Reachability in fixed dimension vector addition systems with states

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

Sławomir Lasota
University of Warsaw
Poland
sl@mimuw.edu.pl

Ranko Lazić
University of Warwick
UK
R.S.Lazic@warwick.ac.uk

Jérôme Leroux
CNRS & University of Bordeaux
France
jerome.leroux@labri.fr

Filip Mazowiecki
Max Planck Institute for Software Systems
Germany
filipm@mpi-sws.org

Abstract

The reachability problem is a central decision problem for formal verification based on vector addition systems with states (VASS), which are equivalent to Petri nets and form one of the most studied and applied models of concurrency. Reachability for VASS is also inter-reducible with a plethora of problems from a number of areas of computer science. In spite of recent progress, the complexity of the reachability problem remains unsettled, and it is closely related to the lengths of shortest VASS runs that witness reachability.

We consider VASS of fixed dimension, and obtain three main results. For the first two, we assume that the integers in the input are given in unary, and that the control graph of the given VASS is flat (i.e., without nested cycles). We obtain a family of VASS in dimension 3 whose shortest reachability witnessing runs are exponential, and we show that the reachability problem is NP-hard in dimension 7. These results resolve negatively questions that had been posed by the works of Blondin et al. in LICS 2015 and Englert et al. in LICS 2016, and contribute a first construction that distinguishes 3-dimensional flat VASS from 2-dimensional VASS.

Our third result, by means of a novel family of products of integer fractions, shows that 4-dimensional VASS can have doubly exponentially long shortest reachability witnessing runs. The smallest dimension for which this was previously known is 14.

Keywords reachability problem, vector addition systems, Petri nets

1 Introduction

Context. Vector addition systems with states (shortly, VASS) [20, cf. Section 5.1], [24], vector addition systems without states (shortly, VAS) [27], and Petri nets [37], are equally expressive with well-known straightforward mutual translations. They form a long established model of concurrency with extensive applications in modelling and analysis of hardware [7, 28], software [6, 19, 25] and database [4, 5] systems, as well as chemical [1], biological [2, 36] and business [32, 43] processes (where the references are illustrative).

Two central decision problems in the context of formal verification based on that model are the following. Stated in terms of the first formalism, the input of both problems is a VASS $\mathcal{V}$, and two configurations $p(\mathbf{v})$ and $q(\mathbf{w})$.

Coverability asks whether $\mathcal{V}$ has a run starting at $p(\mathbf{v})$ and finishing at some configuration $q(\mathbf{w}')$ such that $\mathbf{w}' \geq \mathbf{w}$. Thus the final configuration of the run needs to have control that is in the given target state q and resources that are component-wise no smaller than the given target vector $\mathbf{w}$. In applications, $q(\mathbf{w})$ is typically seen as a minimal unsafe configuration, and indeed the coverability problem is fundamental for verifying safety properties.

Reachability asks whether $\mathcal{V}$ has a run starting at $p(\mathbf{v})$ and finishing at $q(\mathbf{w})$. Thus the run needs to reach the given target configuration exactly. It has turned out that verification of liveness properties amounts to solving the reachability problem [22]. Moreover, a plethora of problems from formal languages [10], logic [8, 12, 13, 26], concurrent systems [16, 18], process calculi [35], linear algebra [23] and other areas (the references are again illustrative, cf. Schmitz's recent survey [40]) are inter-reducible with the reachability problem.

The coverability problem was found ExpSpace-complete already in the 1970s [33, 38], and the reachability problem was proved decidable in the early 1980s [34]. However, the complexity of the latter has become one of the most studied open questions in the theory of verification. The best upper and lower bounds are both very recent, and are given by an Ackermannian function [29] and a tower of exponentials [11], respectively.

Submitted to LICS, 8–12 July 2020, Beijing
2020.

Fixed dimension VASS. The gaps in the state of the art on the complexity of the reachability problem are particularly vivid when the dimension is fixed. For concreteness, we focus on VASS, bearing in mind that corresponding statements in terms of VAS or Petri nets can be obtained by means of the standard translations; we refer to [40, Section 2.1] for an overview of the details, noting that in some cases the dimension is affected by a small additive constant.

The only broadly settled cases are for dimensions 1 and 2, as shown in the following table, where 'unary' and 'binary' specify how the integers in the input to the reachability problem are encoded.

	unary VASS	binary VASS
dimension 1	NL-complete [42]	NP-complete [21]
dimension 2	NL-complete [14]	PSpace-complete [3]

For dimensions $d \geq 3$, the best known bounds are from [29] and [11], namely membership of the fast-growing primitive recursive class $\mathbf{F}_{d+4}$ and hardness for $(d-13)$-ExpSpace when $d \geq 13$, respectively, which hold with both unary and binary encodings. In particular, for $3 \leq d < 13$, no better lower bounds have been known than NL for unary VASS and PSpace for binary VASS, whereas the $\mathbf{F}_{d+4}$ upper bound is far above elementary already for $d = 3$.

Flat control. The structural restriction of flatness, which is essentially that the control graph contains no nested cycles, has long played a prominent role in a number of settings in verification, cf. e.g. [9]. In fact, all the tight upper bounds for dimensions 1 and 2 recalled above can be seen as due to the effective flattability of 2-dimensional VASS [30].

Regarding the complexity of reachability for flat VASS, there has been a marked contrast in the state of the art depending on the encoding.

Binary: Thanks to reducibility to existential Presburger arithmetic [3, 17], we have NP membership, even when the dimension is not fixed. And already for dimension 1, we have NP hardness.

Unary: With the exception of dimensions 1 and 2 for which we have the NL memberships, no better upper bound than NP has been known in dimension 3 or higher. And for any fixed dimension, no better lower bound than NL has been obtained.

Interestingly, from the results of Rosier and Yen [39], we have that the coverability problem for fixed dimension flat VASS is in NP with the binary encoding and in NL with the unary encoding, which not provably better than the reachability problem as just discussed.

Main results. The NL memberships of reachability for unary VASS in dimension 2 and of coverability for unary VASS in any fixed dimension were obtained by proving that polynomially bounded witnessing runs always exist. It is therefore pertinent to ask:

> Do polynomially bounded witnessing runs exist for reachability for unary flat VASS in fixed dimensions greater than 2?

Our first main result, presented in Section 3, provides a negative answer immediately in dimension 3. We believe this is very significant for the continuing quest to understand the reachability problem, for which as we have seen there is currently a huge complexity gap already in dimension 3. Namely, 3-dimensional VASS have so far been distinguished from 2-dimensional VASS only by means of the infamous example of Hopcroft and Pansiot [24, proof of Lemma 2.8], which shows that, in contrast to the latter, the former do not have semi-linear reachability sets and are hence not flattable. However, we now have a new distinguishing feature which is present even under the restriction of flatness.

Even if polynomially bounded witnessing runs do not exist, it is conceivable that the decision problem nevertheless has low complexity, so we next ask:

> Is reachability for unary flat VASS in NL in fixed dimensions greater than 2?

We show that this is unlikely in Section 4, where our second main result establishes NP hardness in dimension 7. This provides the first concrete indication that the reachability problem is harder than the coverability problem for fixed dimension flat VASS.

Lastly, we turn to binary VASS in fixed dimensions d, where without the flat assumption, the enormous complexity gap between PSpace hardness and $\mathbf{F}_{d+4}$ membership remains for $3 \leq d \leq 13$. Given that exponentially bounded witnessing runs exist for $d = 2$ [14] (which yields PSpace membership) but not for $d = 14$ [11], we ask:

> Do exponentially bounded witnessing runs exist for reachability for binary VASS in fixed dimensions from 3 to 13?

A negative answer is provided in Section 5 by our third main result, which exhibits a family of 4-dimensional VASSes whose shortest witnessing runs are doubly exponentially long.

Technical contributions. In all three of the main results, we make use of a key technical pattern first seen in [11], namely checking divisibility of a counter x by a large integer as follows: ensure that a counter y is initially equal to x, then multiply x weakly (which a priori may nondeterministically produce an erroneous smaller result) by many integer fractions greater 1 whose product is c/d, and finally verify that $\mathsf{x} = \mathsf{y} \cdot (c/d)$ by subtracting c from x and d from y repeatedly until they are both 0. The divisibility by the large integer is ensured because the check succeeds if and only if the weak multiplications are all exact. However, much additional development has been involved:

1. For the exponentially long shortest runs in Section 3, we employ the factorial fractions also seen in [11], but in reverse order, with the construction stripped to its

essentials to minimise the dimension, and with a detailed analysis of the corresponding large integer the divisibility, by which the construction checks.

2. The NP hardness in Section 4 is by a reduction that builds on the development in the previous section, adding careful machinery that facilitates exact computations on exponentially large integers.
3. To obtain the doubly exponentially long shortest runs in Section 5, we have developed an intricate new family of sequences of fractions, where in contrast to the much simpler factorial equations, the number of distinct fractions in a sequence is logarithmic in relation to both the numerators and the denominators as well as to the length of the sequence.

