Improved Lower Bounds for Reachability in Vector Addition Systems

Wojciech Czerwiński
University of Warsaw, Poland

Sławomir Lasota
University of Warsaw, Poland

Łukasz Orlikowski
University of Warsaw, Poland

Abstract

We investigate computational complexity of the reachability problem for vector addition systems (or, equivalently, Petri nets), the central algorithmic problem in verification of concurrent systems. Concerning its complexity, after 40 years of stagnation, a non-elementary lower bound has been shown recently: the problem needs a tower of exponentials of time or space, where the height of tower is linear in the input size. We improve on this lower bound, by increasing the height of tower from linear to exponential. As a side-effect, we obtain better lower bounds for vector addition systems of fixed dimension.

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.ICALP.2021.128

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *Wojciech Czerwiński*: Supported by the ERC grant LIPA, agreement no. 683080.
Sławomir Lasota: Supported by the NCN grant 2017/27/B/ST6/02093.

1 Introduction

Petri nets [38] are a long established model of concurrency, with extensive applications in modelling and analysis of hardware [6, 25], software [16, 5, 20] and database [3, 4] systems, as well as chemical [1], biological [37, 2] and business [43, 31] processes (the references on applications are illustrative). The model admits various alternative but essentially equivalent presentations, most notably *vector addition systems* (VAS) [22], and *vector addition systems with states* (VASS) [17, Sec.5], [19]. The central algorithmic problem for this model is reachability: whether from the given initial configuration there exists a sequence of valid execution steps that reaches the given final configuration. Each of the alternative presentations admits its own formulation of the reachability problem, all of them being equivalent due to straightforward polynomial-time translations that preserve reachability; for further details, see e.g. Schmitz's survey [42, Section 2.1]. For instance, in terms of VAS, the problem is stated as follows: given a finite set T of integer vectors in d-dimensional space and two d-dimensional vectors $\mathbf{v}$ and $\mathbf{w}$ of nonnegative integers, does there exist a walk from $\mathbf{v}$ to $\mathbf{w}$ such that it stays within the nonnegative orthant, and its every step modifies the current position by adding some vector from T? Emphasizing VASS, a natural extension of VAS with finite control, in the sequel we use the name 'VASS reachability problem'.

Importance of the problem is widespread, as a plethora of problems from formal languages [8], logic [21, 11, 10, 7], concurrent systems [15, 13], process calculi [35], linear algebra [18] and other areas (the references are again illustrative) are known to admit reductions from the VASS reachability problem; for more such problems and a wider discussion, we refer to [42].

State of the art. The complexity of the VASS reachability problem was remaining unsettled over the past half century. The late 1970s and the early 1980s saw the initial burst of activity. After an incomplete proof by Sacerdote and Tenney [40], decidability of the problem was established by Mayr [33, 34], whose proof was then simplified by Kosaraju [23]. Building on the further refinements made by Lambert in the 1990s [24], there has been substantial progress over the past ten years [26, 27, 28], culminating in the first upper bound on the complexity [29], recently improved to Ackermannian [30].

In contrast to the progress on refining the proof of decidability and obtaining upper bounds on the complexity, Lipton's landmark result that the VASS reachability problem requires exponential space [32] has remained the state of the art on lower bounds for over 40 years. Moreover, in conjunction with an apparent tightness of Lipton's construction, this has led to the conjecture that the problem is ExpSpace-complete becoming common in the community. The conjecture has been falsified by a recent breakthrough construction of [9] that shows the VASS reachability problem is hard for the class Tower of all decision problems that are solvable in time or space bounded by a tower of exponentials whose height is an elementary (bounded by a tower of exponentials of fixed height) function of the input size.

Contribution. This paper provides a further improvement on the lower bound. The construction of [9] proves hardness of the VASS reachability problem for the class Tower, with respect to elementary reductions. However, in terms of the more fine-grained hierarchy defined with respect to polynomial-time reductions only, the result states that the problem needs a tower of exponentials of time or space, where the height of tower is linear in input size. As our primary result we increase the height of the tower from linear to exponential, namely we prove that the problem actually needs a tower of exponentials of time or space, where the height of the tower is itself exponential in input size:

$$\left.2^{2^{2^{\cdots^{n}}}}\right\} \text{ height } \mathcal{O}(n) \qquad \rightsquigarrow \qquad \left.2^{2^{2^{\cdots^{n}}}}\right\} \text{ height } 2^{\mathcal{O}(n)}.$$

In addition, as a side effect of our improved construction we obtain better lower bounds for VASS of fixed dimension. It has been known, as shown in [9], that the reachability problem for VASS in dimension d is $\mathcal{O}(d)$-ExpSpace-hard (specifically: d-ExpSpace-hardness for dimension $d + 13$). We show that the reachability problem for VASS in dimension d is $2^{\mathcal{O}(d)}$-ExpSpace-hard (specifically: 2^d-ExpSpace-hardness for dimension $2d + 13$).

Clearly, these new lower bounds do not formally exclude the VASS reachability problem from still being in Tower (and hence being Tower-complete with respect to elementary reductions). However, we believe that the result makes this less likely. Intuitively speaking, we rule out a natural algorithmic scheme of solving the reachability problem, where each additional control state or dimension leads to an additional exponential blowup of time or space. Presently, the most optimistic scheme must suffer, for each additional control state or dimension, from at least *doubling* of the number of exponentials.

Organisation of the paper. We start by defining the model, in Section 2, and then the reachability problem, in Section 3. In the latter section we also recall the lower bounds of [9] and formally formulate our improved ones. The following two sections are devoted to proofs. First, in Section 4 we show 2^d-ExpSpace-hardness of the reachability problem for VASS of dimension $2d + 13$, as a preparation before Section 5, in which we prove our primary result: without restriction on dimension, the problem needs a tower of exponentials of time or space, of height exponential in input size.

2 Counter programs

As mentioned in the introduction, Petri nets [38], VAS [22], and VASS [17, 19] are alternative presentations of the same model of concurrent processes, in the sense that between each pair there exist straightforward polynomial-time translations that preserve reachability [42, Sec. 2.1]. Following [12] and [9], instead of working directly with VASS, as our primary language we choose imperative nondeterministic programs operating on variables called counters, that range over nonnegative integers. These programs may be seen as user-friendly presentations of VASS.

Counter programs. A counter program is a sequence of commands, each of which is of one of the following three types:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')

except that the last command is of the form:

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero).

Counters are only allowed to have nonnegative values, they may be incremented or decremented but, notably, counters may not be zero-tested. As an illustration, consider the following program:

1: **goto** 7 **or** 2
2: $\mathsf{x} \mathrel{+}= 1$
3: $\mathsf{x}' \mathrel{-}= 1$
4: $\mathsf{y} \mathrel{+}= 1$
5: $\mathsf{y} \mathrel{+}= 1$
6: **goto** 1
7: **halt if** $\mathsf{x}' = 0$.

