace-hardness for unary (not necessarily flat though) VASSes.

Theorem 1.2. *The reachability problem for unary* 5*-VASSes is* PSpace*-hard.*

Next we lower the dimension for which ExpSpace-hardness is known from 10 [15] to 6.

Theorem 1.3. *The reachability problem for binary* 6*-VASSes is* ExpSpace*-hard.*

Notice that Theorem 1.3 clearly shows also PSpace-hardness for unary 6-VASSes (as the unary representation is at most exponentially bigger than the binary one), but for PSpace-hardness we can eliminate one dimension in the proof of Theorem 1.2.

We also show that only two dimensions more than needed for ExpSpace-hardness is enough to get Tower-hardness.

Theorem 1.4. *The reachability problem for unary* 8*-VASSes is* Tower*-hard.*

In order to prove our results we crucially exploit two known techniques designed to force counters of VASSes to be equal to zero at some particular configurations along the run, namely simulate zero-tests on some counters. The first technique is based on triples of the form (B, C, BC) and was introduced in [4] in order to simulate $C/2$ zero-tests for counters bounded by value B. This idea was later improved in [13] and in [6] to handle many counters by just one triple. Based on this technique we design our novel quadratic pair technique. The second technique was introduced in [6] and uses a single controlling-counter in order to perform a linear number of zero-tests. It turns out that none of these two tools dominate the other one, they are useful in different situations.

Organisation of the paper. In Section 2 we introduce preliminary notions and recall necessary facts about the above mentioned two techniques of zero-testing. In Section 2 we also introduce the quadratic pair technique and prove related facts about counter automata. Then in Section 3 we briefly describe ideas beyond our proofs, in some cases it might be even sufficient to read this section in order to understand in-depth our arguments. In Sections 4, 5, 6 and 7 we prove in detail Theorems 1.1, 1.2, 1.3 and 1.4, respectively. Finally in Section 8 we comment about the limitations of our techniques and mention possible future research directions.

2 PRELIMINARIES

Basic notions. For $a, b \in \mathbb{N}$ we write $[a, b]$ to denote the set $\{a, a+1, \ldots, b-1, b\}$. For a vector $v \in \mathbb{N}^d$ and $i \in [1, d]$ we write $v[i]$ to denote the i-th coordinate of vector v. By 0^d we denote vector $v \in \mathbb{N}^d$ with all coordinates equal to zero.

Vector Addition Systems with States. A d-dimensional Vector Addition System with States (d-VASS) consists of a finite set of *states* Q and a finite set of transitions $T \subseteq Q \times \mathbb{Z}^d \times Q$. A *configuration* of a d-VASS is a pair $(q, v) \in Q \times \mathbb{N}^d$, we often write it as $q(v)$ instead of (q, v). For a configuration $c = q(v)$ and $i \in [1, d]$, we denote by $c[i]$ the value $v[i]$. The set of all the configurations is denoted by $\mathrm{Conf} = Q \times \mathbb{N}^d$. A transition (p, u, q) can be *fired* in a configuration $r(v)$ if $p = r$ and $u + v \in \mathbb{N}^d$. We write then $p(v) \xrightarrow{(p,u,q)} q(u+v)$. The *effect* of a transition (p, u, q) is the vector u, we write $\mathrm{eff}((p, u, q)) = u$. A sequence $\rho = (c_1, t_1, c_1'), (c_2, t_2, c_2'), \ldots, (c_n, t_n, c_n') \in \mathrm{Conf} \times T \times \mathrm{Conf}$ is a *run* of a VASS $V = (Q, T)$ if, for all $i \in [1, n]$, we have $c_i \xrightarrow{t_i} c_i'$ and, for all $i \in [1, n-1]$, we have $c_i' = c_{i+1}$. We naturally extend the notion of *effect* to runs by defining $\mathrm{eff}(\rho) = \mathrm{eff}(t_1) + \ldots + \mathrm{eff}(t_n)$. Such a run ρ is said to be *from* the configuration c_1 *to* the configuration c_n'. We write then $c_1 \xrightarrow{\rho} c_n'$, slightly overloading the notation, or simply $c_1 \longrightarrow c_n'$ if there is some ρ such that $c_1 \xrightarrow{\rho} c_n'$. By $\textsc{Reach}(\mathrm{src}, V) = \{c \mid \mathrm{src} \longrightarrow c\}$ we denote the set of all the configurations reachable from the configuration src and we call it the *reachability set*. We also write simply $\textsc{Reach}(\mathrm{src})$ if the VASS V is clear from the context.

The following problem is the main focus of this paper, for different values of $d \in \mathbb{N}$.

Reachability problem for d-VASSes
Input: A d-VASS V and two of its configurations src, trg
Question: Does $\mathrm{src} \longrightarrow \mathrm{trg}$ in V?

The size of VASS V, denoted $\mathrm{size}(V)$, is the total number of bits needed to represent states and transitions of V. A *state-cycle* in a VASS V is a cycle in the graph (Q, E) with vertices being states of V and edges being defined as

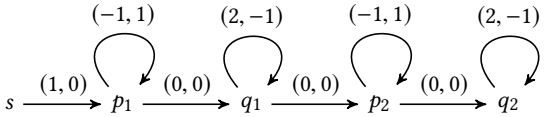
$(p, q) \in E$ if there is some transition $(p, u, q) \in T$. We say that a VASS V is *flat* if for each state $q \in Q$ there is at most one state-cycle in V which contains q. In other words a VASS is flat if there are no nested cycles in its state structure. If numbers in transitions of a VASS are encoded in unary then we call it a *unary VASS*. Similarly a *binary VASS* is a VASS with transitions encoded in binary.

Counter programs. A very useful formalism to describe some VASSes are counter programs. A *counter program* is a sequence of instructions of the form either $x \mathrel{+}= a$ or loop P, where P is another counter program. Such a counter program with d counters can be transformed in a natural way to a corresponding d-VASS. Thus in the rest of the paper in many places we use terms VASS and counter programs almost interchangeably. A precise definition can be found in [6], we recall here examples provided in [6].

Example 2.1. The following counter program

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

represents the 2-VASS presented below, state names are chosen arbitrarily.

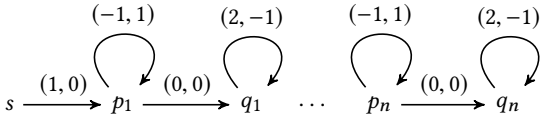


We often use macro **for** $i := 1$ **to** n **do**, by which we represent just the counter program in which the body of the for-loop is repeated n times. Notice that in such a for-loop we can use the index i in the body of the loop.

Example 2.2. The following counter program uses the macro **for**. For $n = 2$ it is equivalent to the above example.

```
1: x += 1
2: for i := 1 to n do
3:     loop
4:         x -= 1    y += 1
5:     loop
6:         x += 2    y -= 1
```

The counter program represents the following 2-VASS.



Sometimes we add to counter programs an instruction P_1 or P_2, where P_1 and P_2 are counter programs. It is easy to see that such an instruction can be as well easily simulated by a nondeterministic choice in VASSes.

Bounded counter automata. A counter automaton is a VASS with special zero-test transitions, which can be fired only if a particular counter has value exactly zero. It is folklore that the reachability problem for counter automata is undecidable in general. However restricted versions of the problem are natural problems complete for natural complexity classes. We say that a run of a counter automaton is *B-bounded* if *the sum of all the counters* on that run has values smaller than B. Notice that here we use a slightly unusual notion of boundedness: we demand the sum of all the counters to be bounded by B, not every single counter by itself. This is however only a small technical change. A run is *accepting* if it starts in the distinguished initial state with all the counters equal to zero and finishes in the distinguished accepting state also with all the counters equal to zero. Consider the following problem:

The f-bounded reachability problem for d-counter automata
Input: A d-counter automaton $\mathcal{A}$, number $n \in \mathbb{N}$ given in unary
Question: Does $\mathcal{A}$ have an $f(n)$-bounded accepting run?

The following theorem is folklore. The proof can be found in [11] (Theorem 3.1) while in [20] (Section 4.1) it is argued that small modifications in the definition of the Tower function do not change the class.

Theorem 2.3. *The f-bounded reachability problem for three-counter automata is* Tower*-complete for $f(n) = \mathrm{Tower}(n)$ defined as* $\mathrm{Tower}(1) = 2$, $\mathrm{Tower}(n+1) = 2^{\mathrm{Tower}(n)}$ *for any $n > 1$.*

Theorem 2.3 will be used in the Tower-hardness proof in Section 7. Actually in the case of $f = \mathrm{Tower}$ even the problem for two-counter automaton is Tower-complete, but this simplification is not needed in our construction.

