



**HAL**  
open science

# New Lower Bounds for Reachability in Vector Addition Systems

Wojciech Czerwiński, Ismaël Jecker, Sławomir Lasota, Jérôme Leroux, Łukasz Orlikowski

► **To cite this version:**

Wojciech Czerwiński, Ismaël Jecker, Sławomir Lasota, Jérôme Leroux, Łukasz Orlikowski. New Lower Bounds for Reachability in Vector Addition Systems. FSTTCS, Dec 2023, Telangana, India. pp.35:1–35:22, 10.4230/LIPICS.FSTTCS.2023.35. hal-04782385

**HAL Id: hal-04782385**

**<https://hal.science/hal-04782385v1>**

Submitted on 14 Nov 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.


# 1 New Lower Bounds for Reachability in Vector 2 Addition Systems

3 **Wojciech Czerwiński** ✉ 

4 University of Warsaw

5 **Ismaël Jecker**

6 University of Warsaw

7 **Sławomir Lasota** 

8 University of Warsaw, Poland

9 **Jérôme Leroux**

10 LaBRI, CNRS, Univ. Bordeaux, France

11 **Łukasz Orlikowski**

12 University of Warsaw

## 13 — Abstract —

---

14 We investigate the dimension-parametric complexity of the reachability problem in vector addition  
15 systems with states (VASS) and its extension with pushdown stack (pushdown VASS). Up to  
16 now, the problem is known to be  $\mathcal{F}_d$ -hard for VASS of dimension  $3d + 2$  (the complexity class  
17  $\mathcal{F}_d$  corresponds to the  $k$ th level of the fast-growing hierarchy), and no essentially better bound  
18 is known for pushdown VASS. We provide a new construction that improves the lower bound for  
19 VASS:  $\mathcal{F}_k$ -hardness in dimension  $2d + 3$ . Furthermore, building on our new insights we show a new  
20 lower bound for pushdown VASS:  $\mathcal{F}_k$ -hardness in dimension  $\frac{d}{2} + 6$ . This dimension-parametric lower  
21 bound is strictly stronger than the upper bound for VASS, which suggests that the (still unknown)  
22 complexity of the reachability problem in pushdown VASS is higher than in plain VASS (where it is  
23 Ackermann-complete).

24 **2012 ACM Subject Classification** Theory of computation → Concurrency; Theory of computation  
25 → Verification by model checking; Theory of computation → Logic and verification

26 **Keywords and phrases** vector addition systems, reachability problem, pushdown vector addition  
27 system, lower bounds

28 **Digital Object Identifier** 10.4230/LIPIcs.STACS.2022.35

29 **Funding** *Wojciech Czerwiński*: Supported by the ERC grant INFSYS, agreement no. 950398.

30 *Sławomir Lasota*: Supported by the ERC project ‘Lipa’ within the EU Horizon 2020 research and  
31 innovation programme (No. 683080) and by the NCN grant 2021/41/B/ST6/00535.

32 *Jérôme Leroux*: Supported by the grant ANR-17-CE40-0028 of the French National Research Agency  
33 ANR (project BRAVAS).



© Sławomir Lasota;

licensed under Creative Commons License CC-BY 4.0

39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022).

Editors: Petra Berenbrink and Benjamin Monmege; Article No. 35; pp. 35:1–35:24

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

34 **1** Introduction

35 Petri nets, equivalently presentable as vector addition systems with states (VASS), are an  
 36 established model of concurrency with widespread applications. The central algorithmic  
 37 problem for this model is the *reachability problem* which asks whether from a given initial  
 38 configuration there exists a sequence of valid execution steps reaching a given final config-  
 39 uration. For a long time the complexity of this problem remained one of the hardest open  
 40 questions in verification of concurrent systems. In 2019 Leroux and Schmitz made a significant  
 41 breakthrough by providing an Ackermannian upper bound [14]. With respect to the hardness,  
 42 the exponential space lower bound, shown by Lipton already in 1976 [16], remained the only  
 43 known for over 40 years until a breakthrough non-elementary lower bound by Czerwiński,  
 44 Lasota, Lazic, Leroux and Mazowiecki in 2019 [3, 4]. Finally, a matching Ackermannian lower  
 45 bound announced in 2021 independently by two teams, namely Czerwiński and Orlikowski [5]  
 46 and Leroux [12], established the exact complexity of the problem.

47 However, despite the fact that the exact complexity of the reachability problem for VASS  
 48 is settled, there are still significant gaps in our understanding of the problem. One such gap  
 49 is the complexity of the reachability problem *parametrised by the dimension*, namely deciding  
 50 the reachability problem for  $d$ -dimensional VASS ( $d$ -VASS) for fixed  $d \in \mathbb{N}$ . Currently, the  
 51 exact complexity bounds are only known for dimensions one and two. In these cases, the  
 52 complexity depends on the representations of numbers in the transitions, either unary or  
 53 binary. For binary VASS (where the numbers are represented in binary) the reachability  
 54 problem is known to be NP-complete for 1-VASS [9] and PSPACE-complete for 2-VASS [2].  
 55 For unary VASS the problem is NL-complete for both 1-VASS (folklore) and 2-VASS [8].

56 Much less is known for higher dimensions, and it is striking that even in the case of  
 57 3-VASS we have a huge complexity gap. The best complexity upper bound comes from the  
 58 above mentioned work of Leroux and Schmitz [14], where it is proved that the reachability  
 59 problem for  $(d-4)$ -VASS is in  $\mathcal{F}_d$  (here  $\mathcal{F}_d$  denotes the  $d$ th level of the Grzegorzczuk hierarchy  
 60 of complexity classes, which corresponds to the fast growing function hierarchy). In particular  
 61 this shows that the reachability problem for 3-VASS is in  $\mathcal{F}_7$  (recall that  $\mathcal{F}_3 = \text{TOWER}$ ).

62 The recent Ackermann-hardness results provide lower bounds for the reachability problem  
 63 in fixed dimensions. The result of Czerwiński and Orlikowski [5] yields  $\mathcal{F}_d$ -hardness for  
 64  $6d$ -VASS, while the result of Leroux [12] establishes  $\mathcal{F}_d$ -hardness for  $(4d+5)$ -VASS. Lasota  
 65 improved upon these results and showed  $\mathcal{F}_d$ -hardness of the problem for  $(3d+2)$ -VASS [10].  
 66 In [6], additional lower bound results were obtained for specific fixed dimensions: PSPACE-  
 67 hardness for unary 5-VASS, EXPSPACE-hardness for binary 6-VASS and TOWER-hardness  
 68 for unary 8-VASS.

69 To summarise, despite significant research efforts there are still several natural problems  
 70 related to the VASS reachability problem that present significant complexity gaps:

71 **Q<sub>1</sub>**: What is the complexity of the reachability problem for VASS of dimension 3? It is  
 72 known to be PSPACE-hard and in  $\mathcal{F}_7$ ;

73 **Q<sub>2</sub>**: What is highest dimension for which the complexity of the reachability problem is  
 74 elementary? It is known to fall within the range of 2 to 8;

75 **Q<sub>3</sub>**: What is the smallest constant  $C$  such that the complexity of the reachability problem  
 76 for  $d$ -VASS is in  $\mathcal{F}_{Cd+o(d)}$ ? It is known to fall within the range of  $\frac{1}{3}$  to 1.

77 In this work, we focus on addressing Question **Q<sub>3</sub>**. We present new and improved lower  
 78 bounds, first in the standard setting of VASS, and then in the setting of pushdown VASS  
 79 (PVASS) which extend the VASS model by incorporating a pushdown stack.

80 **VASS reachability.** Our first main result is a new complexity lower bound for the reachability  
81 problem which improves the gap:

82 ▶ **Theorem 1.** *The reachability problem for  $(2d + 3)$ -VASS is  $\mathcal{F}_d$ -hard.*

83 A preliminary version of this result was presented in in [11]. In this revised version, we  
84 aim to present the result in a conceptually simple framework by using the notion of *triples*, a  
85 formalism that was originally developed in [3], and was heavily used in [5] and [10]. We view  
86 this contribution as an important step towards understanding the complexity of the VASS  
87 reachability problem parametrised by the dimension.

88 **PVASS reachability.** The decidability of the reachability problem for PVASS has been an  
89 important open problem for over a decade [1]. Despite efforts of the community, it remains  
90 unknown even for PVASS of dimension 1 (1-PVASS), namely automata with one counter  
91 and one pushdown stack.

92 ▶ **Conjecture 2.1.** *The reachability problem for PVASS is decidable.*

93 It is important to acknowledge that the PVASS setting is complex, and few decidability  
94 results are known. Some progress has been made in the study of the *coverability* problem, a  
95 variant of the reachability problem which asks whether from a given initial configuration  
96 there exists a sequence of valid execution steps that reaches a configuration *greater* than the  
97 given final configuration. Notably, the coverability problem has been shown to be decidable  
98 for 1-PVASS [15] (and mentioned to be in EXPSpace).

99 Interestingly, despite the slow progress on determining upper bounds for PVASS, there is  
100 limited knowledge about lower bounds as well. To the best of our knowledge, the only lower  
101 bound that is not directly implied by the results on VASS concerns (again) the coverability  
102 problem for 1-PVASS, which has been established as PSPACE-hard [7].

103 As for our contribution, our second main result is the first complexity lower bound for  
104 the PVASS reachability problem that is not immediately inherited from VASS:

105 ▶ **Theorem 2.** *The reachability problem for  $(\lfloor \frac{d}{2} \rfloor + 6)$ -PVASS is  $\mathcal{F}_d$ -hard.*

106 Notably, Theorem 2 implies that for sufficiently large  $d$  the reachability problem for  
107  $d$ -PVASS is harder than the problem for  $(d + 1)$ -VASS (which is a subclass of  $d$ -PVASS as  
108 the pushdown stack can keep track of one VASS counter). Indeed the problem for  $d$ -PVASS  
109 is  $\mathcal{F}_{2d-12}$ -hard by Theorem 2, while the problem for  $(d + 1)$ -VASS is in  $\mathcal{F}_{d+5}$  by [14].

110 While our results indicate that the reachability problem for PVASS is harder than for  
111 VASS, some known results about PVASS hint that even higher lower bounds might be proved:  
112 In [13] it was shown that PVASS are able to weakly compute functions of hyper-Ackermannian  
113 growth rate. Based on this observation we propose the following two conjectures:

114 ▶ **Conjecture 2.2.** *There exists a fixed dimension  $d \in \mathbb{N}$  such that the reachability problem  
115 for  $d$ -PVASS is ACKERMANN-hard.*

116 ▶ **Conjecture 2.3.** *The reachability problem for PVASS is HYPERACKERMANN-hard.*

## 117 2 Preliminaries

118 **Fast-growing hierarchy.** Let  $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$  be the set of positive integers. We define  
119 the complexity classes  $\mathcal{F}_i$  corresponding to the  $i$ th level in the Grzegorzczuk Hierarchy [18,

## 35:4 New Lower Bounds for Reachability in Vector Addition Systems

120 Sect. 2.3, 4.1]. To this aim we choose to use the following family of functions  $\mathbf{F}_i : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ ,  
 121 indexed by  $i \in \mathbb{N}$ :

$$122 \quad \mathbf{F}_0(n) = n + 2, \quad \mathbf{F}_{i+1} = \widetilde{\mathbf{F}}_i \quad \text{where} \quad \widetilde{\mathbf{F}}(n) = F^{n-1}(1) = \underbrace{F \circ F \circ \dots \circ F}_{n-1}(1). \quad (1)$$

123  
 124 In particular,  $\mathbf{F}_1(n) = 2n - 1$  and  $\mathbf{F}_i(1) = 1$  for all  $i \in \mathbb{N}_+$ . Using the functions  $\mathbf{F}_i$ , we define  
 125 the complexity classes  $\mathcal{F}_i$ , indexed by  $i \in \mathbb{N}_+$ , of problems solvable in deterministic time  
 126  $\mathbf{F}_i \circ \mathbf{F}_{i-1}^m(n)$  for some  $m \in \mathbb{N}$ :

$$127 \quad \mathcal{F}_i = \bigcup_{m \in \mathbb{N}} \text{DTIME}(\mathbf{F}_i \circ \mathbf{F}_{i-1}^m(n)).$$

128 Intuitively speaking, the class  $\mathcal{F}_i$  contains all problems solvable in time  $\mathbf{F}_i(n)$ , and is closed  
 129 under reductions computable in time of lower order  $\mathbf{F}_{i-1}^m(n)$ , for some fixed  $m \in \mathbb{N}_+$ . In  
 130 particular,  $\mathcal{F}_3 = \text{TOWER}$  (problems solvable in a tower of exponentials of time or space  
 131 whose height is an elementary function of input size). The classes  $\mathcal{F}_i$  are robust with respect  
 132 to the choice of fast-growing function hierarchy (see [18, Sect.4.1]). For  $i \geq 3$ , instead of  
 133 deterministic time, one could equivalently take nondeterministic time, or space as all these  
 134 definitions collapse.