It repeats the block of three commands in lines 2–5 some number of times chosen nondeterministically (possibly zero, although not infinite because x' is decreasing and hence its initial value bounds the number of iterations) and then halts provided counter x' is zero.

We emphasise that counters are not permitted to have negative values. In the example above, that is why the decrement in line 3 works also as a non-zero test.

We assume that initially all counters are set to 0. A run of a counter program is a sequence of commands, as expected. We say that such a run is *halting* if, and only if it has successfully executed its **halt** command (which is necessarily the program's last); otherwise, the run is either *partial* or *infinite*. Observe that, due to a decrement that would cause a counter to become negative, or due to an unsuccessful terminal check for zero, a partial run may fail to continue because it is blocked from further execution. Moreover, due to nondeterministic jumps, the same program may have various runs in each of the three categories: halting runs, maximal partial runs, and infinite runs.

By the *size* of a program we mean the number of commands in it, and by its *dimension* we mean the number of counters. We sometimes speak also of *program fragments*, which are not ended with a **halt** command.

3 The reachability problem

Counter programs can be seen as presentations of VASS, where the latter are required to start with all vector components zero and to finish with vector components zero as specified by the **halt** command, and hence the VASS reachability problem can be stated as:

VASS REACHABILITY PROBLEM
Input: A counter program.
Question: Does it have a halting run?

We remark that restricting further to programs where no counter is required to be zero finally (i.e., where the last command is just **halt**) turns this problem into the VASS *coverability* problem, which is concerned with reachability of just a control location, with no requirement on the final values of counters. Lipton's EXPSPACE lower bound [32] holds already for the coverability problem, which is in fact EXPSPACE-complete [39].

For stating our results we recall the standard complexity classes bases on exponential hierarchy of fast-growing functions. We write $\mathcal{T}(n)$ for a tower of exponentials:

$$\mathcal{T}_2(k,n) \;=\; 2^{2^{2^{\cdots^{n}}}} \Big\} \text{ height } k \qquad\qquad \mathcal{T}(n) \;=\; \mathcal{T}_2(n,n) \;=\; 2^{2^{2^{\cdots^{n}}}} \Big\} \text{ height } n.$$

Formally, let $\mathcal{T}_2(0,n) = n$ and $\mathcal{T}_2(k+1,n) = 2^{\mathcal{T}_2(k,n)}$. For a fixed positive integer k, the class k-EXPSPACE contains all decision problems solvable in space $\mathcal{T}_2(k, \text{poly}(n))$. The class TOWER contains all decision problems solvable in time or space bounded by a tower of exponentials whose height is an elementary function of the input size, namely

$$\text{TOWER} \;=\; \bigcup_{f \text{ elementary}} \text{SPACE}(\mathcal{T}(f(n)))$$

with $f(\_)$ ranging over all *elementary* functions, i.e., functions bounded by $\mathcal{T}_2(k,\_)$ for some k. TOWER is thus closed under elementary reductions.

For the results of this paper we need a fine-grained hierarchy inside TOWER, with respect to dependency of the height of exponentials on the input size. For a fixed elementary function f, let $f(n)$-TOWER denote the class of all decision problems solvable in (time or) space $\mathcal{T}(f(\text{poly}(n)))$, i.e., in (time or) space bounded by a tower of exponentials of height $f(n^k)$, for some k, where n is the input size:

$$f(n)\text{-TOWER} \;=\; \bigcup_{k\in\mathbb{N}} \text{SPACE}(\mathcal{T}(f(n^k))).$$

Each class $f(n)$-TOWER is a strict subclass of TOWER as it is closed under polynomial-time reductions but, contrarily to TOWER, not under arbitrary elementary reductions.

We recall the results of [9]. First, once one fixes the dimension of VASS, the reachability problem requires a tower of exponentials of space, where the height of the tower is linear in the dimension:

▶ **Theorem 1** ([9], Cor. 5). *For any positive integer d, the* VASS *reachability problem in dimension $d+13$ is d-*EXPSPACE*-hard, with respect to polynomial-time*[1] *reductions.*

[1] All results mentioned in this section are actually shown using linear-time reductions.

The main result of [9] is a non-elementary lower bound of the VASS reachability problem: TOWER-hardness with respect to elementary reductions. The construction of [9] shows that the problem requires a tower of exponentials of space, where the height of tower is linear in the input size, and hence the result can be stated in terms of the fine-grained hierarchy as follows:

▶ **Theorem 2** ([9], Thm. 4). *The* VASS *reachability problem is hard for* n-TOWER*, with respect to polynomial-time reductions.*

As our first main result, we improve the lower bound of Theorem 1, namely we prove that solving the VASS reachability problem in dimension $2d + 13$ requires at least $\mathcal{T}_2(2^d, n)$ space:

▶ **Theorem 3.** *For any positive integer* d*, the* VASS *reachability problem in dimension* $2d + 13$ *is* 2^d*-*EXPSPACE*-hard.*

Clearly, without restriction of dimension, Theorem 3 does not exclude membership of the problem in TOWER, neither it excludes its membership in n-TOWER. Our primary result excludes the latter possibility: it states that the VASS reachability problem requires a tower of exponentials of space, where the height of tower is exponential in the input size, thus improving the bound of Theorem 2:

▶ **Theorem 4.** *The* VASS *reachability problem is hard for* 2^n*-*TOWER*, with respect to polynomial-time reductions.*

Again, Theorem 4 does not formally exclude membership of the VASS reachability problem in TOWER, it makes it however less imaginable as mentioned in the introduction.

Theorems 3–4 are proved in the following two sections. The proofs are a refinement and an optimisation of the construction of [9], involving certain amount of programming effort in the setting of counter programs.

4 Proof of Theorem 3

In this and in the next section we proceed by reductions from space-bounded variants of the halting problem for the standard model of (deterministic) Minsky machines [36]. The reader is referred to [14, Theorem 3.1] for translations between space-bounded 3-counter Minsky machines and Turing machines.

Following [9], for technical reasons we prefer to work with factorials instead of exponentials. We write $\mathcal{F}(k, \_)$ for the tower of factorials of height k: $\mathcal{F}(0, n) = n$ and $\mathcal{F}(k+1, n) = \mathcal{F}(k, n)!$. We note that all the complexity classes from the previous section are robust with respect to the choice of the fast-growing function hierarchy: exponential function can be equivalently replaced by factorial [41, Section 4.1].

Our proof is a refinement of the reduction of [9], from the following bounded version of the halting problem for Minsky machines with 3 counters, which is $(k-1)$-EXPSPACE-complete for any fixed positive integer k:

k-EXP-BOUNDED HALTING PROBLEM

Input: A 3-counter Minsky machine $\mathcal{M}$ of size n.

Question: Does it have a halting run where all counters are bounded by $\mathcal{F}(k, n)$?