For the PSpace-hardness and ExpSpace-hardness proofs in Sections 5 and 6, respectively we need a more subtle problem. We call a counter automata to be *B-bounded* if all its accepting runs are B-bounded. Let us consider the following promise problem:

The reachability problem for f-bounded d-counter automata
Input: An $f(n)$-bounded d-counter automaton $\mathcal{A}$, number $n \in \mathbb{N}$ given in unary
Question: Does $\mathcal{A}$ have an accepting run?

Notice that the assumption of f-boundedness makes the reachability problem for f-bounded counter automata easier than the problem of f-bounded reachability for not necessarily bounded counter automata. Indeed, if one can check f-bounded reachability for any counter automata then in particular for f-bounded counter automata, for which it is equivalent to the reachability problem. Thus the following theorem is harder to prove in our setting of a promise problem than in the more classical scenario.

Theorem 2.4. *The reachability problem for f-bounded d-counter automata is*

(1) PSpace*-hard for $f(n) = 2^n$ and $d = 2$*
(2) ExpSpace*-hard for $f(n) = 2^{2^n}$ and $d = 3$.*

The proof of Theorem 2.4 is omitted here, it can be found in the arxiv version of the paper [8].

In the next two paragraphs we present two different techniques, which can be used to simulate zero-tests in bounded counter automata by a VASS without zero-tests.

Controlling-counter technique. Here we describe the technique of controlling-counter presented in [6]. The essence of this technique is to add a new counter, called *controlling-counter*, which is modified in an appropriate way in the existing transitions and demanded to have value zero in both source and target configurations of the run. This enforces that some other counters need to have zero values in particular configurations along the run. If a counter is forced to be zero at some moment of the run we say that a *zero-test* is performed on that counter at this moment or it is *zero-tested*.

Assume that configurations $c_1, \ldots, c_n$ are some of the configurations on run ρ from configuration src to configuration trg and let

$$c_0 \xrightarrow{\rho_1} c_1 \xrightarrow{\rho_2} \ldots \xrightarrow{\rho_n} c_n \xrightarrow{\rho_{n+1}} c_{n+1}.$$

Let counter x have value zero at both source c_0 and target c_{n+1} of the run ρ and let values of counter x in configurations $c_1, \ldots, c_n$ be $x_1, \ldots, x_n$ respectively, namely $c_i[x] = x_i$ for all $i \in [1, n]$. Let x'_i be the effect of run ρ_i on counter x, namely $x'_1 = x_1$ and $x'_i = x_i - x_{i-1}$ for $i \in [2, n]$. Clearly in order to assure $x_1 = x_2 = \ldots = x_n = 0$ it is enough to assure $x_1 + \ldots + x_n = 0$. Notice that for each $i \in [1, n]$ we have $x_i = x'_1 + x'_2 + \ldots + x'_i$. Therefore

$$x_1 + \ldots + x_n = nx'_1 + (n-1)x'_2 + \ldots + 2x'_{n-1} + x'_n.$$

Thus if there is a controlling-counter y with the property that $c_0[y] = 0$ and for each $i \in [1, n]$ we have $\mathrm{eff}(\rho_i)[y] = (n + 1 - i) \cdot \mathrm{eff}(\rho_i)[x]$ then we have that

$$c_{n+1}[y] = nx'_1 + (n-1)x'_2 + \ldots + 2x'_{n-1} + x'_n = x_1 + \ldots + x_n.$$

Therefore $\mathrm{trg}[y] = 0$ implies that $c_i[x] = 0$ for all $i \in [1, n]$.

This idea can be extended to one counter controlling many counters. Here we recall Lemma 10 from [6] (see also arxiv version [7]) stating this generalised version, which will be used in our proofs.

Lemma 2.5. *Let* $\mathrm{src} \xrightarrow{\rho} \mathrm{trg}$ *be a run of a* $(d+1)$*-VASS* V *and let* $\mathrm{src} = c_0, c_1, \ldots, c_{n-1}, c_n = \mathrm{trg}$ *be some of the configurations on* ρ*. Let* ρ_j *for* $j \in [1, n]$ *be the parts of the run* ρ *starting in* c_{j-1} *and finishing in* c_j*, namely*

$$c_0 \xrightarrow{\rho_1} c_1 \xrightarrow{\rho_2} \ldots \xrightarrow{\rho_{n-1}} c_{n-1} \xrightarrow{\rho_n} c_n.$$

Let $S_1, \ldots, S_d \subseteq [0, n]$ *be the sets of indices of* c_j*, in which we want to zero-test counters numbered* $1, \ldots, d$*, respectively and let* $N_{j,i} = |\{k \geq j \mid k \in S_i\}|$ *for* $i \in [1, d], j \in [0, n]$ *be the number of zero-tests, which we want to perform on the* i*-th counter starting from configuration* c_j *(in other words after the run* ρ_j *for* $j > 0$*). Then if:*

(1) $\mathrm{src}[d+1] = \sum_{i=1}^{d} N_{0,i} \cdot \mathrm{src}[i]$*;*
(2) *for each* $j \in [1, n]$ *we have* $\mathrm{eff}(\rho_j, d+1) = \sum_{i=1}^{d} N_{j,i} \cdot \mathrm{eff}(\rho_j, i)$*; and*
(3) $\mathrm{trg}[d+1] = 0$

then for each $i \in [1, d]$ *and for each* $j \in S_i$ *we have* $c_j[i] = 0$.

Multiplication triples technique. The technique of multiplication triples was introduced in [4]. If values of counter x along run ρ are upper-bounded by B then we say that x is B*-bounded* on ρ. The essence of this idea is that a VASS starting with some three counters b, c and d having values B, C and BC, respectively, can perform $C/2$ zero-tests on a B-bounded counter.

Let us introduce a macro **flush**(x, y, z), which stands for a counter program:

1: **loop**
2: $\quad x \mathrel{-}= 1 \quad y \mathrel{+}= 1 \quad z \mathrel{-}= 1$

In other words **flush**(x, y, z) transfers value of counter x to counter y (but maybe not the whole value) while keeping value $x + y$ constant and decreases counter z by the transferred value.

Now it is easy to see how a zero-test on a B-bounded counter x can be performed. Assume as above that values of counters (b, c, d) are (B, C, BC) and initial value of x is 0. Then initially $x + b = B$ and as x is B-bounded we can keep this invariant along the run by decreasing b when x is increased and increasing b when x is decreased. Then zero-test on x is performed as follows:

1: **flush**(b, x, d)
2: **flush**(x, b, d)
3: c −= 2

In order to see that the above counter program indeed zero-tests x notice that maximal decrease of d during this program is $2B$ and it is so only if $x = 0$, $b = B$ at the beginning of the program and both flushes were fully realised (so in particular $x = 0$ also at the end of the program). Counter c is decreased by 2 in that program, therefore it can be fired at most $C/2$ times, as the initial value of c equals C. Thus in order to reach $d = 0$ at the end of the program each firing of zero-test must result in decreasing d by exactly $2B$. This in turn implies that zero-test can be indeed fired only if $x = 0$.

An extension of this technique to many counters zero-tested by the use of just one triple (b, c, d) was introduced in [6] and elegantly described by Lasota in [13]. We recall the argument here in order to be self-contained. Assume now that we have m counters $x_1, \ldots, x_m$ which all have value zero at the beginning of the counter program and their sum $x_1 + \ldots + x_m$ is bounded by B along the run. Then triple (B, C, BC) allows for $C/2$ zero-tests on any of $x_1, \ldots, x_m$. We show how to perform zero-test on counter x_1, zero-testing other counters is very similar, we comment about it in a moment. Notice that in lines 1-5 we are flushing values from counters with bigger indices to counters with smaller

Algorithm 1

1: **flush**(x_2, x_1, d)
2: **flush**(x_3, x_2, d)
3: ...
4: **flush**(x_m, x_{m-1}, d)
5: **flush**(b, x_m, d)
6: **flush**(x_m, b, d)
7: **flush**(x_{m-1}, x_m, d)
8: ...
9: **flush**(x_2, x_3, d)
10: **flush**(x_1, x_2, d)
11: c −= 2

indices and in lines 6-10 we do the same process backwards.

The main idea is similar as above: we argue that d can be decreased maximally by $2B$ by the zero-test program and if it is decreased exactly by $2B$ then $x_1 = 0$ at the moment of zero-test and values of all the counters x_i and b are the same before and after the zero-test. Clearly the rest of the argument works as before, so it suffices to show the above property. Let us denote for a moment counter b by x_{m+1}, let a_i be the value of x_i at the beginning of the program and a'_i be the value of x_i after **flush** in line 5. Clearly $a_1 + \ldots + a_m + a_{m+1} = a'_1 + \ldots + a'_m + a'_{m+1} = B$. Notice that the total decrease of d in lines 1-5 is bounded by $a_2 + \ldots + a_{m+1}$ and total decrease of d in lines 6-10 is bounded by $a'_1 + \ldots + a'_m$. Therefore total decrease is bounded by: $a_2 + \ldots + a_{m+1} + a'_1 + \ldots + a'_m = B - a_1 + B - a'_{m+1} = 2B - (a_1 + a'_{m+1})$. Thus clearly total decrease of z is at most $2B$ and it equals $2B$ if: 1) $a_1 = a'_{m+1} = 0$; and 2) all the flushes are fully realised.