### 135 2.1 Counter programs

136 A *counter program* (or simply a *program*) is a sequence of (line-numbered) commands, each  
 137 of which is of one of the following types:

138  $x += 1$  (increment the counter  $x$  by one)  
 $x -= 1$  (decrement the counter  $x$  by one)  
**goto**  $L$  or  $L'$  (nondeterministically jump to either line  $L$  or line  $L'$ )  
**zero?**  $x$  (zero test: continue if counter  $x$  equals 0)

139 Counter programs *with a pushdown stack* (or simply *programs with stack*) are enhanced  
 140 versions of plain counter programs that incorporate a stack containing a word over a fixed  
 141 stack alphabet  $\mathbf{S}$ . The stack content is modified by using two additional types of commands:

142 **PUSH**( $s$ ) (push the symbol  $s \in \mathbf{S}$  at the top of the stack)  
**POP**( $s$ ) (pop the symbol  $s \in \mathbf{S}$  if it is at the top of the stack)

143 The command **POP**( $s$ ) fails if the stack is empty or if the top symbol is different from  $s$ . A  
 144 *configuration* of a counter program with pushdown stack consists, as expected, of a valuation  
 145 of its counters plus a stack content.

146 Counters are only allowed to have nonnegative values.

147 **Conventions.** We are particularly interested in counter programs *without zero tests*, i.e.,  
 148 ones that use no zero test command. In the sequel, unless specified explicitly, counter  
 149 programs are implicitly assumed to be without zero tests. Moreover, we use the syntactic  
 150 sugar **loop**, which iterates a sequence of command a nondeterministic number of times (see  
 151 Example 3). Finally, we write consecutive increments and decrements of different variables  
 152 on a single line and we use the following shorthands:

153  $x += m$  (increment the counter  $x$  by  $m$ )  
 $x -= m$  (decrement the counter  $x$  by  $m$ )  
 $x \rightarrow y$  (decrement the counter  $x$  by one and increment the counter  $y$  by one)  
 $x \xrightarrow{m} y$  (decrement the counter  $x$  by  $m$  and increment the counter  $y$  by  $m$ )

154 ► **Example 3.** As an illustration, consider three different presentations of the same the  
 155 program with three counters  $C = \{x, y, z\}$ :

```

156      1: goto 2 or 6          1: loop                1: loop  x  $\rightarrow$  y  z += 2
      2: x -= 1                  2:   x -= 1          2: z += 1
      3: y += 1                  3:   y += 1
      4: z += 2                  4:   z += 2
      5: goto 1 or 1          5: z += 1
      6: z += 1
  
```

157 The program repeats the block of commands in lines 2–4 some number of times chosen  
 158 nondeterministically (possibly zero, although not infinite because  $x$  is decreasing, and hence  
 159 its initial value bounds the number of iterations) and then increments  $z$ .

160 We emphasise that counters are only permitted to have nonnegative values. In the  
 161 program above, that is why the decrement in line 2 works also as a non-zero test.

162 **Runs.** Consider a program with counters  $X$ . By  $\mathbb{N}^X$  we denote the set of all valuations of  
 163 counters. Given an initial valuation of counters, a *run* (or *execution*) of a counter program  
 164 is a finite sequence of executions of commands, as expected. A run which has successfully  
 165 finished we call *complete*; otherwise, the run is *partial*. Observe that, due to a decrement  
 166 that would cause a counter to become negative, a partial run may fail to continue because it  
 167 is blocked from further execution. Moreover, due to nondeterminism of **goto**, a program  
 168 may have various runs from the same initial valuation.

169 Two programs  $\mathcal{P}, \mathcal{Q}$  may be *composed* by concatenating them, written  $\mathcal{P} \mathcal{Q}$ . We silently  
 170 assume the appropriate re-numbering of lines referred to by **goto** command in  $\mathcal{Q}$ .

171 Consider a distinguished set of counters  $Z \subseteq X$ . A run of  $\mathcal{P}$  is *Z-zeroing* if it is complete  
 172 and all counters from  $Z$  are zero at the end. Given two counter valuations  $r, r' \in \mathbb{N}^X$  we say  
 173 that the program *Z-computes* the valuation  $r$  from  $r'$  if  $\mathcal{P}$  has exactly one *Z-zeroing* run from  
 174  $r'$  and the end configuration is  $r$ . We also say that the program *Z-computes nothing* from  $r'$  if  
 175  $\mathcal{P}$  has no *Z-zeroing* run from  $r'$ . For instance, in Example 3 the programs  $\{x\}$ -compute, from  
 176 any valuation satisfying  $x = n \in \mathbb{N}$  and  $y = z = 0$ , the valuation satisfying  $x = 0$  (trivially),  
 177  $y = n$  and  $z = 2n + 1$ .

178 **Maximal iteration.** The proofs of this paper often focus on the number of iterations of  
 179 the **loop** construct. Consider a program  $\mathcal{P}$  containing a *flat loop*, i.e., a loop whose body  
 180 consists of only increment or decrement commands, such that each counter appears in at  
 181 most one of these commands (like the programs in Example 3). We say that this loop is  
 182 *maximally iterated* in a given run of a  $\mathcal{P}$  if some counter that is decremented in its body is  
 183 zero at the exit from the loop. In particular, a maximally iterated loop could not be iterated  
 184 any further without violation of the nonnegativity constraint on the decremented counter.  
 185 For instance, the loop in Example 3 is maximally iterated by the  $\{x\}$ -zeroing runs. Needless  
 186 to say, maximal iteration needs not happen in general, for instance the program in Example 3  
 187 has multiple complete runs that do not admit this property.

### 188 **3 Main results and structure of the paper**

189 Counter programs (without zero test) provide an equivalent presentation to the standard  
 190 models of Petri nets and VASS, and the transformations between these different models are  
 191 straightforward. For instance, a program can be transformed into a VASS by taking one

192 state for each line of the program, and adding an appropriate transition corresponding to  
 193 each counter update instruction. Note that the dimension of the VASS obtained is equal to  
 194 the number of counters of the program. In this paper, we focus solely on counter programs,  
 195 and we prove that the following problem is  $\mathcal{F}_d$ -hard for every  $d \geq 3$ :

196 ► **Problem 1.** *Input:* A program  $\mathcal{P}$  using  $2d + 3$  counters.

197 **Question:** *Is there a complete run of  $\mathcal{P}$  that starts and ends with all counters equal to 0?*

198 The equivalence between programs and VASS then directly leads to our first main result:

199 ► **Theorem 1.** *The reachability problem for  $(2d + 3)$ -VASS is  $\mathcal{F}_d$ -hard.*

200 For programs with a pushdown stack,  $\mathcal{F}_d$ -hardness can be achieved with less counters. We  
 201 show that the following problem is  $\mathcal{F}_d$ -hard for every  $d \geq 3$ :

202 ► **Problem 2.** *Input:* A program with stack  $\mathcal{Q}$  using  $\lfloor \frac{d}{2} \rfloor + 6$  counters.

203 **Question:** *Is there a complete run of  $\mathcal{Q}$  that starts and ends with all counters equal to 0?*

204 Again, the equivalence between programs and VASS yields our second main result:

205 ► **Theorem 2.** *The reachability problem for  $(\lfloor \frac{d}{2} \rfloor + 6)$ -PVASS is  $\mathcal{F}_d$ -hard.*

206 Let us now introduce the known  $\mathcal{F}_d$ -hard problem that we will reduce to Problems 1 and 2.  
 207 Fortunately, we do not have to search too far for it: counter programs with two counters *and*  
 208 *zero tests* are Turing-complete [17]. This implies that the reachability problem is undecidable  
 209 in that setting. However, similarly to Turing machines, decidability is recovered by imposing  
 210 limitations on the executions, such as bounding their length, the maximal counter size, or the  
 211 number of zero tests. For our purposes the latter is the easiest to use, thus, we present the  
 212 problem that we will reduce from, a variation of the “ $\mathbf{F}_k$ -bounded Minsky Machine Halting  
 213 Problem” proved to be  $\mathcal{F}_d$ -complete in [18, Section 2.3.2]:

214 ► **Problem 3.** *Input:* A program  $\mathcal{P}$  with two zero-tested counters, and a bound  $n \in \mathbb{N}$ .

215 **Question:** *Is there a complete run of  $\mathcal{P}$  that starts and ends with all counters equal to 0 and  
 216 does exactly  $\mathbf{F}_d(n)$  zero tests?*

217 The rest of this section is devoted to the presentation of the tools we use to reduce Problem 3  
 218 to Problems 1 and 2. Following the structure of similar reductions presented in [5] and [10],  
 219 our reduction is divided into two main steps.

220 In the first step, we show how a program *without zero test* can simulate a bounded number  
 221 of zero tests. To achieve this we rely on the concept of *triples*, which are specific counter  
 222 valuations that allow to eliminate the zero tests and instead verify whether a particular  
 223 invariant still holds at the term of the run. However, doing so requires an initial triple  
 224 directly proportional to the number of zero tests we aim to simulate. Since we intend to  
 225 simulate  $\mathbf{F}_d(n)$  zero tests, which is a rather large number, directly applying this approach  
 226 would result in an excessively large program.

227 Thus, in the second step, we construct compact *amplifiers*. These amplifiers are small  
 228 programs that compute functions of substantial magnitude (such as  $\mathbf{F}_d$ ) while using a small  
 229 number of counters (namely  $2d + 4$ , or  $\lfloor \frac{d}{2} \rfloor + 4$  counters along with a stack).

230 We now define formally the notions required for these two steps. This will allow us to  
 231 state the main lemmas proved in this paper, and use them to construct the reduction proving  
 232 our main result. The proofs of the lemmas can then be found in the following sections.