Let h be a fixed positive integer. Given a 3-counter Minsky machine $\mathcal{M}$ of size n, the reduction of [9, Lem. 6] transforms $\mathcal{M}$, in time $\mathcal{O}(n+h)$, into a counter program $\mathcal{P}$ with $h+13$ counters, including $h+2$ counters $\mathsf{x}_{-1}, \mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_h$ which are required to be zero by the final **halt** command, plus 11 other counters, of the form[2]

$$\underbrace{\mathcal{H}_{-1}}_{\text{size } \mathcal{O}(n)} \quad \underbrace{\mathcal{H}_0 \quad \mathcal{H}_1 \quad \cdots \quad \mathcal{H}_h}_{\text{constant size each}} \quad \underbrace{\mathcal{H}_{h+1}}_{\text{size } \mathcal{O}(n)} \quad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h = 0,$$

for some program fragments $\mathcal{H}_{-1}, \mathcal{H}_0, \ldots, \mathcal{H}_{h+1}$ (for technical convenience, the indexing starts at -1). We refer to the program fragments $\mathcal{H}_{-1}, \mathcal{H}_0, \ldots, \mathcal{H}_{h+1}$ as *segments*. Furthermore, the first segment $\mathcal{H}_{-1}$ and the last one $\mathcal{H}_{h+1}$ are of size $\mathcal{O}(n)$, the remaining ones are of constant size, and the reduction is correct due to:

▷ Claim 5 (Correctness). $\mathcal{M}$ has a halting run with all counters bounded by $\mathcal{F}(h+1, n)$ if, and only if, $\mathcal{P}$ has a halting run.[3]

Moreover, the counter program $\mathcal{P}$ has the following two crucial properties:

(i) Each counter x_j appearing in the final **halt** command appears in segments $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$ only, for $j = -1, 0, \ldots, h$:

$$\underbrace{\mathcal{H}_{-1}}_{\mathsf{x}_{-1} \text{ only here}} \quad \overbrace{\underbrace{\mathcal{H}_0}_{\mathsf{x}_1 \text{ only here}} \quad \mathcal{H}_1}^{\mathsf{x}_0 \text{ only here}} \quad \overbrace{\mathcal{H}_2}^{\mathsf{x}_2 \text{ only here}} \quad \mathcal{H}_3 \quad \ldots \quad \underbrace{\mathcal{H}_{h-1}}_{\mathsf{x}_{h-1} \text{ only here}} \quad \overbrace{\mathcal{H}_h}^{\mathsf{x}_h \text{ only here}} \quad \mathcal{H}_{h+1}.$$

We say, to some extent informally, that $\mathcal{P}$ is a *relay-race* with respect to counters $\mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h$, by which we mean the last appearance of every counter x_j, for $j < h$, is in the same segment as the first appearance of the next counter x_{j+1}, and counters x_j and x_{j+2}, for $j < h-1$, never appear in the same segment.

(ii) Each segment $\mathcal{H}_j$, for $j = 0, 1, \ldots, h$, is obtained from the same[4] program fragment $\mathcal{H}$, not using counters $\mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h$, by replacing a counter x appearing in $\mathcal{H}$ by x_{j-1}, and another counter x' appearing in $\mathcal{H}$ by x_j. We denote the replacement as:

$$\mathcal{H}_j \quad = \quad \mathcal{H}[\mathsf{x} \mapsto \mathsf{x}_{j-1},\ \mathsf{x}' \mapsto \mathsf{x}_j]. \tag{1}$$

Let $h = 2^d$ for some positive integer d. We prove Theorem 3 by transforming $\mathcal{P}$ into an equivalent counter program $\widetilde{\mathcal{P}}$ of (slightly larger) size $\mathcal{O}(n + d \cdot 2^d)$, but of exponentially smaller dimension $2d+13$. We will rely on the relay-race property with respect to h counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_{h-1}$ (call these counters *relay-race counters*), and thus the transformation will not affect counters x_{-1} and x_h. The intuitive idea is to re-cycle counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_{h-1}$ appearing in segments $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_h$. Our transformation proceeds in $(d-1)$ steps, in each step reducing by half the number of relay-race counters and adding two additional fresh counters.

---

[2] Whenever convenient we compose counter program fragments horizontally, for the ease of presentation.

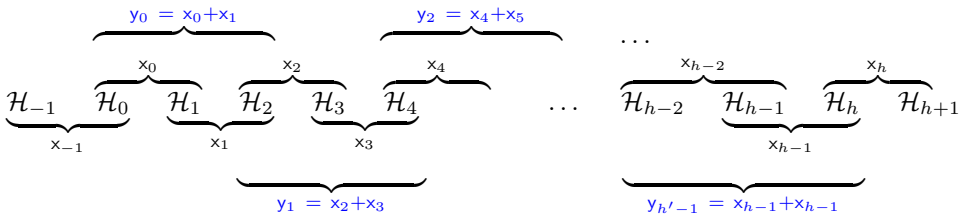
[3] Roughly speaking, $\mathcal{H}_{-1}$ simply computes n, then the sequence of segments $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_h$ is responsible for computation of $(h+1)$th iterate of factorial of n, which is then used in $\mathcal{H}_{h+1}$ for simulating $\mathcal{M}$.

[4] For the ease of presentation we slightly simplify the structure of $\mathcal{P}$; in fact, in order to rigorously express the construction of [9] one needs to use two different program fragments instead of just one $\mathcal{H}$, one of them for even j and the other one for odd j. As will be clear later, this simplification is inessential.

The first step. We split the first step of the transformation into two sub-steps. As the first sub-step, consider the following transformation of $\mathcal{P}$. Let $h' = \frac{1}{2}h = 2^{d-1}$. We introduce h' additional fresh counters $\mathsf{y}_0, \ldots, \mathsf{y}_{h'-1}$, initially set to 0, with the idea that counter y_k invariantly equals to the sum of counters $\mathsf{x}_{2k} + \mathsf{x}_{2k+1}$. This is achieved by adding, immediately after each command anywhere in fragments $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_{h-1}$ that operates on counter x_j, for $j = 0, \ldots, h-1$, an additional command operating in the same way on $\mathsf{y}_{j \text{ div } 2}$:

$$\mathsf{x}_j \mathrel{*}= 1 \qquad \mapsto \qquad \mathsf{x}_j \mathrel{*}= 1 \quad \mathsf{y}_{j \text{ div } 2} \mathrel{*}= 1,$$

where j div 2 is the largest nonnegative integer k such that $2k \leq j$, and $* \in \{+, -\}$. This addition yields a counter program $\mathcal{P}'$ (recall that both h and h' are even):



(2)

Furthermore, we remove counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_{h-1}$ from the **halt** command, and put counters $\mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1}$ instead:

$$\textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h = 0 \qquad \mapsto \qquad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_h, \mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1} = 0.$$

▷ Claim 6. $\mathcal{P}'$ satisfies invariantly $\mathsf{y}_k = \mathsf{x}_{2k} + \mathsf{x}_{2k+1}$, for $k = 0, 1, \ldots, h'-1$.