One can easily see that if all the flushes are fully realised and $a_1 = 0$ then final values of x_i are the same as the original ones, so the zero-test indeed works as required. In order to zero-test counter different than x_1, say x_i, we perform the same procedure, but we apply flushes in different order so that x_i takes the place of counter x_1.

Recall now that an accepting run of a bounded counter automaton is from the distinguished initial state with all counters having zero values to the distinguished final state with all counters having zero values. Thus we can summarise the reasoning described above in the following lemma.

Lemma 2.6. *For each d-counter automaton $\mathcal{A}$ which on its B-bounded accepting run fires at most C zero-tests one can construct a unary $(d+3)$-VASS V with two distinguished states q_I, q_F such that: $\mathcal{A}$ has an accepting run if and only if there is a run from $q_I(B, 2C, 2BC, 0^d)$ to $q_F(B, 0^{d+2})$ in V.*

Quadratic pairs technique. We emphasise here that in order to apply this technique we need to work with B-bounded counter automata, rather than with B-bounded runs of not necessarily bounded counter automata, in contrast to the multiplication triple technique. This is because in the multiplication triple technique the counters are checked to be bounded, while in the quadratic pairs technique the counters are not checked to be bounded, we need to know in advance that they are B-bounded. The essence of this idea is that a VASS starting with some two counters b and c having values $2B$ and $4B^2$, respectively, can perform B zero-tests on a B-bounded counter.

We first illustrate this technique for one counter x which is $B/2$-bounded and then show how to easily generalise it to more counters. Assume that values of (b, c) are (B, B^2) and the initial value of x is 0. Then initially $x + b = B$, thus we have that $(x + b)^2 = B^2 = c$. The idea of the technique is that we keep the invariant $(x + b)^2 = c$ along the run as long as all the performed zero-tests are correct. If at some moment an incorrect zero-test is fired then $(x + b)^2 < c$ and this inequality holds till the end of the run implying in particular that $0 < c$. Thus checking $c = 0$ at the end of the run shows that all the performed zero-tests were correct.

The zero-test on x is performed as follows:

1: **flush**(b, x, c)
2: **flush**(x, b, c)
3: b −= 1 c += 1

If initially $c = B^2$ and $x + b = B$ then after lines 1-2 still $x + b = B$ and $c \geq B^2 - 2B$ where the equality holds if and only if both flushes were fully realised. Thus after line 3 we have $x + b = B - 1$ and $c \leq B^2 - 2B + 1 = (B-1)^2 = (x+b)^2$ and the equality holds iff both flushes were fully realised, so in particular the zero-test was correct.

Thus after $\ell \leq B/2$ zero-tests performed on x we have $b + x = B - \ell \geq B - B/2 = B/2$. As we know that x is $B/2$-bounded then applying at most $B/2$ zero-tests on x is possible (as $x \leq B - B/2 = B/2$). We know after $\ell \leq B/2$ zero-tests that they were correct if $(x + b)^2 = c$. But after ℓ zero-tests $x + b = B - \ell$, so it is not immediately clear how to check whether the equality $(x + b)^2 = c$ holds. We check it by performing auxiliary zero-tests at the end of the run. Namely as the last step we allow for arbitrary decrease of counter x and arbitrarily many zero-tests on x with the aim of reaching $x + b = 0$. Then checking whether $(x + b)^2 = c$ boils down to checking whether $c = 0$. After each such an auxiliary zero-test the following invariant is kept: all the zero-tests are correct only if $(x + b)^2 = c$, otherwise $(x + b)^2 < c$; and additionally if all the zero-tests are correct then it is possible to have $(x + b)^2 = c$. Thus in order to check whether all the zero-tests were correct it is enough to check at the very end whether $c = 0$, similarly as in the multiplication triple technique.

One can easily observe that extending this technique to many counters $x_1, \ldots, x_m$ which are $B/2$-bounded (recall that this means that its sum is bounded by $B/2$) is straightforward. The only modification is the implementation of

zero-tests which decrease the counter c exactly by two times of the current value of $b + x_1 + \ldots + x_\ell$. This is realised exactly as in the multiplication triple technique, namely as presented in Algorithm 1. The above reasoning can be summarised in the following lemma.

Lemma 2.7. *For each B-bounded d-counter automaton $\mathcal{A}$ which on any its accepting run fires at most B zero-tests one can construct in polynomial time a unary $(d+2)$-VASS $V_{\mathcal{A}}$ with two distinguished states q_I, q_F such that the following are equivalent:*

(1) *$\mathcal{A}$ has an accepting run*
(2) *there is a run from $q_I(2B, 4B^2, 0^d)$ to $q_F(0^{d+2})$ in $V_{\mathcal{A}}$.*

Using Lemma 2.7 and Theorem 2.4 one can pretty easily get some hardness results for VASS reachability problems, namely Corollaries 2.9 and 2.10. As an intermediate tool for these results we formulate the following lemma.

Lemma 2.8. *For each B-bounded d-counter automaton $\mathcal{A}$ with s states one can construct in polynomial time a unary $(d+2)$-VASS $V_{\mathcal{A}}$ with two distinguished states q_I, q_F such that the following are equivalent:*

(1) *$\mathcal{A}$ has an accepting run*
(2) *there is a run from $q_I(\bar{B}, \bar{B}^2, 0^d)$ to $q_F(0^{d+2})$ in $V_{\mathcal{A}}$ where $\bar{B} = 2sd \cdot B^{d-1}$.*

Proof. First observe that if there is an accepting run of the counter automaton $\mathcal{A}$ then there is also an accepting run with no repeating configuration. Notice that the number of zero-tests in a run with no repeating configuration is bounded by the total number of B-bounded configurations in $\mathcal{A}$ with at least one counter equal to zero. The number of such configurations with zero counter value can be bounded by $s \cdot d \cdot B^{d-1}$. Indeed, there are at most s choices of the state of the configuration, at most d choices of the counter, which equals to zero and at most B^{d-1} choices for values of the other counters (some configurations are counted many times, but this only strengthens the bound). Thus if there is an accepting run then there is an accepting run with at most $sdB^{d-1} = \bar{B}/2$ zero-tests performed for $\bar{B}$ defined in the lemma statement. Notice now that if $\mathcal{A}$ is B-bounded then it is also $\bar{B}$-bounded as $B \leq \bar{B}$. Then using Lemma 2.7 applied to $\bar{B}$-bounded d-counter automaton $\mathcal{A}$ finishes the proof. □

The following corollaries are immediate consequences of Theorem 2.4 and Lemma 2.8.

Corollary 2.9. *Given $n, s \in \mathbb{N}$ represented in unary and a unary 4-VASS V with distinguished states q_I, q_F it is* PSpace*-hard to decide whether there is a run from $q_I(4s \cdot 2^n, 16s^2 \cdot 4^n, 0, 0)$ to $q_F(0^4)$.*

Corollary 2.10. *Given $n, s \in \mathbb{N}$ represented in unary and a unary 5-VASS V with distinguished states q_I, q_F it is* ExpSpace*-hard to decide whether there is a run from $q_I(6s \cdot 4^{2^n}, 36s^2 \cdot 16^{2^n}, 0^3)$ to $q_F(0^5)$.*

3 OVERVIEW

Here we provide short sketches of the proofs of Theorems 1.1, 1.2, 1.3 and 1.4. In the following sections we prove these theorems in detail. Let us emphasise which techniques are used in which proofs. Let us denote the controlling-counter technique by (CC), the multiplication triple technique by (MT) and the quadratic pairs technique by (QP). Then to prove Theorem 1.1 we use (CC), to prove Theorem 1.2 we use (CC) and (QP), to prove Theorem 1.3 we use (MT) and (QP) and to prove Theorem 1.4 we use (CC) and (MT).

