Reachability in Vector Addition Systems is Ackermann-complete

Wojciech Czerwiński
University of Warsaw
wczerwin@mimuw.edu.pl

Łukasz Orlikowski
University of Warsaw
lo418363@students.mimuw.edu.pl

November 16, 2021

Abstract

Vector Addition Systems and equivalent Petri nets are a well established models of concurrency. The central algorithmic problem for Vector Addition Systems with a long research history is the reachability problem asking whether there exists a run from one given configuration to another. We settle its complexity to be Ackermann-complete thus closing the problem open for 45 years. In particular we prove that the problem is $\mathcal{F}_k$-hard for Vector Addition Systems with States in dimension $6k$, where $\mathcal{F}_k$ is the k-th complexity class from the hierarchy of fast-growing complexity classes.

1 Introduction

The model of Vector Addition Systems (VASes) is a fundamental computation model well suited to model concurrent phenomena. Together with essentially equivalent Petri nets it is long studied and has numerous applications in modelling and analysis of computer systems and natural processes. The central algorithmic problem for VASes is the reachability problem, which asks whether there exists a run from one given configuration to another. The reachability problem has a long research history. In 1976 it was shown to be ExpSpace-hard by Lipton [18]. Decidability of the reachability problem was first proven by Mayr in 1981 [19]. The construction was simplified later by Kosaraju [10] and Lambert [11]. Their approach was to use an equivalent model of VASes with states (VASSes) and in certain situations, when the answer to the problem is not clear use a nontrivial decomposition of the system into simpler ones. This technique is called the KLM decomposition after the names of its three inventors. Despite a substantial effort of the community for a long time there was no known upper complexity bound for the VASS reachability problem. There were however important results in the special cases when the dimension is fixed. Haase et al. shown an NP-completeness of the problem in binary encoded one-dimensional VASSes [7]. In dimension one the reachability problem for unary encoded VASSes can be easily shown NL-complete. In 2015 Blondin et al. have shown that the reachability problem for two-dimensional VASSes is PSpace-complete in the case when transitions are encoded in binary [1]. Further improvement came soon after that, a year later Englert et al. proved that the same problem in the case of unary encodings of transitions is NL-complete [5].

In 2015 Leroux and Schmitz have obtained the first upper complexity bound for the reachability problem proving that it belongs to the cubic-Ackermannian complexity class denoted also $\mathcal{F}_{\omega^3}$ [16]. The same authors have improved their result recently in 2019 showing that the problem can be solved in the Ackermann complexity class (denoted $\mathcal{F}_\omega$) [17]. They have actually shown that the reachability problem for k-dimensional VASSes (denoted k-VASSes) can be solved in the complexity class $\mathcal{F}_{k+4}$, where $\mathcal{F}_i$ is the hierarchy of complexity classes related to the hierarchy of fast-growing functions F_i. In the meanwhile in [2] it was shown that the reachability problem is Tower-hard, recall that Tower $= \mathcal{F}_3$. Thus the complexity gap was decreased to the gap between Tower and Ackermann complexity classes.

Our contribution   In this paper we close the above mentioned complexity gap. Our main result is actually a more detailed hardness result, which depends on the dimension of the input VASS.

Theorem 1. For each $k \geq 3$ the reachability problem for $6k$-VASSes is $\mathcal{F}_k$-hard.

In particular the reachability problem for 18-VASSes is Tower-hard, as Tower $= \mathcal{F}_3$. An immediate consequence of Theorem 1 is that reachability problem for VASSes is Ackermann-hard. Together with [17] it implies the following.

Corollary 2. The VASS reachability problem is Ackermann-complete.

Recently Jérôme Leroux independently has shown Ackermann-hardness of the VASS reachability problem [13]. He relies on similar known techniques, but his new contribution is substantially different than ours.

Organisation of the paper   In Section 2 we introduce preliminary notions. Next, in Section 3 we present known approach to the problem, introduce technical notions necessary to show our result and formulate the main technical Lemma 7. In Section 4 we present the main technique, which led to our result, namely the technique of performing many zero-tests by using only one additional counter. We also present there two examples of application of this technique. The examples are not necessary to understand the main construction, but are interesting in their own and introduce mildly the new technique. In Section 5 we prove the main technical result, namely the Lemma 7. Finally, in Section 6 we present possible future research directions.

2   Preliminaries

Basic notions   For $a, b \in \mathbb{N}$, $b \geq a$ we write $[a, b]$ for the set $\{a, a + 1, \ldots, b - 1, b\}$. For a vector $v \in \mathbb{Z}^d$ and $i \in [1, d]$ we write $v[i]$ for the i-th entry of v. For a vector $v \in \mathbb{Z}^d$ and the set of indices $S \subseteq [1, d]$ by $v[S] \in \mathbb{Z}^{|S|}$ we denote vector v restricted to the indices in S. By 0^d we represent the d-dimensional vector with all entries being 0.

Vector Addition Systems   A d-dimensional Vector Addition System with States (d-VASS) consists of a finite set of *states* Q and a finite set of *transitions* $T \subseteq Q \times \mathbb{Z}^d \times Q$. Configuration of a VASS is a pair $(q, v) \in Q \times \mathbb{N}^d$, usually written $q(v)$. We write Conf $= Q \times \mathbb{N}^d$. Transition $(p, t, q) \in T$ can be fired in the configuration $r(v) \in$ Conf if $p = r$ and $v + t \in \mathbb{N}^d$. Then we write $p(v) \xrightarrow{(p,t,q)} q(v + t)$. The *effect* of transition (p, t, q) is a vector $t \in \mathbb{N}^d$, we write eff$((p, t, q)) = t$. A sequence of triples $\rho = (c_1, t_1, c_1'), (c_2, t_2, c_2'), \ldots, (c_n, t_n, c_n') \in$ Conf $\times T \times$ Conf is a *run* of 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 extend naturally the definition of the effect to runs, eff$(\rho) = t_1 + \ldots + t_n$. 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 $c_1 \xrightarrow{\rho} c_n'$. We also say then that the configuration c_1 *reaches* the configuration c_n' or c_n' *is reachable from* c_1. By $\text{REACH}(\text{src}, V) = \{c \mid \text{src} \longrightarrow c\}$ we denote the set of all the configurations reachable from configuration src and we call it the *reachability set*. We also write simply $\text{REACH}(\text{src})$ if VASS V is clear from the context. The following problem is the main focus of this paper.

> **Reachability problem for VASSes**
>
> **Input** A VASS V and two its configurations src, trg
>
> **Question** Does src $\longrightarrow$ trg in V?

The size of VASS V, denoted $\text{SIZE}(V)$, is the total number of bits needed to represent states and transitions of V. A Vector Addition System (VAS) is a VASS with only one state (thus the state can be ignored). It is folklore that reachability problems for VASSes and for VASes are interreducible in polynomial time, therefore one can wlog. focus on one of them. In this paper we decide to work with VASSes as they form a more robust model.

Counter programs   We often work with VASSes which have a special sequential form: each run of such a VASS performs first some sequence of operations, then some other sequence of operations etc. Such VASSes can be very conveniently described as counter programs. A counter program is a sequence of instructions, each one being either the counter values modifications of the form $x_1 \mathrel{+}= a_1 \quad \ldots \quad x_d \mathrel{+}= a_d$ or a loop of the form

1: **loop**
2:     P

where P is another counter program. Such a counter program with k instructions and d counters $x_1, \ldots, x_d$ represents a d-VASS V with states $q_1, \ldots, q_k, q_{k+1}$ (and some other ones) such that:

- there are two distinguished states of V, the state q_1 called the *source* state of V and the state q_{k+1} called the *target* state of V;

- if the i-th instruction is of the form $x_1 \mathrel{+}= a_1 \quad \ldots \quad x_d \mathrel{+}= a_d$ then in V there is a transition $q_i \xrightarrow{v} q_{i+1}$ where $v[j] = a_j$ if $x_j \mathrel{+}= a_j$ is listed in the sequence of increments and $v[j] = 0$ otherwise;

- if the i-th instruction is the loop with body equal to counter program P then in V there are transitions $q_i \xrightarrow{0^d} \text{src}_P$ and $\text{trg}_P \xrightarrow{0^d} q_i$ where src_P and trg_P are source and target states of VASS V_P represented by program P

- if the i-th instruction is the loop then in V there is a transition $q_i \xrightarrow{0^d} q_{i+1}$.

