Reachability in Vector Addition Systems is Ackermann-complete

Wojciech Czerwiński
University of Warsaw

Łukasz Orlikowski
University of Warsaw

Abstract

Vector Addition Systems and equivalent Petri nets are well established models of concurrency. The central algorithmic problem for Vector Addition Systems with 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.

2012 ACM Subject Classification Theory of computation → Parallel computing models

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

Digital Object Identifier 10.4230/LIPIcs...

Funding *Wojciech Czerwiński*: Supported by the ERC grant LIPA, agreement no. 683080.

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. It was considered in the 70-ties and shown to be ExpSpace-hard by Lipton in 1976 [11]. Decidability of the reachability problem was first proven by Mayr in 1981 [12]. The construction was simplified later by Kosaraju [6] and Lambert [7]. Their approach was to use an equivalent model of VAS with states (VASS) 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. 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 [4]. Another interesting result was NP-completeness of the problem in binary encoded one-dimensional VASSes [1].

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 $\mathcal{F}_{\omega^3}$ [9]. The same authors have improved their result recently in 2019 showing that the problem can be solved in the Ackermann complexity class (denoted $\mathcal{F}_\omega$) [10]. They have actually shown that the reachability problem for d-dimensional VASSes (denoted d-

VASSes) can be solved in the complexity class $\mathcal{F}_{d+4}$, 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. 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 depend on the dimension of the considered 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 [10] it implies the following.

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

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 $S \subseteq [1, d]$ and 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 $\mathrm{Conf} = Q \times \mathbb{Z}^d$. Transition $(p, t, q) \in T$ can be fired in configuration $r(v) \in \mathbb{N}^d$ 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 vector $t \in \mathbb{N}^d$, we write $\mathrm{eff}((p, t, q)) = t$. A sequence of triples $\rho = (c_1, t_1, c_1'), (c_2, t_2, c_2'), \ldots, (c_n, t_n, c_n') \in \mathrm{Conf} \times T \times \mathrm{Conf}$ 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, $\mathrm{eff}(\rho) = t_1 + \ldots + t_n$. Such a run ρ is said to be *from* configuration c_1 *to* configuration c_n'. We write then $c_1 \xrightarrow{\rho} c_n'$ slightly overloading the notation or simply $c_1 \longrightarrow c_n'$ if ρ is irrelevant. We also say then that configuration c_1 *reaches* a configuration c_n' or c_n' *is reachable from* c_1. By $\textsc{Reach}(\mathrm{src}, V) = \{c \mid \mathrm{src} \longrightarrow c\}$ we denote the set of all the configurations reachable from configuration src and we call it a *reachability set*. Of also write simply $\textsc{Reach}(\mathrm{src})$ is VASS V is clear from the context. The following problem is the main focus of this paper.

maybe reformulate

Reachability problem for VASSes
Input A VASS V and two its configurations src, trg
Question Is trg reachable from src in V?

A Vector Addition System (VAS) is a VASS with only one state (thus the state can be ignored). It is a 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.

accepting run

Counter programs

We often work with VASSes in 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. Counter program is a sequence of instructions, each one being either the counter values modifications of the form $\mathsf{x_1} \mathrel{+}= a_1 \quad \ldots \quad \mathsf{x_d} \mathrel{+}= a_d$ or a loop of the form

```
loop
    P
```

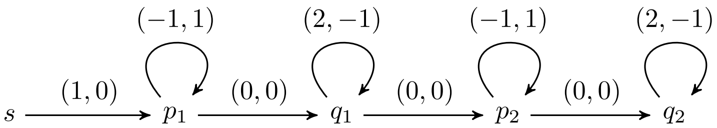
where P is another counter program. Such a counter program with k instructions and d counters $x_1, \ldots, x_d$ represents a d-VASS V with states $q_1, \ldots, q_k, q_{k+1}$ (and some other ones) such that:

- state q_1 is the *source* state of V and q_{k+1} is the *target* state
- if the i-th line is of the form $\mathsf{x_1} \mathrel{+}= a_1 \quad \ldots \quad \mathsf{x_d} \mathrel{+}= a_d$ then in V there is a transition $q_i \xrightarrow{v} q_{i+1}$ for $v[i] = a_d$
- if the i-th line is the loop with body equal counter program P then in V there are transitions $q_i \xrightarrow{0^d} \mathrm{src}_P$ and $\mathrm{trg}_P \xrightarrow{0^d} q_i$ where src_P and trg_P are source and target states of VASS V_P represented by program P
- if the i-th line is the loop then in V there is a transition $q_i \xrightarrow{0^d} q_{i+1}$.