Proof of Theorem 1.1: NP-hardness in unary, flat 4-VASSes. This is the easiest proof out of the four presented ones. We reduce from the Subset Sum problem asking whether there is a subset of the set $\{s_1, \ldots, s_n\} \subseteq \mathbb{N}$ which sums up to a given number $s \in \mathbb{N}$. The main challenge is that numbers s_i and s in Subset Sum are encoded in binary, while transitions in our 4-VASS are encoded in unary. We use VASSes very similar to the one from Example 2.2 in order to be able to obtain exponential counter values out of unary encoded numbers in VASS transitions. If we add the third counter, which is a controlling-counter, we are able to construct a flat, unary 3-VASS, which produces a number s_i on a distinguished counter. Then we reduce the Subset Sum problem as follows: we have a distinguished counter called the *summing* counter, to which we first add value s using a 3-VASS (then altogether we have four counters). Then for each $i \in [1, n]$ we construct a 3-VASS, which produces number s_i and then nondeterministically: either subtracts s_i from the summing counter or does not touch the summing counter. After processing all the 3-VASSes for $s_1, \ldots, s_n$ we check the summing counter to be zero: it is easy to observe that there exists a run reaching zero if and only if the instance of Subset Sum is positive.

Proof of Theorem 1.2: PSpace-hardness in unary 5-VASSes. By Corollary 2.9 to show PSpace-hardness it is enough to design for given $s, n \in \mathbb{N}$ a 5-VASS, or in other words a five counter program of size polynomial in s and n which constructs on its first four counters (x_1, x_2, x_3, x_4) values $(4s \cdot 2^n, 16s^2 \cdot 4^n, 0, 0)$ under the condition that $x_5 = 0$. Indeed, then checking whether it reaches valuation 0^5 at its end is PSpace-hard by Corollary 2.9. We construct the pair $(4s \cdot 2^n, 16s^2 \cdot 4^n)$ on (x_1, x_2) in the following way. We start with $(x_1, x_2) = (4s, 16s^2)$ and then exactly n times multiply x_1 by 2 and x_2 by 4. The multiplications are realised as flushing x_1 or x_2 to x_3 and then flushing it back from x_3 to x_1 or x_2 simultaneously multiplying it by 2 or 4, respectively. We assume that multiplications are exact by forcing appropriate counters x_1, x_2 and x_3 to be exactly zero after the flushes. This is realised by the use of the controlling-counter technique, the counter x_5 controls x_1, x_2 and x_3 thus if $x_5 = 0$ at the end of the run then all the multiplications were indeed exact. Thus indeed after this phase the five counters have values $(4s \cdot 2^n, 16s^2 \cdot 4^n, 0, 0, x_5)$ under the condition that $x_5 = 0$.

Proof of Theorem 1.3: ExpSpace-hardness in binary 6-VASSes. The idea is similar to the proof of Theorem 1.2. By Corollary 2.10 to show ExpSpace-hardness it is enough to design for given $s, n \in \mathbb{N}$ a 6-VASS, or in other words a six counter program of size polynomial in s and n which constructs on its first five counters $(x_1, x_2, x_3, x_4, x_5)$ values $(6s \cdot 4^{2^n}, 36s^2 \cdot 16^{2^n}, 0, 0, 0)$ under the condition that $x_6 = 0$. Indeed, then checking whether it reaches valuation 0^6 at its end is ExpSpace-hard by Corollary 2.10. We construct the pair $(6s \cdot 4^{2^n}, 36s^2 \cdot 16^{2^n})$ on (x_1, x_2) in the following way. We start from setting $(x_1, x_2) = (6s, 36s^2)$ and then 2^n times we perform the following: 1) flush x_1 to x_3, 2) flush back x_3 to x_1 while multiplying by 4, 3) flush x_2 to x_3, 4) flush back x_3 to x_2 while multiplying by 16. After each flush we perform a zero-test to assure that the flush was full. Additionally after these multiplications we perform a zero-test on x_4. This time we cannot use the controlling-counter technique easily, as the number of zero-tests is equal to $4 \cdot 2^n + 1$, which is super-linear. In order to simulate $4 \cdot 2^n + 1$ zero-tests (even on big counters) we use the multiplication triples technique. We produce triple $(B, 8 \cdot 2^n + 2, B \cdot (8 \cdot 2^n + 2))$ on counters (x_4, x_5, x_6) for some big guessed value $B \in \mathbb{N}$ and use it to implement $4 \cdot 2^n + 1$ zero-tests on B-bounded counters. Using this triple and checking that at the end of the run counter x_6 has value zero guarantees that indeed all the flushes were full. So after this phase we indeed have values $(6s \cdot 4^{2^n}, 36s^2 \cdot 16^{2^n}, 0, 0, 0)$ on the first five counters.

Proof of Theorem 1.4: Tower-hardness in binary 8-VASSes. We reduce from the Tower(n)-bounded reachability problem for three-counter automata. This construction uses both the multiplication triples technique and the controlling-counter technique in an interplay. The aim is, similarly as in the proof of Theorem 1.3, to construct a triple of the form

(Tower(n), C, $C \cdot$Tower(n)) for appropriately big C. We first show that there exists a 7-VASS, which is a 2^k-amplifier (more precisely speaking an f-amplifier for $f(k) = 2^k$). The notion of an amplifier was defined in [6], we recall it in Section 7. Roughly speaking a 2^k-amplifier from a triple (B, C, BC) produces a triple $(2^B, C', C' \cdot 2^B)$ for some guessed value $C' \in \mathbb{N}$. In short words the construction of the amplifier works as follows: we start from a triple $(1, C', C')$ for guessed C' and then using the triple (B, C, BC) multiply exactly $B/8$ times the first and the third coordinate of the triple $(1, C', C')$ by exactly $2^8 = 256$. After these multiplications we therefore get a triple $(2^{8 \cdot B/8}, C', C' \cdot 2^{8 \cdot B/8}) = (2^B, C', C' \cdot 2^B)$ as needed. Using the trick from the previous paragraph we are able to achieve it by the use of just one additional counter and therefore the 2^k-amplifier has only seven counters. We use then the eighth counter as a controlling-counter: we compose the 2^k-amplifier exactly n times and assure by the controlling-counter that the appropriate counters in the places of composition have value exactly zero, which guarantees that composition works correctly. As the number of compositions is linear this can be achieved by a single controlling-counter and thus the whole construction uses only eight counters.

4 NP-HARDNESS FOR 4-VASSES

We reduce from the following problem:

Subset Sum problem
Input Number $s_0 \in \mathbb{N}$, set of numbers
$S = \{s_1, \ldots, s_n\} \subseteq \mathbb{N}$, all encoded in binary
Question Is there a subset of S summing up exactly to s_0?

For an instance of Subset Sum we design a four-counter program P and show that there is a run of P starting in 0^4 and finishing in 0^4 iff the instance is positive. Our counter program has four counters: x and y, which will be used to generate numbers s_i, the summing counter z and the controlling counter c. The counter program P consists of counter program P_0 and for each $i \in [1, n]$ counter programs P_i and P_i' in the following way:

1: P_0
2: **for** $i := 1$ **to** n **do**
3: $\quad P_i \quad$ or $\quad P_i'$

The counter program P_0 will be constructed such that in every run reaching 0^4 its effect on counter z is exactly s_0. On the other hand in such runs the effect of counter programs P_i on z for $i \in [1, n]$ will be exactly $-s_i$, while P_i' will have no effect on z.

Let us assume that 2^k is the smallest power of 2 strictly bigger than all the numbers $s_0, s_1, \ldots, s_n$, namely all s_i can be encoded in k bits. For each $i \in [0, n]$ let $s_i = \langle b^i_{k-1} \cdots b^i_0 \rangle_2$ be the bit representation of s_i. We show now how the counter program P_0 is constructed. For simplicity we first do not include the controlling counter c in the program.

1: $x \mathrel{+}= b^0_{k-1}$
2: **for** $j := k-2$ **downto** 0 **do**
3: $\quad$ **loop**
4: $\quad\quad x \mathrel{-}= 1 \quad y \mathrel{+}= 1$
5: $\quad$ **loop**
6: $\quad\quad x \mathrel{+}= 2 \quad y \mathrel{-}= 1$
7: $\quad x \mathrel{+}= b^0_j$

8: **loop**
9: $x \mathrel{-}= 1 \quad z \mathrel{+}= 1$

One can easily see that if loops in lines 3-4, 5-6 and 8-9 are fired maximal possible number of times then the final values of (x, y, z) are $(0, 0, s_0)$. Therefore to guarantee that $z = s_0$ after the program P_0 it is enough to assure that $x = 0$ each time line 4 is left, $y = 0$ each time line 6 is left and $x = 0$ when line 9 is left. In order to achieve that we use the counter c. However its behaviour depends as well on programs P_i and P'_i, so we first present them, also without the controlling counter. We first show the counter program P_i also without the counter c.