2 Preliminaries

Vector addition systems with states. A *vector addition system with states* in dimension d (d-VASS, or simply VASS if the dimension is irrelevant) is a pair $\mathcal{V} = (Q, T)$ consisting of a finite set Q of states and a finite set of transitions $T \subseteq Q \times \mathbb{Z}^d \times Q$. The size of a VASS is $|Q| + |T| \cdot s$, where s is the maximum on the representation size of a vector in T. A *configuration* of a d-VASS is a pair $(p, \mathbf{v}) \in Q \times \mathbb{N}^d$, denoted $p(\mathbf{v})$, consisting of a state p and a nonnegative integer vector $\mathbf{v}$. A run of a d-VASS is a sequence of configurations

$$p_0(\mathbf{v}_0), \ \ldots, \ p_k(\mathbf{v}_k), \tag{1}$$

such that for every $1 \leq i \leq k$ there is a transition $\alpha_i = (p_{i-1}, \mathbf{w}_i, p_i) \in T$ satisfying $\mathbf{v}_{i-1} + \mathbf{w}_i = \mathbf{v}_i$. The sequence of transitions

$$\alpha_1, \ \ldots, \ \alpha_k \tag{2}$$

we call the *path* of the run (1).

We are interested in the complexity of the *reachability problem*: given a d-VASS and two configurations $p(\mathbf{v})$, $q(\mathbf{w})$ does there exist a run from $p(\mathbf{v})$ to $q(\mathbf{w})$. W.l.o.g. we can restrict $\mathbf{v} = \mathbf{w} = 0$ to be the zero vectors, as the general case polynomially reduces to such restricted case. Indeed, it suffices to add a new initial state whose only out-going transition adds $\mathbf{v}$, and likewise a new final state whose only in-going transition substracts $\mathbf{w}$. In the sequel we usually assume that VASS is additionally equipped with a pair of configurations, a source $p(\mathbf{v})$ and a target $q(\mathbf{w})$, thus $\mathcal{V} = (Q, T, p(\mathbf{v}), q(\mathbf{w}))$. This means that we do not distinguish between a VASS and a VASS reachability instance. Runs from $p(\mathbf{v})$ to $q(\mathbf{w})$ we call *halting runs* of $\mathcal{V}$.

In this paper we study the reachability problem under two further restrictions. The first restriction assumes that the dimension d is fixed. In this case it may matter, for the complexity of the reachability problem, whether the numbers appearing in the vectors in T are encoded in unary or binary. We will thus distinguish these two cases, and speak of unary, respectively binary VASS. Note that in the unary case one can assume w.l.o.g. all vectors in T to be either the zero vector, or the unit vector $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ with single 1 on some i-th coordinate, or inverse $-e_i$ thereof.

The second restriction is *flatness* and concerns cycles in runs (see e.g. [3, 30]). A run (1), or actually its path (2), is called a *simple path* if there is no repetition of states along the path; it is called a *simple cycle* if there is no repetition of states along the path except for the first and the last states which are equal: $p_0 = p_k$. A VASS is *flat* if every state admits at most one simple cycle on it. Intuitively, a flat VASS has no nested cycles.

Counter programs. We are going to represent VASSes by counter programs. A *counter program* is a numbered sequence of commands of the following types:

x += n	(increment counter x by n)
x −= n	(decrement counter x by n)
goto L **or** L'	(jump to either line L or line L')

except that the first and the last command of the program, respectively, are of the form

initialise to 0	(initialise all counters to zero);
halt if $x_1, \ldots, x_l = 0$	(terminate provided all listed counters are zero).

We note that in the unary case, increments x += m and decrements x −= m can be written as m consecutive unitary increments x += 1 and decrements x −= 1, respectively, introducing only linear blow-up. Clearly, in the binary case this would lead to an exponential blow-up.

Indeed, a counter program $\mathcal{P}$ represents a VASS (in fact, a VASS reachability instance) of dimension equal to the number of counters used in $\mathcal{P}$, with a separate state for every line in $\mathcal{P}$. The increment and decrement commands in P are simulated by transition vectors of the VASS. The source and target configurations of the VASS correspond to the first and last line of $\mathcal{P}$. The size of the VASS is linear with respect to the size of the program. This convenient representation was adopted e.g. in [11, 15].

Accordingly with runs of a VASS, we speak of runs of a counter program (in particular, values of counters along a run are nonnegative) with the proviso that the initial value of all counters is 0. A run is *halting* if it has successfully executed its (necessarily last) **halt** command; otherwise, the run is *partial*. The reachability problem for a VASS translates into the question whether there exists a halting run in a counter program.

Note that a counter program does not need to test for zero all counters in the final **halt** command; for the sake of presentation it is convenient to allow for halting runs with non-zero final value of certain (irrelevant) counters. On the other hand, formally, our intention is that a counter program represents a VASS reachability instance with the zero target vector. This incompatibility can be circumvented by assuming that counter programs are implicitly completed with additional loops allowing to decrease every untested counter just before executing the **halt** command.

When analysing fragments of counter programs which neither start with **initialise** nor end with **halt**, we consider explicit *initial* and *final* values of counters. Note however that due to nondeterministic **goto** command, final values are not uniquely determined by initial ones.

When writing counter program we use the following syntactic sugar. First, we write **goto** L instead of **goto** L **or** L. Second, whenever a program has the following form:

```
1: goto 4 or 2
2: <iterated commands>
3: goto 1
4: <remaining commands>.
```

and therefore it repeats the block of commands in line 2 some nondeterministically chosen number of times (possibly zero, possibly infinite), we write this program as:

```
loop
    <iterated commands>
<remaining commands>.
```

In the sequel we will only occasionally use **goto** commands *explicitly*. Observe that a counter program without explicit **goto** commands, but using *unnested* **loop** commands (which *implicitly* use **goto** commands), always represents a flat VASS.

We end this section with examples of counter programs performing certain weak computations. A program fragment *weakly* computes a number b in counter x, if all runs end with $\mathsf{x} \leq b$, and there is a run that ends with $\mathsf{x} = b$. On the way we also introduce macros to be used later to facilitate writing complex programs.

As a preparation, consider the program in Algorithm I which weakly computes the initial value of x multiplied by $\frac{c}{d}$.

Algorithm I Weak multiplication by $\frac{c}{d}$, for $c > d$.

```
loop
    x −= 1    y += 1
loop
    x += $c$    y −= $d$
```

Let x_0, y_0 and x_1, y_1 be initial and final values, respectively, of counters x, y. We claim that the sum of final values is at most $\frac{c}{d}$ times larger than the sum of initial values. Moreover, it is exactly $\frac{c}{d}$ times larger if, and only if, both loops are iterated *maximally*: the first loop exits only when the counter x, decreased in its every iteration, reaches the minimal possible value 0; and likewise the second loop exits only when the counter y reaches 0. Enforcing maximal iteration of loops will be our fundamental technical objective in the sequel.

Claim 1. *Let x', y' be the values of counters* x, y *at the exit from the first loop. Then $x_1 + y_1 \leq (x_0 + y_0) \cdot \frac{c}{d}$. Moreover, $x_1 = (x_0 + y_0) \cdot \frac{c}{d}$ if, and only if, $x' = y_1 = 0$.*

Proof. As $x' + y' = x_0 + y_0$ and $c > d$ we get:

$$x_1 + y_1 \leq x' + \frac{c}{d} \cdot y' \leq \frac{c}{d} \cdot (x_0 + y_0). \tag{3}$$

We now concentrate on the second part of the claim. If $x' = y_1 = 0$ then $d \mid (x_0 + y_0)$ and thus $x_1 = (x_0 + y_0) \cdot \frac{c}{d}$. For the opposite direction, if $y_1 \neq 0$ then $x_1 < x_1 + y_1 \leq (x_0 + y_0) \cdot \frac{c}{d}$. If $x' \neq 0$ then by (3) we get

$$x_1 \leq x' + \frac{c}{d} \cdot y' < \frac{c}{d} \cdot (x' + y') = \frac{c}{d} \cdot (x_0 + y_0). \qquad \square$$

The counter program $\mathcal{W}_b$ shown in Algorithm II weakly computes a number b, assuming that $b_m \ldots b_0 = \textsc{Bin}(b)$ is the binary representation of b (thus the oldest bit $b_m = 1$). The **halt** command is omitted as no zero-testing is relevant in this example.