We use that claim for entailing:

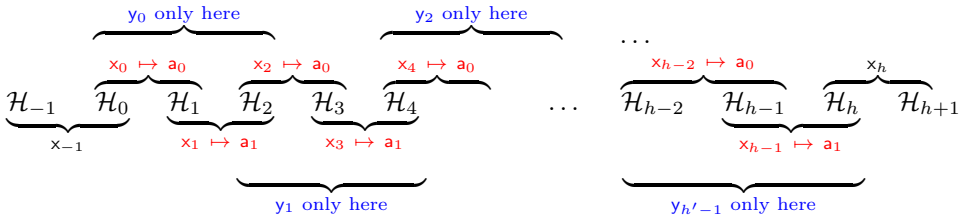
▷ Claim 7 (Correctness). $\mathcal{P}$ has a halting run if, and only if, $\mathcal{P}'$ has one.

Proof. We show that halting runs of $\mathcal{P}$ are in one-to-one correspondence with halting runs of $\mathcal{P}'$, using Claim 6 in both directions. In one direction, every halting run of $\mathcal{P}$, ending with all counters $\mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h$ equal 0, has a corresponding halting run of $\mathcal{P}'$ ending with $\mathsf{x}_{-1}, \mathsf{x}_h$ and all added counters $\mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1}$ equal 0 as well. For the opposite direction consider any halting run of $\mathcal{P}'$. As every counter y_k is 0 at the end of the run, the sum of every two consecutive counters $\mathsf{x}_{2k} + \mathsf{x}_{2k+1}$ is 0 at the end, and hence also each individual counter x_j is *forced* to be 0 as well (without even checking that it is so, in the final **halt** command). Therefore, dropping operations on the added counters yields a halting run of $\mathcal{P}$. ◁

As the second sub-step, we introduce two fresh counters a_0, a_1, and replace each operation on every counter x_j, for $j = 0, 1, \ldots, h-1$, anywhere in segments $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_{h-1}$, by the analogous operation on $\mathsf{a}_{j \text{ mod } 2}$.

$$\mathsf{x}_j \mathrel{*}= 1 \qquad \mapsto \qquad \mathsf{a}_{j \text{ mod } 2} \mathrel{*}= 1. \tag{3}$$

The replacement removes h counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_{h-1}$ definitely from $\mathcal{P}'$, yielding a counter program $\mathcal{P}''$:



(4)

with the same **halt** command as $\mathcal{P}'$. Note that the size of $\mathcal{P}''$ is larger by $\mathcal{O}(2^d)$ than the size of $\mathcal{P}$, but its dimension decreased from $2^d + 13$ to $2^{d-1} + 15$, as $h = 2^d$ counters have been removed and another $h' + 2 = 2^{d-1} + 2$ counters have been introduced instead.

We have thus completed the description of a transformation $\mathcal{P} \mapsto \mathcal{P}''$. Formally, by composing the two sub-steps, we see that $\mathcal{P}''$ has the form:

$$\underbrace{\mathcal{H}_{-1}}_{\text{size } \mathcal{O}(n)} \quad \underbrace{\mathcal{H}''_0 \quad \mathcal{H}''_1 \quad \cdots \quad \mathcal{H}''_h}_{\text{constant size each}} \quad \underbrace{\mathcal{H}_{h+1}}_{\text{size } \mathcal{O}(n)} \quad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_h, \mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1} = 0,$$

where each segment $\mathcal{H}''_i$ is obtained from $\mathcal{H}_i$ by the simultaneous substitution, for all $j = 0, 1, \ldots, h-1$:

$$\mathsf{x}_j \mathrel{*}= 1 \qquad \mapsto \qquad \mathsf{a}_{j \bmod 2} \mathrel{*}= 1 \quad \mathsf{y}_{j \operatorname{div} 2} \mathrel{*}= 1. \tag{5}$$

The following fact is crucial for correctness, i.e., for proving that the second sub-step, and hence also the whole step, preserves reachability:

▷ Claim 8. In every halting run of $\mathcal{P}''$, $\mathsf{a}_{j \bmod 2} = 0$ at the end of $\mathcal{H}''_{j+1}$ for every $j \in \{0, \ldots, h-1\}$.

Proof. Consider any halting run of $\mathcal{P}''$. By induction on j we prove that for every *even* $j \in \{0, 1, \ldots, h-1\}$, $\mathsf{a}_0 = 0$ at the end of $\mathcal{H}''_{j+1}$, and $\mathsf{a}_1 = 0$ at the end of $\mathcal{H}''_{j+2}$.

Let $j \geq 0$ and asssume the claim holds for smaller values of j. Let $v_0 \in \mathbb{N}$ (resp. $v_1 \in \mathbb{N}$) denote the value of counter a_0 (resp. counter a_1), at the beginning of fragment $\mathcal{H}''_j$ (resp. $\mathcal{H}''_{j+1}$). By induction assumption (or due to initial 0 values of counters) we have $v_0 = v_1 = 0$. According to the substitution (5), every operation on a_0 in $\mathcal{H}''_j$ and $\mathcal{H}''_{j+1}$ (recall that these are the only fragments where x_j appears) is also performed on $\mathsf{y}_{j \operatorname{div} 2}$. Likewise, every operation on a_1 in $\mathcal{H}''_{j+1}$ and $\mathcal{H}''_{j+2}$ (these are the only fragments where x_{j+1} appears) is also performed on $\mathsf{y}_{j \operatorname{div} 2}$. Also according to (5), these are the only operations on $\mathsf{y}_{j \operatorname{div} 2}$ performed along the run. Therefore, denoting by v'_0 (resp. v'_1) the value of counter a_0 (resp. counter a_1), at the end of fragment $\mathcal{H}''_{j+1}$ (resp. $\mathcal{H}''_{j+2}$), we know that the value of counter $\mathsf{y}_{j \operatorname{div} 2}$ at the end of the run is $(v'_0 - v_0) + (v'_1 - v_1) = v'_0 + v'_1$. As the counter $\mathsf{y}_{j \operatorname{div} 2}$ is checked to be 0 by the final **halt** command, we deduce $v'_0 + v'_1 = 0$. Finally, since both values are necessarily nonnegative, we deduce $v'_0 = v'_1 = 0$, as required. ◁

▷ Claim 9 (Correctness). $\mathcal{P}'$ has a halting run if, and only if, $\mathcal{P}''$ has one.