1: $x \mathrel{+}= b^i_{k-1}$
2: **for** $j := k-2$ **downto** 0 **do**
3: **loop**
4: $x \mathrel{-}= 1 \quad y \mathrel{+}= 1$
5: **loop**
6: $x \mathrel{+}= 2 \quad y \mathrel{-}= 1$
7: $x \mathrel{+}= b^i_j$
8: **loop**
9: $x \mathrel{-}= 1 \quad z \mathrel{-}= 1$

The only difference between P_i for $i \geq 1$ and P_0 is that in P_i in the loop in lines 8-9 the counter z is decreased, while in P_0 it was increased. One can easily observe that if P_i for $i \geq 1$ starts with valuation $(x, y, z) = (0, 0, N)$ and all the loops are iterated a maximal number of times then it finishes with valuation $(x, y, z) = (0, 0, N - s_i)$. Counter program P'_i (also with counter c ignored) is the same as P_i with the only difference that in line 9 counter z is not decreased, but kept unchanged. Intuitively the run in VASS choses to use P_i if the number s_i have to be taken into the sum and P'_i if the number s_i is not taken into the sum. We can see now that counter programs P_0, P_i and P'_i have the promised properties under the condition that counters x and y are zero in the appropriate places. In order to assure it we add the controlling counter c. One can observe that P_0, P_i and P'_i differ only on the operation done to z in line 9: in P_0 it is increase by 1, in P'_i it is increased by 0 and in P_i it is increased by -1. Therefore we write one parametrised program to represent all the three counter programs. The presented counter program $\bar{P}(i, \text{sign})$ satisifies $P_0 = \bar{P}(0, 1)$, $P_i = \bar{P}(i, -1)$ and $P'_i = \bar{P}(i, 0)$.

1: $x \mathrel{+}= b^i_{k-1} \quad c \mathrel{+}= b^i_{k-1} \cdot k(n-i+1)$
2: **for** $j := k-2$ **downto** 0 **do**
3: **loop**
4: $x \mathrel{-}= 1 \quad y \mathrel{+}= 1 \quad c \mathrel{-}= n+1-i$
5: **loop**
6: $x \mathrel{+}= 2 \quad y \mathrel{-}= 1 \quad c \mathrel{+}= (k+1)(n-i)+(j+1)$
7: $x \mathrel{+}= b^i_j \quad c \mathrel{+}= b^i_j \cdot (k(n-i)+(j+1))$
8: **loop**
9: $x \mathrel{-}= 1 \quad z \mathrel{+}= \text{sign} \quad c \mathrel{-}= k(n-i)+1$

The only parts in $\bar{P}(i, \text{sign})$, which are nontrivial to understand are the effects of transitions on the controlling counter c. Let us recall from Lemma 2.5 that if counter c controls counter x then any increment of $x \mathrel{+}= a$ should be matched by $c \mathrel{+}= Na$, where N is the number of zero-tests which are planned to be performed on x in the remaining part of the run.

It is clear that there exist appropriate changes of c, which fulfil Lemma 2.5 and they are not too big, so for an intuitive understanding of the program one does not need the next paragraph. However in order to prove that we above counter program indeed satisfies the needed conditions we need to meticulously inspect all the cases, which we do below.

In order to count the needed changes on c we need to count how many times zero-tests are performed on the controlled counters x and y. In each program $\bar{P}$ the counter y is zero-tested $k-1$ times in the line 6, while counter x is zero-tested $k-1$ times in the line 4 and once in line 9, so altogether k times. Therefore in line 1 in program $\bar{P}(i, \text{sign})$ counter x is waiting for all the tests in programs $\bar{P}$ with first parameter being $i, i+1, \ldots, n$, altogether $n-i+1$ programs $\bar{P}$. Thus the number of zero-tests waiting for x is exactly $k(n-i+1)$ and each increment $x \mathrel{+}= b^i_{k-1}$ should be matched by increment $c \mathrel{+}= b^i_{k-1} \cdot k(n-i+1)$. Similarly in line 9 in program $\bar{P}(i, \text{sign})$ counter x is waiting for $k(n-i)$ zero-tests in programs $\bar{P}$ with first parameter being $i+1, \ldots, n$ plus one last zero-test after the loop in lines 8-9 in program $\bar{P}(i, \text{sign})$. Thus $x \mathrel{-}= 1$ has to be matched with $c \mathrel{-}= k(n-i)+1$. A similar calculation shows that in line 7 increment $x \mathrel{+}= b^i_j$ has to be matched with $c \mathrel{+}= b^i_j \cdot (k(n-i)+(j+1))$. A bit more involved calculation is needed in case of lines 4 and 6, as there both counters x and y are modified, so modification of c has to reflect both changes. In line 4 counter x is waiting for $k(n-i)$ zero-tests in next programs, one zero-test in line 9 and $j+1$ zero-tests in line 4 in the further iterations of the for loops, so altogether for $k(n-i)+(j+2)$ zero-tests. In the same line counter y is waiting for $(k-1)(n-i)+(j+1)$ zero-tests, thus the total change on c is the $-k(n-i)-(j+2)+(k-1)(n-i)+(j+1) = -(n-i)-1$. Similarly one can count that in line 6 counter x awaits for $k(n-i)+(j+1)$ zero-tests, while counter y awaits for $(k-1)(n-i)+(j+1)$ zero-tests. Therefore c should be incremented by $2(k(n-i)+(j+1))-(k-1)(n-i)-(j+1) = (k+1)(n-i)+(j+1)$.

One can easily observe that all the changes performed on c are of polynomial size, therefore the reduction from Sumset Sum is indeed performed in polynomial time. Finally, one can easily see that the constructed counter program contains no nested loop constructions, so it is indeed flat, as required in the statement of the Theorem 1.1. This finishes the proof of the Theorem 1.1.

5 PSPACE-HARDNESS FOR 5-VASSES

Due to Corollary 2.9 in order to prove Theorem 1.2 it is enough to show the following lemma.

Lemma 5.1. *For each $s, n \in \mathbb{N}$ one can construct in polynomial time a unary 5-VASS of size polynomial in s and n with distinguished states q_I, q_F such that for each run from $q_I(0^5)$ to $q_F(x_1, x_2, x_3, x_4, 0)$ we have $(x_1, x_2, x_3, x_4) = (4s \cdot 2^n, 16s^2 \cdot 4^n, 0, 0)$.*

Indeed, let V_1 be the 5-VASS from Lemma 5.1 with distinguished states q^1_I, q^1_F. Our aim is to reduce the problem from Corollary 2.9 to the reachability problem in unary 5-VASSes. Let V_2 be a 4-VASS from Corollary 2.9 with distinguished states q^2_I, q^2_F for which we want to check whether there is a run from $q^2_I(4s \cdot 2^n, 16s^2 \cdot 4^n, 0, 0)$ to $q^2_F(0^4)$. Let V_2' be V_2 extended with the fifth coordinate in such a way that all the transitions have zero on this fifth coordinate. We construct now a unary 5-VASS V with distinguished states q^1_I, q^2_F which is a disjoint union of V_1 and V_2' with additional transition from q^1_F to q^2_I labelled by 0^5. It is then immediate to see that the following are equivalent:

- there is a run from $q^2_I(4s \cdot 2^n, 16s^2 \cdot 4^n, 0, 0)$ to $q^2_F(0^4)$ in V_2
- there is a run from $q^1_I(0^5)$ to $q^2_F(0^5)$ in V,

which finishes the proof of Theorem 1.2. Thus the rest of this section focuses on the proof of Lemma 5.1.

Proof of Lemma 5.1. In the proof we prefer to use the terminology of counter programs instead of VASSes, but recall that counter programs are just syntactic sugar to present VASSes in a human-readable way. In our construction

we actually do not use the counter x_4. Our aim is to construct on (x_1, x_2) values $(4s \cdot 2^n, 16s^2 \cdot 4^n)$. We start with setting (x_1, x_2) to $(4s, 16s^2)$. Then we need to multiply n times x_1 by 2 and x_2 by 4. We realise it by flushing x_1 to x_3 and then flushing it back from x_3 to x_1 while simultaneously multiplying by 2, and similarly with x_2 but multiplying it by 4. In order to assure that all the multiplications are exact we perform a zero-test after each flush, then we are sure that all the flushes are full. Before explaining how we realise zero-tests we can already present how our counter program works. Let us define the following macro **multiply**(x, y, c) for two counters x and y and number $c \in \mathbb{N}$.

1: **loop** $x \mathrel{-}= 1$ $y \mathrel{+}= 1$
2: **zero-test**(x)
3: **loop** $x \mathrel{+}= c$ $y \mathrel{-}= 1$
4: **zero-test**(y)

Using the macro **multiply**(x, y, c) we can briefly describe our counter program as follows.