Algorithm II Counter program fragment $\mathcal{W}_b$.

```
initialise to 0
for $i$ := $m$ downto 0 do
    loop
        x −= 1    y += 1
    loop
        x += 2    y −= 1
    if $b_i = 1$ then x += 1
```

We use **for** and **if then** preprocessing macros with the following semantics. The macro

for i := m **downto** 0 **do**
    <program fragment>

is understood as $(m + 1)$-fold repetition of <program fragment>:

<program fragment> $(i = m)$
<program fragment> $(i = m - 1)$
...
<program fragment> $(i = 0)$

for $i = m, m - 1, \ldots, 0$. It is important that i is not a counter but a meta-variable that is treated as a constant in every program fragment. By convention we use different fonts for counters and meta-variables: i is a counter while i is a meta-variable. Furthermore in every copy, say

<program fragment> $(i = k)$

at every appearance of the macro

if $\varphi(i)$ **then** <optional program fragment>

the formula $\varphi(i)$ is evaluated and, if it evaluates positively then macro is replaced by <optional program fragment>, otherwise the macro is removed. Specifically, consider for example $m = 1$ and $b = 2$, i.e., $b_1 = 1$ and $b_0 = 0$. Unfolding of the macros appearing in $\mathcal{W}_2$ yields the counter program

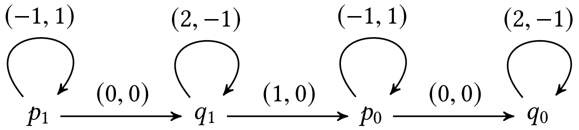


Figure 1. A 2-VASS represented by the program $\mathcal{W}_2$. The first coordinate corresponds to the value of counter x and the second one to the value of counter y.

shown in Algorithm III. Figure 1 shows the corresponding 2-VASS.

Algorithm III Unfolding of macros in $\mathcal{W}_2$.

```
 1: initialise to 0
 2: loop
 3:     x −= 1    y += 1
 4: loop
 5:     x += 2    y −= 1
 6: x += 1
 7: loop
 8:     x −= 1    y += 1
 9: loop
10:     x += 2    y −= 1
```

Clearly, we do not want **for** and **if then** to be full-fledged commands operating on program counters, as this would make counter programs as powerful as Minsky machines, hence undecidable. They are just pre-processing macros that operate on meta-variables i only, and constitute syntactic sugar helpful in writing repetitive program fragments.

Proposition 2.1. *The program $\mathcal{W}_b$ weakly computes b.*

Proof. By Claim 1, the program $\mathcal{W}_b$ weakly multiplies x by 2 in lines (3)–(6). Combining this with addition of a bit in (7) gives weak computation of b. □

Remark 1. *The programs in Algorithm I and Algorithm II represent flat VASS.*

3 Exponential shortest runs

In this section we prove the following theorem.

Theorem 3.1. *There is a family of unary flat 3-VASS $(\mathcal{V}_n)_{n\in\mathbb{N}}$ of size $O(n^2)$ such that every halting run of $\mathcal{V}_n$ is of length exponential in n.*

The VASS $\mathcal{V}_n$ are represented by the counter programs $\mathcal{P}_n$ shown in Algorithm IV. The idea of multiplying by consecutive fractions $\frac{2}{1}, \frac{3}{2} \ldots \frac{n+1}{n}$ comes from [11] (cf. Algorithms I,II therein), however, we need to apply the multiplications in the reverse order. The size of $\mathcal{P}_n$ is quadratic in n, as the **for** macro unfolds n times, and the constants appearing in the increment/decrement commands, like $i+1$ in x += $i+1$, are written in unary.

Algorithm IV Counter program $\mathcal{P}_n$.

```
 1: initialise to 0
 2: x += 1    y += 1
 3: loop
 4:     x += 1    y += 1
 5: for $i$ := $n$ down to 1 do
 6:     loop
 7:         x −= 1    z += 1
 8:     loop
 9:         x += $i+1$    z −= $i$
10: loop
11:     x −= $n+1$    y −= 1
12: halt if y = 0.
```

Consider any run that reaches (but not yet executes) line 12. For every $i = n, \ldots, 1$ let x_i and z_i be the values of counters x and z, respectively, at the exit from the loop in lines 8–9. Similarly, let x'_i be the value of counter x at the exit from the loop in lines 6–7. For uniformity we write x_{n+1} and z_{n+1} for the values of x and z, respectively, just before entering the **for** macro, and call these values initial. Notice that x_{n+1} is equal to the value of counter y at that point and $z_{n+1} = 0$.

Claim 2. *For all $i = 1, \ldots, n$*

$$x_i + z_i \leq (x_{i+1} + z_{i+1}) \cdot \frac{i+1}{i}.$$

Proof. Follows by Claim 1. □

In the next claim we will focus on runs that *maximally iterate* both loops in the body of **for**, by which we mean:

- the value of x is 0 at the exit of the loop in lines 6–7;
- the value of z is 0 at the exit of the loop in lines 8–9;

Claim 3. *We have $x_1 \leq x_{n+1} \cdot (n+1)$. The equality holds if, and only if, $z_i = x'_i = 0$ for all $i = 1, \ldots, n$.*

Proof. By Claim 2 we get

$$x_1 + z_1 \leq (x_{n+1} + z_{n+1}) \cdot \prod_{i=1}^{n} \frac{i+1}{i} = (\mathrm{x}_{n+1} + \mathrm{z}_{n+1}) \cdot (n+1).$$

Since $z_{n+1} = 0$ this implies the inequality.

Now we step to the second part of the claim. If $z_i = x'_i = 0$ for all $i = 1, \ldots, n$ then by Claim 1 we get $x_i = x_{i+1} \cdot \frac{i+1}{i}$ for every i, which implies $x_1 = x_{n+1} \cdot (n+1)$.

Conversely, suppose for some i we have $z_i \neq 0$ or $x'_i \neq 0$. Then by Claim 1 we get $x_i + z_i < (x_{i+1} + z_{i+1}) \cdot \frac{i+1}{i}$. Combined with Claim 2 this yields $x_1 + z_1 < (x_{n+1} + z_{n+1}) \cdot (n+1)$, which concludes the proof as $z_{n+1} = 0$. □

For a finite subset $X \subseteq \mathbb{N}$ of natural numbers, we write $\textsc{Lcm}(X)$ for the least common multiple of all numbers in X. For $k \in \mathbb{N}$, let $\textsc{Div}(k) \subseteq \{2, \ldots, k-1\}$ denote the set of all proper divisors of k. In particular $\textsc{Div}(k) = \emptyset$ if, and only if k is prime. We will use the number $N(n)$ defined as

$$N(n) \quad := \quad \textsc{Lcm}(\{2, \ldots, n\} - \textsc{Div}(n+1)).$$

The following two claims conclude the proof of Theorem 3.1.

Claim 4. *The function N grows exponentially with respect to n. The binary representation $\textsc{Bin}(N(n))$ is computable in time polynomial with respect to n.*

Proof. For the exponential upper bound we recall that $N(n) \leq n!$. For the exponential lower bound we use the prime number theorem (proved independently by Jacques Hadamard and Charles Jean de la Vallée Poussin in 1896): the numer of primes $\pi(n)$ between 2 and n is at least $\pi(n) \geq c \cdot n^{\epsilon} - 1$ for some constants $c > 0$ and $0 < \epsilon < 1$. Since $\textsc{Lcm}(\{2 \ldots n\})$ must be divisible by all prime numbers between 2 and n and each prime number is at least 2 we get $\textsc{Lcm}(\{2, \ldots, n\}) \geq 2^{c \cdot n^{\epsilon} - 1}$. The claim follows as $N(n) \geq \textsc{Lcm}(\{2, \ldots, n\})/(n+1)$.

Computation of $N(n)$ it done by exhaustive enumeration of all non-divisors of $n+1$, computing their prime decompositions, and combining them into prime decomposition of $N(n)$. □

Claim 5. *For every initial value x_{n+1} of counter x there is at most one halting run. Such a run exists if, and only if, x_{n+1} is a positive multiple of $N(n)$.*

Proof. Recall that the last loop in $\mathcal{P}_n$ in line 11 decreases simultaneously y by 1 and x by $n+1$. Therefore, the run halts only if $\mathsf{x} \geq \mathsf{y} \cdot (n+1)$. By Claim 3 we have $\mathsf{x} \leq \mathsf{y} \cdot (n+1)$. Thus every halting run satisfies the equality $\mathsf{x} = \mathsf{y} \cdot (n+1)$. By Claim 3 we know that is possible only if $\mathsf{z}_i = \mathsf{x}'_i = 0$ for all $i = 1, \ldots, n$, which uniquely determines the run for a given x_{n+1}.

It remains to prove that a halting run exists if, and only if, x_{n+1} is a positive multiple of $N(n)$. Notice that by Claim 2 and Claim 3 in a halting run

$$\mathsf{x}_i = \mathsf{x}_{i+1} \cdot \frac{i+1}{i} = \ldots = \mathsf{x}_{n+1} \cdot \prod_{j=i}^{n} \frac{j+1}{j} = \mathsf{x}_{n+1} \cdot \frac{n+1}{i}.$$

Therefore $\mathsf{x}_{n+1} \cdot (n+1)$ must be always divisible by all numbers in $\{1 \ldots n\}$, and hence x_{n+1} needs to be a multiple of $N(n)$. Conversely, if x_{n+1} is a multiple of $N(n)$ then there is a run where all loops are iterated maximally, satisfying $\mathsf{x}_i = \mathsf{x}_{i+1} \cdot \frac{i+1}{i}$ and thus halting. □

4 NP-hardness

This section is devoted to proving NP-lower bound for flat VASS in fixed dimension.