▶ **Example 3.** The following counter program

```
x += 1
loop
    x -= 1    y += 1
loop
    x += 2    y -= 1
loop
    x -= 1    y += 1
loop
    x += 2    y -= 1
```

represents the following 2-VASS, state names are chosen arbitrary.



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

▶ **Example 4.** The following counter program uses the macro **for**. For $n = 2$ it is equivalent to the above example.

```
x += 1
for i := 1 to n do
    loop
        x -= 1    y += 1
    loop
        x += 2    y -= 1
```

The counter program represents the following 2-VASS.

$$\begin{tikzcd}
s \arrow[r, "{(1,0)}"] & p_1 \arrow[loop above, "{(-1,1)}"] \arrow[r, "{(0,0)}"] & q_1 \arrow[loop above, "{(2,-1)}"] \arrow[r, "{(0,0)}"] & \cdots \arrow[r, "{(0,0)}"] & p_n \arrow[loop above, "{(-1,1)}"] \arrow[r, "{(0,0)}"] & q_n \arrow[loop above, "{(2,-1)}"]
\end{tikzcd}$$

We say that a counter program has an *accepting run* if there is a run in the corresponding VASS from source state with zero counter values to target state with zero counter values.

Other situations

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 [14, 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 $F_2 = 2^n$ and $F_3 = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n} = \mathrm{Tower}(n)$. We define the Ackermann function as $A(n) = F_n(n)$. We often also say that Ackermann function is the function F_ω from the fast growing hierarchy.

Based on functions F_k we define complexity classes $\mathcal{F}_k$ also following definitions in [14]. The complexity class $\mathcal{F}_k$ contains all the problems, which can be solved in time $f \circ g$, where $f \in F_{k-1}$ and $g \in F_k$. The idea is that problems in $\mathcal{F}_k$ can be solved by a F_{k-1} 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 $\mathrm{Tower}(n)$, but also for example those, which can be solved in time $\mathrm{Tower}(2^{2^n})$, as $\mathrm{Tower}(2^{2^n}) = \mathrm{Tower}(n) \circ 2^{2^n}$. It is well known that complexity classes $\mathcal{F}_k$ have natural complete problems connected with counter automata, which we formulate precisely in Section 3.

Correct the definition

3 Outline

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

Weak computation

It has been well known since a long time that some decidable problems for VASSes are Ackermann-hard. An early example of that phenomena is Ackermann-hardness of inclusion of two finite reachability sets of VASSes [13]. There is a well-known family of VASSes V_d in growing dimension $d+1$ with reachability sets being finite, but of size roughly equal $F_d(n)$, where $n = |V_d|$ equals the size of V_d. In such a d-VASS there are runs, with counters reaching at some moment value roughly $F_d(n)$, but there are also runs reaching the same states with pretty low counter values. On a very high level one can see our construction as forcing VASSes to choose the runs, which achieve very high counter values.

define size of VASS

We remind here the basic intuition behind $(d+1)$-VASSes with finite reachability sets, which may reach counter values around $F_d(n)$. A simple idea is to construct a VASS, which directly implements the F_d function, it uses its last counter in order to n times compose the F_{d-1} function, which is implemented by its other counters.

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

```
1: x1 += 1    x3 += n
2: loop
3:     loop
4:         x1 += 2    x2 −= 1
5:     loop
6:         x1 −= 1    x2 += 1
7:     x3 −= 1
```

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

```
1: x1 += 1    x3 += 1    ...    xd += 1    xd+1 += n
2: loop
3:     V_{d-1}
4:     loop
5:         x1 −= 1    xd += 1
6:     xd+1 −= 1    x3 += 1    ...    xd−1 += 1
```