233 **Triples.** The concept of *triples* plays a central role in all the constructions presented within  
 234 this paper. Given a set of counters  $X$ , three distinguished counters  $a, b, c \in X$  and  $A, B \in \mathbb{N}$ ,  
 235 we denote by  $\text{TRIPLE}(A, B, a, b, c, X)$  the counter valuation satisfying

$$236 \quad a = A, \quad b = B, \quad c = A \cdot (4^B - 1), \quad x = 0 \text{ for every } x \in X \setminus \{a, b, c\}. \quad (2)$$

238 Informally we sometimes call such valuation a *B-triple*, or simply a *triple* over the counters  
 239  $a, b, c$ . The interest of triples lies in their ability to establish invariants that enable the  
 240 detection of unwanted behaviours in counter programs. For instance, in Section 4, we show  
 241 how to use triples to replace zero tests by proving the following lemma:

242 ► **Lemma 4.** *Let  $\mathcal{P}$  be a program using two zero-tested counters. There exists a program  $\mathcal{P}'$*   
 243 *with six counters  $X$  such that for every  $B \in \mathbb{N}_+$  the two following conditions are equivalent:*  
 244 ■ *There exists a complete run of  $\mathcal{P}$  that starts and ends with all counters equal to 0 and*  
 245 *performs exactly  $B$  zero tests;*  
 246 ■ *There exists a complete run of  $\mathcal{P}'$  that starts in some configuration  $\text{TRIPLE}(A, B, a, b, c, X)$*   
 247 *with  $A \in \mathbb{N}_+$  and ends in a configuration where all the counters except  $a$  are zero.*

248 **Amplifiers.** The key contribution of this paper consists in the construction of two families  
 249 of programs that transform *B-triples* into  $\mathbf{F}_d(B)$ -triples. We formalise this type of programs  
 250 through the notion of *amplifiers*. Let  $F : \mathbb{N}_+ \rightarrow \mathbb{N}_+$  be a monotone function satisfying  
 251  $F(n) \geq n$  for every  $n \in \mathbb{N}_+$ . Consider a program  $\mathcal{P}$  using the set of counters  $X$ , out of which we  
 252 distinguish six counters  $a, b, c, b', c', t \in X$ , and a subset of counters  $Z \subseteq X \setminus \{a, b, c, b', t\}$  that  
 253 contains  $c'$ . The tuple  $(\mathcal{P}, (a, b, c), (a, b', c'), t, Z)$  is called *F-amplifier* if for all  $A, B \in \mathbb{N}_+$ :

- 254 ■  $\mathcal{P}$  *Z*-computes  $\text{TRIPLE}(A \cdot 4^{B-F(B)}, F(B), a, b, c, X)$  from  $\text{TRIPLE}(A, B, a, b', c', X)$  if  $A$  is  
 255 divisible by  $4^{(F(B)-B)}$ ;
- 256 ■  $\mathcal{P}$  *Z*-computes nothing from  $\text{TRIPLE}(A, B, a, b', c', X)$  if  $A$  is not divisible by  $4^{(F(B)-B)}$ .

257 An amplifier transforms *B-triples* on its *input counters*  $a, b', c'$  into  $F(B)$ -triples on its *output*  
 258 *counters*  $a, b, c$ . Remark that the counter  $a$  is involved in both input and output. The  
 259 counters in  $Z$ , called *end counters*, are intuitively speaking assumed to be 0-checked after the  
 260 completion of a run of  $\mathcal{P}$ . The auxiliary counter  $t$  does not play a direct role apart from *not*  
 261 being an input counter, an output counter nor an end counter. This will prove useful in our  
 262 constructions. We note that no condition is imposed on the runs that start from a counter  
 263 valuation that is not a triple on the input counters, nor on the runs that are not *Z*-zeroing.

264 In Section 5 we construct of a family of  $\mathbf{F}_d$ -amplifiers:

265 ► **Lemma 5.** *For every  $d \in \mathbb{N}_+$  there exists an  $\mathbf{F}_d$ -amplifier of size  $\mathcal{O}(d)$  that uses  $2d + 4$*   
 266 *counters out of which  $d$  are end counters.*

267 Furthermore, in Section 6 we extend the notion of amplifiers to programs with stack, and  
 268 we demonstrate how using a stack in an efficient manner can replace three quarters of the  
 269 counters used in the previous construction:

270 ► **Lemma 6.** *For every  $d \in \mathbb{N}_+$  there exists an  $\mathbf{F}_d$ -amplifier of size  $\mathcal{O}(d)$  that uses a stack*  
 271 *and  $\lfloor \frac{d}{2} \rfloor + 4$  counters out of which  $\lfloor \frac{d}{2} \rfloor$  are end counters.*

272 **Proof of the main theorems.** While the proofs of our key lemmas are delegated to the  
 273 appropriate sections, we can already show how these lemmas yield a reduction from Problem 3  
 274 to Problems 1 and 2. Let us consider an instance of Problem 3, that is, a 2-counter program



275 with zero tests  $\mathcal{P}$  and an integer  $d \in \mathbb{N}$ . We transform this instance into an instance  $\mathcal{P}''$  of  
 276 Problem 1 and an instance  $\mathcal{Q}''$  of Problems 2. These two programs rely on the program  $\mathcal{P}'$   
 277 given by Lemma 4, and the  $\mathbf{F}_d$ -amplifiers  $\mathcal{P}_d$  and  $\mathcal{Q}_d$  given by Lemmas 5 and 6.

278 Program $\mathcal{P}''$ :	Program $\mathcal{Q}''$ :
1: $\mathbf{b}' += n$	1: $\mathbf{b}' += n$
2: <b>loop</b> $\mathbf{a} += 1$ $\mathbf{c}' += 4^n - 1$	2: <b>loop</b> $\mathbf{a} += 1$ $\mathbf{c}' += 4^n - 1$
3: $\mathcal{P}_d$	3: $\mathcal{Q}_d$
4: $\mathcal{P}'$	4: $\mathcal{P}'$
5: <b>loop</b> $\mathbf{a} -= 1$	5: <b>loop</b> $\mathbf{a} -= 1$

279 We now prove that this is a valid reduction from Problem 3 to Problem 1. The proof for  
 280 Problem 2 is analogous since the programs  $\mathcal{P}_d$  and  $\mathcal{Q}_d$  have identical effect on triples.

281 We need to show that  $\mathcal{P}''$  has a complete run that starts and ends with all counters equal  
 282 to 0 if and only if  $\mathcal{P}$  has a complete run that starts and ends with all counters equal to 0 and  
 283 that does exactly  $\mathbf{F}_d(n)$  zero tests. In order to prove it, let us consider the structure of a  
 284 hypothetical run  $\pi$  of  $\mathcal{P}''$  that starts and ends with all counters having value zero. Starting  
 285 from the configuration where all the counters are zero, Lines 1–2 generate an arbitrary  
 286  $n$ -triple: Progressing through line 1 and performing  $A \in \mathbb{N}_+$  iterations of line 2 results in  
 287 the configuration  $\text{TRIPLE}(A, n, \mathbf{a}, \mathbf{b}', \mathbf{c}', \mathbf{X})$ . Next, it is important to note that the subrun  
 288 of  $\pi$  involving  $\mathcal{P}_d$  zeroes all the end counters, since these counters remain unchanged after  
 289 the invocation of  $\mathcal{P}_d$  and they have value zero at the end of  $\pi$ . Consequently, according to  
 290 the definition of an amplifier,  $\mathcal{P}_d$  transforms the  $n$ -triple  $\text{TRIPLE}(A, n, \mathbf{a}, \mathbf{b}', \mathbf{c}', \mathbf{X})$  into an  
 291  $\mathbf{F}_d(n)$ -triple  $\text{TRIPLE}(A', \mathbf{F}_d(n), \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{X})$ . From there, the subrun involving  $\mathcal{P}'$  must end in  
 292 a configuration where every counter except  $\mathbf{a}$  has value zero since the final line of  $\mathcal{P}''$  can  
 293 only decrement  $\mathbf{a}$ . Therefore, the run  $\pi$  exists if and only if there exists a run of  $\mathcal{P}'$  that  
 294 bridges the gap, starting from an  $\mathbf{F}_d(n)$ -triple  $\text{TRIPLE}(A', \mathbf{F}_d(n), \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{X})$  and ending in a  
 295 configuration where all the counters except  $\mathbf{a}$  are 0. By Lemma 4 we know that such a run of  
 296  $\mathcal{P}'$  exists if and only if  $\mathcal{P}$  has a complete run that starts and ends with all counters equal to  
 297 0 and does exactly  $\mathbf{F}_d(n)$  zero tests, which shows that our reduction is valid.

298 To conclude, let us remark that, as defined here, the program  $\mathcal{P}''$  uses  $2d + 7$  counters:  
 299 the call to  $\mathcal{P}_d$  requires  $2d + 4$  counters, and while  $\mathcal{P}'$  shares the output counters  $\mathbf{a}, \mathbf{b}$  and  $\mathbf{c}$  of  
 300  $\mathcal{P}_d$ , it uses three more counters. We now argue that four counters can be saved, so that our  
 301 program matches the definition of Problem 1.

- 302 ■ The value of the input counter  $\mathbf{b}'$  is never incremented in the program  $\mathcal{P}_d$  we construct.<sup>1</sup>  
 303 Therefore, since in  $\mathcal{P}''$  the call to  $\mathcal{P}_d$  *always* starts with the value  $\mathbf{b}' = n$ , we can get rid  
 304 of the counter  $\mathbf{b}'$  by replacing the call to  $\mathcal{P}_d$  with  $n$  consecutive copies of  $\mathcal{P}_d$  in which  
 305 each instruction decrementing  $\mathbf{b}'$  is replaced with a jump to the next copy.
- 306 ■ The second optimisation consists in reusing the counters of  $\mathcal{P}_d$ . Since  $\mathcal{P}_d$  is an amplifier,  
 307 at the term of the run it is sufficient to check that the end counters are 0 to ensure that  
 308 *all* the counters except the output counters  $\mathbf{a}$  and  $\mathbf{c}$  are 0. Therefore, while the call to  $\mathcal{P}'$   
 309 in  $\mathcal{P}''$  needs to keep the  $d$  end counters of  $\mathcal{P}_d$  untouched, there are still  $d$  freely reusable  
 310 counters (not counting  $\mathbf{a}, \mathbf{b}$  and  $\mathbf{c}$  that are already reused), and we can pick any three of  
 311 these to use in  $\mathcal{P}'$  instead of adding fresh ones.

312 For the program  $\mathcal{Q}''$  it is not possible to save counters is that way, but one of the extra  
 313 counters of  $\mathcal{P}'$  can be loaded on the stack, which results in a program with  $2d + 6$  counters.

---

<sup>1</sup> Remark that this is *not* stated explicitly by Lemma 5, but it is a trivial property of the corresponding construction presented in Section 5.

## 314 4 Triples as a replacement for zero tests

315 Let  $\mathcal{P}$  be a program with zero tests using two counters  $x$  and  $y$ . The goal of this subsection  
 316 is to construct a program that simulates  $\mathcal{P}$  *without using zero tests*.

317 ► **Lemma 4.** *Let  $\mathcal{P}$  be a program using two zero-tested counters. There exists a program  $\mathcal{P}'$*   
 318 *with six counters  $X$  such that for every  $B \in \mathbb{N}_+$  the two following conditions are equivalent:*  
 319 ■ *There exists a complete run of  $\mathcal{P}$  that starts and ends with all counters equal to 0 and*  
 320 *performs exactly  $B$  zero tests;*  
 321 ■ *There exists a complete run of  $\mathcal{P}'$  that starts in some configuration  $\text{TRIPLE}(A, B, a, b, c, X)$*   
 322 *with  $A \in \mathbb{N}_+$  and ends in a configuration where all the counters except  $a$  are zero.*

323 The six counters of program  $\mathcal{P}'$  will consists in two counters  $x, y$  simulating the two counters  
 324 of  $\mathcal{P}$ , three counters  $a, b, c$  containing the initial triple, and an auxiliary counter  $t$ . The idea  
 325 behind the construction is that we will replace the zero tests with the two gadgets  $\text{Zero}(x)$   
 326 and  $\text{Zero}(y)$  defined as follows:

327 <u>Program <math>\text{Zero}(x)</math>:</u>	<u>Program <math>\text{Zero}(y)</math>:</u>
1: <b>loop</b> $a \rightarrow t \quad c \rightarrow t$	1: <b>loop</b> $a \rightarrow t \quad c \rightarrow t$
2: <b>loop</b> $y \rightarrow x \quad c \rightarrow t$	2: <b>loop</b> $x \rightarrow y \quad c \rightarrow t$
3: <b>loop</b> $t \rightarrow a \quad c \rightarrow a$	3: <b>loop</b> $t \rightarrow a \quad c \rightarrow a$
4: <b>loop</b> $x \rightarrow y \quad c \rightarrow a$	4: <b>loop</b> $y \rightarrow x \quad c \rightarrow a$
5: $b \text{ -- } 1$	5: $b \text{ -- } 1$

328 The functioning of these two programs revolves around the following invariant:

329 INVARIANT:	$(a + x + y + t) \cdot 4^b = a + x + y + t + c$ and $t = 0$ ;
330 BROKEN INVARIANT:	$(a + x + y + t) \cdot 4^b < a + x + y + t + c$ .