Theorem 4.1. *The reachability problem for unary flat 7-VASS is* NP*-hard.*

As mentioned in the introduction NP-membership is already known (even in binary VASS of unrestricted dimension). Thus as a corollary we get the following result.

Corollary 4.2. *The reachability problem for flat d-VASS is* NP*-complete for every fixed $d \geq 7$.*

To prove Theorem 4.1 we reduce from the Subset Sum problem: given a set of positive integers $S = \{s_1, \ldots, s_k\} \subseteq \mathbb{N} - \{0\}$ and an integer $s_0 > 0$, determine if some subset $R \subseteq S$ satisfies $\sum_{s \in R} s = s_0$. Note that all the numbers $s_0, s_1, \ldots, s_k$ are encoded in binary.

Fix an instance $s_0, s_1, \ldots, s_k$ of the Subset Sum problem and let n be the smallest natural number such that $N(n) \geq s_0, s_1, \ldots, s_k$. By Claim 4 the number n as well as the binary representation

$$\textsc{Bin}(N(n)) = b_m \ldots b_0$$

is computable in time polynomial with respect to the sizes of binary representations of $s_0, s_1, \ldots, s_k$. Recall that $b_m = 1$. We are going to define a unary counter program $\mathcal{P}$ of polynomial size using 7 counters, as a function of $s_0, s_1, \ldots, s_k$ and n, which halts if, and only if the instance $s_0, s_1, \ldots, s_k$ is positive.

Here is the main obstacle for the reduction: the numbers in the Subset Sum problem are represented in binary, while the numbers in a counter program (to represent a VASS) are to be represented in unary. Thus the challenge is to perform exact computation of exponential numbers, using a fixed number of 7 counters.

Initial part of $\mathcal{P}$. We face the challenge by combining the weak computation given by Proposition 2.1 (that allows us to compute *at most* a required value b) with the insight of the proof of Theorem 3.1 (that enforces that the computed value is simultaneously *at least* b). Counter program $\mathcal{I}$ shown in Algorithm V implements this idea. The first half of the program, namely lines 1–8, weakly computes in counter e the value $N(n)$, and in counter f the value $N(n) \cdot (k+1)$, very much like the counter program $\mathcal{W}_b$. Note a slight difference compared to Algorithm II: the oldest bit $b_m = 1$ is treated in a different way than other bits b_i for $0 \leq i < m$, by initializing counters e and f to 1 and $k+1$, respectively, which excludes a trivial halting run that would never iterate any loop and end with the value of y equal 0. Then the second part of $\mathcal{I}$ checks, very much like the counter program $\mathcal{P}_n$, if the values are computed exactly. (Notice that the lines (9)–(16) are exactly the same as lines 5–12 of Algorithm IV.)

Using Claim 5 we obtain:

Claim 6. *Counter program $\mathcal{I}$ has exactly one halting run that computes $N(n)$ and $N(n) \cdot (k+1)$ in counters e and f, respectively, and 0 in the remaining counters $\mathsf{x}, \mathsf{x}', \mathsf{y}$ and z.*

The initial part of $\mathcal{P}$ consists of the counter program $\mathcal{I}$ without the last **halt** command in line (16), denoted below by $\mathcal{I}'$.

The remaining part of $\mathcal{P}$ will exploit the values of counters e and f computed by $\mathcal{I}'$ for turning weak computations of exponential numbers into exact ones. The counter y will

Algorithm V Counter program $\mathcal{I}$.

```
initialise to 0
x += 1    y += 1    e += 1    f += $k+1$
for $i$ := $m-1$ to 0 do
    loop
        x −= 1    x′ += 1    e −= 1    f −= $k+1$
    loop
        x += 2    x′ −= 1    e += 2    f += $2(k+1)$
    if $b_i = 1$ then x += 1    e += 1    f += $k+1$
for $i$ := $n$ down to 1 do
    loop
        x −= 1    z += 1
    loop
        x += $i+1$    z −= $i$
loop
    x −= $n+1$    y −= 1
halt if y = 0                              // removed in $\mathcal{I}'$
```

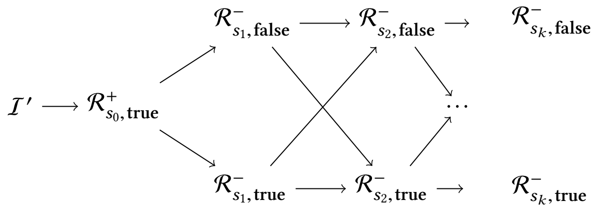
be never modified in the remaining part of $\mathcal{P}$, and will be listed in the final **halt** command of $\mathcal{P}$.

Remaining part of $\mathcal{P}$**.** The remaining part of $\mathcal{P}$ uses a distinguished counter u, initially set to 0, a counter program fragment $\mathcal{R}^{+}_{s_0,\mathbf{true}}$ and a number of counter program fragments $\mathcal{R}^{-}_{s,p}$ for $s \in \{s_1, s_2 \dots s_k\}$ and $p \in \{\mathbf{true}, \mathbf{false}\}$. We call these program fragments *components*. In every halting run of $\mathcal{P}$, the component $\mathcal{R}^{-}_{s,\mathbf{true}}$ decrements u by s while the other component $\mathcal{R}^{-}_{s,\mathbf{false}}$ has no effect on counter u. Likewise, the component $\mathcal{R}^{+}_{s_0,\mathbf{true}}$ increments u by s_0. Finally, u is zero-tested by the final **halt** command. The whole program $\mathcal{P}$ is shown in Algorithm VI.

Algorithm VI Counter program $\mathcal{P}$.

```
    $\mathcal{I}'$
    $\mathcal{R}^{+}_{s_0,\mathbf{true}}$
    goto $f_1$ or $t_1$
$f_1$: $\mathcal{R}^{-}_{s_1,\mathbf{false}}$    goto $f_2$ or $t_2$
$t_1$: $\mathcal{R}^{-}_{s_1,\mathbf{true}}$     goto $f_2$ or $t_2$
$f_2$: $\mathcal{R}^{-}_{s_2,\mathbf{false}}$    goto $f_3$ or $t_3$
$t_2$: $\mathcal{R}^{-}_{s_2,\mathbf{true}}$     goto $f_3$ or $t_3$
       ...
$f_k$: $\mathcal{R}^{-}_{s_k,\mathbf{false}}$    goto $h$
$t_k$: $\mathcal{R}^{-}_{s_k,\mathbf{true}}$
$h$:   halt if y, u, f = 0
```

For $1 \leq i \leq k$, we use f_i, respectively t_i, to denote the the first line of the program fragment $\mathcal{R}^{*}_{s_i,\mathbf{false}}$, respectively $\mathcal{R}^{*}_{s_i,\mathbf{true}}$. Every component $\mathcal{R}^{*}_{s_i,p}$, for $0 \leq i < k$ and $* \in \{+,-\}$, is followed by **goto** f_{i+1} **or** t_{i+1}. Thus for every $i = \{1 \dots k\}$ either $\mathcal{R}^{-}_{s_i,\mathbf{false}}$ or $\mathcal{R}^{-}_{s_i,\mathbf{true}}$ is executed, and the control flow in $\mathcal{P}$ can be outlined by the following directed graph, where the arrows correspond to **goto** commands:


$\mathcal{I}'$
$\mathcal{R}^{+}_{s_0,\mathbf{true}}$
$\mathcal{R}^{-}_{s_1,\mathbf{false}}$
$\mathcal{R}^{-}_{s_2,\mathbf{false}}$
$\mathcal{R}^{-}_{s_k,\mathbf{false}}$
$\dots$
$\mathcal{R}^{-}_{s_1,\mathbf{true}}$
$\mathcal{R}^{-}_{s_2,\mathbf{true}}$
$\mathcal{R}^{-}_{s_k,\mathbf{true}}$

Observe that every halting run of $\mathcal{P}$ determines a subset $R \subseteq \{1, \dots, k\}$ such that for every $i \in R$ the component $\mathcal{R}^{-}_{s_i,\mathbf{true}}$ is executed, while for every $i \notin R$ the component $\mathcal{R}^{-}_{s_i,\mathbf{false}}$ is executed.