Our aim is to show by induction on d that $(1, 0, 1^{d-2}, k-1) \xrightarrow{V_d} (F_d(k), 0^d)$. We start the induction from $d = 2$, one can easily see that indeed $(1, 0, n) \xrightarrow{V_2} (2^n, 0, 0)$. For induction step assume by induction assumption that $(1, 0, 1^{d-3}, k) \xrightarrow{V_{d-1}} (F_{d-1}(k), 0^{d-1})$. Let us denote by U_d the lines 2-6 of V_d. We therefore have that

$$
\begin{aligned}
(1, 0, 1^{d-3}, k-1, \ell) &\xrightarrow{(3)} (F_{d-1}(k), 0^{d-1}, \ell) \xrightarrow{(4-5)} (1, 0^{d-1}, F_{d-1}(k) - 1, \ell) \\
&\xrightarrow{(6)} (1, 0, 1^{d-3}, F_{d-1}(k) - 1, \ell - 1),
\end{aligned}
$$

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

$$
\begin{aligned}
(1, 0, 1^{d-3}, 1, k-1) &\xrightarrow{(3-6)} (1, 0, 1^{d-3}, F_{d-1}(1) - 1, k-1) \\
&\xrightarrow{(3-6)} (1, 0, 1^{d-3}, F_{d-1} \circ F_{d-1}(1) - 1, k-2) \\
&\xrightarrow{(3-6)} \ldots \xrightarrow{(3-6)} (1, 0, 1^{d-3}, \underbrace{F_{d-1} \circ \ldots \circ F_{d-1}}_{k-1}(1) - 1, 0) \\
&\xrightarrow{(3)} (\underbrace{F_{d-1} \circ \ldots \circ F_{d-1}}_{k}(1), 0^d) = (F_d(k), 0^d).
\end{aligned}
$$

which finishes the induction step. Therefore indeed V_d can lift its counter values as high at $F_d(n)$.

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 related to 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 exactly zero. Such transitions are called *zero-tests*. equal to? We say that a run of counter automaton is *accepting* if it starts in the distinguished initial state with all counters equal zero and finishes in distinguished accepting state also with all counters equal zero. A run is N-bounded if all the counters along this run have values not Or maybe counter values are important at acceptance?

exceeding N. It is a folklore that the following problem is $\mathcal{F}_k$-hard for $k \geq 3$ (for a similar problem see [14, 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 $k \geq 3$, automaton $\mathcal{A}$ and number $n \in \mathbb{N}$ constructs a $6k$-VASS together with source and target configurations s and t such that $s \longrightarrow t$ 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-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 the intuitive level the argument proceeds as follows. Assume a machine (in our case VASS) has access to triples of the form (M, x, Mx). Then it can use them to perform exactly x sequences of actions, whatever this 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 x. 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 x sequences of actions can decrease the third counter maximally by Mx and moreover if this counter was decreased by exactly Mx 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 zero we check whether there were exactly x sequences and in all the sequences there were exactly M actions. Below the exploit this idea and more precisely explain what kind of VASS we need to prove $\mathcal{F}_k$-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 test-counter $t \in [1, d]$ is an *M-generator* if:

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

We call the counter 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 guarantied 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.

▶ **Lemma 5.** *For any d-VASS (V, s, q) with $d \geq 9$, 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. We first run the d-VASS (V, s, q), which outputs a triple $(c_1, c_2, c_3, 0^{d-3})$ under the condition that the last counter equals zero. In the rest of the run we do not modify the last counter in order to assure (by setting $\mathrm{trg}[c_d] = 0$) that indeed this last counter equals zero at output of V. We need to simulate three counters of 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 decrease 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 guarantied 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.

Maybe counter should be everywhere written as x, not just x

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 $(c_2, c_3) = (x, Mx)$ generated by the M-generator for that. Recall that using this pair we are able to perform x 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 0 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 inspect the code to see that it indeed reflect 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 are 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 indeed runs from src to trg of $V_\mathcal{A}$ are in one-to-one correspondence with M-bounded runs of automaton $\mathcal{A}$. ◀

Our approach is therefore to construct $6k$-VASSes of size $\mathrm{poly}(n)$ which are $F_k(n)$-generators. We will do it by induction on k, using $6(k-1)$-VASSes, which are $F_{k-1}(n)$-generators.