If the last instruction is a loop we often omit q_{k+1} as the only transition incoming to it is $q_k \xrightarrow{0^d} q_{k+1}$ and treat q_k as the target state.

Example 3. The following counter program

1: $x \mathrel{+}= 1$
2: **loop**
3:     $x \mathrel{-}= 1 \quad y \mathrel{+}= 1$
4: **loop**
5:     $x \mathrel{+}= 2 \quad y \mathrel{-}= 1$
6: **loop**
7:     $x \mathrel{-}= 1 \quad y \mathrel{+}= 1$
8: **loop**
9:     $x \mathrel{+}= 2 \quad y \mathrel{-}= 1$

represents the 2-VASS presented below, state names are chosen arbitrary. Notice that in the program there are five instructions: line 1 and loops in lines 2-3, 4-5, 6-7 and 8-9, so the corresponding VASS has five states and can be depicted as follows.
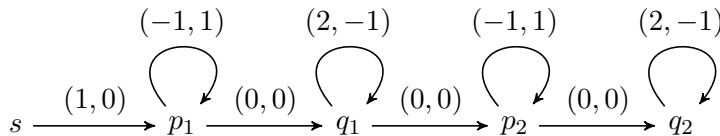
We often use macro **for** $i := 1$ **to** n **do**, by which we represent just the counter program in which the body of the for-loop is repeated n times. We do not allow for the use of variable i inside the for-loop.
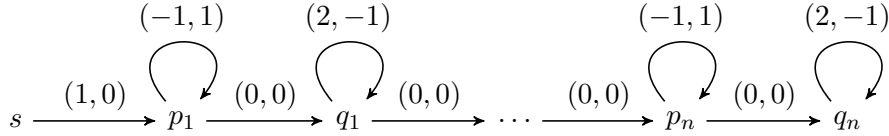
Example 4. The following counter program uses the macro **for**. For $n = 2$ it is equivalent to the above example.

```
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.

For a counter program V we write $u_{\text{in}} \xrightarrow{V} u_{\text{out}}$ if there is a run of V starting in counter valuation u_{in} in the source state of V and finishing in counter valuation u_{out} in the target state of V.

Fast-growing functions and its complexity classes   We introduce here a hierarchy of fast-growing functions and the corresponding complexity classes. There are many known variants of the definition of the fast-growing function hierarchy. Notice however, that the definition of the corresponding complexity classes $\mathcal{F}_i$ is robust and does not depend on the small changes in the definitions of the fast-growing hierarchy (for the robustness argument see [20, Section 4]).

Let $F_1(n) = 2n$ and let $F_k(n) = \underbrace{F_{k-1} \circ \ldots \circ F_{k-1}}_{n}(1)$ for any $k > 1$. Therefore we have

$F_2(n) = 2^n$ and $F_3(n) = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{n} = \text{Tower}(n)$. We define the function F_ω as $F_\omega(n) = F_n(n)$. The Ackermann function is defined exactly to be the F_ω function from the fast-growing hierarchy.

Based on functions F_k we define complexity classes $\mathcal{F}_k$ also following definitions in [20]. The complexity class $\mathcal{F}_k$ contains all the problems, which can be solved in time $f \circ g$, where $g \in F_k$ and f belongs to levels F_i for $i < k$ closed under composition and limited primitive recursion. The idea is that problems in $\mathcal{F}_k$ can be solved by some easier-then-F_k reduction to an F_k-solvable problem. For example the class $\mathcal{F}_3$, also called Tower, contains all the problems, which can be solved in the time $\text{Tower}(n)$, but also for example those, which can be solved in time $\text{Tower}(2^{2^n})$, as $\text{Tower}(2^{2^n}) = \text{Tower}(n) \circ 2^{2^n}$. It is well known that complexity classes $\mathcal{F}_k$ have natural complete problems concerning counter automata, which we formulate precisely in Section 3.

3   Outline

Here we outline the proof of our main result, Theorem 1. We introduce gradually intuitions, which led us to this contribution.

Counter automata and $\mathcal{F}_k$-hardness   We follow some of the ideas of the previous lower bound result showing Tower-hardness [2]. In particular we reduce from a similar problem concerning counter automata. Counter automata are extensions of VASSes in which transitions may have an additional condition that they are fired only if certain counter is equal to exactly zero. Such transitions are called *zero-tests*. We say that a run of counter automaton is *accepting* if it starts in the distinguished initial state with all counters equal to zero and finishes in the distinguished accepting state also with all counters equal to zero. A run is N-bounded if all the

counters along this run have values not exceeding N. It is a folklore that the following problem is $\mathcal{F}_k$-hard for $k \geq 3$ (for a similar problem see [20, Section 2.3.2]):

F_k-reachability for counter automaton

Input Three-counter automaton $\mathcal{A}$, number $n \in \mathbb{N}$

Question Does $\mathcal{A}$ have an $F_k(n)$-bounded accepting run?

Our aim is to provide a polynomial time reduction, which for each $k \geq 3$, three-counter automaton $\mathcal{A}$ and number $n \in \mathbb{N}$ constructs a $6k$-VASS together with source and target configurations src and trg such that src $\longrightarrow$ trg iff $\mathcal{A}$ has an $F_k(n)$-bounded accepting run. This will finish the proof of Theorem 1.

Multiplication triples As suggested above the main challenge in showing $\mathcal{F}_k$-hardness is the need to simulate $F_k(n)$-bounded counters and provide zero-tests for them. We first recall an idea from [2] which reduces the problem to constructing three counters with appropriate properties. On an intuitive level the argument proceeds as follows. Assume a machine (in our case VASS) has access to triples of the form (M, y, My). Then it can use them to perform exactly y sequences of actions, whatever these actions exactly are, and in each sequence perform exactly M actions. The idea is that in each sequence VASS decreases the second counter by one therefore assuring that the number of sequences is exactly y. It uses the first counter to assure that in each sequence the number of actions is at most M. During each action the third counter is decreased by one, thus each sequence of actions decreases the third counter by at most M. Therefore y sequences of actions can decrease the third counter maximally by My and moreover if this counter was decreased by exactly My it means that in every sequence exactly the maximal possible number M of actions was performed. Thus by checking at the end of the whole process whether the second and the third counters are equal to zero we check whether there were exactly y sequences and in all the sequences there were exactly M actions. Below we exploit this idea roughly speaking for simulating zero-tests on counters which are bounded by some value M an arbitrary number y of times. So typically in our application M will be bounded and y is a guessed arbitrarily big number. We also explain below more precisely what kind of VASS we need to prove the $\mathcal{F}_k$-hardness of the reachability problem.

We say that a $(d+3)$-VASS V for $d \geq 0$ together with its initial configuration c, accepting state q and a set of test counters $T \subseteq [1, d]$ is an *M-generator* if:

- all the configurations of the form $q(x, y, z, v)$ with $v \in \mathbb{N}^d$ in the set $\text{REACH}(c, V)$ such that $v[t] = 0$ for all $t \in T$ fulfil $v = 0^d$, $x = M$ and $z = My$;

- for each $y \in \mathbb{N}$ we have $q(M, y, My, 0^d) \in \text{REACH}(c, V)$.

We call the counters x, y, z the *output counters*. In other words an M-generator generates triples (x, y, z) on its output counters such that we are guaranteed that they are of the form (M, y, My) and moreover each such triple can be generated. We also say briefly that (V, c, q, t) is an M-generator.

The following lemma shows that it is enough to focus on the construction of M-generators, as they allow for simulation of M-bounded counters. The same idea and a similar statement was already present in [2], but we prove the lemma in order to be self-contained.

Lemma 5. For any d-VASS (V, s, q, T) with $d \geq 12$ and $|T| \leq 4$, which is an M-generator, and a three-counter automaton $\mathcal{A}$ one can construct in polynomial time a d-VASS $V_{\mathcal{A}}$ with configurations src and trg such that src $\longrightarrow$ trg iff $\mathcal{A}$ has an M-bounded accepting run.

Proof. The construction of the d-VASS $V_\mathcal{A}$ proceeds as follows. The configuration src of $V_\mathcal{A}$ is the source configuration of the M-generator (V, s, q, T). We first run the d-VASS (V, s, q, T), which outputs a triple $(c_1, c_2, c_3, 0^{d-3})$ under the condition that the test counters equal zero. For technical simplicity we assume that the test counters are the ones with the biggest indices. In the rest of the run we do not modify the test counters in order to assure (by setting $\text{trg}[t] = 0$ for all $t \in T$) that indeed these test counters equals zero at output of V. Recall that as $d \geq 12$ and we have at most 4 test counters then we are at least 8 counters beside the test ones, namely: $c_1, \ldots, c_8$. We need to simulate three counters of the automaton $\mathcal{A}$, say counters x, y and z. In order to assure that each run of $V_\mathcal{A}$ corresponds to an M-bounded run of $\mathcal{A}$ we add for each counter c another counter $\bar{c}$ such that at any time after an initialisation phase it holds $c + \bar{c} = M$. We will use the counters c_1, c_4 and c_5 to simulate counters x, y and z, respectively and the counters c_6, c_7 and c_8 to simulate counters $\bar{x}$, $\bar{y}$ and $\bar{z}$, respectively. We thus need to set $c_6 = c_7 = c_8 = M$ in the initialisation phase, which is realised by the following program fragment