1: $x_1 \mathrel{+}= 4s$ $x_2 \mathrel{+}= 16s^2$
2: **for** $i := 1$ **to** n **do**
3: **multiply**$(x_1, x_3, 2)$
4: **multiply**$(x_2, x_3, 4)$

It is easy to see that after the above counter program indeed (x_1, x_2, x_3) are equal to $(4s \cdot 2^n, 16s^2 \cdot 4^n, 0)$ as supposed. Thus it remains to explain how do we realise zero-tests. We use the controlling-counter technique described in Section 2 and used also in Section 4 (and in Section 7 later). The counter x_5 is the controlling-counter in our counter program and it controls counters x_1, x_2 and x_3. Recall that in this technique each operation on one of the controlled counters $x_i \mathrel{+}= a$ is matched by an operation of the controlling-counter $x_5 \mathrel{+}= Na$, where N is the number of zero-tests which will be performed on the counter x_i in the rest of the run after this operation. By Lemma 2.5 we know that if the value of the controlling-counter x_5 is equal to zero at the end of the run then all the zero-tests on controlled counters were correct as well. Recall also that a bit counterintuitively a zero-test on controlled counter is not reflected in the counter program by any code, the only effect of a zero-test on some counter x_i is that less zero-tests will be performed on x_i in the future, thus changes of x_i are reflected now in the controlling-counter in a slightly different way (N decreases by one). Thus the above presented counter program after implementing the zero-tests looks as follows.

1: $x_1 \mathrel{+}= 4s$ $x_2 \mathrel{+}= 16s^2$
2: **for** $i := 1$ **to** n **do**
3: **loop** $x_1 \mathrel{-}= 1$ $x_3 \mathrel{+}= 1$ $x_5 \mathrel{+}= n + 1 - i$
4: **loop** $x_1 \mathrel{+}= 2$ $x_3 \mathrel{-}= 1$ $x_5 \mathrel{-}= 2$
5: **loop** $x_2 \mathrel{-}= 1$ $x_3 \mathrel{+}= 1$ $x_5 \mathrel{+}= n - i$
6: **loop** $x_2 \mathrel{+}= 4$ $x_3 \mathrel{-}= 1$ $x_5 \mathrel{+}= 2n - 2i - 1$

Let us check carefully that the operations on the controlling-counter x_5 are correct. In line 3 the counter x_1 awaits for $n + 1 - i$ zero-tests (and is increased by -1) while the counter x_3 awaits for $2(n + 1 - i)$ zero-tests (and is increased by 1), so the counter x_5 should be increased by $(n + 1 - i) \cdot (-1) + 2(n + 1 - i) \cdot 1 = n + 1 - i$. In line 4 the counter x_1 awaits for $n - i$ zero-tests and the counter x_3 awaits for $2(n + 1 - i)$ zero-tests, so the counter x_5 should be increased by $(n - i) \cdot 2 + 2(n + 1 - i) \cdot (-1) = -2$. In line 5 the counter x_2 awaits for $n + 1 - i$ zero-tests and the counter x_3 awaits for $2(n + 1 - i) - 1$ zero-tests, so the counter x_5 should be increased by $(n + 1 - i) \cdot (-1) + (2(n + 1 - i) - 1) \cdot 1 = n - i$. In line 6 the counter x_1 awaits for $n - i$ zero-tests and the counter x_3 awaits for $2(n + 1 - i) - 1$ zero-tests, so the counter

x_5 should be increased by $(n-i)\cdot 4 + (2(n+1-i)-1)\cdot(-1) = 2n-2i-1$. So the above counter program satisfies the conditions of Lemma 2.5. Thus by Lemma 2.5 indeed if $x_5 = 0$ at the end of the counter program then (x_1, x_2, x_3, x_4) have values $(4s\cdot 2^n, 16s^2\cdot 4^n, 0, 0)$ which finishes the proof. □

6 EXPSPACE-HARDNESS FOR 6-VASSES

Similarly as in Section 5 using Corollary 2.10 and the following Lemma 6.1 one can easily derive Theorem 1.3. As the argument is totally analogous to the argument in the beginning of Section 5 and easy to see we do not repeat it here.

Lemma 6.1. *For each $s, n \in \mathbb{N}$ one can construct in polynomial time a binary 6-VASS with distinguished states q_I, q_F such that for each run from $q_I(0^6)$ to $q_F(x_1, x_2, x_3, x_4, x_5, 0)$ we have $(x_1, x_2, x_3, x_4, x_5) = (6s\cdot 4^{2^n}, 36s^2\cdot 16^{2^n}, 0, 0, 0)$.*

The rest of this section focuses on the proof of Lemma 6.1.

Proof of Lemma 6.1. The main idea is quite similar to the proof of Lemma 5.1, namely to set at the beginning $x_1 = 6s$, $x_2 = 36s^2$ and then 2^n times multiply the counters x_1 and x_2 by values 4 and 16, respectively. The multiplications are realised by the use of the counter x_3, namely we use the macro **multiply** from the proof of Lemma 5.1 with the counter x_3 as the second argument. After these multiplications we also zero-test counter x_4 once, the purpose of it will be explained later. The main difference between the proofs of Lemmas 5.1 and 6.1 is the way we implement zero-tests. In the proof of Lemma 5.1 we used the controlling-counter technique, but here it is not sufficient and we are forced to use the multiplication triples techniques which is more powerful, but uses more counters. We are ready to present the demanded counter program, it roughly speaking looks as follows.

1: x_1 += $6s$ x_2 += $36s^2$
2: **for** i := 1 **to** 2^n **do**
3: **multiply**$(x_1, x_3, 4)$
4: **multiply**$(x_2, x_3, 16)$
5: **zero-test**(x_4)

Here however we cannot expand the macro **for** as it would result in a counter program of exponential size. We need therefore to show how to implement zero-tests and how to iterate exactly 2^n the **multiply** instructions. Notice that we need to perform exactly $4\cdot 2^n$ zero-tests inside the **multiply** instructions and then one zero-test on counter x_4. If our counters are B-bounded for some B then thanks to Lemma 2.6 in order to simulate these $4\cdot 2^n + 1$ zero-tests it is enough to have a triple $(B, 8\cdot 2^n + 2, (8\cdot 2^n + 2)\cdot B)$. The best bound B such that all $x_1 + x_2 + x_3 + x_4 \leq B$ through the whole run is $B = 6s\cdot 4^{2^n} + 36s^2\cdot 16^{2^n}$. At first glance it looks a bit like a problem, as it seems that we need one more time to produce triples with doubly-exponential entries. We perform here however a twist in thinking about triples (B, C, BC), which was one of the main conceptual contributions of [13]. Namely, the triple (B, C, BC) with small B and big C can be both used to implement $C/2$ zero-tests on B-bounded counters and to implement $B/2$ zero-tests on C-bounded counters. In other words: we do not need to compute our big bound B, we just need to guess it nondeterministically. If the guess is too small then the corresponding run will not be accepting, but it is important that there exists an appropriate guess for B.

Thus our counter program first prepares a triple (B, C, BC) on the counters x_4, x_5 and x_6.

1: x_5 += $8\cdot 2^n + 2$
2: **loop**
3: x_4 += 1 x_6 += $8\cdot 2^n + 2$

Notice that n is given in unary, as in the reachability problem for bounded three-counter automata n is given in unary. Thus the above fragment of counter program is of polynomial size, as $8 \cdot 2^n + 2$ can be represented in polynomially many bits, recall that our 6-VASS is binary. It is actually the only place where we use the fact that we deal with binary VASSes, but it is an important one. After this program fragment, the valuation of (x_4, x_5, x_6) equals $(B, 8 \cdot 2^n + 2, (8 \cdot 2^n + 2) \cdot B)$ for some $B \in \mathbb{N}$. Then the zero-tests **zero-test**(x_i) for $i \in \{1, 2, 3\}$ use this triple to perform at most B zero-tests on those counters.

The last part, which remains to be shown is how to afford that the for-loop is fired exactly 2^n times. Recall now that our triple $(B, 8 \cdot 2^n + 2, (8 \cdot 2^n + 2) \cdot B)$ guarantees that in order to reach at the end of the program some value $(B', 0, 0)$ we need to fire exactly $4 \cdot 2^n + 1$ zero-tests, which means that the loop needs to be repeated exactly 2^n times. Finally observe that after firing these zero-tests we obtain counter values of (x_4, x_5, x_6) of the form $(B', 0, 0)$, where $B' + 6s \cdot 4^{2^n} + 36s^2 \cdot 16^{2^n} = B$. However we actually need x_4 to be exactly zero at this point. In order to assure it we apply a last zero-test on counter x_4. This zero-test does not differ from zero-tests on counters x_1, x_2 and x_3 at all, notice that during the run we actually keep the invariant $x_1 + x_2 + x_3 + x_4 = B$, so all the counters x_i for $i \in [1, 4]$ behave symmetrically with respect to zero-testing. Notice now that the only guess for B which allows for $x_4 = 0$ at this point is $B = 6s \cdot 4^{2^n} + 36s^2 \cdot 16^{2^n}$, in all the other cases the run of our counter program will not reach $x_6 = 0$. Summarising, the counter program has the following code.