Amplifiers

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

- if $p_{\text{in}}(a, x, ax, 0^{d-3}) \longrightarrow p_{\text{out}}(v, b, y, z)$ for $v \in \mathbb{N}^d$ with $v[T] = 0$ then $v = 0^{d-3}$, $b = f(a)$ and $z = by$
- for each $y \in \mathbb{N}$ there exists an $x \in \mathbb{N}$ such that $p_{\text{in}}(a, x, ax, 0^{d-3}) \longrightarrow p_{\text{out}}(0^{d-3}, f(a), y, f(a)y)$.

In other words, intuitively, if an amplifier inputs triples (a, x, ax) it outputs triples $(f(a), y, f(a)y)$ and moreover each such triple can be outputted 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 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 $f(a)$-generator of size linear in $|V| + a$.*

Proof. We construct the $f(n)$-generator as follows. In its initial state we have a loop with the effect $(0, 1, a)$, thus after x applications of it we get vector $(0, x, ax)$. Then a transition with effect $(a, 0, 0)$ leading to the input state of the f-amplifier, so the f-amplifier inputs triple (a, x, ax). Immediately from the definition of 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 $v[t]$ fulfil $v = 0^{d-3}$, $x = f(a)$, $z = f(a)y$ and additionally such configurations for all $y \in \mathbb{N}$ can be reached, which finishes the proof. ◀

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, which is an F_k-amplifier.*

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 amplifier for much smaller functions without adding too many new counters. The proof of Lemma 7 is presented in Section 5.

4 Zero-tests

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

▶ **Example 8.** Imagine first you are given a VASS run ρ and you 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,

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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. 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 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 value $3y_1 + 2y_2 + y_3$. The realise it in the following way. Every increase of x by a during ρ_1 is reflected in the increase of the added counter by $3a$. Similarly, an increase of x by a on ρ_2 is reflected by the increase of the added counter by $2a$ and on ρ_3 just by a. After configuration c_3 the added counter is not modified. Therefore testing the added counteer for 0 on 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.

▶ **Lemma 9.** *Let ρ be a run of a $(d+1)$-VASS V starting from configuration* src *and finishing in configuration* trg*, with the aim that we want to check first d counters for being zero in some configurations on ρ and the last counter will be used to perform these tests. Let $c_1, \ldots, c_n$ be configurations on ρ and let $c_0 = \mathrm{src}$, $c_{n+1} = \mathrm{trg}$. Let ρ_i for $i \in [1, n+1]$ be the part of ρ starting in c_{i-1} and finishing in c_i. Let $S_1, \ldots, S_n \subseteq [1, d]$ be the counters, which we want to zero-test in configurations $c_1, \ldots, c_n$, respectively and let $n_{k,i} = |\{j \geq k \mid i \in S_j\}|$ for $i \in [1, d], j \in [1, n+1]$ be the number of zero-tests, which we want to perform on the i-th counter after the part ρ_k. Then if:*

(1) $\mathrm{src}[d+1] = \sum_{i=1}^{d} n_{0,i} \cdot \mathrm{src}[i]$*;*

(2) *for each $k \in [1, n+1]$ we have $\mathrm{eff}(\rho_k, d+1) = \sum_{i=1}^{d} n_{k,i} \cdot \mathrm{eff}(\rho_k, i)$; and*

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

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

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

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

$$\mathrm{SUM} = \sum_{i \in [1,n], j \in S_i} c_i[j] = \sum_{i \in [1,n], j \in S_i} (c_0[j] + \sum_{k=1}^{i} \mathrm{eff}(\rho_k, j)).$$

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

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

where the first equation follows from (1) and (2), the second one from the fact that $\mathrm{eff}(\rho_{n+1}, d+1) = 0$ and the last one follows from (3). This shows that $\mathrm{trg}[d+1] = 0$ indeed implies $\mathrm{SUM} = 0$ and finishes the proof. ◀

In such a situation as in the lemma we say the last counter *controls* the other counters.

Below we present two examples of the application of Lemma 9: 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. We often name the counter which guaranties that other counter are equal zero at certain moments the *checksum* counter.