```
1: c₂ −= 1
2: loop
3:     c₁ −= 1    c₃ −= 1    c₆ += 1    c₇ += 1    c₈ += 1
```

Counter c_2 is decreased here by 1, while counter c_3 is decreased by at most value of c_1, which is M. As explained before the only option for counter c_3 to reach value 0 at the end of the run is to match each decrease of c_2 by 1 by a decrease by M. Therefore we are guaranteed that in any run reaching configuration trg the initialisation phase indeed sets $\bar{x} = \bar{y} = \bar{z} = M$ and also we have $x = y = z = 0$ after this phase. The configuration trg of $V_\mathcal{A}$ is defined as follows: the state corresponds to the accepting state of three-counter automaton $\mathcal{A}$, eight first counter values correspond to counter values of accepting configuration of $\mathcal{A}$, namely $(c_1, \ldots, c_8) = (0^5, M, M, M)$ and all the other counters of $V_\mathcal{A}$ are equal zero in the configuration trg.

Next VASS $V_\mathcal{A}$ simulates operations of counter automaton $\mathcal{A}$, namely increments, decrements and zero-tests. Concretely speaking there is a copy of $\mathcal{A}$ inside of $V_\mathcal{A}$ with slightly modified transitions. Simulation of operation $c \mathrel{+}= a$ (for both positive and negative a) in $\mathcal{A}$ is straightforward, we add operations $c \mathrel{+}= a$ and $\bar{c} \mathrel{-}= a$ to $V_\mathcal{A}$. It is more challenging to simulate **zero-test**(c) in $\mathcal{A}$, we use the pair of counters $(c_2, c_3) = (b, Mb)$ generated by the M-generator for that. Recall that using this pair we are able to perform exactly b sequences of exactly M actions. The idea is that for checking whether $c = 0$ (and thus $\bar{c} = M$) we first transfer value of $\bar{c}$ to c (i.e. decrement $\bar{c}$ and increment c) and simultaneously decrement c_3. Then we transfer value of c back to $\bar{c}$ also decrementing c_3. In that way we can decrement c_3 at most $2M$ times and decrement by exactly $2M$ can happen only if the initial value of c was zero and also final value of c is zero as well. If we decrement c_2 by 2 we assure that indeed c_3 needs to be decremented by $2M$ and hence the **zero-test**(c) can be simulated as follows.

```
1: c₂ −= 2
2: loop
3:     c += 1    c̄ −= 1    c₃ −= 1
4: loop
5:     c −= 1    c̄ += 1    c₃ −= 1
```

Let us inspect the code to see that it indeed reflects the above story. Recall that we keep all the time the invariant $c + \bar{c} = M$, so $\bar{c} \leq M$. Therefore the loop in lines 2-3 is fired at most M times. Similarly the loop in lines 4-5 is fired at most M times. Thus indeed the result of loops in lines 2-5 is the decrease of counter c_3 by at most $2M$ and decrease by exactly $2M$ corresponds to initial and final value of c being zero. Thus lines 1-5 indeed simulate faithfully the zero-test.

As increments, decrements and zero-tests of $\mathcal{A}$ can be simulated faithfully by $V_\mathcal{A}$ one can see that runs from src to trg of $V_\mathcal{A}$ are in one-to-one correspondence with M-bounded runs of automaton $\mathcal{A}$. $\qquad\square$

Our approach is therefore to construct $6k$-VASSes of not too big size which are $F_k(n)$-generators.

Amplifiers At this moment it is natural to introduce a notion of *amplifier*, which can be used to produce an N-generator from an M-generator for N much bigger than M. For a function $f : \mathbb{N} \to \mathbb{N}$ we say that a d-VASS V together with its *input state* p_{in}, *output state* p_{out} and set of *test counters* $T \subseteq [1, d]$ is an f-*amplifier* if the following holds

- if $p_{\text{in}}(M, x, Mx, 0^{d-3}) \longrightarrow p_{\text{out}}(v, b, y, z)$ for $v \in \mathbb{N}^d$ with $v[T] = 0$ then $v = 0^{d-3}$, $b = f(M)$ and $z = by$; and

- for each $y \in \mathbb{N}$ there exists an $x \in \mathbb{N}$ such that $p_{\text{in}}(M, x, Mx, 0^{d-3}) \longrightarrow p_{\text{out}}(0^{d-3}, f(M), y, f(M) \cdot y)$.

In other words, intuitively, if an amplifier inputs triples (M, x, Mx) it outputs triples $(f(M), y, f(M) \cdot y)$ and moreover each such triple can be output if an appropriate triple is delivered to the input. In the above case we call the first three counters the *input* counters and the last three counters the *output* counters, but in general we do not impose any order of the input and output counters. Notice that notions of amplifier and generators are very much connected as suggested by the following claim.

Lemma 6. For any $d \geq 3$ if there exists a d-dimensional f-amplifier V then exists a d-dimensional $f(M)$-generator of size linear in $\text{SIZE}(V) + \log(M)$ with the same number of test counters as V.

Proof. We construct the $f(M)$-generator as follows. In its initial state we have a loop with the effect $(0, 1, M, 0^{d-3})$, thus after x applications of it we get a vector $(0, x, Mx, 0^{d-3})$. Then a transition with effect $(M, 0, 0, 0^{d-3})$ leads to the input state of the f-amplifier, so the f-amplifier inputs a tuple $(M, x, Mx, 0^{d-3})$. Immediately from the definition of an amplifier we get that all the runs reaching the output state of the amplifier with vectors of the form (v, x, y, z) with $v \in \mathbb{N}^{d-3}$ such that $v[t] = 0$ for all test counters $t \in T$ fulfil $v = 0^{d-3}$, $x = f(M)$, $z = f(M) \cdot y$ and additionally such configurations for all $y \in \mathbb{N}$ can be reached, which finishes the proof. $\square$

Observe that taking into account Lemmas 5 and 6 in order to prove Theorem 1 it is enough to show the following lemma.

Lemma 7. For each $k \geq 1$ there exists a $6k$-VASS of size exponential in k (at most C^k for some constant $C \in \mathbb{N}$) which is an F_k-amplifier with at most four test counters.

The advantage of amplifiers over generators is that we can easily compose them. Notice that having two VASSes: a $(d_1 + 3)$-VASS being an f_1-amplifier and a $(d_2 + 3)$-VASS being an f_2-amplifier it is easy to construct a $(d_1 + d_2 + 3)$-VASS being an $f_1 \circ f_2$-amplifier just by using sequential composition of the f_2-amplifier and f_1-amplifier. However, the drawback of the construction is that the dimension grows substantially. The main challenge in the proof of Lemma 7 is to build amplifiers for much bigger functions from amplifiers for much smaller functions without adding too many new counters. The proof of Lemma 7 is presented in Section 5.

Big counter values In order to prove F_k-hardness for VASS reachability problem one should in particular construct VASSes V_k in which the shortest run from some source configuration to some target configuration has length at least $F_k(n)$, where $n = \text{SIZE}(V_k)$. Notice that some configuration on such a run needs to have some counters of value at least roughly $F_k(n)$. It has been known since a long time that such VASSes exist and it is relatively easy to construct them. The hard part is to design a VASS, in which *every* run reaches high counter values and on a very high level of abstraction one can see our construction as mainly achieving this goal.

Below we present an example family of VASSes V_k in growing dimension $k+1$ with reachability sets being finite, but which can reach values of counters up to $F_k(n)$. Knowing this construction is absolutely not needed to understand our construction and this part of the paper can be omitted during the reading without any harm. We present it however as we believe that it helps to distinguish which parts of the proof of Lemma 7 are pretty standard and which were the real challenge. In short words the real challenge was to force the runs to have values zero at some precise points, we show in Section 4 in details how to guarantee this.

For $d = 2$ we have the following 3-VASS V_2.