1: x_5 += $8 \cdot 2^n + 2$
2: **loop**
3: x_4 += 1 x_6 += $8 \cdot 2^n + 2$
4: x_1 += $6s$ x_2 += $36s^2$
5: **loop**
6: **multiply**$(x_1, x_3, 4)$
7: **multiply**$(x_2, x_3, 16)$
8: **zero-test**(x_4)

Thus checking whether $x_6 = 0$ at the end of the program indeed assures that the other values are equal $x_1 = 6s \cdot 4^{2^n}$, $x_2 = 36s^2 \cdot 16^{2^n}$ and $x_3 = x_4 = x_5 = 0$. □

7 TOWER-HARDNESS FOR 8-VASSES

Similarly as in Section 6 due to Lemma 2.6 it is enough to prove the following lemma.

Lemma 7.1. *For each $n \in \mathbb{N}$ one can construct in polynomial time a unary 8-VASS with distinguished states q_I, q_F such that for each run from $q_I(0^8)$ to $q_F(x_1, x_2, x_3, x_4, x_5, x_6, x_7, 0)$ we have $x_1 = \mathrm{Tower}(n)$, $x_3 = x_2 \cdot \mathrm{Tower}(n)$ and $x_4 = x_5 = x_6 = x_7 = 0$.*

The proof that Lemma 7.1 implies Theorem 1.4 is even simpler than the corresponding one for ExpSpace-hardness for 6-VASSes as we do not need to prove any bound on the number of needed zero-tests; it follows immediately from Theorem 2.3 and Lemma 2.6.

Before showing Lemma 7.1 we recall first the notion of amplifier and prove a suitable lemma. Here we define an amplifier in a restrictive setting, especially adjusted to our application. In particular instead of talking about f-amplifier for $f(k) = 2^k$ we just talk about amplifiers, as we only apply here the notion of amplifiers to this particular function f.

A 7-VASS V together with its two distinguished states q_{in} and q_{out} is an *amplifier* if the following holds:

- if $q_{\text{in}}(B, C, BC, 0^4) \longrightarrow q_{\text{out}}(0^4, B', C', D')$ in V then $B' = 2^n$ and $D' = B' \cdot C'$; and
- for each $C' \in \mathbb{N}$ there exists $C \in \mathbb{N}$ such that $q_{\text{in}}(B, C, BC, 0^4) \longrightarrow q_{\text{out}}(0^4, 2^B, C', 2^B \cdot C')$ in V.

We first show the following lemma.

Lemma 7.2. *There exists an amplifier.*

Proof. We denote counters of the constructed 7-VASS as x_i, for $i \in [1, 7]$. The main idea of the amplifier is that we initialise (x_5, x_6, x_7) as $(1, C', C')$ for some guessed $C' \in \mathbb{N}$ and then multiply $B/8$ times both x_5 and x_7 by $2^8 = 256$. The multiplication uses counter x_4, similarly as in the proof of Lemma 6.1. Namely we use the macro **multiply**(x, y, c) for two counters x and y and a number $c \in \mathbb{N}$ defined as follows.

1: **loop** x −= 1 y += 1
2: **zero-test**(x)
3: **loop** x += c y −= 1
4: **zero-test**(y)

The zero-tests for x_i, where $i \in [4, 7]$ will use the triple (B, C, BC) on counters (x_1, x_2, x_3) in order to be implemented. The code of the amplifier is the following.

1: x_5 += 1
2: **loop** x_6 += 1 x_7 += 1
3: **loop**
4: **multiply**$(x_5, x_4, 256)$
5: **multiply**$(x_7, x_4, 256)$
6: **loop** x_2 −= 1

In each iteration of the loop we fire exactly four zero-tests, twice on counter x_4, once on counter x_5 and once on counter x_7. Our aim is to have exactly $B/8$ iterations of the loop. By Lemma 2.6 using the triple (B, C, BC) guarantees that we can perform exactly $B/2$ zero-tests on C-bounded counters. Notice that here, similarly as in Section 6 we use triples (B, C, BC) in an unusual way: to apply small number $(B/2)$ of zero-tests on big (C-bounded) counters. As we demand that after finishing the main loop (in lines 3-5) $x_1 = x_3 = 0$ we know that exactly $B/2$ zero-tests were performed, this implies that exactly $B/8$ iterations of the main loop were performed, as in each one there are four zero-tests. Thus each run of our program, which reaches a configuration $(0^4, B', C', D')$ fulfils that $B' = 256^{B/8} = 2^B$ and $D' = B' \cdot C'$, which means that the program satisfies the first condition of being an amplifier. To show that the second condition of being an amplifier also holds observe that for each C' it suffices to have $C \geq C'(1 + 2^B)$ and such a $C \in \mathbb{N}$ surely always exists, which finishes the proof of Lemma 7.2. □

We are now ready to prove Lemma 7.1.

Proof of Lemma 7.1. In the proof for each $n \in \mathbb{N}$ we construct an eight counter program P_n representing the VASS demanded in the statement of Lemma 7.1. Very roughly speaking P_n just uses n times the amplifier from Lemma 7.2.

Let $\mathbf{ampl}_i$ denote the code of amplifier from Lemma 7.2 with additional counter x_8 with updates depending in the parameter i (to be specified later). The code of the counter program P_n is roughly speaking the following.

1: x_1 += 1
2: **loop**
3: x_2 += 1 x_3 += 1
4: **for** i := 1 **to** n **do**
5: $\mathbf{ampl}_i$
6: **zero-test**(x_1, x_2, x_3, x_4)
7: **loop** x_5 −= 1 x_1 += 1
8: **loop** x_6 −= 1 x_2 += 1
9: **loop** x_7 −= 1 x_3 += 1
10: **zero-test**(x_5, x_6, x_7)

Recall here that the **for**-operator is a macro here, so in fact the program P_n has the lines 5-10 repeated n times with different values of i in $\mathbf{ampl}_i$. Notice also that the eighth counter x_8 seemingly does not occur in the program. It is used for implementing the zero-tests, we explain its role in a moment. Observe however first that if the **zero-test** procedures correctly zero-test the listed counters then the program P_n performs what it is supposed to perform. After lines 1-3 values of the seven first counters are $(1, C_0, C_0, 0^4)$ for some arbitrarily guessed $C_0 \in \mathbb{N}$. It is easy to show by induction on i that after i iterations of the for-loop the counters have values $(\text{Tower}(i), C_i, C_i \cdot \text{Tower}(i), 0^4)$ for some arbitrarily guessed $C_i \in \mathbb{N}$. Indeed, if after the $i-1$ iterations values were $(\text{Tower}(i-1), C_{i-1}, C_{i-1} \cdot \text{Tower}(i-1))$ then after the amplifier in line 5 of the i-th iteration and zero-testing counters x_i for $i \in [1,4]$ by definition of the amplifier counter values are $(0^4, \text{Tower}(i), C_i, C_i \cdot \text{Tower}(i))$ for some arbitrarily guessed $C_i \in \mathbb{N}$. Lines 7-10 transfer the triple $(\text{Tower}(i), C_i, C_i \cdot \text{Tower}(i))$ from counters (x_5, x_6, x_7) to counters (x_1, x_2, x_3) and thus the proof of the induction step is finished.

It remains to show how the zero-tests are implemented. We use here the controlling-counter technique summarised in Lemma 2.5. The controlling-counter technique is more useful here than the multiplication triples technique, because to implement multiplication triple technique we need three additional counters, while to implement a linear number of zero-tests we need just one additional controlling-counter. Recall that in the technique of controlling-counter we have an additional controlling-counter (in our case x_8) which starts from zero in the initial configuration. For each other counter (in our case x_i for $i \in [1,7]$) which x_8 controls each modification of this counter of the form x_i += a is matched by a modification of the controlling-counter x_8 += Na, where N is the number of zero-tests which will be applied to the counter x_i after this modification. Notice that the commands **zero-test** in the program P_n are actually not transformed into any real code in this technique, they only mark a point in the code where the controlling-counter x_8 slightly changes its behaviour. With such a modifications of x_8 we are guarantied by Lemma 2.5 that in each run in which the controlling-counter finishes with value zero all the controlled counters in all the zero-tested places indeed have value zero. Thus it is enough to add the suitable modifications of the counter x_8. Each of the counters x_i for $i \in [1,7]$ is zero-tested n times in P_n, counters x_1, x_2, x_3, x_4 in line 6 while counters x_5, x_6, x_7 in line 10. Thus we need to add in line 1 operation x_8 += n and in line 3 operation x_8 += $2n$. In the i-th iteration of the for-loop in line 7 counter x_5 awaits for $n-i+1$ zero-tests while counter x_1 awaits for $n-i$ zero-tests. This means that we need to modify counter x_8 by $(-1) \cdot (n-i+1) + 1 \cdot (n-i) = -1$. Similarly in lines 8 and 9 we also need to add the operation x_8 −= 1. Similarly in the program $\mathbf{ampl}_i$ we need to add for each operation x_i += a where $i \in [1,7]$ an operation x_8 += $(n-i+1) \cdot a$ as

each such counter awaits for $(n - i + 1)$ zero-tests ($i - 1$ of the zero-tests where already performed in the previous $i - 1$ iterations of the for-loop). By Lemma 2.5 we are guarantied that the **zero-test** operations are correct, thus indeed at the end of P_n we finish the counter valuation $(0^4, \text{Tower}(i), C_i, C_i \cdot \text{Tower}(i), 0)$, which finishes the proof of Lemma 7.1. For clarity we add the code of the counter program P_n below.