332 Remark that the broken invariant is more specific than the negation of the invariant. We now  
 333 present a technical claim showing that  $\text{Zero}(x)$  and  $\text{Zero}(y)$  accurately replace zero tests.

334 ► **Claim 6.1.** *Let  $z \in \{x, y\}$ . From each configuration of  $\text{Zero}(z)$  where  $b > 0$ ,  $z = 0$  and the*  
 335 *invariant holds there is a unique complete run that maintains the invariant and preserves the*  
 336 *values of  $x$  and  $y$ . All other runs starting from this configuration, as well as all runs starting*  
 337 *with a broken invariant or a holding invariant with a value of  $z$  greater than 0, end with a*  
 338 *broken invariant.*

339 **Proof.** We show the result for  $\text{Zero}(x)$ : the proof can easily be transferred to  $\text{Zero}(y)$  by  
 340 exchanging the roles of  $x$  and  $y$ . Let us start by observing that  $\text{Zero}(x)$  globally decrements  
 341 the value of  $b$  by 1 and preserves the sum  $a + x + y + t + c$  since every line preserves it.  
 342 Therefore, in order to maintain a holding invariant,  $\text{Zero}(x)$  needs to quadruple the value of  
 343 the sum of counters  $a + x + y + t$ . Similarly, to repair a broken invariant  $\text{Zero}(x)$  needs to  
 344 increase the value of  $a + x + y + t$  by more than quadrupling it. We now study  $\text{Zero}(x)$  in  
 345 detail to show that the latter is impossible, and that the former only occurs under specific  
 346 conditions that imply the statement of the claim. We split our analysis in two:

347 **Lines 1–2:** The loop on line 1, respectively 2, increases  $a + x + y + t$  by at most  $a$ , respectively  $y$ .  
 348 Therefore the value of  $a + x + y + t$  is at most doubled, which occurs only if initially  
 349  $t = x = 0$  and if then both loops are maximally iterated, resulting in  $a = y = 0$ .

350 **Lines 3–4:** The loop on line 3, respectively 4, increases  $a + x + y + t$  by at most  $t$ , respectively  $x$ .  
 351 Therefore the value of  $a + x + y + t$  is at most doubled, which happens only if  $a = y = 0$   
 352 upon reaching line 3 and if then both loops are maximally iterated, resulting in  $t = x = 0$ .

## 35:10 New Lower Bounds for Reachability in Vector Addition Systems

353 Combining both parts, we get that the value of  $\mathbf{a} + \mathbf{x} + \mathbf{y} + \mathbf{t}$  is at most quadrupled through  
 354 a run of  $\mathit{Zero}(\mathbf{x})$ , which only occurs if initially  $\mathbf{t} = \mathbf{x} = \mathbf{0}$  and all the loops are maximally  
 355 iterated. This immediately implies that  $\mathit{Zero}(\mathbf{x})$  cannot repair a broken invariant. Moreover,  
 356 to maintain a holding invariant it is necessary to actually quadruple this value, therefore  $\mathbf{x}$   
 357 needs to be 0 at the start; the maximal iteration of the loops will cause  $\mathbf{t}$  to be 0 at the end;  
 358 and the content of  $\mathbf{y}$  will be completely moved to  $\mathbf{x}$  by line 2 and then back to  $\mathbf{y}$  by line 4,  
 359 remaining unchanged as required by the statement. ◀

360 We now use  $\mathit{Zero}(\mathbf{x})$  and  $\mathit{Zero}(\mathbf{y})$  to transform  $\mathcal{P}$  into a program  $\mathcal{P}'$  satisfying Lemma 4.  
 361 Formally, the program  $\mathcal{P}'$  is obtained by applying the following modifications to  $\mathcal{P}$ :

- 362 ■ We add an increment (resp. decrement) of  $\mathbf{a}$  to each line of  $\mathcal{P}$  featuring a decrement  
 363 (resp. increment) of  $\mathbf{x}$  or  $\mathbf{y}$  so that every line preserves the value of  $\mathbf{a} + \mathbf{x} + \mathbf{y} + \mathbf{t}$ ;
- 364 ■ We replace each zero tests “ $\mathbf{zero? x}$ ” with a copy of the program  $\mathit{Zero}(\mathbf{x})$ , and each zero  
 365 test “ $\mathbf{zero? y}$ ” with a copy of the program  $\mathit{Zero}(\mathbf{y})$ ;

366 The first modification ensures that all the lines of  $\mathcal{P}'$  except the calls to  $\mathit{Zero}(\mathbf{x})$  and  $\mathit{Zero}(\mathbf{y})$   
 367 maintain the invariant. We observe that for every complete run  $\pi'$  of  $\mathcal{P}'$  that starts in a triple  
 368  $\text{TRIPLE}(A, B, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{X})$  and ends in a configuration where all counters except  $\mathbf{a}$  are zero, the  
 369 invariant is satisfied both at the beginning (by the definition of a triple) and at the end.  
 370 Therefore, according to Claim 6.1 the invariant is never broken throughout  $\pi'$ , indicating  
 371 that the calls to  $\mathit{Zero}(\mathbf{x})$  and  $\mathit{Zero}(\mathbf{y})$  accurately simulate zero tests. Consequently,  $\pi'$  can  
 372 be transformed into a matching run  $\pi$  of  $\mathcal{P}$  with same values of  $\mathbf{x}$  and  $\mathbf{y}$ . In particular,  $\pi$   
 373 starts and ends with both counters equal to 0. Additionally, the value of  $\mathbf{b}$  goes from  $B$  to 0  
 374 along  $\pi'$ . Since this value is decremented by one by each call to  $\mathit{Zero}(\mathbf{x})$  or  $\mathit{Zero}(\mathbf{y})$ ,  $\pi'$  goes  
 375 through exactly  $B$  such calls, which translates into  $\pi$  performing exactly  $B$  zero tests.

376 To conclude the proof of Lemma 4, remark that we can also transform every run of  $\mathcal{P}$   
 377 that starts and ends with both counters equal to 0 and performs  $B$  zero tests into a matching  
 378 run of  $\mathcal{P}'$  starting from some triple  $\text{TRIPLE}(A, B, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{X})$  and ending in a configuration  
 379 where all counters except  $\mathbf{a}$  are zero. However, we must be cautious in choosing the initial  
 380 value  $A$  of  $\mathbf{a}$  to be sufficiently high. This ensures that we can increment  $\mathbf{x}$  and  $\mathbf{y}$  as high as  
 381 required, despite the matching decrements of  $\mathbf{a}$  added in  $\mathcal{P}'$ .

### 382 **5 Amplifiers defined by counter programs**

383 In this section we build amplifiers of polynomial size for the family of functions  $(\mathbf{F}_d)_{d \in \mathbb{N}}$ :

384 ► **Lemma 5.** *For every  $d \in \mathbb{N}_+$  there exists an  $\mathbf{F}_d$ -amplifier of size  $\mathcal{O}(d)$  that uses  $2d + 4$   
 385 counters out of which  $d$  are end counters.*

386 The rest of this section is devoted to an inductive proof of Lemma 5. First we build an  
 387  $\mathbf{F}_1$ -amplifier  $\mathcal{P}_1$  with 6 counters out of which one is an end counter (Lemma 7). Next we  
 388 show how to lift an arbitrary  $F$ -amplifier with  $d$  counters into an  $\widetilde{F}$ -amplifier by adding two  
 389 counters out of which one is an end counter (Lemma 8). Applying  $d - 1$  times our lifting  
 390 process to the program  $\mathcal{P}_1$  yields an  $\mathbf{F}_d$ -amplifier using  $2d + 4$  counters, proving Lemma 5.

391 **Strong amplifiers.** For the purpose of induction step, namely for lifting  $F$ -amplifiers to  
 392  $\widetilde{F}$ -amplifiers, we need a slight strengthening of the notion of amplifier. An  $F$ -amplifier  
 393  $(\mathcal{P}, (\mathbf{a}, \mathbf{b}, \mathbf{b}), (\mathbf{a}, \mathbf{b}', \mathbf{c}'), \mathbf{t}, \mathbf{Z})$  is called *strong* if every run  $\pi$  of  $\mathcal{P}$  satisfies the following conditions  
 394 (let  $\Sigma \mathbf{Z}$  stand for the sum of all counters in  $\mathbf{Z}$ ):

- 395 1. The value of the sum  $\mathbf{a} + \mathbf{c} + \mathbf{t} + \Sigma \mathbf{Z}$  is the same at the start and at the end of  $\pi$ ;
- 396 2. If  $(\mathbf{a} + \mathbf{c} + \mathbf{t} + \Sigma \mathbf{Z} - \mathbf{c}') \cdot 4^{b'} < \mathbf{a} + \mathbf{c} + \mathbf{t} + \Sigma \mathbf{Z}$  holds at the start of  $\pi$ , it also holds at the end.

## 397 5.1 Construction of the $\mathbf{F}_1$ -amplifier $\mathcal{P}_1$

398 Consider the following program with 6 counters  $X = \{a, b, c, b', c', t\}$ :

Program  $\mathcal{P}_1$ :

```

1: loop a  $\rightarrow$  t
2: loop t  $\rightarrow$  a   c' -= 3   c += 3
3: b' -= 1   b += 1
399 4: loop
5:   loop c  $\rightarrow$  t   c' -= 1   t += 1
6:   loop a  $\xrightarrow{4}$  c   c' -= 4   c += 1   t += 3
7:   loop c  $\rightarrow$  a   c' -= 1   t += 1
8:   loop t  $\rightarrow$  c   c' -= 1   c += 1
9:   b' -= 1   b += 2

```

400 The program consists of an *initialisation* step lines 1–3, and an *iteration* step lines 4–9.

401 ► **Lemma 7.** *The program  $(\mathcal{P}_1, X, (a, b, c), (a, b', c'), t, \{c'\})$  is a strong  $\mathbf{F}_1$ -amplifier.*

402 As an  $\mathbf{F}_1$ -amplifier,  $\mathcal{P}_1$  is expected to map each input  $\text{TRIPLE}(A, B, a, b', c', X)$  such that  
403  $4^{\mathbf{F}_1(B)-B}$  divides  $A$  to the output  $\text{TRIPLE}(A \cdot 4^{B-\mathbf{F}_1(B)}, \mathbf{F}_1(B), a, b, c, X)$ . Since  $\mathbf{F}_1(B) =$   
404  $2B - 1$ , transforming the initial value  $b' = B$  into the final value  $b = \mathbf{F}_1(B)$  is easy:  $\mathcal{P}_1$  first  
405 decrements  $b$  by 1 and increments  $b$  by 1 once (line 3), and then increments  $b$  by 2 whenever  
406 it decrements  $b'$  by 1 (line 9). It is more complicated to transform the initial value  $a = A$   
407 into the final value  $a = A \cdot 4^{B-\mathbf{F}_1(B)}$ : we need to divide  $\mathbf{F}_1(B) - B = B - 1$  times the content  
408 of  $a$  by 4. We prove that  $\mathcal{P}_1$  does so by studying the following invariant:

409 INVARIANT:  $(a + c + t) \cdot 4^{b'} = a + c + t + c'$  and  $t = 0$ ;

410 BROKEN INVARIANT:  $(a + c + t) \cdot 4^{b'} < a + c + t + c'$ .  
411

412 Notice that saying that the invariant is broken is more specific than saying that the invariant  
413 does not hold. We now present two technical claims describing how the invariant evolves  
414 along both steps of  $\mathcal{P}_1$ . The proof of the claims can be found in Appendix A.