```
1: loop
2:     loop
3:         x₁ += 2     x₂ -= 1
4:     loop
5:         x₁ -= 1     x₂ += 1
6:     x₃ -= 1
```

In general we construct $(k+1)$-dimensional VASS V_k in the following way from k-dimensional VASS V_{k-1}.

```
1: loop
2:     V_{k-1}
3:     loop
4:         x₁ -= 1     x_k += 1
5:     x_{k+1} -= 1     x₃ += 1     ...     x_{k-1} += 1
```

It is quite easy to see that the reachability set of V_k is finite when starting from any counter valuation and one can show it easily by induction on k. Therefore it remains to show the following proposition.

Proposition 8. For each $k \in \mathbb{N}$ it holds $(1, 0, 1^{k-2}, m-1) \xrightarrow{V_k} (F_k(m), 0^k)$.

Proof. We show the proposition by induction on k. We start the induction from $k = 2$, one can easily see that indeed $(1, 0, m-1) \xrightarrow{V_2} (2^m, 0, 0)$. For the induction step assume that $(1, 0, 1^{k-3}, m-1) \xrightarrow{V_{k-1}} (F_{k-1}(m), 0^{k-1})$. We therefore have for any $\ell \in \mathbb{N}$ that

$$(1, 0, 1^{k-3}, m-1, \ell) \xrightarrow{(2)} (F_{k-1}(m), 0^{k-1}, \ell) \xrightarrow{(3-4)} (1, 0^{k-2}, F_{k-1}(m) - 1, \ell)$$
$$\xrightarrow{(5)} (1, 0, 1^{k-3}, F_{k-1}(m) - 1, \ell - 1),$$

where by $\xrightarrow{(i)}$ we denote the transformation on counters caused by line i of program V_k. Therefore we have

$$(1, 0, 1^{k-3}, 1, m-1) \xrightarrow{(2-5)} (1, 0, 1^{k-3}, F_{k-1}(1) - 1, m-1)$$
$$\xrightarrow{(2-5)} (1, 0, 1^{k-3}, F_{k-1} \circ F_{k-1}(1) - 1, m-2)$$
$$\xrightarrow{(2-5)} \ldots \xrightarrow{(2-5)} (1, 0, 1^{k-3}, \underbrace{F_{k-1} \circ \ldots \circ F_{k-1}}_{m-1}(1) - 1, 0)$$
$$\xrightarrow{(2)} (\underbrace{F_{k-1} \circ \ldots \circ F_{k-1}}_{m}(1), 0^k) = (F_k(m), 0^k).$$

which finishes the induction step. $\qquad\square$

4 Zero-tests

The main contribution of this paper is a novel technique of zero-testing, which allows for performing many zero-tests simultaneously. This will be the key technique in the proof of Lemma 7. We prefer to introduce it mildly before using it in Section 5 and present first how it works on a few simple examples. Already on these examples its power is visible.

Example 9. Imagine first that we are given a VASS run ρ and we want to test some counter x for being exactly zero in three moments along this run: in configurations c_1, c_2 and c_3. Assume that value of x is zero at the beginning of ρ. Let value of x in these configurations be x_1, x_2 and x_3, respectively. A naive way to solve this problem is to add three new counters, which are copies of x, but the first one stops copying effects of transitions on x in c_1, the second one in c_2 and the third one in c_3. In that way the additional counters keep values of x_1, x_2 and x_3 till the end of the run and can be checked there for zero (just by setting the target configuration to zero on these counters). We show here how to perform these three zero-tests using just one additional counter, we call it the *controlling counter* and say that it *controls* the other counters. Let ρ_1 be the part of ρ before c_1, ρ_2 the part in between c_1 and c_2 and ρ_3 the part in between c_2 and c_3. Let y_1, y_2 and y_3 be the effects of ρ_1, ρ_2 and ρ_3 on x, respectively. We can easily see that $x_1 = y_1$, $x_2 = y_1 + y_2$ and $x_3 = y_1 + y_2 + y_3$. Notice that we can check whether all the x_1, x_2 and x_3 are equal to zero by checking whether its sum $x_1 + x_2 + x_3$ equals zero, as all the x_i are nonnegative. We have $x_1 + x_2 + x_3 = 3y_1 + 2y_2 + y_3$, so it is enough to check whether $3y_1 + 2y_2 + y_3 = 0$. Instead of adding three new counters we add only one, which computes the value $3y_1 + 2y_2 + y_3$. We realise it in the following way. Every increase of x by a during ρ_1 is reflected by an increase of the controlling counter by $3a$. Similarly, an increase of x by a on ρ_2 is reflected by an increase of the controlling counter by $2a$ and on ρ_3 just by a. After configuration c_3 the controlling counter is not modified. Therefore testing the controlling counter for 0 at the end of the run (by setting appropriately the target configuration on that counter) checks indeed whether $x_1 = x_2 = x_3 = 0$.

This approach can be generalised to more zero-tests and moreover to zero-tests on different counters, as shown by the following lemma. In the lemma the aim of the last (controlling) counter is to allow for zero-testing the i-th counter on configurations with indices in S_i, for all $i \in [1, d]$. Notice also that configurations c_i are not necessarily all the configurations on the run ρ, but some subset, which can be restricted only to those in which we want to perform some zero-test.

Lemma 10. Let $\text{src} \xrightarrow{\rho} \text{trg}$ be a run of a $(d+1)$-VASS V and let $\text{src} = c_0, c_1, \ldots, c_{n-1}, c_n = \text{trg}$ 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) $\text{src}[d + 1] = \sum_{i=1}^{d} N_{0,i} \cdot \text{src}[i]$;

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

(3) $\text{trg}[d + 1] = 0$

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

Before we proceed with the proof of Lemma 10 we comment a bit on the lemma and see how the example above fulfils its conditions. The condition (1) assures that at the configuration src the controlling $(d+1)$-th counter is appropriately related to the other ones. In the example it is trivially fulfilled by the equation $0 = 0$. The condition (2) assures that the effect of each part of the run ρ is appropriately reflected by the controlling counter. On the example it is fulfilled as each change of the x-counter by a on ρ_1, ρ_2 and ρ_3 is reflected by the change of controlling counter by $3a$, $2a$ and a, respectively. Notice that in the case of our example the condition (2) is satisfied even if we set ρ_j to be single transitions and n equals length of the run. However in many applications of Lemma 10 condition (2) is true only if we set ρ_j to be some groups of transitions forming a longer sub-run of ρ, this is why we formulate the lemma in such a stronger version. Condition (3) assures that the controlling counter is indeed zero at the end of the run.

Proof. Notice first that in order to check whether for each $i \in [1, d]$ and for each $j \in S_i$ we have $c_j[i] = 0$ it is enough to check whether the sum of all $c_j[i]$ is zero, namely to check whether $\text{SUM} = \sum_{i \in [1,d], j \in S_i} c_j[i] = 0$.

For each $i \in [1, d]$ and $j \in S_i$ the value $c_j[i]$ is the sum of the initial value of the i-th counter and effects of all the runs ρ_k before configuration c_j on the i-th counter. In other words $c_j[i] = c_0[i] + \sum_{k=1}^{j} \text{eff}(\rho_k, i)$. We therefore have

$$\text{SUM} = \sum_{i \in [1,d], j \in S_i} c_j[i] = \sum_{i \in [1,d], j \in S_i} \left(c_0[i] + \sum_{k=1}^{j} \text{eff}(\rho_k, i) \right).$$

In the rightmost expression $c_0[i]$ occurs exactly $N_{0,i}$ times and each $\text{eff}(\rho_k, i)$ occurs exactly $N_{k,i}$ times. Therefore

$$\text{SUM} = \text{src}[d+1] + \sum_{k=1}^{n} \text{eff}(\rho_k, d+1) = \text{trg}[d+1],$$

where the first equation follows from (1) and (2). By (3) we have $\text{trg}[d+1] = 0$ which implies that indeed $\text{SUM} = 0$ and finishes the proof. $\square$

Below we present two examples of the application of Lemma 10: an example of a 3-VASS with transitions represented in unary (shortly unary 3-VASS) with exponential shortest run and an example of a 7-VASS with transitions represented in binary (shortly binary 7-VASS) with doubly-exponential shortest run. Low dimensional unary VASSes with exponential shortest runs and binary VASSes with doubly-exponential shortest run have been presented in [3]. We present here the new examples in order to illustrate our technique, but also to provide another set of nontrivial VASS examples in low dimensions.

Example 11. We consider the 2-VASS from Example 4 with one controlling counter added, the counter c. We analyse runs starting from all zeros and finishing in all zeros. We will observe that adding the controlling counter forces the loops in lines 3-4 (or states p_i) and in lines 5-6 (or states q_i) to be performed maximal possible number of times, namely the x counter is zero when run leaves line 4 and the y counter is zero when run leaves line 6. We have added comments in the program below to emphasise where which counters are forced to be zero. This forces the run to visit configuration $(2^n, 0, 0)$ in line 6 and therefore to be exponential in VASS size.