The components $\mathcal{R}^{*}_{a,p}$**.** The component $\mathcal{R}^{*}_{a,p}$ is shown in Algorithm VII. By $\textsc{Bin}_m(a)$ we mean the binary representation of the number $a < 2^{m+1}$ consisting of exactly $m+1$ bits, and hence padded with oldest 0 bits if necessary. The aim of every $\mathcal{R}^{*}_{a,\mathbf{true}}$ is to increment (when $* = +$) or decrement (when $* = -$) a from the counter u. Counters e and f are used to ensure that the computation is exact. After the auxiliary counter v is initialised to 1, in every iteration of the **for** loop in lines 2-8 counter v is weakly multiplied by 2, so after i iterations its value is at most 2^i. In lines 5 and 11 counter f is decremented, in both cases together with counter e, hence the total decrement of f is at most the initial value of counter e. Now recall that in a halting run of $\mathcal{P}$ the values of e and f output by $\mathcal{I}'$ are $N(n)$ and $N(n) \cdot (k+1)$, respectively. As every halting run of $\mathcal{P}$ passes through exactly $k+1$ components and f is zero-tested by the final **halt** command of $\mathcal{P}$, every of the components forcedly decrements f by exactly $N(n)$. Also forcedly, after i iterations of the for loop in lines 2-8 the value of counter v is exactly 2^i. This in consequence implies that the counter u is incremented (respectively, decremented) in lines 6 and 12 by exactly a times, hence by a in total. Lines 13-14 are to revert the roles of counters e and e'.

Algorithm VII Component $\mathcal{R}^{*}_{a,p}$. Let $\textsc{Bin}_m(a) = a_m \dots a_0$.

```
 1: v += 1
 2: for $j$ := 0 to $m-1$ do
 3:     loop
 4:         v −= 1    v′ += 1
 5:         if $b_j = 1$ then e −= 1    e′ += 1    f −= 1
 6:         if $p \wedge (a_j = 1)$ then u *= 1
 7:     loop
 8:         v += 2    v′ −= 1
 9: loop
10:     v −= 1
11:     e −= 1    e′ += 1    f −= 1
12:     if $p \wedge (a_m = 1)$ then u *= 1
13: loop
14:     e += 1    e′ −= 1
```

Note that the oldest bit a_m, irrespectively of its value 0 or 1, is treated differently (in lines 9–12) from the other bits $a_{m-1} \dots a_0$ of $\textsc{Bin}_m(a)$ (treated in the body of the **for** loop in lines 3–8). This is because the auxiliary counter v needs to be multiplied by 2 exactly m times, which happens in the course of m iterations of the **for** loop, while the number of bits in $\textsc{Bin}(a)$ is $m + 1$, thus larger by 1. Consequently, in lines 9–12 the value of v is not flashed to v' nor restored back from v', and hence v is forcedly 0 at the end of $\mathcal{R}^*_{a,\mathbf{true}}$ and can be reused by the following commands of $\mathcal{P}$. No **if** macro is used in line 11 as, due to the choice of m, the oldest bit b_m of $\textsc{Bin}(N(n))$ is 1.

The above analysis applies equally well to every component $\mathcal{R}^*_{a,\mathbf{false}}$, as its computation is exactly the same as that of $\mathcal{R}^*_{a,\mathbf{true}}$, except that the value of u is not changed.

Proof of Theorem 4.1. The size of $\mathcal{P}$ is polynomial in n, k and m and it can be computed in time polynomial with respect to the size of the input: s_0, $S = \{s_1, \dots, s_k\}$. $\mathcal{P}$ represents a flat VASS since its explicite **goto** commands form a directed acyclic graph, and **loop** macros are not nested.

We prove that $\mathcal{P}$ has a halting run if, and only if, the instance $\{s_0\}, \{s_1, \dots, s_k\}$ of the subset problem is positive.

($\Longleftarrow$) Fix a subset $R \subseteq S$ with $\sum_{s \in R} s = s_0$. We define a halting run ρ that starts (cf. Claim 6) by executing $\mathcal{I}'$ to compute $N(n)$ and $N(n) \cdot (k + 1)$ in counters e and f, respectively, and 0 in the remaining counters x, y and z. Then $\mathcal{R}^+_{s_0,\mathbf{true}}$ is executed, and finally for every $1 \le i \le k$, if $i \in R$ then ρ jumps to $\mathcal{R}^-_{s_i,\mathbf{true}}$, otherwise ρ jumps to $\mathcal{R}^-_{s_i,\mathbf{false}}$. Inside every component $\mathcal{R}^*_{s_i,p}$ the run ρ *iterates all loops maximally*, by which we mean:

- the value of v is 0 at the exit of the loop in lines 3–6, and at the exit of the loop in lines 9–12;
- the value of v' is 0 at the exit of the loop in lines 7–8;
- the value of e' is 0 at the exit of the loop in lines 13–14.

It remains to observe that by iterating all loops maximally, in every component $\mathcal{R}^*_{s_i,p}$, for $0 \le i \le k$, the counter f will be decremented by exactly $N(n)$, and thus the value of f at the end of ρ is zero. Moreover, $\mathcal{R}^+_{s_0,\mathbf{true}}$ sets the counter u to s_0, and for every $s_i \notin R$ the value of counter u is preserved by $\mathcal{R}^-_{s_i,\mathbf{false}}$, and for every $s_i \in R$ the counter u is decremented by s_i in $\mathcal{R}^-_{s_i,\mathbf{true}}$. Thus the value of the counter u is 0 at the end of ρ, as well as the values of y and f, as required by the final **halt**.

($\Longrightarrow$) Consider a halting run ρ of $\mathcal{P}$, and recall that after $\mathcal{I}'$ the counter y is not modified any more, and zero-tested by the final **halt** command of $\mathcal{P}$. By Claim 6 the values of e and f after $\mathcal{I}'$ are $N(n)$ and $N(n) \cdot (k + 1)$, respectively.

The sum of counters e and e' is invariantly equal $N(n)$ as decrement of one is always accompanied by increment of the other. Thus in every component $\mathcal{R}^*_{a,p}$ visited by ρ, the counter f is decreased by at most the initial value of e, hence by at most $N(n)$. Finally, by construction of $\mathcal{P}$ the run ρ passes through exactly $k + 1$ components $\mathcal{R}^*_{a,p}$. Therefore, as f is zero-tested by the final **halt** command, we deduce.

Claim 7. *The run ρ decreases counter f by exactly $N(n)$ in every visited component $\mathcal{R}^*_{a,p}$.*

In consequence, the initial values of component $\mathcal{R}^*_{s_i,p}$, for $0 \le i \le k$, satisfy:

$$\mathsf{e} = N(n)$$
$$\mathsf{f} = N(n) \cdot (k + 1 - i)$$
$$\mathsf{v} = \mathsf{v}' = \mathsf{e}' = 0.$$

Using Claim 7 we deduce.

Claim 8. *The run ρ iterates all loops maximally in every visited component $\mathcal{R}^*_{a,p}$, except possibly the last loop in line* (14) *in the last two components $\mathcal{R}^*_{s_k,p}$.*

Possible non-maximal iteration of the last loop in $\mathcal{R}^*_{s_k,\mathbf{true}}$ and $\mathcal{R}^*_{s_k,\mathbf{false}}$ has no impact on the further analysis of the run ρ. As a direct corollary we deduce:

Claim 9. *The run ρ executes the command* $\mathsf{u} \mathrel{*}= 1$ *exactly a times in every visited component $\mathcal{R}^*_{a,\mathbf{true}}$.*

Therefore, the value of u is incremented by s_0 in component $\mathcal{R}^+_{s_0,\mathbf{true}}$. Let $R \subseteq \{s_1, \dots, s_k\}$ be the set of all s_i such that ρ passes through $\mathcal{R}^-_{s_i,\mathbf{true}}$. Again by Claim 9, for every $s_i \in R$ the value of u is decreased by s_i in component $\mathcal{R}^-_{s_i,\mathbf{true}}$, and for every $s_i \notin R$ the value of u is preserved in component $\mathcal{R}^-_{s_i,\mathbf{false}}$. Since u is zero-tested by the final **halt** command, the instance of the Subset Sum problem is necessarily positive.

***Dimension** 7.* To estimate the dimension of the VASS represented by $\mathcal{P}$, notice that $\mathcal{I}'$ uses counters $\mathsf{x}, \mathsf{x}', \mathsf{y}, \mathsf{z}, \mathsf{e}, \mathsf{f}$ and components $\mathcal{R}^*_{s_i,p}$ use additionally $\mathsf{v}, \mathsf{v}', \mathsf{e}', \mathsf{u}$. However, by Claim 6 the final values of $\mathsf{x}, \mathsf{x}', \mathsf{z}$ computed by $\mathcal{I}'$ are 0 in every halting run of $\mathcal{P}$, and hence the three counters can be reused in components $\mathcal{R}^*_{s_i,p}$, which reduces the number of counters to 7. □

5 Doubly exponential shortest runs

In this section we prove the following result:

Theorem 5.1. *There is a family of binary 4-VASS $(\mathcal{V}_n)_{n \in \mathbb{N}}$ of size $O(n^3)$ such that every halting run of $\mathcal{V}_n$ is of length doubly exponential in n.*

We define the *description size* of an irreducible fraction $\frac{p}{q}$ as $\max\{p, q\}$. We start with a key technical lemma stating existence of arbitrarily long increasing sequences of rationals greater than 1, of description size exponential with respect to k, with the property that the result of multiplying consecutive exponential powers of these rationals has only exponential (and not doubly exponential) description size.