415 ► **Claim 7.1.** *From each configuration where  $b' > 0$ ,  $c = 0$  and the invariant holds, there is  
416 a unique run through the initialisation step that maintains the invariant and preserves the  
417 value of  $a$ . All the other runs starting from this configuration, as well as the runs starting  
418 with a broken invariant, end with a broken invariant.*

419 ► **Claim 7.2.** *From each configuration where  $b' > 0$ ,  $a$  is divisible by 4 and the invariant  
420 holds, there is a unique run through the iteration step that maintains the invariant and  
421 divides the value of  $a$  by 4. All the other runs starting from this configuration, as well as  
422 those starting with a broken invariant or a holding invariant with a value of  $a$  not divisible  
423 by 4, end with a broken invariant.*

424 We proceed with the proof of Lemma 7. Let  $A, B \in \mathbb{N}$  and let  $\pi$  be a  $\{c'\}$ -zeroing run of  
425 the program  $\mathcal{P}_1$  starting from  $\text{TRIPLE}(A, B, a, b', c', X)$ . Initially the counters satisfy:

$$426 \quad a = A, \quad b' = B, \quad c' = A \cdot (4^B - 1), \quad b = c = t = 0.$$

427 This directly implies that the invariant holds at the beginning of  $\pi$ . Let us analyse the  
428 values of the counters at the end of  $\pi$ . We immediately get  $c' = 0$  since  $\pi$  is  $\{c'\}$ -zeroing.  
429 Note that this implies that the invariant cannot be broken as the counters always hold

## 35:12 New Lower Bounds for Reachability in Vector Addition Systems

430 non-negative integer values. As a consequence, Claims 7.1 and 7.2 imply that the invariant  
 431 still holds at the end of  $\pi$ , and that  $\pi$  is the unique  $\{c'\}$ -zeroing run of  $\mathcal{P}_1$  starting from  
 432  $\text{TRIPLE}(A, B, a, b', c', X)$ . To conclude the proof, we now show that at the end of  $\pi$  all the  
 433 counters match  $\text{TRIPLE}(A \cdot 4^{B-\mathbf{F}_1(B)}, \mathbf{F}_1(B), a, b, c, X)$ :

$$434 \quad a = A \cdot 4^{B-\mathbf{F}_1(B)}, \quad b = \mathbf{F}_1(B), \quad c = A \cdot (4^B - 4^{B-\mathbf{F}_1(B)}), \quad b' = c' = t = 0.$$

435 First, the invariant directly yields  $t = 0$ , and also  $b' = 0$  by using the fact that  $c' = 0$ . As  $b'$   
 436 starts with value  $B$  and is decremented by one along the initialisation step and each iteration  
 437 step, we get that  $\pi$  visits the iteration step  $B - 1$  times. In turn, this implies that the final  
 438 value of  $b$  is  $2B - 1 = \mathbf{F}_1(B)$ , and by Claims 7.1 and 7.2 we also get that the final value of  $a$   
 439 is  $\frac{A}{4^{B-1}} = A \cdot 4^{B-\mathbf{F}_1(B)}$ . Combining this with the fact that the initial value  $A \cdot 4^B$  of the sum  
 440  $a + c + t + c'$  is preserved along  $\pi$  finally yields the appropriate value for  $c$ .

441 Note that the run  $\pi$  exists if and only if  $4^{B-\mathbf{F}_1(B)}$  divides  $A$ , otherwise Claim 7.2 implies  
 442 that the invariant is broken before the end of the run. This proves that  $\mathcal{P}_1$  is an  $\mathbf{F}_1$ -amplifier.  
 443 The fact that  $\mathcal{P}_1$  is a *strong*  $\mathbf{F}_1$ -amplifier then directly follows from Claims 7.1 and 7.2.

### 444 5.2 Construction of the $\widetilde{F}$ -amplifier $\widetilde{\mathcal{P}}$ from an $F$ -amplifier $\mathcal{P}$

445 Let  $(\mathcal{P}, X, (a, b, c), (a, b', c'), t, Z)$  be a strong  $F$ -amplifier for some function  $F : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ .  
 446 We construct a strong  $\widetilde{F}$ -amplifier  $\widetilde{\mathcal{P}}$  out of  $\mathcal{P}$ . The program  $\widetilde{\mathcal{P}}$  uses the counters of  $\mathcal{P}$  plus  
 447 two fresh input counters  $b''$  and  $c''$ , and it shares the output counters of  $\mathcal{P}$ :

**Program  $\widetilde{\mathcal{P}}$ :**

```

1: loop a  $\rightarrow$  t
2: loop t  $\rightarrow$  a   c'' -= 3   c += 3
3: b'' -= 1   b += 1
4: loop
448 5:   loop a  $\rightarrow$  t
6:   loop t  $\rightarrow$  a   c'' -= 3   a += 3
7:   loop c  $\rightarrow$  c'   c'' -= 3   c' += 3
8:   loop b  $\rightarrow$  b'
9:    $\mathcal{P}$ 
10:  b'' -= 1
  
```

449 Similarly to  $\mathcal{P}_1$  the program  $\widetilde{\mathcal{P}}$  consists of an *initialisation* step lines 1–3 (differing from  $\mathcal{P}_1$   
 450 only by renaming counters), and an *iteration* step lines 4–10 (differing significantly from  $\mathcal{P}_1$ ).

451 **► Lemma 8.** *For every strong  $F$ -amplifier  $(\mathcal{P}, X, (a, b, c), (a, b', c'), t, Z)$ , the program  $(\widetilde{\mathcal{P}}, X \cup$   
 452  $\{b'', c''\}, (a, b, c), (a, b'', c''), Z \cup \{c''\})$  is a strong  $\widetilde{F}$ -amplifier.*

453 The proof of Lemma 8 can be found in Appendix B. Here, we provide an overview of  
 454 the main intuitions behind it. As an  $\widetilde{F}$ -amplifier,  $\widetilde{\mathcal{P}}$  is expected to map each input  
 455  $\text{TRIPLE}(A, B, a, b'', c'', X \cup \{b'', c''\})$  such that  $4^{\widetilde{F}(B)-F(B)}$  divides  $A$  to the output  $\text{TRIPLE}(A \cdot$   
 456  $4^{F(B)-\widetilde{F}(B)}, \widetilde{F}(B), a, b, c, X \cup \{b'', c''\})$ . The intended behaviour of  $\widetilde{\mathcal{P}}$  is straightforward:  
 457 Since for all  $n \in \mathbb{N}_+$  the value  $\widetilde{F}(n)$  is obtained by applying the function  $F$  to 1 for  $n - 1$   
 458 consecutive times, we expect  $\widetilde{\mathcal{P}}$  to apply the program  $\mathcal{P}$  exactly  $B - 1$  times to transform a  
 459 1-triple into an  $\widetilde{F}(B)$ -triple. To show that  $\widetilde{\mathcal{P}}$  behaves as expected, we study the following  
 460 invariant:

$$461 \quad \text{INVARIANT:} \quad (a + c + t) \cdot 4^{b''} = a + c + t + c'' \quad \text{and} \quad t = 0;$$

$$462 \quad \text{BROKEN INVARIANT:} \quad (a + c + t) \cdot 4^{b''} < a + c + t + c''.$$

464 The starting configuration  $\text{TRIPLE}(A, B, \mathbf{a}, \mathbf{b}'', \mathbf{c}'', \mathbf{X} \cup \{\mathbf{b}'', \mathbf{c}''\})$  satisfies the invariant. The  
 465 program  $\widetilde{\mathcal{P}}$  is designed such that every run  $\pi$  starting from such a configuration then satisfies:  
 466 ■ If along  $\pi$  all the loops are maximally iterated and all the calls to  $\mathcal{P}$  are  $Z$ -zeroing, then  
 467 the invariant holds until the end of  $\pi$ . Moreover,  $\pi$  then matches the expected behaviour  
 468 of  $\widetilde{\mathcal{P}}$  described above. In particular,  $\pi$  will correctly amplify  $B - 1$  times via  $\mathcal{P}$  a triple  
 469  $\text{TRIPLE}(A, 1, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{X})$ , thus ending in  $\text{TRIPLE}(A \cdot 4^{F(B) - \widetilde{F}(B)}, \widetilde{F}(B), \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{X} \cup \{\mathbf{b}'', \mathbf{c}''\})$ .  
 470 ■ However, if  $\pi$  fails to maximally iterate one loop, or does a call to  $\mathcal{P}$  that is not  $Z$ -zeroing,  
 471 then the invariant is irremediably broken, which implies that  $\pi$  is not  $(Z \cup \{\mathbf{c}''\})$ -zeroing.  
 472 This proves that  $\widetilde{\mathcal{P}}$  is a strong  $\widetilde{F}$ -amplifier.

## 473 6 Amplifiers defined by counter programs with a pushdown stack

474 In this section, we implement more efficient  $\mathbf{F}_d$ -amplifiers using a stack:

475 ► **Lemma 6.** *For every  $d \in \mathbb{N}_+$  there exists an  $\mathbf{F}_d$ -amplifier of size  $\mathcal{O}(d)$  that uses a stack*  
 476 *and  $\lfloor \frac{d}{2} \rfloor + 4$  counters out of which  $\lfloor \frac{d}{2} \rfloor$  are end counters.*

477 Our construction is based on the amplifiers from Section 5. The main intuitive idea is to  
 478 ‘delegate’ some counters to the stack. The stack alphabet consists exactly of those counters  
 479 which are delegated, and the value of each delegated counter  $x$  corresponds to the number of  
 480 occurrences of the symbol  $x$  on the stack. Therefore, ‘delegated counters’ can be understood  
 481 as a synonym of ‘stack symbols’ in the sequel. This idea motivates the following definition.  
 482 Let  $\mathbf{S}$  and  $\mathbf{X}$  be two disjoint sets of delegated, respectively non-delegated, counters. We define  
 483 the function

$$484 \quad h_{\mathbf{X}, \mathbf{S}} : \mathbb{N}^{\mathbf{X}} \times \mathbf{S}^* \rightarrow \mathbb{N}^{\mathbf{X} \cup \mathbf{S}}$$

485 that maps a configuration, i.e., a valuation  $v$  of the non-delegated counters of  $\mathbf{X}$  together with  
 486 a stack content  $s \in \mathbf{S}^*$ , to a valuation of all the counters from  $\mathbf{X} \cup \mathbf{S}$ , as follows:  $h_{\mathbf{X}, \mathbf{S}}(v, s) = v'$ ,  
 487 where  $v'(x) = v(x)$  for  $x \in \mathbf{X}$ , and  $v'(x)$  is the number of occurrences of  $x$  in  $s$  for  $x \in \mathbf{S}$ .