▶ **Example 10.** We consider the 2-VASS from Example 4 with one checksum counter added, the counter c. We analyse runs starting from all zeros and finishing in all zeros. We will observe that adding the checksum 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. 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
5:     loop
6:         x += 2    y −= 1    c += n − i − 1
7: loop
8:     x −= 1
```

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

$$\begin{tikzcd} s \arrow[r, "{(1,0,n)}"] & p_1 \arrow[loop above, "{(-1,1,0)}"] \arrow[r] & q_1 \arrow[loop above, "{(2,-1,n-2)}"] \arrow[r] & \cdots \arrow[r] & p_n \arrow[loop above, "{(-1,1,0)}"] \arrow[r] & q_n \arrow[loop above, "{(2,-1,-1)}"] \arrow[r] & t \arrow[loop above, "{(-1,0,0)}"] \end{tikzcd}$$

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 last configurations with 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 9. Indeed, to see this we need to 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 11.** 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 VASS. In Example 10 the number of zero-tests both on x and y counters was comparable to size of VASS. Therefore different behaviour of checksum 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 [5], in which one can have a doubly-exponential run.

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

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 as n is represented in binary. Notice however nothing forces the run to reach so high value of x. Our aim is now to add a checksum 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, wherefore 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 zero-tests, while y is awaiting $(z-1)$ zero-tests. Therefore the correct changes on checksum counter c should be increase by 0 in line 4 and increase by $z \cdot 2 + (z-1) \cdot (-1) = z - 2$ in line 6. The counter program thus should be the following.

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

One can however easily observe that the operation $\mathsf{c} \mathrel{+}= \mathsf{z} - 2$ is not a valid VASS operation, as $\mathsf{z} - 2$ 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. We add three additional counters z', $\bar{\mathsf{z}}$ and $\bar{\mathsf{z}}'$ such that all the time after line 1 we have invariants $\mathsf{z} + \bar{\mathsf{z}} = n$ and $\mathsf{z}' + \bar{\mathsf{z}}' = n$. Notice that with that invariants we can easily check whether $\mathsf{z} = 0$ just by performing two operations: $\bar{\mathsf{z}} \mathrel{-}= n$ and then $\bar{\mathsf{z}} \mathrel{+}= n$, similarly we test whether $\mathsf{z}' = 0$. We introduce a macro for these operations here, writing **zero-test**(z) and **zero-test**(z').

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

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

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
5: loop
6:     z += 1    z' -= 1
7:     z̄ -= 1    z̄' += 1
8: zero-test(z')
9: c -= 2
```

The aim of line 2 is to perform $\mathsf{c} \mathrel{+}= \mathsf{z}$ and keep $\mathsf{z}+\mathsf{z}'$ constant. We need to keep $\mathsf{z}+\mathsf{z}'$ constant to be able to reconstruct in a moment original value of z. In line 3 we update $\bar{\mathsf{z}}$ and $\bar{\mathsf{z}}'$ 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 $\mathsf{c} \mathrel{+}= \mathsf{z}-2$, not $\mathsf{c} \mathrel{+}= \mathsf{z}$.

One can easily check that Lemma 9 applies to our situation and therefore if we demand $c = 0$ and $z = 0$ in the line 7 of the main VASS then both the loops in lines 3-4 and in lines 5-6 have to be executed maximally each time. Therefore at the end of the run we have $x = 2^n$, which is doubly-exponential in VASS size.

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, which is an F_k-amplifier. We prove by induction on k this fact additionally strengthened by the assumption that 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 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 till 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 $\mathsf{x}_2 \mathrel{-}= 1$ (in lines 2 or 7) and at most n operations $\mathsf{x}_3 \mathrel{-}= 1$ (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 loop in lines 1-6 has to be fired exactly $\frac{x-1}{2}$ times, as the final value of x_2 also need to be 0. Therefore final values of (x_4, x_5, x_6) are $(2n, \frac{x-1}{2}, 2n\frac{x-1}{2})$, which finishes the proof for $k = 1$.

For an induction step assume that V_{k-1} is $(6k-6)$-dimensional VASS and an F_{k-1}-amplifier. We aim at constructing $6k$-VASS, which is an F_k-amplifier. 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. The main challenge is to achieve it without adding

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

new counters for each composition. We show here how we obtain it by adding only six new counters. We crucially use the Lemma 9.