Lemma 5.2. *For each $k \geq 1$ there are k rational numbers*

$$1 < f_1 < \ldots < f_k = 1 + \frac{1}{4^k}, \qquad (4)$$

of description size bounded by 4^{k^2+k}, such that the description size of the product

$$(f_1)^{2^1} \cdot (f_2)^{2^2} \cdot \ldots \cdot (f_k)^{2^k}$$

is bounded by $4^{2(k^2+k)}$.

Proof. For $1 \leq i \leq k$ put $r_i := \frac{4^k + 2^{k-i}}{4^k}$, and observe the following (straightforward) equalities:

$$\left(\frac{1}{r_i}\right)^{2^1} \cdot \left(\frac{1}{r_i}\right)^{2^2} \cdot \ldots \cdot \left(\frac{1}{r_i}\right)^{2^{i-1}} \cdot r_i^{2^i} = r_i^2.$$

Multiplying all these equalities yields the equality:

$$f_1^{2^1} \cdot f_2^{2^2} \cdot \ldots \cdot f_k^{2^k} = f, \qquad (5)$$

for

$$f_i = \frac{r_i}{r_{i+1} \cdot \ldots \cdot r_k} \qquad f = (r_1 \cdot \ldots \cdot r_k)^2.$$

As numerators and denominators of all r_i are bounded by 4^{k+1}, numerators and denominators of all f_i are bounded by 4^{k^2+k}, and numerator and denominator of f are bounded by $4^{2(k^2+k)}$, as required.

It remains to argue that the (in)equalities (4) hold. We notice the following relation between r_i and r_{i-1}, for $1 < i \leq k$:

$$r_i^2 = \left(1 + \frac{2^{k-i}}{4^k}\right)^2 > 1 + \frac{2^{k+1-i}}{4^k} = r_{i-1}, \qquad (6)$$

which implies

$$\frac{f_i}{f_{i-1}} = \frac{r_i \cdot (r_i \cdot \ldots \cdot r_k)}{r_{i-1} \cdot (r_{i+1} \cdot \ldots \cdot r_k)} = \frac{r_i^2}{r_{i-1}} > 1$$

and hence $f_1 < f_2 < \ldots < f_k$. For $i = k$ we have $f_k = r_k = 1 + \frac{1}{4^k}$. It thus remains to show $f_1 > 1$, which is equivalent to

$$r_1 > r_2 \cdot \ldots \cdot r_k. \qquad (7)$$

By (6) we deduce

$$r_k^{2^i} > r_{k-i},$$

by induction on i, which implies the following inequality:

$$r_k^{2^{k-1}-1} = r_k^{1+2+4+\ldots+2^{k-2}} > r_2 \cdot \ldots \cdot r_k.$$

For (7) it suffices to show, relying on the above inequality, that

$$r_1 > r_k^{2^{k-1}-1}.$$

Put $N := 2^{k-1} - 1$ for convenience. We thus need to prove:

$$r_1 > \left(1 + \frac{1}{4^k}\right)^N. \qquad (8)$$

By inspecting the expansion of the right-hand side

$$\left(1 + \frac{1}{4^k}\right)^N = \sum_{i=0}^{N} \binom{N}{i} \cdot \frac{1}{4^{ik}}$$

we observe that the right-hand side is bounded by the sum of first N elements of a geometric progression, which, in turn, is bounded by the sum of the whole infinite one:

$$\left(1 + \frac{1}{4^k}\right)^N \leq 1 + \frac{N}{4^k} + \frac{N^2}{4^{2k}} + \ldots + \frac{N^N}{4^{Nk}} < \frac{1}{1 - \frac{N}{4^k}}.$$

Thus for showing (8) it is sufficient to prove the inequality

$$r_1 > \frac{1}{1 - \frac{N}{4^k}},$$

which is equivalent to

$$\left(1 - \frac{2^{k-1} - 1}{4^k}\right)\left(1 + \frac{2^{k-1}}{4^k}\right) > 1.$$

The latter inequality is easily verified to hold true as

$$\frac{1}{4^k} > \frac{2^{k-i} - 1}{4^k} \cdot \frac{2^{k-i}}{4^k}.$$

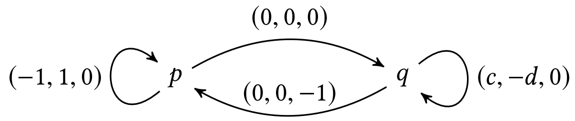
The inequality (8) is proved, and hence so is Lemma 5.2. □

Idea of the construction. A distinguished counter x in the 4-VASS $\mathcal{V}_k$ will play a special role: in every halting run, x will be multiplied by consecutive powers

$$(f_k)^{2^k}, \ (f_{k-1})^{2^{k-1}}, \ \ldots, \ (f_1)^{2^1},$$

as in Lemma 5.2, and every such multiplication will have to be exact. As denominator of the irreducible form of f_k has to be at least 2, the counter x, just before the very first multiplication by $(f_k)^{2^k}$, has to be divisible by the denominator of f_k to the power 2^k, which is necessarily doubly exponential in k. In consequence, every halting run will have to be doubly exponentially long.

As before, the main difficulty is to turn weak multiplications into exact ones. To this aim we will rely on the equation (5) and on a well-known weakly exponentiating 3-VASS gadget of Hopcroft and Pansiot [24]:



The counter program representing the gadget is outlined in Algorithm VIII, with counters x, y and z corresponding to dimension 1, 2 and 3, respectively.

Algorithm VIII Counter program fragment $\mathcal{HP}(c, d)$.

```
loop
    loop
        x −= 1    y += 1
    loop
        x += c    y −= d
    z −= 1
```

Proposition 5.3. *Consider program fragment $\mathcal{HP}(c,d)$ for an irreducible fraction $\frac{c}{d} > 1$, and initial values x_0, y_0, z_0 of counters* x, y *and* z. *In every run, the respective final values x_1, y_1, z_1 satisfy*

$$x_1 + y_1 \le (x_0 + y_0) \cdot \left(\frac{c}{d}\right)^{z_0 - z_1}.$$

Moreover, there is a run satisfying $x_1 = (x_0 + y_0) \cdot \left(\frac{c}{d}\right)^{z_0}$ if, and only if, $x_0 + y_0$ is divisible by d^{z_0}. In this case $y_1 = z_1 = 0$.

Proof. The two inner loops (lines 2–5) coincide with the counter program fragment shown in Algorithm I. As the outer loop is executed $z_1 - z_0$ times, the first part follows by Claim 1.

For the second part, assume $x_0 + y_0$ is divisible by d^{z_0}, and consider the unique run where all the loops are iterated maximally, by which we mean:

- the outer loop (lines 1–6) is executed exactly z_0 times;
- whenever execution of the first inner loop (lines 2–3) ends, the value of x is 0;
- whenever execution of the second inner loop (lines 4–5) ends, the value of y is 0;

Thus every execution of the two inner loops necessarily multiplies the sum x+y by $\frac{c}{d}$, and consequently, after i iterations of the outer loop the values of respective counters x', y', z' satisfy

$$x' = (x_0 + y_0) \cdot \left(\frac{c}{d}\right)^{z_0 - i} \qquad y' = 0 \qquad z' = z_0 - i. \quad (9)$$

Repeating the multiplication z_0 times yields $x_1 = \left(\frac{c}{d}\right)^{z_0}$ and $y_1 = z_1 = 0$, as required.

Conversely, suppose $x_1 = (x_0 + y_0) \cdot \left(\frac{c}{d}\right)^{z_0}$. As c and d are co-primes, the sum of initial values $x_0 + y_0$ is thus forcedly divisible by d^{z_0}. By the first part we know that the outer loop has been iterated maximally, hence $z_1 = 0$. Then $y_1 = 0$ follows by the first part. □

Construction of $\mathcal{V}_k$. Fix $k \ge 1$. Let $f_i = \frac{a_i}{b_i}$, for $i \le i \le k$, be the fractions from Lemma 5.2, and let $f = \frac{a}{b}$ be the result of their multiplication as in (5). We thus have:

$$\left(\frac{a_1}{b_1}\right)^2 \cdot \left(\frac{a_2}{b_2}\right)^{2^2} \cdot \ldots \cdot \left(\frac{a_k}{b_k}\right)^{2^k} = \frac{a}{b}. \qquad (10)$$

The 4-VASS V_k, shown in Figure 2, is represented by the counter program in Algorithm IX using four counters t, x, y and z. The constants appearing in increment and decrement commands are represented in binary, and hence can be exponential in k while keeping the size of $\mathcal{V}_k$ polynomial.

The size of $\mathcal{V}_k$ is in $O(k^3)$. Indeed, the length of $\mathcal{V}_k$ is $O(k)$, and binary representation of constants appearing in the commands are of length $O(k^2)$.