488 Using this definition, we establish a notion of *simulation* between programs with or  
 489 without stack. Given an  $F$ -amplifier  $\mathcal{P}$  with set of counters  $\mathbf{X} \cup \mathbf{S}$ , we say that a counter  
 490 program with a stack  $\mathcal{Q}$  *simulates*  $\mathcal{P}$  if it satisfies the two following conditions:

- 491 ■ For every  $A, B \in \mathbb{N}_+$  such that  $A$  is divisible by  $4^{(F(B) - B)}$  there exists a run of  $\mathcal{Q}$   
 492 between two configurations  $x$  and  $y$  satisfying  $h_{\mathbf{X}, \mathbf{S}}(x) = \text{TRIPLE}(A, B, \mathbf{a}, \mathbf{b}', \mathbf{c}', \mathbf{X})$  and  
 493  $h_{\mathbf{X}, \mathbf{S}}(y) = \text{TRIPLE}(A \cdot 4^{(B - F(B))}, F(B), \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{X})$ .
- 494 ■ For every run of  $\mathcal{Q}$  between two configurations  $x$  and  $y$  there exists a run of  $\mathcal{P}$  between  
 495  $h_{\mathbf{X}, \mathbf{S}}(x)$  and  $h_{\mathbf{X}, \mathbf{S}}(y)$ ;

496 We say that such a program with stack  $\mathcal{Q}$  is an  $F$ -amplifier.

497 The rest of this section is devoted to the proof of Lemma 6. We rely on the constructions  
 498 of Section 5, and similarly proceed in two steps. First, we transform the  $\mathbf{F}_1$ -amplifier  $\mathcal{P}_1$  into  
 499 a program with stack  $\mathcal{Q}_1$  that simulates  $\mathcal{P}_1$  with four counters, as it delegates the two other  
 500 counters to the stack (Lemma 9). Next, we adapt the constructions used to lift  $F$ -amplifiers  
 501 into  $\widetilde{F}$ -amplifier. This time, we will have *two* constructions that can be applied alternatively:  
 502 the first introduces one counter and one delegated counter, and the second introduces two  
 503 delegated counters (Lemma 10). Therefore, for every  $d \in \mathbb{N}$ , starting with the program  $\mathcal{Q}_1$   
 504 and applying alternatively our two lifting constructions yields a  $\mathbf{F}_d$ -amplifier with  $\lfloor \frac{d}{2} \rfloor + 4$   
 505 counters (as the other counters are delegated to the stack), which proves Lemma 6.

506 **6.1 Construction of the  $F_1$ -amplifier  $Q_1$** 

507 Consider the following program with 4 counters  $X = \{a, b, c, t\}$  and the stack alphabet  
 508  $S = \{b', c'\}$ , which is obtained from the program  $\mathcal{P}_1$  defined in Section 5 by replacing each  
 509 decrement on  $b'$  and  $c'$  by the corresponding pop operation:

**Program  $Q_1$ :**

1: **loop**  $a \rightarrow t$   
 2: **loop**  $t \rightarrow a$  POP( $c'c'c'$ )  $c += 3$   
 3: POP( $b'$ )  $b += 1$   
 4: **loop**  
 510 5: **loop**  $c \rightarrow t$  POP( $c'$ )  $t += 1$   
 6: **loop**  $a \xrightarrow{4} c$  POP( $c'c'c'c'$ )  $c += 1$   $t += 3$   
 7: **loop**  $c \rightarrow a$  POP( $c'$ )  $t += 1$   
 8: **loop**  $t \rightarrow c$  POP( $c'$ )  $c += 1$   
 9: POP( $b'$ )  $b += 2$

511 **► Lemma 9.** *The program  $Q_1$  simulates the  $F_1$ -amplifier  $(\mathcal{P}_1, X, (a, b, c), (a, b', c'), t, \{c''\})$ .*

512 **Proof.** Let  $h$  denote the function  $h_{\{a,b,c,t\},\{b',c'\}}$  that transforms configurations of  $Q_1$  into  
 513 configurations of  $\mathcal{P}_1$ . The program  $Q_1$  is a constrained version of  $\mathcal{P}_1$ : every line is identical  
 514 with the added restriction that lines 2, 3, 6 and 9 can only be fired if the appropriate symbol  
 515 is at the top of the stack. Therefore, we immediately get the second condition required for  
 516  $Q_1$  to simulate  $\mathcal{P}_1$ : for every run  $\pi$  of  $Q_1$  between two configurations  $x$  and  $y$ , the run  $\pi'$  of  
 517  $\mathcal{P}$  that starts in  $h(x)$  and uses the same lines as  $\pi$  is a valid run of  $\mathcal{P}$  that ends in  $h(y)$ .

518 To conclude, we show that we can transform the  $\{b', c'\}$ -zeroing runs of  $\mathcal{P}_1$  (thus in  
 519 particular the runs that witness the  $F_1$ -amplifier behaviour) into runs of  $Q_1$ . To do so we rely  
 520 on the fact that the counters  $b'$  and  $c'$  are only decreasing along the runs of  $\mathcal{P}_1$ . Formally,  
 521 given a  $\{b', c'\}$ -zeroing run  $\pi$  of  $\mathcal{P}_1$  between two configurations  $x$  and  $y$ , let  $u_\pi \in \{b', c'\}^*$  be  
 522 the word listing, in order, the occurrences of the decrements of  $b'$  and  $c'$  along  $\pi$ . We define  
 523 a configuration  $x'$  of  $Q_1$  as follows: the counters  $a, b, c, t$  match the content they have in the  
 524 starting configuration  $x$  of  $\pi$ , and the stack content is the reverse of the word  $u_\pi$  so that the  
 525 first letter of  $u_\pi$  is at the top of the stack. This definition guarantees that:

- 526 **■** We have  $h(x') = x$ . Indeed, since  $\pi$  is  $\{b', c'\}$ -zeroing, the value of the counters  $b'$   
 527 and  $c'$  in the initial configuration  $x$  is equal to the number of times these counters are  
 528 decremented;
- 529 **■** There exists a run  $\pi'$  of  $Q_1$  that starts from  $x'$  and follows the same lines as  $\pi$ : whenever  
 530 a popping instruction appears the adequate symbol will be at the top of the stack. As the  
 531 lines of  $\mathcal{P}_1$  and  $Q_1$  are analogous, the configuration  $y'$  reached by  $\pi'$  satisfies  $h(y') = y$ .  $\blacktriangleleft$

532 **6.2 Construction of the  $\widetilde{F}$ -amplifiers  $\widetilde{Q}$  and  $\overline{Q}$  from an  $F$ -amplifier  $Q$** 

533 In Section 5, we showed how to lift an  $F$ -amplifier  $\mathcal{P}$  into an  $\widetilde{F}$ -amplifier  $\widetilde{\mathcal{P}}$ . We now show  
 534 two different manners of adapting the construction of  $\widetilde{\mathcal{P}}$  in order to lift a program with  
 535 stack  $Q$  simulating  $\mathcal{P}$  into a program with stack simulating  $\widetilde{\mathcal{P}}$ .

536 **► Lemma 10.** *Let  $Q$  be a program simulating a strong  $F$ -amplifier  $(\mathcal{P}, X, (a, b, c), (a, b', c'), t, Z)$   
 537 without delegating the counters  $a, b, c$  and  $t$ .*

- 538 **■** *If  $Q$  delegates  $b'$  but not  $c'$ , then  $\widetilde{Q}$  simulates  $(\widetilde{\mathcal{P}}, X \cup \{b'', c''\}, (a, b, c), (a, b'', c''), Z \cup \{c''\})$   
 539 while delegating two input counters  $b''$  and  $c''$  in addition to the counters delegated by  $Q$ .*

- 540 ■ If  $\mathcal{Q}$  delegates  $b'$  and  $c'$ , then  $\overline{\mathcal{Q}}$  simulates  $(\widetilde{\mathcal{P}}, X \cup \{b'', c''\}, (a, b, c), (a, b'', c''), Z \cup \{c''\})$   
541 while delegating only one input counter  $b''$  in addition to the counters delegated by  $\mathcal{Q}$ .



## 35:16 New Lower Bounds for Reachability in Vector Addition Systems

542

**Program  $\widetilde{Q}$ :**

```

1: loop a  $\rightarrow$  t
2: loop t  $\rightarrow$  a POP(c''c''c'') c += 3
3: POP(b'') b += 1
4: loop
5:   loop a  $\rightarrow$  t
6:   loop t  $\rightarrow$  a POP(c''c''c'') a += 3
7:   loop c  $\rightarrow$  c' POP(c''c''c'') c' += 3
8:   loop b -= 1 PUSH(b')
9:   Q
10:  POP(b'')
```

**Program  $\overline{Q}$ :**

```

1: loop a  $\rightarrow$  t
2: loop t  $\rightarrow$  a c'' -= 3 c += 3
3: POP(b'') b += 1
4: loop
5:   loop a  $\rightarrow$  t
6:   loop t  $\rightarrow$  a c'' -= 3 a += 3
7:   loop
8:     loop b -= 1 PUSH(b')
9:     c -= 1 c'' -= 3 PUSH(c')
10:    loop b -= 1 PUSH(b')
11:    PUSH(c')
12:    loop b -= 1 PUSH(b')
13:    PUSH(c')
14:    loop b -= 1 PUSH(b')
15:    PUSH(c')
16:    loop b -= 1 PUSH(b')
17:    Q
18:    POP(b'')
```

543 The proof of Lemma 10 can be found in Appendix C. To convey the intuition behind it we  
544 analyse the differences between the two programs. The main difference concerns the counters  
545 delegated to the stack: If  $Q$  delegates only  $b'$ , then the starting configurations for the calls  
546 to  $Q$  are easy to setup as the stack simply contains a sequence of  $b'$ . Therefore  $Q$  can be  
547 lifted via  $\widetilde{Q}$  which delegates both  $b''$  and  $c''$ . However, if  $Q$  delegates both  $b'$  and  $c'$ , then  
548 the starting configurations required for the calls to  $Q$  are more complex: the stack needs to  
549 contain the symbols  $b'$  and  $c'$  in a specific order. This prevents us from delegating both  $b''$   
550 and  $c''$  to the stack, thus we need to lift  $Q$  via  $\overline{Q}$  which delegates only  $b''$ . and keeps  $c''$  as  
551 a standard counter. A second difference between  $\widetilde{Q}$  and  $\overline{Q}$  concerns the loops updating  $b'$   
552 and  $c'$  in the iteration step. To understand what is happening here, let us have a look at  
553 what happens when we replace the push and pop instructions by increments and decrements:

554

<pre> 1: <b>loop</b> c <math>\rightarrow</math> c' c'' -= 3 c' += 3 2: <b>loop</b> b <math>\rightarrow</math> b'</pre>	<pre> 1: <b>loop</b> 2:   <b>loop</b> b <math>\rightarrow</math> b' 3:   c -= 1 c' += 1 c'' -= 3 4:   <b>loop</b> b <math>\rightarrow</math> b' 5:   c' += 1 6:   <b>loop</b> b <math>\rightarrow</math> b' 7:   c' += 1 8:   <b>loop</b> b <math>\rightarrow</math> b' 9:   c' += 1 10:  <b>loop</b> b <math>\rightarrow</math> b'</pre>
--	---

555 While these two sequences of instructions are different, we can remark that their global effect  
556 is identical, in the sense that every counter update realisable by the left one is also realisable  
557 by the right one, and reciprocally. However, if  $b'$  and  $c'$  are delegated to the stack then  
558 the sequence of instruction on the right is more powerful, as it performs the same number  
559 of increments of  $b'$  and  $c'$ , but *in any order*, which allows to create many different stack  
560 contents. This is required so that  $\overline{Q}$  can construct the stack contents needed to call  $Q$ .