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), starting triple (s_1, s_2, s_3), checksum counter c, two auxiliary counters a_1, a_2 and $6k - 12$ counters, which are the counters of V_{k-1}, which are neither input nor output.

We first present the code for F_k-amplifier V_k, which uses illegal constructions like "**for** $i := 1$ **to** n **do**" or "$\mathsf{c} \mathrel{+}= a$" where a is 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 $(F_k(n), y, F_k(n)y)$ for some $y \in \mathbb{N}$. Let V'_{k-1} be the modified version of V_{k-1} in which the checksum counter c is also appropriately modified: each modification $\mathsf{x} \mathrel{+}= 1$ (or $\mathsf{x} \mathrel{-}= 1$) in the i-th iteration for-loop for counter x being any output or test counter of V_{k-1} is accompanied by modification $\mathsf{c} \mathrel{+}= i$ (or $\mathsf{c} \mathrel{-}= i$). Recall that test counters contain all the input counters s_1, s_2, s_3 and maybe 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}(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
```

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 arbitrarily 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 operation:

- in line 5 from a triple $(F_{k-1}^{(i-1)}(1), a_{i-1}, F_{k-1}^{(i-1)}(1) \cdot a_{i-1})$ on (s_1, s_2, s_3) we compute a triple $(F_{k-1}^{(i)}(1), a_i, F_{k-1}^{(i)}(1) \cdot a_i)$ on (o_1, o_2, o_3) under the condition that test counters of V_{k-1} are zero after its run;
- in lines 6-11 we copy 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 counters (s_1, s_2, s_3).

The checksum counter c controls counters o_1, o_2 and o_3 and the test counters of V_{k-1}, 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} 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} 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 test counters of V_{k-1} and o_j counters is tested exactly n times in the program. This is why in line 1 we update $\mathsf{c} \mathrel{+}= n$ and in line 3 we have $\mathsf{c} \mathrel{+}= 2n$. In lines 7, 9 and 11 each counter s_j awaits for $n - i$ tests, while counter o_j awaits for one more test, this is why counter c is increased here by $-1 = (n - i) \cdot 1 + (n + 1 - i) \cdot (-1) = -1$. We can easily check that counter c together with the counters it controls fulfil conditions of Lemma 9. Therefore indeed 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 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 $(F_{k-1}^{(i)}(1), a_i, F_{k-1}^{(i)}(1) \cdot a_i)$ 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 checksum counter equals 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 $\mathsf{c} \mathrel{+}= ai$ are implemented.

For that we use the input counters (i_1, i_2, i_3) and auxiliary counters a_1 and a_2. Assume that the for-loop has the following shape

1: **for** i := 1 **to** n **do**
2: $\quad\langle\text{body}\rangle$

and inside the $\langle\text{body}\rangle$ we have operations $\mathsf{c} \mathrel{+}= i$. The initial value of i_1 equals n. We exploit this in order to implement the for-loop as follows.

1: **loop**
2: $\quad\langle\text{body}\rangle$
3: $\quad \mathsf{i_1} \mathrel{-}= 1 \quad \mathsf{a_2} \mathrel{+}= 1$

As i_1 is one of test counters, so we are guarantied that the loop indeed will be iterated exactly n times. Now we show how to implement operation $\mathsf{c} \mathrel{+}= i_1$, which together shows how to implement $\mathsf{c} \mathrel{+}= n$ in lines 1 and 3 and $\mathsf{c} \mathrel{+}= i$ in the i-iteration of the for-loop. Operation $\mathsf{c} \mathrel{-}= i$ is implemented totally analogously to $\mathsf{c} \mathrel{+}= i$. At the beginning we have $i_1 = n$, $a_1 = a_2 = 0$. We will keep the invariant $i_1 + a_1 + a_2 = n$. Notice that in the i-th iteration values of counters are $(i_1, a_1, a_2) = (i, 0, n-i)$. We implement the increment $\mathsf{c} \mathrel{+}= i$ as follows.

1: **loop**
2: $\quad \mathsf{i_1} \mathrel{-}= 1 \quad \mathsf{a_1} \mathrel{+}= 1 \quad \mathsf{c} \mathrel{+}= 1$
3: **zero-test**($\mathsf{i_1}$)
4: **loop**
5: $\quad \mathsf{i_1} \mathrel{+}= 1 \quad \mathsf{a_1} \mathrel{-}= 1$
6: **zero-test**($\mathsf{a_1}$)