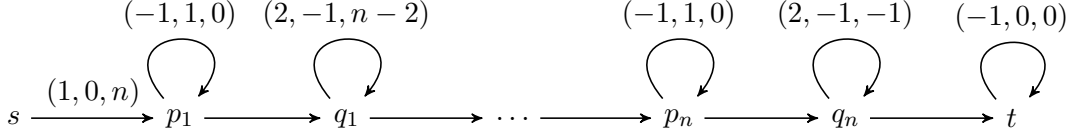
```
1: x += 1    c += n
2: for  i  :=  1  to  n do
3:     loop
4:         x −= 1    y += 1                          // zero-test on x after the loop
5:     loop
6:         x += 2    y −= 1    c += n − i − 1         // zero-test on y after the loop
7: loop
```

8:     $x\ -\!\!= 1$

To see the comparison with Example 4 we also present our VASS in a more traditional way. For simplicity we do not write labels if they are vectors with all entries being zero.

Any run from $s(0,0,0)$ to $t(0,0,0)$ crosses through states $p_1,\ldots,q_n$ and therefore can be divided into $2n+1$ parts $\rho_1,\ldots,\rho_{2n+1}$ by cutting in the last configurations in appropriate states. We want the counter x to be zero-tested in states $p_1,\ldots,p_n$ and counter y to be zero-tested in states $q_1,\ldots,q_n$, so we set $S_1=\{1,3,\ldots,2n-1\}$ and $S_2=\{2,4,\ldots,2n\}$. One can easily check that the considered VASS fulfils conditions of Lemma 10. Condition (1) is trivially fulfilled as $0=0$ and condition (3) is fulfilled as we demand to reach $t(0,0,0)$, so the controlling counter is equal to zero at the target configuration. In order to see that condition (2) is also fulfilled we show that it is even fulfilled for each transition (not only for runs in between of zero-tests). Let us consider four lines: 1, 4, 6 and 8. In line 1 counter x is awaiting n zero-tests, so increase of x by 1 should be reflected by increase of c by n. In line 4 counter x is awaiting $(n+1)-i$ zero-tests and so similarly counter y. Therefore in line 4 counter c should be increased by $(n+1-i)\cdot 1 + (n+1-i)\cdot(-1) = 0$. In line 6 counter x is awaiting $n-i$ zero-tests, while counter y is awaiting $(n+1)-i$ zero-tests, therefore counter c should be increased here by $(n-i)\cdot 2 + (n+1-i)\cdot(-1) = 2n-2i-n-1+i = n-1-i$. In line 8 counter x is not awaiting for any zero-test, so counter c should not be changed because of its changes. Summarising all of that we see that $c=0$ at the target configuration forces the run to perform maximal number of times loops in the states p_i and q_i and therefore there is only one run of our 3-VASS, which in particular visits the configuration $q_n(2^n,0,0)$ and has exponential length.

Example 12. Here we present an example of a binary 7-VASS with doubly-exponential shortest run. This result is not needed in the proof of Lemma 7, we show it however in order to illustrate how to use the technique of performing many zero-tests in the case when the number of zero-tests is bigger then the size of the VASS. In Example 11 the number of zero-tests both on x and y counters was comparable to the size of the VASS. Therefore different behaviour of controlling c in different phases of the run could be implemented by different behaviour of c in different states. In the current example this is not possible. Let us recall the well known Hopcroft-Pansiot example of a 3-VASS from [9]. As n is given in binary it can have a doubly-exponential run.

1: $x\ +\!\!= 1$ $z\ +\!\!= n$
2: **loop**
3:     **loop**
4:         $x\ -\!\!= 1$ $y\ +\!\!= 1$
5:     **loop**
6:         $x\ +\!\!= 2$ $y\ -\!\!= 1$
7:     $z\ -\!\!= 1$

We can observe that there is a run which finishes with counter values $(x,y,z) = (2^n,0,0)$ and 2^n is doubly-exponential wrt. the VASS size. Notice however that nothing forces the run to reach so high value of x. Our aim is now to add a controlling counter c which would force the loops in lines 3-4 and in lines 5-6 to be applied maximal number of times. In the case when $z=0$ at the end of the run we know that the main loop in lines 2-7 is executed exactly n times, therefore we want to test both x and y exactly n times for zero. It is easy to observe that in lines 3-4 both counters are awaiting z zero-tests and in lines 5-6 counter x is awaiting $(z-1)$ zero-tests, while y is awaiting z zero-tests. Therefore the correct updates on controlling counter c should be: increase by 0 in line 4, and increase by $(z-1)\cdot 2 + z\cdot(-1) = z-2$ in line 6. The

counter program thus should be the following.

1: $x \mathrel{+}= 1 \quad z \mathrel{+}= n \quad c \mathrel{+}= n$
2: **loop**
3:     **loop**
4:         $x \mathrel{-}= 1 \quad y \mathrel{+}= 1$
5:     **loop**
6:         $x \mathrel{+}= 2 \quad y \mathrel{-}= 1 \quad c \mathrel{+}= z - 2$
7:     $z \mathrel{-}= 1$

One can however easily observe that the operation $c \mathrel{+}= z - 2$ is not a valid VASS operation, as $z - 2$ is not a constant. Fortunately counter z is a counter bounded by n and in that case we can implement this operation using only VASS operations. Very intuitively in order to implement $c \mathrel{+}= z$ it is enough to decrement z from its current value to value zero and simultaneously increment value of counter c. In order to be able to restore the original value of z and to be able to check whether z reached zero we need to add auxiliary counters. We add three additional counters z', $\bar{z}$ and $\bar{z}'$ such that all the time after line 1 we keep invariants $z + \bar{z} = n$ and $z' + \bar{z}' = n$. Notice that with those invariants we can easily check whether $z = 0$ just by performing two operations: $\bar{z} \mathrel{-}= n$ and then $\bar{z} \mathrel{+}= n$, similarly we test whether $z' = 0$. We introduce a macro for these operations here, writing **zero-test**(z) and **zero-test**(z'). These zero-tests should not be confused with zero-tests on counters x and y. Notice that they are of a very different nature and we implement **zero-test**(z) without using controlling counter c, but by the use of the fact that z is n-bounded.

Therefore in line 1 we have now additionally operation $\bar{z}' \mathrel{+}= n$, in line 7 additionally operation $\bar{z} \mathrel{+}= 1$ and in line 6 instead of operation $c \mathrel{+}= z - 2$ we place the following code of VASS

1: **loop**
2:     $c \mathrel{+}= 1 \quad z \mathrel{-}= 1 \quad z' \mathrel{+}= 1$
3:     $\bar{z} \mathrel{+}= 1 \quad \bar{z}' \mathrel{-}= 1$
4: **zero-test**(z)
5: **loop**
6:     $z \mathrel{+}= 1 \quad z' \mathrel{-}= 1$
7:     $\bar{z} \mathrel{-}= 1 \quad \bar{z}' \mathrel{+}= 1$
8: **zero-test**(z')
9: $c \mathrel{-}= 2$

The aim of line 2 is to perform $c \mathrel{+}= z$ and keep $z + z'$ constant. We need to keep $z + z'$ constant to be able to reconstruct later the original value of z. In line 3 we update $\bar{z}$ and $\bar{z}'$ to keep the invariants and in line 4 we check whether indeed we have added everything from z to c. Lines 5-8 are devoted to moving values of z and z' back to original ones, while the line 9 takes care of subtracting 2 from c, as our aim is to perform $c \mathrel{+}= z - 2$, not $c \mathrel{+}= z$.

One can easily check that Lemma 10 applies to our situation when one defines parts of the run ρ_j corresponding to the single lines of the considered 7-VASS. Therefore if we demand $c = 0$ and $z = 0$ after the line 7 of the 7-VASS then both the loops in lines 3-4 and in lines 5-6 have to be executed maximally each time. Notice that here we need to consider run fragments ρ_j in Lemma 10 to be longer than single transitions. For example all the operations $x \mathrel{+}= 2$, $y \mathrel{-}= 1$ and $c \mathrel{+}= z - 2$ should be contained in one fragment ρ_j and operation $c \mathrel{+}= z - 2$ is implemented as long sequence of transitions, as presented above. Summarising, by Lemma 10 at the end of the run we have $x = 2^n$, which is indeed doubly-exponential in VASS size since n is encoded in binary.

5 Amplifiers

This section is devoted to the proof of Lemma 7. Recall that we need to show that for each $k \geq 1$ there exists a $6k$-VASS of size exponential in k which is an F_k-amplifier. We prove it by induction on k with induction assumption additionally strengthened by the fact that the test counters include all the input counters and at most one additional counter. For $k = 1$ it is not hard to construct a 6-VASS, which is an F_1-amplifier, recall that $F_1(n) = 2n$. The following VASS realises our goal, the input counters are x_1, x_2 and x_3, the output counters are x_4, x_5 and x_6 and the test counters are only the input counters.