1: x_1 += 1 x_8 += n
2: **loop**
3: x_2 += 1 x_3 += 1 x_8 += $2n$
4: **for** i := 1 **to** n **do**
5: x_5 += 1 x_8 += $(n - i + 1)$
6: **loop**
7: x_6 += 1 x_7 += 1 x_8 += $(n - i + 1)$
8: **loop**
9: **multiply**$(x_5, x_4, 256)$
10: **multiply**$(x_7, x_4, 256)$
11: **loop** x_2 −= 1 x_8 −= $(n - i + 1)$
12: **loop** x_5 −= 1 x_1 += 1 x_8 −= 1
13: **loop** x_6 −= 1 x_2 += 1 x_8 −= 1
14: **loop** x_7 −= 1 x_3 += 1 x_8 −= 1

where inside the **multiply** operation in the i-th iteration of the for-loop also the operations x_i += a for $i \in [1, 7]$ are enriched with operations x_8 += $(n - i + 1) \cdot a$. □

8 FUTURE RESEARCH

General remarks. An obvious future goal is to try to get tight complexity bounds for the reachability problem for fixed dimensional VASSes, where there are still a lot of question marks. For each $d \in [3, 7]$ we do not know whether it is elementary or not, moreover for $d \in [3, 5]$ for binary encoding we still cannot exclude that the problem is PSpace-complete, exactly like for 2-VASSes [2]. In order to exclude PSpace-completeness it would be helpful to come up with some say ExpSpace-hard or ExpTime-hard problem, which does not involve bounded counter automata but is anyway convenient for a hardness proof; similarly as Subset Sum is convenient for NP-hardness proof for unary flat 4-VASSes.

One reason why proving hardness results in low dimensional VASSes is so hard may be because of the use of multiplication triple technique: we need there three counters to lift our constructions one level higher. Some partial solution to that problem is the technique of quadratic pairs proposed by us in the paper. It would be interesting to pursue the research in that direction and try to design some other ways of efficient zero-testing.

Short paths. A common technique to prove upper complexity bounds on the reachability problem in VASSes is to show that if there is any reachability path then there is also a short one. In this way the reachability problem was shown to be in PSpace for binary 2-VASSes [2] (reachability path implies exponential length reachability path) and in NL for unary 2-VASSes [10] (reachability path implies polynomial length reachability path). In particular in order to have hope for ExpSpace-hardness for d-VASSes we need to have an example of d-VASS with the shortest path of at least doubly-exponential length. Similarly for Tower-hardness we need an example of a VASS with the shortest path being of tower length. Currently there is a known example of 4-VASS with shortest path being doubly-exponential [5]

(Section 5) which means that we may have hope for decreasing the ExpSpace-hardness from dimension 6 to 4 if we happen to find appropriate techniques. However there are no known examples of 3-VASSes of shortest reachability path bigger then exponential and of 7-VASSes of shortest reachability path bigger then doubly-exponential. Therefore without finding examples of 3-VASSes and 7-VASSes with longer shortest reachability paths we have no hope to prove ExpSpace-hardness for 3-VASSes or Tower-hardness for 7-VASSes. This indicates that a search for hard VASS examples may be actually the most needed and potentially fruitful one.

Two-counter automata. Another way how we can sometimes decrease VASS dimension by one is to use bounded two-counter automata instead of bounded three-counter automata. We managed to achieve it for 2^k-bounded automata in Theorem 2.4 and it allowed us to prove Theorem 1.2 in dimension 5 instead of 6. However this technique does not seem to extend immediately to higher bounds, for example to 2^{2^k}-bounded automata. To our best knowledge the following statement is open, but we conjecture it to be true.

Conjecture 8.1. *The reachability problem for f-bounded two-counter automata (with unary updates) is* ExpSpace*-complete for $f(k) = 2^{2^k}$.*

This conjecture would not immediately give ExpSpace-hardness for binary 5-VASSes as generating the pair $(6s \cdot 4^{2^n}, 36s^2 \cdot 16^{2^n})$ in the proof of Theorem 1.3 currently needs six counters, but would be some step towards it and an interesting result in itself.

ACKNOWLEDGMENTS

We thank Sławomir Lasota for letting us to present his proof of the first item in Theorem 2.4 and we thank anonymous reviewers for helpful remarks. First author is supported by the ERC grant INFSYS, agreement no. 950398. Second author is supported by the Ministry of Science and Higher Education project Szkoła Orłów, project number 500-D110-06-0465160.

REFERENCES

[1] Michael Blondin, Matthias Englert, Alain Finkel, Stefan Göller, Christoph Haase, Ranko Lazic, Pierre McKenzie, and Patrick Totzke. 2021. The Reachability Problem for Two-Dimensional Vector Addition Systems with States. *J. ACM* 68, 5 (2021), 34:1–34:43.

[2] 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 *Proceedings of LICS 2015*. 32–43.

[3] Michael Blondin, Christoph Haase, and Philip Offtermatt. 2021. Directed Reachability for Infinite-State Systems. In *Proceedings of TACAS 2021 (Lecture Notes in Computer Science, Vol. 12652)*. 3–23.

[4] Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. 2019. The reachability problem for Petri nets is not elementary. In *Proceedings of STOC 2019*. ACM, 24–33.

[5] Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. 2020. Reachability in Fixed Dimension Vector Addition Systems with States. In *Proceedings of CONCUR 2020*. 48:1–48:21.

[6] Wojciech Czerwinski and Lukasz Orlikowski. 2021. Reachability in Vector Addition Systems is Ackermann-complete. In *Proceedings of FOCS 2021*. IEEE, 1229–1240.

[7] Wojciech Czerwinski and Lukasz Orlikowski. 2021. Reachability in Vector Addition Systems is Ackermann-complete. *CoRR* abs/2104.13866 (2021).

[8] Wojciech Czerwinski and Lukasz Orlikowski. 2022. Lower Bounds for the Reachability Problem in Fixed Dimensional VASSes. *CoRR* abs/2203.04243 (2022).

[9] Alex Dixon and Ranko Lazic. 2020. KReach: A Tool for Reachability in Petri Nets. In *Proceedings of TACAS 2020 (Lecture Notes in Computer Science, Vol. 12078)*. 405–412.

[10] Matthias Englert, Ranko Lazic, and Patrick Totzke. 2016. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In *Proceedings of LICS 2016*. 477–484.

[11] Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. 1968. Counter Machines and Counter Languages. *Mathematical Systems Theory* 2, 3 (1968), 265–283.

[12] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. 2009. Reachability in Succinct and Parametric One-Counter Automata. In *Proceedings of CONCUR 2009*. 369–383.

[13] Slawomir Lasota. 2022. Improved Ackermannian Lower Bound for the Petri Nets Reachability Problem. In *Proceedings of STACS 2022 (LIPIcs, Vol. 219)*. 46:1–46:15.
[14] Jérôme Leroux. 2021. The Reachability Problem for Petri Nets is Not Primitive Recursive. In *Proceedings of FOCS 2021*. IEEE, 1241–1252.
[15] Jérôme Leroux. 2021. The Reachability Problem for Petri Nets is Not Primitive Recursive. *CoRR* abs/2104.12695 (2021).
[16] Jérôme Leroux and Sylvain Schmitz. 2019. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *Proceedings of LICS 2019*. IEEE, 1–13.
[17] Jérôme Leroux and Grégoire Sutre. 2004. On Flatness for 2-Dimensional Vector Addition Systems with States. In *Proceedings of CONCUR 2004 (Lecture Notes in Computer Science, Vol. 3170)*. 402–416.
[18] Richard J. Lipton. 1976. *The Reachability Problem Requires Exponential Space.* Technical Report. Yale University.
[19] Ernst W. Mayr. 1981. An Algorithm for the General Petri Net Reachability Problem. In *Proceedings of STOC 1981*. 238–246.
[20] Sylvain Schmitz. 2016. Complexity Hierarchies beyond Elementary. *ACM Trans. Comput. Theory* 8, 1 (2016), 3:1–3:36.