Claim 10. *For every $k \ge 0$, the 4-VASS $\mathcal{V}_k$ has a halting run.*

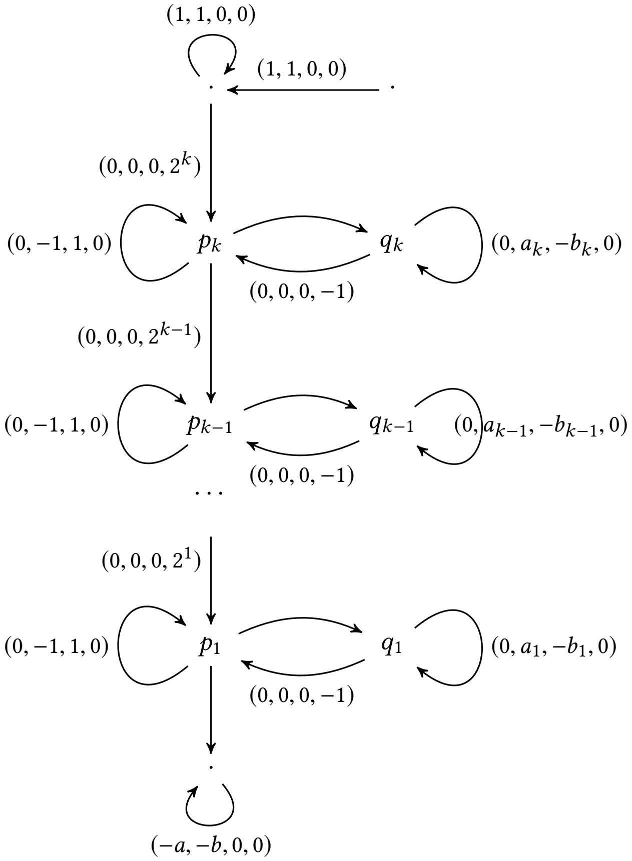


Figure 2. 4-VASS $\mathcal{V}_k$. The consecutive four dimensions correspond to counters t, x, y and z, respectively, of the counter program outlined in Algorithm IX. Unlabeled arrows depict transitions with effect (0, 0, 0, 0).

Algorithm IX Counter program representing $\mathcal{V}_k$

```
initialise to 0
t += 1    x += 1
loop
    t += 1    x += 1
for $i$ := $k$ down to 1 do
    z += $2^i$
    loop
        loop
            x −= 1    y += 1
        loop
            x += $a_i$    y −= $b_i$
        z −= 1
loop
    t −= $b$    x −= $a$
halt if t = 0
```

Proof. Put

$$N := \prod_{i=1\ldots k} (b_i)^{2^i}.$$

By performing the first loop (lines 3–4) exactly N times, the run reaches the following valuation of counters x, y, z:

$$x_k = N \qquad y_k = z_k = 0. \tag{11}$$

Notice that the outer loop (lines 7–12) coincides with the program fragment $\mathcal{HP}(a_i, b_i)$. We use the second part of Proposition 5.3 for consecutive iterations of the **for** macro. The proposition allows us to derive a run where the values x_j, y_j, z_j of counters x, y, z, after $k - j$ iterations of the **for** macro (for $j \in \{0, \ldots, k\}$), satisfy:

$$x_j = N \cdot \left(\frac{a_j}{b_j}\right)^{2^j} \cdot \ldots \cdot \left(\frac{a_k}{b_k}\right)^{2^k} \qquad y_j = z_j = 0. \tag{12}$$

Indeed, by induction with respect to $k-j$ (using (11) as induction base for $j = k$), we argue as follows: if (12) holds then x_j is divisible by $(b_{j-1})^{2^{j-1}}$, and hence by Proposition 5.3 there is a continuation of the run that yields

$$x_{j-1} = x_j \cdot \left(\frac{a_{j-1}}{b_{j-1}}\right)^{2^{j-1}} \qquad y_{j-1} = z_{j-1} = 0.$$

In consequence, for $j = 0$ we obtain, using (10):

$$x_0 = N \cdot \frac{a}{b} \qquad y_0 = z_0 = 0.$$

As the counter t is not modified inside the **for** loop (lines 5–12), its value is still equal to N after **for** loop is finished. Thus, by executing N iterations of the last loop (in lines 13–14) we reach the value 0 of all the four counters t, x, y, z and hence halt in line 15. Summing up, every $\mathcal{V}_k$ admits a halting run. □

Proof of Theorem 5.1. We argue that every halting run of $\mathcal{V}_k$ has length at least doubly exponential in k. Consider an arbitrary halting run, i.e., a run reaching the final value t = 0 in line (15). As before, let x_j, y_j and z_j, for $j = 0, \ldots, k$, stand for the values of counters x, y and z, respectively, after $k - j$ iterations of the **for** macro. Let $x_k = N \geq 1$ be the value of the counters t and x after exiting from the first loop (lines 3–4); cf. (11). The counter t is not modified inside the **for** loop (lines 5–12). Thus the last loop (in lines 13–14) has to be performed exactly N times, which implies

$$x_0 \geq N \cdot \frac{a}{b}. \tag{13}$$

Let $n_k, n_{k-1}, \ldots, n_1$ stand for the number of iterations of the outer loop (lines 7–12) in consecutive iterations of the **for** macro. By the very structure of $\mathcal{V}_k$ we know that, for every $1 \leq i \leq k$,

$$\sum_{j=i}^{k} n_j \leq \sum_{j=i}^{k} 2^j. \tag{14}$$

We aim to show the inequality (13) implies $n_j = 2^j$ for every $j \in \{1, \ldots, k\}$. As the outer loop (lines 7–12) coincides with the program fragment $\mathcal{HP}(a_i, b_i)$, we may apply the first part of Proposition 5.3 to derive, similarly as above:

$$x_j \leq N \cdot \left(\frac{a_j}{b_j}\right)^{n_j} \cdot \ldots \cdot \left(\frac{a_k}{b_k}\right)^{n_k}. \tag{15}$$

Claim 11 will imply that, roughly speaking, the biggest value of x_j is obtained, if in every unfolding of the **for** macro we perform the maximal possible number of iterations of the outer loop, and hence finish with the counter value z = 0.

Claim 11. *Assuming* (14), $\left(\frac{a_1}{b_1}\right)^{n_1} \cdot \left(\frac{a_2}{b_2}\right)^{n_2} \cdot \ldots \cdot \left(\frac{a_k}{b_k}\right)^{n_k} \leq \frac{a}{b}$. *The equality holds if, and only if,* $n_j = 2^j$ *for all* $j \in \{1, \ldots, k\}$.

Proof. For vectors $(n_1, \ldots, n_k)$ satisfying (14), we define the function $f(n_1, \ldots, n_k) = \left(\frac{a_1}{b_1}\right)^{n_1} \cdot \ldots \cdot \left(\frac{a_k}{b_k}\right)^{n_k}$. Thus (10) says that

$$f(2^1, \ldots, 2^k) = \frac{a}{b}.$$

Observe that any other vector $(n_1, \ldots, n_k)$ satisfying (14) is obtained from $(2^1, \ldots, 2^k)$ by applying a number of times one of the following two operations:

1. decrement some n_i by 1
2. decrement some n_i by 1 and increment n_{i-1} by 1.

As any of this operations strictly decreses the value of f, Claim 11 follows. □

By the first part of Claim 11, together with inequalities (14) and (15) we deduce $x_0 \leq N \cdot \frac{a}{b}$ which, combined with (13) yields the equality:

$$x_0 = N \cdot \frac{a}{b}.$$

The latter equality, together with the second part of Claim 11, implies $n_j = 2^j$ for all $j = 1 \ldots k$. As a consequence, the initial value N of x is, due to the second part of Proposition 5.3, divisible by $M = (b_k)^{2^k}$. As $1 < \frac{a_k}{b_k} < 2$, we have $b_k \geq 2$, and hence M is doubly exponential with respect to k. It follows that the length of the run is also doubly exponential, as the first inner loop, in the first iteration of the **for** macro ($i = k$), is necessarily executed $N \geq M$ times. This concludes the proof of Theorem 5.1. □

6 Conclusion

Our three main results have provided non-trivial counterexamples that advance the state of the art in the challenging area of the complexity of the reachability problem for VASS (equivalently, VAS and Petri nets). We have focussed on fixed dimension, and in particular, answered a central question that had remained open since [3] and [14], namely whether reachability for flat VASS given in unary is decidable in nondeterministic logarithmic space for any fixed dimension, by establishing NP hardness in dimension 7.

Two specific matters that remain unresolved by this work are:

- whether NP hardness of reachability for unary flat VASS is obtainable in any dimension less than 7 (and more than 2), and
- whether binary VASS in dimension 3 can have doubly exponential shortest reachability witnesses.

We also remark that, although it has never been made precise, there seems to be an intriguing deep connection between the still open gap from NL hardness to NP membership of reachability for unary flat 3-VASS and the still open gap from PSpace hardness to ExpSpace membership of coverability for 1-GVAS [31, 41].

Finally, we expect that the novel family of sequences of fractions developed in Section 5 will have applications beyond the result obtained here.