```
1: loop
2:     x₂ -= 2    x₅ += 1
3:     loop
4:         x₁ -= 1    x₄ += 1    x₃ -= 1    x₆ += 1
5:     loop
6:         x₁ += 1    x₄ -= 1    x₃ -= 1    x₆ += 1
7: x₂ -= 1
8: loop
9:     x₁ -= 1    x₄ += 2    x₃ -= 1
```

Lines 1-6 are devoted to set the correct values of x_5 and x_6, while lines 7-9 set the correct value of x_4. Assume that at the input we have $(x_1, x_2, x_3) = (n, x, nx)$ and recall that initially $x_4 = x_5 = x_6 = 0$. The proof idea is similar as in the proof of Lemma 5, triple (n, x, nx) is used to perform exactly x sequences of exactly n actions. Observe first that until line 8 the sum $x_1 + x_4$ does not change, thus we have $x_1 + x_4 = n$. This means that loops in lines 3-4, 5-6 and 8-9 all can be fired at most n times. Each such a loop corresponds to one operation $x_2 \mathrel{-}= 1$ (in lines 2 or 7) and at most n operations $x_3 \mathrel{-}= 1$ (in lines 4, 6 and 9). This means that in order to reach $x_3 = 0$ at the end of the run each loop has to be fired exactly n times. Moreover the loop in lines 1-6 has to be fired exactly $\frac{x-1}{2}$ times, as the final value of x_2 also needs to be 0. Therefore final values of (x_4, x_5, x_6) are $(2n, \frac{x-1}{2}, 2n \cdot \frac{x-1}{2})$, which finishes the proof for $k = 1$.

For an induction step assume that V_{k-1} is a $(6k - 6)$-dimensional VASS of size exponential in $k-1$ and an F_{k-1}-amplifier. We aim at constructing a $6k$-VASS, which is an F_k-amplifier and its size is at most $C \cdot \text{SIZE}(V_{k-1})$ for some constant C which does not depend on k. The idea to obtain F_k-amplifier is the following: start from the triple $(1, x, x)$ and apply the F_{k-1}-amplifier n times in a row, where n is the input value. The main challenge is to achieve it without adding new counters for each application. We show here how we obtain it by adding only six new counters. We crucially rely on the Lemma 10.

The F_k-amplifier has the following $6k$ counters: input triple (i_1, i_2, i_3), output triple (o_1, o_2, o_3), an auxiliary triple (s_1, s_2, s_3), controlling counter c, two auxiliary counters y_1, y_2 and $6k - 12$ counters, which are the counters of V_{k-1} being neither its input nor output counters. The triple (s_1, s_2, s_3) will be used inside the F_k-amplifier as an input triple of a F_{k-1}-amplifier. The test counters are the input counters (i_1, i_2, i_3) and the controlling counter c.

We first present the code for an F_k-amplifier V_k, which uses illegal constructions like "**for** $i := 1$ **to** n **do**" or "$c \mathrel{+}= z$" where z is current value of another counter, in order to provide an intuition what V_k does. Then we show how we can implement the mentioned constructions using only legal VASS operations. Assume that input counter values on (i_1, i_2, i_3) are (n, x, nx). Thus we aim at producing on output counters (o_1, o_2, o_3) values $(F_k(n), m, F_k(n) \cdot m)$ for some $m \in \mathbb{N}$. Let $V_{k-1}^{[i]}$ be the modified version of V_{k-1} in which the controlling counter c is also appropriately modified: each modification $x \mathrel{+}= 1$ (or $x \mathrel{-}= 1$) for counter x being any output or test counter of V_{k-1} is accompanied with a modification $c \mathrel{+}= (n+1)-i$ (or $c \mathrel{-}= (n+1)-i$). It is important to emphasise that all the counters of $V_{k-1}^{[i]}$ as well as the counter c are shared among $V_{k-1}^{[i]}$ for different i. Recall that the test counters of $V_{k-1}^{[i]}$ contain all the input counters

13

s_1, s_2, s_3 and possibly one additional counter.