Proof. We show that halting runs of $\mathcal{P}'$ are in one-to-one correspondence with halting runs of $\mathcal{P}''$. As $\mathcal{P}''$ is obtained from $\mathcal{P}'$ by merging all counters x_j, for even j, to one counter a_0, and all counters x_j, for odd j, to one counter a_1, every halting run of $\mathcal{P}'$ has a corresponding halting run of $\mathcal{P}''$ such that a_0 is the sum of values of all counters x_j for even j and a_1 is the sum of values of all counters x_j for odd j. For the opposite direction we rely on Claim 8. According to the claim, replacing in a halting run of $\mathcal{P}''$ each operation on $\mathsf{a}_{j \bmod 2}$ in $\mathcal{H}''_j$ and $\mathcal{H}''_{j+1}$ by the same operation on x_j is safe, namely it is guaranteed that counters x_j stay nonnegative. Therefore we get a halting run of $\mathcal{P}''$. ◁

Iterating steps. A crucial but simple observation, cf. (4), is that the so described step of the transformation preserves the relay-race property, and hence can be iterated:

▷ Claim 10 (Relay-race preservation). The counter program $\mathcal{P}''$ is a relay-race with respect to counters $\mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1}$.

Proof. According to the substitution (5), each counter y_k only appears in three consecutive segments $\mathcal{H}''_{2k}, \mathcal{H}''_{2k+1}$ and $\mathcal{H}''_{2k+2}$. Therefore the relay-race condition is satisfied: the last appearance of every counter y_k, for $k+1 < h'$, is in the same segment $\mathcal{H}''_{2k+2}$ as the first appearance of the next counter y_{k+1}, and counters y_k and y_{k+2}, for $k+2 < h'$, never appear in the same segment. ◁

We apply the transformation step $d-1$ times. Initially, $\mathcal{P}$ has 2^d relay-race counters, and 13 other counters including x_{-1} and x_h. Each ith iteration, for $i = 0, 1, \ldots, d-2$, decreases the number of relay-race counters twice, from 2^{d-i} to 2^{d-i-1}, and adds two additional counters a^i_0 and a^i_1. Therefore, after $d-1$ iterations[5] we arrive at a counter program $\widetilde{\mathcal{P}}$ with just two relay-race counters, say $\mathsf{z}_0, \mathsf{z}_1$, plus $2(d-1)$ added counters, say a^i_0, a^i_1 for $i = 0, 1, \ldots, d-2$, and 13 remaining counters, of the form:

$$\underbrace{\mathcal{H}_{-1}}_{\text{size } \mathcal{O}(n)} \quad \underbrace{\widetilde{\mathcal{H}}_0 \quad \widetilde{\mathcal{H}}_1 \quad \cdots \quad \widetilde{\mathcal{H}}_h}_{\text{size } \mathcal{O}(d) \text{ each}} \quad \underbrace{\mathcal{H}_{h+1}}_{\text{size } \mathcal{O}(n)} \qquad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_h, \mathsf{z}_0, \mathsf{z}_1 = 0. \tag{6}$$

By iterating the substitution (5), one sees that $\widetilde{\mathcal{H}}_i$ is actually obtained from $\mathcal{H}_i$ by the following simultaneous substitution, for $* \in \{+, -\}$ and $j = 0, 1, \ldots, h-1$

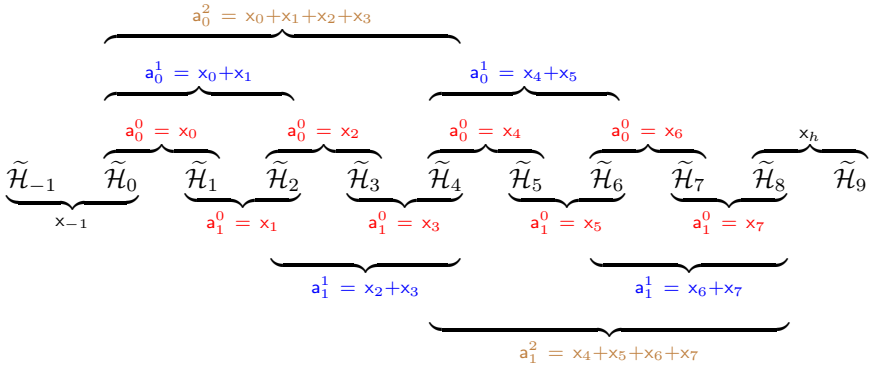
$$\mathsf{x}_j \mathrel{*}= 1 \qquad \mapsto \qquad \mathsf{a}^0_{j_0} \mathrel{*}= 1 \quad \mathsf{a}^1_{j_1} \mathrel{*}= 1 \quad \ldots \quad \mathsf{a}^{d-2}_{j_{d-2}} \mathrel{*}= 1 \quad \mathsf{z}_{j_{d-1}} \mathrel{*}= 1, \tag{7}$$

where $j_{d-1} \ldots j_1 j_0 = \text{BIN}(j)$ is the binary representation of j using d bits, in the order of decreasing significance of bits. Indeed, observe that $j_0 = j \bmod 2$, $j_1 = (j \operatorname{div} 2) \bmod 2$, $j_2 = ((j \operatorname{div} 2) \operatorname{div} 2) \bmod 2$, and so on. Therefore, by iterating the substitution (5), we first replace x_j by $\mathsf{a}^0_{j_0}$ and a relay-race counter $\mathsf{y}_{j \operatorname{div} 2}$, then replace $\mathsf{y}_{j \operatorname{div} 2}$ by $\mathsf{a}^1_{j_1}$ and some further relay-race counter $\mathsf{z}_{(j \operatorname{div} 2) \operatorname{div} 2}$, and so on. In the sequel we uniformly write a^{d-1}_b instead of z_b appearing in (6) and (7), for $b = 0, 1$. Let $\overline{b} = 1 - b$ denote bit negation.

▷ Claim 11. In every halting run of $\widetilde{\mathcal{P}}$, $\mathsf{a}^i_b = 0$ at the end of $\widetilde{\mathcal{H}}_j$ for every $j \in \{1, \ldots, h\}$ such that $j_i = \overline{b}$ and $j_{i-1} = \ldots = j_0 = 0$.

Proof. Follows by iterating the substitution (5) and Claim 8. ◁

▶ **Example 12.** As an illustration consider $d = 3$, and hence $h = 8$. The construction yields a counter program $\widetilde{\mathcal{P}}$ with 6 added counters $\mathsf{a}^0_0, \mathsf{a}^0_1, \mathsf{a}^1_0, \mathsf{a}^1_1, \mathsf{a}^2_0, \mathsf{a}^2_1$, and hence of dimension 19. Halting runs of $\mathcal{P}$ and $\widetilde{\mathcal{P}}$ are in one-to-one correspondence, and the values of counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_7$ in a halting run of $\mathcal{P}$ are related, via the equalities depicted below, to the values of counters $\mathsf{a}^0_0, \mathsf{a}^0_1, \mathsf{a}^1_0, \mathsf{a}^1_1, \mathsf{a}^2_0, \mathsf{a}^2_1$ in the corresponding halting run of $\widetilde{\mathcal{P}}$.