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 $(i_1, a_1) = (i, 0)$ it also finishes in the same valuation, but a side effect is the increment $\mathsf{c} \mathrel{+}= i_1$. Thus it remains to show that we can implement zero-tests. We present how to perform **zero-test**($\mathsf{i_1}$), the **zero-test**($\mathsf{a_1}$) is performed analogously with roles of i_1 and a_1 swapped.

1: $\mathsf{i_2} \mathrel{-}= 2$
2: **loop**
3: $\quad \mathsf{a_1} \mathrel{-}= 1 \quad \mathsf{i_1} \mathrel{+}= 1 \quad \mathsf{i_3} \mathrel{-}= 1$
4: **loop**
5: $\quad \mathsf{a_2} \mathrel{-}= 1 \quad \mathsf{a_1} \mathrel{+}= 1 \quad \mathsf{i_3} \mathrel{-}= 1$
6: **loop**
7: $\quad \mathsf{a_1} \mathrel{-}= 1 \quad \mathsf{a_2} \mathrel{+}= 1 \quad \mathsf{i_3} \mathrel{-}= 1$
8: **loop**
9: $\quad \mathsf{i_1} \mathrel{-}= 1 \quad \mathsf{a_1} \mathrel{+}= 1 \quad \mathsf{i_3} \mathrel{-}= 1$

We aim to show that if $i_1 + a_1 + a_2 = n$ then the total effect of loops in lines 2-8 on counter i_3 is decrease of 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 their values need to be both zero. Therefore it remains to argue about the decrease on counter i_3. The easiest way to see this is to see the loop in lines 2-3 as transferring value of counter a_1 to counter i_1, but maybe not fully, we can write it $a_1 \mapsto i_1$. Similarly next loops correspond to transfers $a_2 \mapsto a_1$, $a_1 \mapsto a_2$ and $i_1 \mapsto a_1$, each of the transfers maybe not be fully realised. The total decrease on c_3 equals exactly the total amount of value transferred during all the four loops. Notice now that value of original a_2 can be used in at most two transfers: $a_2 \mapsto a_1$, $a_1 \mapsto a_2$. Similarly value of original a_1 can be used either only in $a_1 \mapsto a_2$ or in two transfers $a_1 \mapsto i_1$ and $i_1 \mapsto a_1$. Value of original i_1 can be used only in the transfer $i_1 \mapsto a_2$. Therefore total amount of the transfer equals at most $2a_1 + 2a_2 + i_1$ 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, a_1 and a_2 come back to the same values as before the **zero-test**(i_1). This finishes the proof that the above fragment tests i_1 for zero and has no impact on the counters, therefore it faithfully implements **zero-test**(i_1). The proof of Lemma 7 is also finished.

6 Future research

We have settled the complexity of 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. We 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 $\mathcal{F}_7$ [10]. We do not see any way off 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.

An even more future goal is to settle exact complexity of the reachability problem for d-VASSes depending on d. Currently the best published lower bounds are Tower-hardness for $d \geq 18$ (this work), ExpSpace-hardness for $d \geq 14$ [2] and NP-hardness for $d \geq 7$ for unary encoding [3] and the best published upper bounds are stated in [10] 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 (like not containing some kind of bad patterns).

Complexity of the reachability problem for VASS extensions such as pushdown VASSes, branching VASSes or data VASSes 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.

Acknowledgements

We thank Sławomir Lasota for many inspiring and fruitful discussions.

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*, volume 171 of *LIPIcs*, pages 48:1–48:21, 2020.

4 Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in two-dimensional unary vector addition systems with states is nl-complete. In *Proceedings of LICS '16*, pages 477–484, 2016.

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

6 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of STOC '82*, pages 267–281, 1982.

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

8 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.

9 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *Proceedings of LICS'15*, pages 56–67, 2015.

10 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.

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

12 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proceedings of STOC'81*, pages 238–246, 1981.

13 Ernst W. Mayr and Albert R. Meyer. The complexity of the finite containment problem for petri nets. *J. ACM*, 28(3):561–576, 1981.

14 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`.