561 — **References** —

- 562 1 Mohamed Faouzi Atig and Pierre Ganty. Approximating Petri Net Reachability Along  
563 Context-free Traces. In *Proceedings of FSTTCS 2011*, volume 13 of *LIPICs*, pages 152–163,  
564 2011.
- 565 2 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reach-  
566 ability in Two-Dimensional Vector Addition Systems with States Is PSPACE-Complete. In  
567 *Proceedings of LICS 2015*, pages 32–43, 2015.
- 568 3 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki.  
569 The reachability problem for Petri nets is not elementary. In *Proceedings of STOC 2019*, pages  
570 24–33. ACM, 2019.
- 571 4 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki.  
572 The Reachability Problem for Petri Nets Is Not Elementary. *Journal of the ACM*, 68(1):7:1–  
573 7:28, 2021.
- 574 5 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in Vector Addition Systems is  
575 Ackermann-complete. In *Proceedings of FOCS 2021*, pages 1229–1240. IEEE, 2021.
- 576 6 Wojciech Czerwinski and Lukasz Orlikowski. Lower Bounds for the Reachability Problem in  
577 Fixed Dimensional VASSes. In *Proceedings of LICS 2022*, pages 40:1–40:12. ACM, 2022.
- 578 7 Matthias Englert, Piotr Hofman, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Juliusz  
579 Straszynski. A lower bound for the coverability problem in acyclic pushdown VAS. *Inf. Process.*  
580 *Lett.*, 167:106079, 2021.
- 581 8 Matthias Englert, Ranko Lazić, and Patrick Totzke. Reachability in Two-Dimensional Unary  
582 Vector Addition Systems with States is NL-Complete. In *Proceedings of LICS 2016*, pages  
583 477–484. ACM, 2016.
- 584 9 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in  
585 succinct and parametric one-counter automata. In *Proceeding of CONCUR 2009*, volume 5710,  
586 pages 369–383. Springer, 2009.
- 587 10 Slawomir Lasota. Improved Ackermannian Lower Bound for the Petri Nets Reachability  
588 Problem. In *Proceedings of STACS 2022*, volume 219 of *LIPICs*, pages 46:1–46:15. Schloss  
589 Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 590 11 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. *CoRR*,  
591 abs/2104.12695, 2021.
- 592 12 Jérôme Leroux. The Reachability Problem for Petri Nets is Not Primitive Recursive. In  
593 *Proceedings of FOCS 2021*, pages 1241–1252. IEEE, 2021.
- 594 13 Jérôme Leroux, M. Praveen, and Grégoire Sutre. Hyper-Ackermannian bounds for pushdown  
595 vector addition systems. In *Proceedings of CSL-LICS 2014*, pages 63:1–63:10. ACM, 2014.
- 596 14 Jérôme Leroux and Sylvain Schmitz. Reachability in Vector Addition Systems is Primitive-  
597 Recursive in Fixed Dimension. In *Proceedings of LICS 2019*, pages 1–13. IEEE, 2019.
- 598 15 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the Coverability Problem for Pushdown  
599 Vector Addition Systems in One Dimension. In *Proceedings of ICALP 2015*, volume 9135 of  
600 *Lecture Notes in Computer Science*, pages 324–336. Springer, 2015.
- 601 16 Richard J. Lipton. The Reachability Problem Requires Exponential Space. Technical report,  
602 Yale University, 1976.
- 603 17 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall Series in  
604 Automatic Computation. Prentice-Hall, 1967.
- 605 18 Sylvain Schmitz. Complexity Hierarchies beyond Elementary. *TOCT*, 8(1):3:1–3:36, 2016.

606 **A Proof of Claims 7.1 and 7.2**

607 ► **Claim 7.1.** *From each configuration where  $b' > 0$ ,  $c = 0$  and the invariant holds, there is*  
 608 *a unique run through the initialisation step that maintains the invariant and preserves the*  
 609 *value of  $a$ . All the other runs starting from this configuration, as well as the runs starting*  
 610 *with a broken invariant, end with a broken invariant.*

611 ► **Claim 7.2.** *From each configuration where  $b' > 0$ ,  $a$  is divisible by 4 and the invariant*  
 612 *holds, there is a unique run through the iteration step that maintains the invariant and*  
 613 *divides the value of  $a$  by 4. All the other runs starting from this configuration, as well as*  
 614 *those starting with a broken invariant or a holding invariant with a value of  $a$  not divisible*  
 615 *by 4, end with a broken invariant.*

616 Let us recall the program and the invariant mentioned in these two statements:

**Program  $\mathcal{P}_1$ :**

1: **loop**  $a \rightarrow t$   
 2: **loop**  $t \rightarrow a \quad c' -= 3 \quad c += 3$   
 3:  $b' -= 1 \quad b += 1$   
 617 4: **loop**  
 5: **loop**  $c \rightarrow t \quad c' -= 1 \quad t += 1$   
 6: **loop**  $a \xrightarrow{4} c \quad c' -= 4 \quad c += 1 \quad t += 3$   
 7: **loop**  $c \rightarrow a \quad c' -= 1 \quad t += 1$   
 8: **loop**  $t \rightarrow c \quad c' -= 1 \quad c += 1$   
 9:  $b' -= 1 \quad b += 2$

618 INARIANT:  $(a + c + t) \cdot 4^{b'} = a + c + t + c'$  and  $t = 0$ ;

619 BROKEN INARIANT:  $(a + c + t) \cdot 4^{b'} < a + c + t + c'$ .  
 620

621 The initialisation step and the iteration step both decrement  $b'$  by one and preserve the  
 622 right-hand side  $a + c + t + c'$  of the invariant as every line of  $\mathcal{P}_1$  preserves this sum. Hence,  
 623 to maintain the invariant the sum  $a + c + t$  needs to be quadrupled, and to repair a broken  
 624 invariant the sum  $a + c + t$  needs to be increased by an even larger amount. We show that in  
 625 both steps the latter is impossible, and the former only happens if all loops are maximally  
 626 iterated, which implies the modification of the counter  $a$  required by the statements.

627 **Proof of Claim 7.1.** The value of  $a + c + t$  is at most increased by  $3 \cdot (a + t)$  along the  
 628 initialisation part, as line 1 preserves this sum and moves the content of  $a$  to  $t$ , then line 2  
 629 increases this sum by at most 3 times the value of  $t$ . Therefore  $a + c + t$  is at most quadrupled,  
 630 which implies that the initialisation step cannot repair a broken invariant. Moreover, to  
 631 maintain a holding invariant the program  $\mathcal{P}_1$  needs to quadruple this sum. This happens  
 632 if and only if initially  $b' > 0$ ,  $c = 0$  and  $c' \geq 3 \cdot (a + t)$  (this last condition is implied by  
 633 the invariant); and if then both loops are maximally iterated. Finally, remark that upon  
 634 maximal iteration of the loops  $a$  and  $t$  keep their initial values, as required. ◀

635 **Proof of Claim 7.2.** We divide our analysis of the iteration step in two parts:

636 **Lines 5–6** The loop on line 5, respectively line 6, increases  $a + c + t$  by at most the value of  
 637  $c$ , respectively  $a$ . Therefore the value of  $a + c + t$  is at most doubled, which occurs only if  
 638 initially  $t = 0$  and then both loops are maximally iterated, resulting in  $a = 0$ .

639 **Lines 7–8** The loop on line 7, respectively 8, increases  $a + c + t$  by at most the value of  $c$ ,  
 640 respectively  $t$ . Therefore the value of  $a + c + t$  is at most doubled, which occurs only if  
 641  $a = 0$  upon reaching line 7 and then both loops are maximally iterated, resulting in  $t = 0$ .  
 642 Combining the two parts, we get that the value of  $a + c + t$  is at most quadrupled by the  
 643 iteration step. This directly implies that it is not possible to repair a broken invariant.  
 644 Moreover, to maintain a holding invariant this sum needs to be quadrupled. This happens if  
 645 and only if at the start of the iteration step  $b' > 0$ ,  $a$  is divisible by 4,  $t = 0$  and  $c' \geq 3 \cdot (a + c)$   
 646 (note that the last two conditions are implied by the invariant); and if then the four loops  
 647 are maximally iterated.<sup>2</sup> To conclude, remark that maximally iterating lines 6–7 results in  
 648 dividing the value of  $a$  by 4: line 6 transfers one fourth of the value of  $a$  to  $c$ , which is then  
 649 transferred back to  $a$  by line 7. ◀

## 650 **B** Proof of Lemma 5

651 ▶ **Lemma 8.** *For every strong  $F$ -amplifier  $(\mathcal{P}, \mathcal{X}, (a, b, c), (a, b', c'), t, Z)$ , the program  $(\widetilde{\mathcal{P}}, \mathcal{X} \cup$   
 652  $\{b'', c''\}, (a, b, c), (a, b'', c''), Z \cup \{c''\})$  is a strong  $\widetilde{F}$ -amplifier.*

653 The proof is structured as follows: We begin with a technical lemma implying that  $\widetilde{\mathcal{P}}$  satisfies  
 654 the two invariants required to be a *strong* amplifier (Claim 10.1). Then, to show that  $\widetilde{\mathcal{P}}$  is an  
 655  $\widetilde{\mathcal{F}}$ -amplifier, we formalise the expected behaviour of the runs of  $\widetilde{\mathcal{P}}$  (Equations (3)–(7)), we  
 656 show that the runs that fit this expected behaviour  $(Z \cup \{c''\})$ -compute  $\widetilde{\mathcal{F}}$  (Claim 10.2), and  
 657 that the runs that do not fit this expected behaviour are not  $(Z \cup \{c''\})$ -zeroing (Claim 10.3).

658 **Invariants of  $\widetilde{\mathcal{P}}$ .** Before delving into the intricate functioning of  $\widetilde{\mathcal{P}}$ , we show two invariants  
 659 that hold for every run. On top of being prerequisites for  $\widetilde{\mathcal{P}}$  to qualify as a *strong* amplifier,  
 660 these invariants offer valuable assistance in proving the next results of this section.

661 ▶ **Claim 10.1.** *The initialisation step and the iteration step of  $\widetilde{\mathcal{P}}$  both preserve the value of*  
 662  $a + c + t + \Sigma Z + c''$  *and either preserve or decrease the value of  $(a + c + t + \Sigma Z) \cdot 4^{b''}$ .*

663 **Proof.** We start by observing that the sum  $a + c + t + \Sigma Z + c''$  stays constant along every run  
 664 of  $\widetilde{\mathcal{P}}$ : every command line preserves it, including line 9 since  $\mathcal{P}$  is a strong  $F$ -amplifier. We  
 665 now study the effect of the initialisation and iteration steps on the value of  $(a + c + t + \Sigma Z) \cdot 4^{b''}$ .

666 **Initialisation:** The sum  $a + c + t + \Sigma Z$  is preserved in lines 1 and 3 and is increased in line 2  
 667 by at most three times the value of  $t$ , thus it is at most quadrupled by the initialisation  
 668 step. Since  $b''$  is decremented by 1 during the initialisation step, this proves that the  
 669 value of  $(a + c + t + \Sigma Z) \cdot 4^{b''}$  is either preserved or decreased.