[5] One additional dth iteration could be also performed, which would however result in replacing 2 counters by 3 other ones, and hence the dimension would increase.

Indeed, iterating the proof of Claims 7 and 9 three times, we learn that a_b^i is the sum of values of all counters x_j where $j_i = b$, for instance $\mathsf{a}_0^1 = \mathsf{x}_0 + \mathsf{x}_1 + \mathsf{x}_4 + \mathsf{x}_5$. However, according to Claim 11, we have $\mathsf{a}_0^1 = 0$ at the end of $\widetilde{\mathcal{H}}_2$, and hence $\mathsf{a}_0^1 = \mathsf{x}_4 + \mathsf{x}_5$ in $\widetilde{\mathcal{H}}_4 \dots \widetilde{\mathcal{H}}_6$.

In particular, we obtain (cf. Claim 14 below) 8-ExpSpace-hardness for VASS in dimension 19 which, according to Theorem 1, has been known before to be 6-ExpSpace-hard. ⌟

▷ Claim 13 (Correctness). $\mathcal{P}$ has a halting run if, and only if, $\widetilde{\mathcal{P}}$ has one.

Proof. Iterating Claims 7 and 9 we deduce that halting runs of $\mathcal{P}$ are in one-to-one correspondence with halting runs of $\widetilde{\mathcal{P}}$. In one direction, every halting run of $\mathcal{P}$ has a corresponding halting run of $\widetilde{\mathcal{P}}$ where each counter a_b^i, for $i = 0, 1, \dots, d-2$ and $b = 0, 1$, invariantly equals to the sum of all counters x_j with $j_i = b$: $\mathsf{a}_b^i = \sum_{j \,:\, j_i = b} \mathsf{x}_j$. For the opposite direction we deduce, using Claim 11, that dropping operations on the added counters in every halting run of $\widetilde{\mathcal{P}}$, and replacing each operation on $\mathsf{a}_{j \bmod 2}$ in $\widetilde{\mathcal{H}}_j$ and $\widetilde{\mathcal{H}}_{j+1}$ by the same operation on x_j, yields a halting run of $\mathcal{P}$. ◁

The dimension of $\widetilde{\mathcal{P}}$ is $2d + 13$. The size of every segment $\widetilde{\mathcal{H}}_j$, for $j = 0, 1, \dots, h$, is $\mathcal{O}(d)$ times the constant size of $\mathcal{H}_j$, therefore the size of $\widetilde{\mathcal{P}}$ is $\mathcal{O}(n + d \cdot 2^d)$.

Due to Claims 5 and 13, the reduction $\mathcal{M} \mapsto \mathcal{P}$ of [9], composed with the transformation $\mathcal{P} \mapsto \widetilde{\mathcal{P}}$ just described, yield the required reduction $\mathcal{M} \mapsto \widetilde{\mathcal{P}}$:

▷ Claim 14. Let d be a positive integer. Given a 3-counter Minsky machine $\mathcal{M}$ of size n, one can compute in time $\mathcal{O}(n + d \cdot 2^d)$ a counter program $\widetilde{\mathcal{P}}$ of dimension $2d + 13$ such that $\mathcal{M}$ has a halting run with counters bounded by $\mathcal{F}(2^d + 1, n)$ if, and only if, $\widetilde{\mathcal{P}}$ has a halting run.

For every fixed d the reduction is computable in linear time with respect to the input size n, and hence the VASS reachability problem in dimension $2d + 13$ is 2^d-ExpSpace-hard.

5 Proof of Theorem 4

In order to prove Theorem 4 we refine further our reduction from Section 4. The refinement involves certain amount of low-level programming with counter programs.

Formally, we will set up a linear-time reduction from the following problem:

EXP-TOWER HALTING PROBLEM
Input: A 3-counter Minsky machine $\mathcal{M}$ of size n.
Question: Does it have a halting run where all counters are bounded by $\mathcal{F}(2^n, n)$?

We argue, similarly as before, that the problem is complete for the class 2^n-Tower, with respect to polynomial-time reductions, cf. [14, Theorem 3.1] and [41, Section 4.1].

In this section we strongly rely on condition (ii) from Section 4: each segment $\mathcal{H}_j$ in (6), for $j = 0, 1, \dots, h$, is obtained by the substitution (1) applied to the same program fragment $\mathcal{H}$ of constant size. Combining this substitution with the substitution (7) we deduce that each $\widetilde{\mathcal{H}}_j$, for $j = 0, 1, \dots, h$, is obtained from $\mathcal{H}$ by applying the following two substitutions:

$$\begin{array}{lcl} \mathsf{x} \mathrel{*}= 1 & \mapsto & \mathcal{C}^*_{j-1} \\ \mathsf{x}' \mathrel{*}= 1 & \mapsto & \mathcal{C}^*_j \end{array} \tag{8}$$

where program fragments $\mathcal{C}^*_j$ are defined, for $* \in \{+, -\}$ and $j = 0, 1, \dots, h-1$, as the sequence of commands:

$$\mathcal{C}^*_j \quad = \quad \mathsf{a}^0_{j_0} \mathrel{*}= 1 \quad \mathsf{a}^1_{j_1} \mathrel{*}= 1 \quad \dots \quad \mathsf{a}^{d-2}_{j_{d-2}} \mathrel{*}= 1 \quad \mathsf{a}^{d-1}_{j_{d-1}} \mathrel{*}= 1.$$

We apply here the proviso that the former substitution, referring to x, is not applied when $j = 0$, and the latter one, referring to x', is not applied when $j = h$. Indeed, recalling (1) and (4), x in $\mathcal{H}_0$ refers actually to x_0, and likewise x' in $\mathcal{H}_j$ refers actually to x_{h+1}.

Relying on the substitutions (8) we are going to optimise the counter program $\widetilde{\mathcal{P}}$, as defined in (6), by replacing all segments $\widetilde{\mathcal{H}}_1, \ldots, \widetilde{\mathcal{H}}_{h-1}$ by a *single* program fragment $\overline{\mathcal{G}}$ of size $\mathcal{O}(d)$, and thus obtaining the final counter program $\overline{\mathcal{P}}$ (see (11) below).