```
1:  s₁ += 1    c += n
2:  loop
3:      s₂ += 1    s₃ += 1    c += 2n
4:  for  i  :=  1  to  n do
5:      V_{k-1}^{[i]}(s₁, s₂, s₃, o₁, o₂, o₃)
6:      loop
7:          o₁ -= 1    s₁ += 1    c -= 1
8:      loop
9:          o₂ -= 1    s₂ += 1    c -= 1
10:     loop
11:         o₃ -= 1    s₃ += 1    c -= 1
```

Claim 13. The above counter program is an F_k-amplifier.

Proof. The aim of lines 1-3 is to set the triple (s_1, s_2, s_3) to values $(1, a_0, a_0)$ for some arbitrary guessed $a_0 \in \mathbb{N}$. For a function $f : \mathbb{N} \to \mathbb{N}$ let us denote by $f^{(m)}(n)$ the m-fold application of f to n. Then in lines 4-11 we perform n times, for $i \in \{1, \ldots, n\}$ the following operations:

- in line 5 from a triple $(F_{k-1}^{(i-1)}(1), a_{i-1}, F_{k-1}^{(i-1)}(1) \cdot a_{i-1})$ for some $a_{i-1} \in \mathbb{N}$ on counters (s_1, s_2, s_3) the F_{k-1}-amplifier $V_{k-1}^{[i]}$ computes a triple $(F_{k-1}^{(i)}(1), a_i, F_{k-1}^{(i)}(1) \cdot a_i)$ for some $a_i \in \mathbb{N}$ on counters (o_1, o_2, o_3) under the condition that the test counters of $V_{k-1}^{[i]}$ are zero after its run;

- in lines 6-11 we aim to copy the triple $(F_{k-1}^{(i)}(1), a_i, F_{k-1}^{(i)}(1) \cdot a_i)$ from counters (o_1, o_2, o_3) back to the counters (s_1, s_2, s_3).

The controlling counter c controls counters o_1, o_2 and o_3 for each $i \in [1, d]$ the test counters of $V_{k-1}^{[i]}$, which in particular contain counters s_1, s_2 and s_3. Counter c is designed to test whether each time after line 5 test counters of $V_{k-1}^{[i]}$ are zero and each time after line 11 counters o_1, o_2 and o_3 are zero. The first condition assures that indeed output of amplifier $V_{k-1}^{[i]}$ is computed correctly, while the second condition assures that values of o_j are fully copied back to values of i_j. Notice that each of the test counters of $V_{k-1}^{[i]}$ and o_j counters is tested exactly n times in the program. In order to fulfil condition (2) of Lemma 10 we have the following modifications of counter c in our program. In line 1 we update $c \mathrel{+}= n$ and in line 3 we have $c \mathrel{+}= 2n$, as counters s_1, s_2 and s_3 are here modified and all of them await n tests. In lines 7, 9 and 11 each counter s_j awaits for $n - i$ tests, while counter o_j awaits for $n + 1 - i$ tests, this is why counter c is increased here by $-1 = (n - i) \cdot 1 + (n + 1 - i) \cdot (-1) = -1$. It is easy to verify conditions (1) and (3) of Lemma 10. Indeed, recall that in order to check whether the considered program is an F_k-amplifier we consider only runs starting with values of all the counters of $V_{k-1}^{[i]}$ beside (s_1, s_2, s_3) being zero (which guarantees condition (1)) and finishing with counter value of controlling counter c being zero (which guarantees condition (3)). So counter c together with the counters it controls indeed fulfil conditions of the Lemma 10. Therefore amplifier V_{k-1} computes correctly its output values and values of o_j are correctly transferred to counters s_j. Thus using the induction assumption stating that V_{k-1} is an F_{k-1}-amplifier we can easily show that in the i-th iteration of the for-loop we indeed have values of (o_1, o_2, o_3) equal to $(F_{k-1}^{(i)}(1), a_i, F_{k-1}^{(i)}(1) \cdot a_i)$ for some $a_i \in \mathbb{N}$ guessed nondeterministically. Therefore after n iterations of the for-loop final values of (o_1, o_2, o_3) are $(F_{k-1}^{(n)}(1), a_n, F_{k-1}^{(n)}(1) \cdot a_n) = (F_k(n), a_n, F_k(n) \cdot a_n)$ under the condition that the controlling counter and the input counters are equal to zero at the end of the run. So indeed V_k is an F_k-amplifier with output counters o_1, o_2 and o_3 and test counters i_1, i_2, i_3 and c. □

It remains to show how the for-loop and operations $c \mathrel{+}= (n + 1) - i$ are implemented. Intuitively speaking it is not problematic because we have an access to the triple of input counters (i_1, i_2, i_3) fulfilling $i_1 i_2 = i_3$ and $n = i_1$. Therefore using the idea of multiplication triples we can allow for zero-testing counters bounded by n and thus also for the needed operations. Below we describe these constructions in detail.

For the implementation of the needed operations we use the input counters (i_1, i_2, i_3) and auxiliary counters y_1 and y_2. Assume that the for-loop has the following shape

1: **for** $i := 1$ **to** n **do**
2:     ⟨body⟩

and inside the ⟨body⟩ we have operations $c \mathrel{+}= (n + 1) - i$. The initial value of i_1 equals n. Below we implement the for-loop in such a way that all the time counter value of i_1 is equal to $(n + 1) - i$. So in order to perform $c \mathrel{+}= (n + 1) - i$ it is enough to implement $c \mathrel{+}= i_1$.

1: **loop**
2:     ⟨body⟩
3:     $i_1 \mathrel{-}= 1$ $y_2 \mathrel{+}= 1$

As i_1 is one of the test counters, we are guaranteed that the loop indeed will be iterated exactly n times. Now we show how to implement operation $c \mathrel{+}= i_1$, which together shows how to implement $c \mathrel{+}= n$ in lines 1 and 3 and $c \mathrel{+}= (n + 1) - i$ in the i-th iteration of the for-loop. Operation $c \mathrel{-}= (n + 1) - i$ is implemented totally analogously to $c \mathrel{+}= (n + 1) - i$. The implementation works similarly as in the Example 12, but here we show how to implement $c \mathrel{+}= (n+1) - i$ using only two, not three, auxiliary counters. At the beginning we have $i_1 = n$, $y_1 = y_2 = 0$. We will keep the invariant $i_1 + y_1 + y_2 = n$. Notice that in the i-th iteration values of counters are $(i_1, y_1, y_2) = (n + 1 - i, 0, i - 1)$. We implement the increment $c \mathrel{+}= (n + 1) - i$ as follows.

1: **loop**
2:     $i_1 \mathrel{-}= 1$ $y_1 \mathrel{+}= 1$ $c \mathrel{+}= 1$
3: **zero-test**(i_1)
4: **loop**
5:     $i_1 \mathrel{+}= 1$ $y_1 \mathrel{-}= 1$
6: **zero-test**(y_1)

If zero-tests in lines 3 and 6 are performed correctly then it is easy to see that when the above program fragment starts in valuation $(i_1, y_1) = (n+1-i, 0)$ it also finishes in the same valuation, but a side effect is the increment $c \mathrel{+}= i_1$. Thus it remains to show that we can implement zero-tests or counters i_1 and y_1. We present how to perform **zero-test**(i_1), the **zero-test**(y_1) is performed analogously with roles of i_1 and y_1 swapped. Notice that counter i_1 is here bounded by n, so testing it for zero is much simpler than testing counters s_i or o_i above. On the other hand zero-testing of i_1 is more complicated than the zero-tests in Example 12, as there the tested counters were bounded by the size of VASS transitions. Here the tested counters are also bounded by n, but n is arbitrary, so in order to implement zero-tests we need to use triples (i_1, i_2, i_3). Additional technical complication is caused by the fact that we want to optimise the number of the auxiliary counters from three to two. The **zero-test**(i_1) is performed as follows.

1: $i_2 \mathrel{-}= 2$
2: **loop**
3:     $y_1 \mathrel{-}= 1$ $i_1 \mathrel{+}= 1$ $i_3 \mathrel{-}= 1$
4: **loop**
5:     $y_2 \mathrel{-}= 1$ $y_1 \mathrel{+}= 1$ $i_3 \mathrel{-}= 1$
6: **loop**
7:     $y_1 \mathrel{-}= 1$ $y_2 \mathrel{+}= 1$ $i_3 \mathrel{-}= 1$
8: **loop**