References

[1] David Angeli, Patrick De Leenheer, and Eduardo D. Sontag. 2011. Persistence Results for Chemical Reaction Networks with Time-Dependent Kinetics and No Global Conservation Laws. *SIAM Journal of Applied Mathematics* 71, 1 (2011), 128–146. https://doi.org/10.1137/090779401

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

[3] Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. 2015. Reachability in Two-Dimensional Vector Addition Systems with States Is PSPACE-Complete. In *LICS*. IEEE Computer Society, 32–43. https://doi.org/10.1109/LICS.2015.14

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

[5] Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. 2009. Two-variable logic on data trees and XML reasoning. *J. ACM* 56, 3 (2009), 13:1–13:48. http://doi.acm.org/10.1145/1516512.1516515

[6] Ahmed Bouajjani and Michael Emmi. 2013. Analysis of Recursively Parallel Programs. *ACM Trans. Program. Lang. Syst.* 35, 3 (2013), 10:1–10:49. http://doi.acm.org/10.1145/2518188

[7] Frank P. Burns, Albert Koelmans, and Alexandre Yakovlev. 2000. WCET Analysis of Superscalar Processors Using Simulation With Coloured Petri Nets. *Real-Time Systems* 18, 2/3 (2000), 275–288. https://doi.org/10.1023/A:1008101416758

[8] Thomas Colcombet and Amaldev Manuel. 2014. Generalized Data Automata and Fixpoint Logic. In *FSTTCS (LIPIcs)*, Vol. 29. Schloss Dagstuhl, 267–278. https://doi.org/10.4230/LIPIcs.FSTTCS.2014.267

[9] Hubert Comon and Véronique Cortier. 2000. Flatness Is Not a Weakness. In *CSL (LNCS)*, Vol. 1862. Springer, 262–276. https://doi.org/10.1007/3-540-44622-2_17

[10] Stefano Crespi-Reghizzi and Dino Mandrioli. 1977. Petri Nets and Szilard Languages. *Information and Control* 33, 2 (1977), 177–192. https://doi.org/10.1016/S0019-9958(77)90558-7

[11] Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. 2019. The reachability problem for Petri nets is not elementary. In *STOC*. ACM, 24–33. https://doi.org/10.1145/3313276.3316369

[12] Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. 2014. Ordered Navigation on Multi-attributed Data Words. In *CONCUR (LNCS)*, Vol. 8704. Springer, 497–511. https://doi.org/10.1007/978-3-662-44584-6_34

[13] Stéphane Demri, Diego Figueira, and M. Praveen. 2016. Reasoning about Data Repetitions with Counter Systems. *Logical Methods in Computer Science* 12, 3 (2016). https://doi.org/10.2168/LMCS-12(3:1)2016

[14] Matthias Englert, Ranko Lazić, and Patrick Totzke. 2016. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In *LICS*. ACM, 477–484. http://doi.acm.org/10.1145/2933575.2933577

[15] Javier Esparza. 1998. Decidability and Complexity of Petri Net Problems — An Introduction. In *Lectures on Petri Nets I (LNCS)*, Vol. 1491. Springer, 374–428. https://doi.org/10.1007/3-540-65306-6_20

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

[17] Laurent Fribourg and Hans Olsén. 1997. Proving Safety Properties of Infinite State Systems by Compilation into Presburger Arithmetic. In *CONCUR (LNCS)*, Vol. 1243. Springer, 213–227. https://doi.org/10.1007/3-540-63141-0_15

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

[19] Steven M. German and A. Prasad Sistla. 1992. Reasoning about Systems with Many Processes. *J. ACM* 39, 3 (1992), 675–735. http://doi.acm.org/10.1145/146637.146681

[20] Sheila A. Greibach. 1978. Remarks on Blind and Partially Blind One-Way Multicounter Machines. *Theor. Comput. Sci.* 7 (1978), 311–324. https://doi.org/10.1016/0304-3975(78)90020-8

[21] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. 2009. Reachability in Succinct and Parametric One-Counter Automata. In *CONCUR (LNCS)*, Vol. 5710. Springer, 369–383. https://doi.org/10.1007/978-3-642-04081-8_25

[22] Michel Hack. 1974. The Recursive Equivalence of the Reachability Problem and the Liveness Problem for Petri Nets and Vector Addition Systems. In *SWAT*. IEEE Computer Society, 156–164. https://doi.org/10.1109/SWAT.1974.28

[23] Piotr Hofman and Sławomir Lasota. 2018. Linear Equations with Ordered Data. In *CONCUR (LIPIcs)*, Vol. 118. Schloss Dagstuhl, 24:1–24:17. https://doi.org/10.4230/LIPIcs.CONCUR.2018.24

[24] John E. Hopcroft and Jean-Jacques Pansiot. 1979. On the Reachability Problem for 5-Dimensional Vector Addition Systems. *Theor. Comput. Sci.* 8 (1979), 135–159. https://doi.org/10.1016/0304-3975(79)90041-0

[25] Alexander Kaiser, Daniel Kroening, and Thomas Wahl. 2014. A Widening Approach to Multithreaded Program Verification. *ACM Trans. Program. Lang. Syst.* 36, 4 (2014), 14:1–14:29. http://doi.acm.org/10.1145/2629608

[26] Max I. Kanovich. 1995. Petri Nets, Horn Programs, Linear Logic and Vector Games. *Ann. Pure Appl. Logic* 75, 1–2 (1995), 107–135. https://doi.org/10.1016/0168-0072(94)00060-G

[27] Richard M. Karp and Raymond E. Miller. 1969. Parallel Program Schemata. *J. Comput. Syst. Sci.* 3, 2 (1969), 147–195. https://doi.org/10.1016/S0022-0000(69)80011-5

[28] Hélène Leroux, David Andreu, and Karen Godary-Dejean. 2015. Handling Exceptions in Petri Net-Based Digital Architecture: From Formalism to Implementation on FPGAs. *IEEE Trans. Industrial Informatics* 11, 4 (2015), 897–906. https://doi.org/10.1109/TII.2015.2435696

[29] Jérôme Leroux and Sylvain Schmitz. 2019. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *LICS*. IEEE, 1–13. https://doi.org/10.1109/LICS.2019.8785796

[30] Jérôme Leroux and Grégoire Sutre. 2004. On Flatness for 2-Dimensional Vector Addition Systems with States. In *CONCUR (LNCS)*, Vol. 3170. Springer, 402–416. https://doi.org/10.1007/978-3-540-28644-8_26

[31] Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. 2015. On the Coverability Problem for Pushdown Vector Addition Systems in One

Dimension. In *ICALP, Part II (LNCS)*, Vol. 9135. Springer, 324–336. https://doi.org/10.1007/978-3-662-47666-6_26

[32] Yuliang Li, Alin Deutsch, and Victor Vianu. 2017. VERIFAS: A Practical Verifier for Artifact Systems. *PVLDB* 11, 3 (2017), 283–296. http://www.vldb.org/pvldb/vol11/p283-li.pdf

[33] Richard J. Lipton. 1976. *The reachability problem requires exponential space*. Technical Report 62. Yale University. http://cpsc.yale.edu/sites/default/files/files/tr63.pdf

[34] Ernst W. Mayr. 1984. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comput.* 13, 3 (1984), 441–460. https://doi.org/10.1137/0213029

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

[36] Mor Peleg, Daniel L. Rubin, and Russ B. Altman. 2005. Research Paper: Using Petri Net Tools to Study Properties and Dynamics of Biological Systems. *JAMIA* 12, 2 (2005), 181–199. https://doi.org/10.1197/jamia.M1637

[37] Carl Adam Petri. 1962. *Kommunikation mit Automaten*. Ph.D. Dissertation. UniversitÃďt Hamburg. http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/

[38] Charles Rackoff. 1978. The Covering and Boundedness Problems for Vector Addition Systems. *Theor. Comput. Sci.* 6 (1978), 223–231. https://doi.org/10.1016/0304-3975(78)90036-1

[39] Louis E. Rosier and Hsu-Chun Yen. 1986. A Multiparameter Analysis of the Boundedness Problem for Vector Addition Systems. *J. Comput. Syst. Sci.* 32, 1 (1986), 105–135. https://doi.org/10.1016/0022-0000(86)90006-1

[40] Sylvain Schmitz. 2016. The complexity of reachability in vector addition systems. *SIGLOG News* 3, 1 (2016), 4–21. http://doi.acm.org/10.1145/2893582.2893585

[41] Juliusz Straszyński. 2017. *Complexity of the reachability problem for pushdown Petri nets*. Master's thesis. University of Warsaw, Faculty of Mathematics, Informatics, and Mechanics. https://apd.uw.edu.pl/diplomas/155747

[42] Leslie G. Valiant and Mike Paterson. 1975. Deterministic One-Counter Automata. *J. Comput. Syst. Sci.* 10, 3 (1975), 340–350. https://doi.org/10.1016/S0022-0000(75)80005-5

[43] Wil M. P. van der Aalst. 2015. Business process management as the "Killer App" for Petri nets. *Software and System Modeling* 14, 2 (2015), 685–691. https://doi.org/10.1007/s10270-014-0424-2