We start by defining a single program fragment $\mathcal{C}^*$ of size $\mathcal{O}(d)$ that achieves the effect of any $\mathcal{C}^*_j$ in a way parametric with respect to j. The program fragment $\mathcal{C}^*$ uses a bit-wise representation of the value of j, ranging over $0, \ldots, h-1$. To this aim we introduce d new counters $\mathsf{b}_0, \mathsf{b}_1, \ldots, \mathsf{b}_{d-1}$ to represent the binary representation $\textsc{bin}(j) = j_{d-1} \ldots j_0$ of current value j; thus these counters will only take values 0 or 1. In addition, we introduce another d counters $\overline{\mathsf{b}}_0, \overline{\mathsf{b}}_1, \ldots, \overline{\mathsf{b}}_{d-1}$ to represent negations of bits $\mathsf{b}_0, \mathsf{b}_1, \ldots, \mathsf{b}_{d-1}$. We are going to enforce the following equalities to hold invariantly:

$$\mathsf{b}_i + \overline{\mathsf{b}}_i = 1 \qquad (\text{for } i = 0, \ldots, d-1). \tag{9}$$

In particular, as expected, counters b_i are initialised to 0, while counters $\overline{\mathsf{b}}_i$ are incremented to 1 at the start of $\overline{\mathcal{P}}$. Having (9), one easily implements a conditional construct

$$\textbf{if } \mathsf{b}_i = 1 \textbf{ then } \mathcal{G} \textbf{ else } \mathcal{G}' \tag{10}$$

that executes one of program fragments $\mathcal{G}$, $\mathcal{G}'$ depending on the value of the ith bit b_i:

1: **goto** 2 **or** 5
2: b_i $-= 1$ $\quad$ b_i $+= 1$
3: $\mathcal{G}$
4: **goto** 7
5: $\overline{\mathsf{b}}_i$ $-= 1$ $\quad$ $\overline{\mathsf{b}}_i$ $+= 1$
6: $\mathcal{G}'$
7: ...

Note that exactly one of the two branches fails (because of $\mathsf{b}_i = 0$ or $\overline{\mathsf{b}}_i = 0$) and the other one succeeds. Line 2 is a non-zero test on b_i, while line 5 is, due to the equality (9), a *zero-test* on b_i. The original values of counters $\mathsf{b}_i, \overline{\mathsf{b}}_i$ are retrieved by increments in lines 2 and 5. The conditional construct allows us to write the code for $\mathcal{C}^*$:

if $\mathsf{b}_0 = 1$ **then** $\mathsf{a}_1^0 *= 1$ **else** $\mathsf{a}_0^0 *= 1$
if $\mathsf{b}_1 = 1$ **then** $\mathsf{a}_1^1 *= 1$ **else** $\mathsf{a}_0^1 *= 1$
...
if $\mathsf{b}_{d-1} = 1$ **then** $\mathsf{a}_1^{d-1} *= 1$ **else** $\mathsf{a}_0^{d-1} *= 1$

Also, using the above conditional construct (10) as a building block, one writes down two further program fragments $\mathcal{I}nc$ and $\mathcal{D}ec$, both of size $\mathcal{O}(d)$. Fragment $\mathcal{I}nc$ (resp. $\mathcal{D}ec$) increments (resp. decrements) the current value j represented by counters $\mathsf{b}_0, \ldots, \mathsf{b}_{d-1}$, assuming that this value is below $h-1$ (resp. above 0), by means of bit operations. In order to keep the invariant (9), every increment b_i $+= 1$ of ith bit is accompanied by the corresponding decrement $\overline{\mathsf{b}}_i$ $-= 1$, and symmetrically every decrement b_i $-= 1$ of ith bit is accompanied by the increment $\overline{\mathsf{b}}_i$ $+= 1$.

Let $\overline{\mathcal{H}}$ denote the result of applying the following substitutions to $\mathcal{H}$, for $* \in \{+, -\}$:

$$\mathsf{x} *= 1 \quad \mapsto \quad \mathcal{D}ec \;\; \mathcal{C}^* \;\; \mathcal{I}nc$$
$$\mathsf{x}' *= 1 \quad \mapsto \quad \mathcal{C}^*$$

This exactly implements substitutions (8): assuming the current value of j is represented in counters b_i and $\overline{\mathsf{b}}_i$, as described above, execution of the program fragment $\mathcal{C}^*$ has the same effect as execution of $\mathcal{C}_j^*$; and execution of $\mathcal{D}ec\ \mathcal{C}^*\ \mathcal{I}nc$ has the same effect as decrementing j, executing $\mathcal{C}_j^*$, and incrementing j to retrieve its actual current value, which is equivalent to execution of $\mathcal{C}_{j-1}^*$.

The size of $\overline{\mathcal{H}}$ is $\mathcal{O}(d)$. It remains to wrap up this program fragment inside a loop that increments the value j, represented by counters $\mathsf{b}_0, \ldots, \mathsf{b}_{d-1}$, from 0 to $h-1$. Using an aggregated conditional construct **if** $\mathsf{b}_0 = \mathsf{b}_1 = \ldots = \mathsf{b}_{d-1} = 1$ **then goto** L of size $\mathcal{O}(d)$, defined as expected, we write down the following program fragment $\overline{\mathcal{G}}$, again of size $\mathcal{O}(d)$:[6]

1: $\mathcal{I}nc$
2: **if** $\mathsf{b}_0 = \mathsf{b}_1 = \ldots = \mathsf{b}_{d-1} = 1$ **then goto** 6
3: $\overline{\mathcal{H}}$
4: $\mathcal{I}nc$
5: **goto** 2
6: ...

Assuming the initial values $\mathsf{b}_i = 0$ and $\overline{\mathsf{b}}_i = 1$ for all $j = 0, \ldots, d-1$, $\overline{\mathcal{G}}$ sets values $\mathsf{b}_i = 1$ and $\overline{\mathsf{b}}_i = 0$ for all $j = 0, \ldots, d-1$, once line 6 is reached. We use $\overline{\mathcal{G}}$ as a part of the final counter program $\overline{\mathcal{P}}$. As the substitutions (8) apply fully only when $j = 1, \ldots, h-1$, we keep the first and the last segments $\widetilde{\mathcal{H}}_0$ and $\widetilde{\mathcal{H}}_h$ as defined in (6)–(7), and replace all others by $\overline{\mathcal{G}}$. The counter program $\overline{\mathcal{P}}$ has the following form:

$$\underbrace{\mathcal{H}_{-1}}_{\text{size } \mathcal{O}(n)} \quad \underbrace{\widetilde{\mathcal{H}}_0}_{\text{size } \mathcal{O}(d)} \quad \underbrace{\overline{\mathcal{G}}}_{\text{size } \mathcal{O}(d)} \quad \underbrace{\widetilde{\mathcal{H}}_h}_{\text{size } \mathcal{O}(d)} \quad \underbrace{\mathcal{H}_{h+1}}_{\text{size } \mathcal{O}(n)} \quad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_h, \mathsf{a}_0^{d-1}, \mathsf{a}_1^{d-1} = 0. \tag{11}$$

Its size is $\mathcal{O}(n+d)$ and its dimension is larger by $2d$ than the dimension of $\widetilde{\mathcal{P}}$.

▷ Claim 15 (Correctness). $\widetilde{\mathcal{P}}$ has a halting run if, and only if, $\overline{\mathcal{P}}$ has one.