9:      i_1 $-=$ 1 y_1 $+=$ 1 i_3 $-=$ 1

We aim to show that if $i_1 + y_1 + y_2 = n$ then the total effect of loops in lines 2-8 on the counter i_3 is the decrease by at most $2n$ and the decrease is exactly $2n$ if and only if initially $i_1 = 0$. As in line 1 counter i_2 is decreased by 2 then counter i_3 have to be decreased by exactly $2n$ in the rest of the program fragment, as finally values of both i_2 and i_3 need to be zero. Therefore it remains to argue about the decrease of counter i_3. The easiest way to see this is to see the loop in lines 2-3 as transferring value of counter y_1 to counter i_1, but maybe not fully. We write it $y_1 \mapsto i_1$. Similarly next loops correspond to transfers $y_2 \mapsto y_1$, $y_1 \mapsto y_2$ and $i_1 \mapsto y_1$, each of the transfers may be not fully realised. The total decrease on c_3 equals exactly the total amount of value transferred during all the four loops. Notice now that the value of original y_2 can be used in at most two transfers: $y_2 \mapsto y_1$, $y_1 \mapsto y_2$. Similarly the value of original y_1 can be used either only in $y_1 \mapsto y_2$ or in two transfers $y_1 \mapsto i_1$ and $i_1 \mapsto y_1$. Value of original i_1 can be used only in the transfer $i_1 \mapsto y_2$. Therefore the total amount of the transfer equals at most $2y_1 + 2y_2 + i_1$ and this equals $2n$ only if $i_1 = 0$. Moreover in order to obtain the transfer of exactly $2n$ we need to perform all the transfers fully. Therefore one can easily observe that in that case after the **zero-test**(i_1) values of counters i_1, y_1 and y_2 come back to the same values as before the **zero-test**(i_1). This finishes the proof that the above fragment faithfully implements **zero-test**(i_1).

In order to finish the proof of Lemma 7 it is enough to observe that $\mathrm{SIZE}(V_k)$ is bounded by $C \cdot \mathrm{SIZE}(V_{k-1})$ for some constant $C \in \mathbb{N}$. Indeed V_k was obtained from V_{k-1} by adding several new lines and for each operation $x \mathrel{+}= 1$ (or $x \mathrel{-}= 1$) for any controlled counter x adding an appropriate operation $c \mathrel{+}= (n+1) - i$ (or $c \mathrel{-}= (n+1) - i$). Implementing each operation on counter c requires a few new lines which is responsible by multiplying the size by some constant, but not more. Thus altogether indeed $\mathrm{SIZE}(V_k) \leq C \cdot \mathrm{SIZE}(V_{k-1})$ which finishes the proof.

6   Future research

We have settled the complexity of the reachability problem for VASSes, but there are still many intriguing questions in this topic. Here we present some, which we think need investigation in the future works of our community.

We still lack understanding of VASSes in small dimensions. The most striking example is the reachability problem for 3-VASSes, where the complexity gap is between PSpace-hardness (inherited from dimension 2 [1]) and algorithm working in $\mathcal{F}_7$ [17]. We do not see any way of applying our techniques or any other known techniques of proving lower bounds to dimension 3, as all of them require some additional counter, which helps to enforce the run to be exact at some control configurations. We conjecture that the reachability problem is actually elementary for VASSes in dimension 3 and maybe even in a few higher dimensions. Showing this seems to be a very challenging task.

Another future goal is to settle the exact complexity of the reachability problem for d-VASSes depending on d. After our work showing $\mathcal{F}_d$-hardness in dimension $6d$ and independent work by Jérôme Leroux [13] showing the same in dimension $4d + 9$ a lot of improvement appeared very recently. Sławomir Lasota in [12] building on results in the arXiv version of this paper [4] improved the $\mathcal{F}_d$-hardness result to dimension $3d + 2$. Also Jérôme Leroux improved his own result and showed $\mathcal{F}_d$-hardness for VASSes in dimension $2d + 4$ [14]. Thus the lower bounds to improve are Tower-hardness for $d \geq 10$ [14], ExpSpace-hardness as well for $d \geq 10$ [14] and NP-hardness for $d \geq 7$ for unary encoding [3] and the best published upper bounds are stated in [17] to be $\mathcal{F}_{d+4}$ complexity for dimension d. One can also suspect that even VASSes in higher dimensions can be solved efficiently under some condition on their structure (for example avoiding some kind of bad patterns).

Complexity of the reachability problem for VASS extensions such as pushdown VASSes [15], branching VASSes [6] or data VASSes [8] is almost totally unexplored and even decidability

is not known for them. We hope that techniques introduced in this paper may help better understanding the mentioned extensions of VASSes and in particular prove some complexity lower bounds.

References

[1] Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSpace-complete. In *Proceedings of LICS 2015*, pages 32–43, 2015.

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

[3] Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. Reachability in fixed dimension vector addition systems with states. In *Proceedings of CONCUR 2020*, pages 48:1–48:21, 2020.

[4] Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. *CoRR*, abs/2104.13866, 2021.

[5] Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in two-dimensional unary vector addition systems with states is NL-complete. In *Proceedings of LICS 2016*, pages 477–484, 2016.

[6] Diego Figueira, Ranko Lazic, Jérôme Leroux, Filip Mazowiecki, and Grégoire Sutre. Polynomial-space completeness of reachability for succinct branching VASS in dimension one. In *Proceedings of ICALP 2017*, pages 119:1–119:14, 2017.

[7] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In *Proceedings of CONCUR 2009*, pages 369–383, 2009.

[8] Piotr Hofman, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, Sylvain Schmitz, and Patrick Totzke. Coverability trees for Petri nets with unordered data. In *Proceedings of FoSSaCS 2016*, pages 445–461, 2016.

[9] John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979.

[10] S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of STOC 1982*, pages 267–281, 1982.

[11] Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992.

[12] Sławomir Lasota. Improved Ackermannian lower bound for the VASS reachability problem. *CoRR*, abs/2105.08551, 2021.

[13] Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *Proceedings of FOCS 2021. To appear.* IEEE Computer Society, 2021.

[14] Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. *CoRR*, abs/2104.12695, 2021.

[15] Jérôme Leroux, M. Praveen, Philippe Schnoebelen, and Grégoire Sutre. On functions weakly computable by pushdown Petri nets and related systems. *Log. Methods Comput. Sci.*, 15(4), 2019.

[16] Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *Proceedings of LICS 2015*, pages 56–67, 2015.

[17] Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Proceedings of LICS 2019*, pages 1–13. IEEE, 2019.

[18] Richard J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, 1976.

[19] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proceedings of STOC 1981*, pages 238–246, 1981.

[20] Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. URL: `https://doi.org/10.1145/2858784`, `doi:10.1145/2858784`.