Proof. Again, halting runs of $\widetilde{\mathcal{P}}$ are in one-to-one correspondence with halting runs of $\overline{\mathcal{P}}$. Indeed, a halting run of $\widetilde{\mathcal{P}}$ is faithfully simulated in $\overline{\mathcal{P}}$ by iterating through $j = 0, 1, \ldots, h-1$ in $\overline{\mathcal{G}}$. Conversely, by the very construction of $\overline{\mathcal{P}}$, its every halting run simulates in this way some halting run of $\widetilde{\mathcal{P}}$. Note that $\overline{\mathcal{P}}$ has also other runs which are necessarily partial (i.e., they fail to reach the **halt** command), because of choosing a failing branch in some of the conditional constructs (10). ◁

The optimisation $\mathcal{P} \mapsto \widetilde{\mathcal{P}}$ described above yields a reduction $\mathcal{M} \mapsto \overline{\mathcal{P}}$, whose correctness follows by Claims 5, 13 and 15:

▷ Claim 16. Given a 3-counter Minsky machine $\mathcal{M}$ of size n and a positive integer d, one can compute in time $\mathcal{O}(n+d)$ a counter program $\overline{\mathcal{P}}$ of dimension $4d+13$ such that $\mathcal{M}$ has a halting run with counters bounded by $\mathcal{F}(2^d+1, n)$ if, and only if, $\overline{\mathcal{P}}$ has a halting run.

Putting $d = n$, we obtain a linear-time reduction from the EXP-TOWER HALTING PROBLEM. In consequence, the VASS reachability problem is 2^n-TOWER-hard, with respect to polynomial-time reductions.

[6] As remarked in the footnote 4 in Section 4, $\overline{\mathcal{G}}$ adapts straightforwardly if, instead of just one $\mathcal{H}$, two different program fragments are used, say $\mathcal{H}_0$ and $\mathcal{H}_1$, one of them for even j and the other one for odd j. It is enough to replace line 3 by **if** $\mathsf{b}_0 = 1$ **then** $\overline{\mathcal{H}}_1$ **else** $\overline{\mathcal{H}}_0$

References

1 David Angeli, Patrick De Leenheer, and Eduardo D. Sontag. Persistence results for chemical reaction networks with time-dependent kinetics and no global conservation laws. *SIAM Journal of Applied Mathematics*, 71(1):128–146, 2011. doi:10.1137/090779401.

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

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

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

5 Ahmed Bouajjani and Michael Emmi. Analysis of recursively parallel programs. *ACM Trans. Program. Lang. Syst.*, 35(3):10:1–10:49, 2013. doi:10.1145/2518188.

6 Frank P. Burns, Albert Koelmans, and Alexandre Yakovlev. WCET analysis of superscalar processors using simulation with coloured Petri nets. *Real-Time Systems*, 18(2/3):275–288, 2000. doi:10.1023/A:1008101416758.

7 Thomas Colcombet and Amaldev Manuel. Generalized data automata and fixpoint logic. In *FSTTCS*, volume 29 of *LIPIcs*, pages 267–278. Schloss Dagstuhl, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.267.

8 Stefano Crespi-Reghizzi and Dino Mandrioli. Petri nets and Szilard languages. *Information and Control*, 33(2):177–192, 1977. doi:10.1016/S0019-9958(77)90558-7.

9 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In Moses Charikar and Edith Cohen, editors, *Proc. STOC 2019*, pages 24–33. ACM, 2019.

10 Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. Ordered navigation on multi-attributed data words. In *CONCUR*, volume 8704 of *LNCS*, pages 497–511. Springer, 2014. doi:10.1007/978-3-662-44584-6_34.

11 Stéphane Demri, Diego Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:1)2016.

12 Javier Esparza. Decidability and complexity of Petri net problems — an introduction. In *Lectures on Petri Nets I*, volume 1491 of *LNCS*, pages 374–428. Springer, 1998. doi:10.1007/3-540-65306-6_20.

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

14 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2(3):265–283, 1968. doi:10.1007/BF01694011.

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

16 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.

17 Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.*, 7:311–324, 1978. doi:10.1016/0304-3975(78)90020-8.

18 Piotr Hofman and Sławomir Lasota. Linear equations with ordered data. In *CONCUR*, volume 118 of *LIPIcs*, pages 24:1–24:17. Schloss Dagstuhl, 2018. doi:10.4230/LIPIcs.CONCUR.2018.24.

19 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

20 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. A widening approach to multithreaded program verification. *ACM Trans. Program. Lang. Syst.*, 36(4):14:1–14:29, 2014. doi:10.1145/2629608.

21 Max I. Kanovich. Petri nets, Horn programs, linear logic and vector games. *Ann. Pure Appl. Logic*, 75(1–2):107–135, 1995. doi:10.1016/0168-0072(94)00060-G.

22 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

23 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC*, pages 267–281. ACM, 1982. doi:10.1145/800070.802201.

24 Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

25 Hélène Leroux, David Andreu, and Karen Godary-Dejean. Handling exceptions in Petri net-based digital architecture: From formalism to implementation on FPGAs. *IEEE Trans. Industrial Informatics*, 11(4):897–906, 2015. doi:10.1109/TII.2015.2435696.

26 Jérôme Leroux. The general vector addition system reachability problem by Presburger inductive invariants. *Logical Methods in Computer Science*, 6(3), 2010. doi:10.2168/LMCS-6(3:22)2010.

27 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL*, pages 307–316. ACM, 2011. doi:10.1145/1926385.1926421.

28 Jérôme Leroux. Vector addition systems reachability problem (A simpler solution). In *Turing-100*, volume 10 of *EPiC Series in Computing*, pages 214–228. EasyChair, 2012. URL: http://www.easychair.org/publications/paper/106497.

29 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *LICS*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.

30 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Proc. LICS 2019*, pages 1–13. IEEE, 2019.

31 Yuliang Li, Alin Deutsch, and Victor Vianu. VERIFAS: A practical verifier for artifact systems. *PVLDB*, 11(3):283–296, 2017. URL: http://www.vldb.org/pvldb/vol11/p283-li.pdf, doi:10.14778/3157794.3157798.

32 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976. URL: http://cpsc.yale.edu/sites/default/files/files/tr63.pdf.

33 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *STOC*, pages 238–246. ACM, 1981. doi:10.1145/800076.802477.

34 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984. doi:10.1137/0213029.

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

36 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967. URL: https://dl.acm.org/citation.cfm?id=1095587.

37 Mor Peleg, Daniel L. Rubin, and Russ B. Altman. Research paper: Using Petri net tools to study properties and dynamics of biological systems. *JAMIA*, 12(2):181–199, 2005. doi:10.1197/jamia.M1637.

38 Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962. URL: http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/.

39 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

40 George S. Sacerdote and Richard L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). In *STOC*, pages 61–76. ACM, 1977. doi:10.1145/800105.803396.

41 Sylvain Schmitz. Complexity hierarchies beyond elementary. *TOCT*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.

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

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