670 **Iteration:** The sum  $a + c + t + \Sigma Z$  is increased in lines 5–6 by at most three times the value of  
 671  $a + t$ ; it is increased in line 7 by at most three times the value of  $c$ ; and it is then preserved  
 672 in lines 8, 9 and 10 (since  $\mathcal{P}$  is a strong amplifier). Hence the value of  $a + c + t + \Sigma Z$  is at  
 673 most quadrupled by each occurrence of the iteration step. Since  $b''$  is decremented by 1,  
 674 this shows that the value of  $(a + c + t + \Sigma Z) \cdot 4^{b''}$  is either preserved or decreased. ◀

<sup>2</sup> The fact that  $a$  is divisible by 4 is what allows line 6 to be maximally iterated: if it is not the case, the run would eventually get stuck with a content of  $a$  smaller than four but greater than zero.

675 **Expected behaviour of  $\widetilde{\mathcal{P}}$ .** The intended behaviour of  $\widetilde{\mathcal{P}}$  is straightforward. We start  
 676 with a  $B$ -triple over the counters  $\mathbf{a}, \mathbf{b}'', \mathbf{c}''$ . The initialisation step establishes a 1-triple over  
 677  $\mathbf{a}, \mathbf{b}, \mathbf{c}$ . Next, in the iteration step, this 1-triple is first moved to  $\mathbf{a}, \mathbf{b}', \mathbf{c}'$ , and then  $\mathcal{P}$  is invoked  
 678 to transform it into a  $F(1)$ -triple over  $\mathbf{a}, \mathbf{b}, \mathbf{c}$ . By repeating the iteration step  $B - 2$  more  
 679 times, we obtain a  $F^{B-1}(1) = \widetilde{\mathcal{F}}(B)$ -triple over  $\mathbf{a}, \mathbf{b}, \mathbf{c}$ , as expected from a  $\widetilde{\mathcal{F}}$ -amplifier. We  
 680 now formalise this expected behaviour as a set of equations.

681 Let  $\pi$  be a run of  $\widetilde{\mathcal{P}}$  that visits the iteration step  $n$  times. Let  $w_0(\pi)$  denote the valuation  
 682 of the counter set  $X \cup \{\mathbf{b}'', \mathbf{c}''\}$  at the start of  $\pi$ , and  $x_n(\pi)$  denote the valuation of the counter  
 683 set  $X$  at the end of  $\pi$ . Moreover, for every  $i = 0, 1, \dots, n - 1$ , we use  $x_i(\pi)$  and  $y_i(\pi)$  to denote  
 684 the valuation of  $X$  at the start of the  $(i + 1)$ th iteration step of  $\pi$  and at the start of the  
 685  $(i + 1)$ th call to the program  $\mathcal{P}$ , respectively. This notation allows us to formally express the  
 686 expected behaviour described earlier:

$$687 \quad w_0(\pi) = \text{TRIPLE}(A, B, \mathbf{a}, \mathbf{b}'', \mathbf{c}'', X \cup \{\mathbf{b}'', \mathbf{c}''\}) \quad (3)$$

$$688 \quad x_0(\pi) = \text{TRIPLE}(A, 1, \mathbf{a}, \mathbf{b}, \mathbf{c}, X) \quad (4)$$

$$689 \quad x_i(\pi) = \text{TRIPLE}(A \cdot 4^{i+1-F^i(1)}, F^i(1), \mathbf{a}, \mathbf{b}, \mathbf{c}, X) \quad (5)$$

$$690 \quad y_i(\pi) = \text{TRIPLE}(A \cdot 4^{i+2-F^i(1)}, F^i(1), \mathbf{a}, \mathbf{b}', \mathbf{c}', X) \quad (6)$$

$$691 \quad x_{B-1}(\pi) = \text{TRIPLE}(A \cdot 4^{B-\widetilde{F}(B)}, \widetilde{F}(B), \mathbf{a}, \mathbf{b}, \mathbf{c}, X) \quad (7)$$

693 The individual counter values corresponding to these equations are listed in Figure 1.

694 We split the set of runs of  $\widetilde{\mathcal{P}}$  in two parts: the *good* runs, for which we show that  
 695 Equations (3)–(7) hold, and the *bad* runs, that we prove to be non  $(Z \cup \{\mathbf{c}''\})$ -zeroing.  
 696 Formally, we say that a run of  $\widetilde{\mathcal{P}}$  is *good* if it goes through  $B - 1$  iteration steps; if all  
 697 the loops visited along it are maximally iterated; and if all its calls to the program  $\mathcal{P}$  are  
 698  $Z$ -zeroing. By opposition, we describe as *bad* the runs that fail to satisfy at least one of these  
 699 conditions.

700 **Good runs.** We prove that the good runs of  $\widetilde{\mathcal{P}}$  compute the function  $\widetilde{F}$ :

701 **► Claim 10.2.** *Let  $A, B \in \mathbb{N}_+$  be two positive integers. Every good run of  $\widetilde{\mathcal{P}}$  starting in*  
 702  *$\text{TRIPLE}(A, B, \mathbf{a}, \mathbf{b}'', \mathbf{c}'', X \cup \{\mathbf{b}'', \mathbf{c}''\})$  satisfies Equations (3)–(7), thus in particular it ends in*  
 703  *$\text{TRIPLE}(A \cdot 4^{B-\widetilde{F}(B)}, \widetilde{F}(B), \mathbf{a}, \mathbf{b}, \mathbf{c}, X \cup \{\mathbf{b}'', \mathbf{c}''\})$ . Moreover, there exists such a run if and*  
 704 *only if  $4^{\widetilde{F}(B)-B}$  divides  $A$ .*

705 **Proof.** Let  $\pi$  be a good run of  $\widetilde{\mathcal{P}}$  starting in  $\text{TRIPLE}(A, B, \mathbf{a}, \mathbf{b}'', \mathbf{c}'', X)$ . We immediately get  
 706 that Equation (3) is satisfied. To show that  $\pi$  satisfies Equations (4)–(7), we prove via three  
 707 inductive steps that Figure 1 is an accurate depiction of the valuations  $x_i(\pi)$  and  $y_i(\pi)$  for  
 708 every  $i \in \{0, 2, \dots, B - 1\}$ :

	<b>a</b>	<b>b</b>	<b>b'</b>	<b>c</b>	<b>c'</b>
$w_0(\pi) :$	$A$	$0$	$0$	$0$	$0$
$x_0(\pi) :$	$A$	$1$	$0$	$A \cdot 2 - \mathbf{a}$	$0$
$x_i(\pi) :$	$A \cdot 4^{i+1-F^i(1)}$	$F^i(1)$	$0$	$A \cdot 4^{i+1} - \mathbf{a}$	$0$
$y_i(\pi) :$	$A \cdot 4^{i+2-F^i(1)}$	$0$	$F^i(1)$	$0$	$A \cdot 4^{i+2} - \mathbf{a}$
$x_{B-1}(\pi) :$	$A \cdot 4^{B-\widetilde{F}(B)}$	$\widetilde{F}(B)$	$0$	$A \cdot 4^B - \mathbf{a}$	$0$

■ **Figure 1** Individual counter values corresponding to the expressions in Equations (3)–(7).

709 1. First, starting from a valuation satisfying  $c'' \geq a$ , the effect of the initialisation step with  
710 maximal iteration of the flat loops is equivalent to the following sequence of assignments:

$$711 \quad b'' := b'' - 1, \quad b := b + 1, \quad c'' := c'' - a, \quad c := a.$$

712 This maps the first row of Figure 1 to its second row, thus Equation (4) holds.

713 2. Next, starting from a valuation satisfying  $c'' \geq a$ , the effect of lines 5–8 of  $\widetilde{\mathcal{P}}$  with  
714 maximal iteration of the flat loops is equivalent to the following sequence of assignments:

$$715 \quad a := 2 \cdot a, \quad b' := b, \quad b := 0, \quad c'' := c'' - (a + c), \quad c' := c' + 2c, \quad c := 0.$$

716 This maps the third row of Figure 1 to its fourth row, thus whenever Equation (5) holds  
717 for some  $0 \in \{1, 2, \dots, B - 2\}$ , so does Equation (6).

718 3. Finally, as the program  $\mathcal{P}$  is a strong  $F$ -amplifier, for all  $i \in \{0, 1, \dots, B - 2\}$  it  $Z$ -computes  
719  $\text{TRIPLE}(A \cdot 4^{i+2-F^{i+1}(1)}, F^{i+1}(1), a, b, c, X)$  from  $\text{TRIPLE}(A \cdot 4^{i+2-F^i(1)}, F^i(1), a, b', c', X)$ .  
720 Therefore, as every call to  $\mathcal{P}$  along  $\pi$  is  $Z$ -zeroing since  $\pi$  a good run, we get that if  
721 Equation (6) holds for some  $i \in \{0, 1, \dots, B - 2\}$  then Equation (5) holds for  $i + 1$ .

722 To show that the run  $\pi$  ends in  $\text{TRIPLE}(A \cdot 4^{B-\widetilde{F}(B)}, \widetilde{F}(B), a, b, c, X \cup \{b'', c''\})$ , we still  
723 need to address the values of counters  $b''$  and  $c''$  (as Equation 7 only specifies the value of the  
724 counter set  $X$ ). We directly get that the value of  $b''$  is 0 at the end of  $\pi$ :  $b''$  starts with value  
725  $B$  and is decremented once in the initialisation step and in each of the  $B - 1$  occurrences of  
726 the iteration step. Moreover, we also get that  $c''$  is 0 at the end of  $\pi$  since Claim 10.1 yields  
727 that the value of  $a + c + t + \Sigma Z + c''$  is constantly equal to  $A \cdot 4^B$  along  $\pi$ .

728 Finally, concerning the existence of the run  $\pi$ , remark that, while the register updates  
729 mentioned in Item 1 and 2 can be applied irrespective of the values of  $A$  and  $B$ , the  $Z$ -zeroing  
730 calls to  $\mathcal{P}$  described in Item 3 can be fulfilled if and only if  $A$  is divisible by a sufficiently  
731 large power of 2. More specifically, the run  $\pi$  described in this proof exists if and only if  
732  $4^{\widetilde{F}(B)-B}$  divides  $A$ .  $\blacktriangleleft$

733 **Bad runs.** We prove that the bad runs of  $\widetilde{\mathcal{P}}$  do not  $(Z \cup \{c''\})$ -compute anything:

734  $\blacktriangleright$  **Claim 10.3.** *Let  $A, B \in \mathbb{N}_+$  be two positive integers. Every bad run of  $\widetilde{\mathcal{P}}$  starting in*  
735  $\text{TRIPLE}(A, B, a, b'', c'', X \cup \{b'', c''\})$  *is not  $(Z \cup \{c''\})$ -zeroing.*

736 **Proof.** Let  $\pi$  be a run of  $\widetilde{\mathcal{P}}$  starting in  $\text{TRIPLE}(A, B, a, b'', c'', X)$ . At the start of  $\pi$  we have:

$$737 \quad a = A, \quad b'' = B, \quad c'' = a \cdot (4^{b''} - 1),$$

738 and all other counters are 0. In particular, this implies  $b = b' = c = c' = t = 0$ , thus

$$739 \quad (a + c + t + \Sigma Z) \cdot 4^{b''} = a + c + t + \Sigma Z + c''. \quad (8)$$

741 We start by observing that Claim 10.1 implies that  $\pi$  is  $Z$ -zeroing if and only if Equation (8)  
742 holds after every step of  $\pi$  and  $b' = 0$  at the end of  $\pi$ : The right-hand of Equation (8) side  
743 is preserved, and the value of the left hand-side never increases, thus if it ever decreases it  
744 remains smaller than the right-hand side until the term of  $\pi$ , which in particular implies that  
745 the value of  $c''$  is not 0.

746 Therefore we can immediately deduce that if  $\pi$  visits the iteration step less than  $B - 1$   
747 times, then  $b' > 0$  at the end of  $\pi$ , thus  $\pi$  is not  $(Z \cup \{c''\})$ -zeroing by Equation (8). For the  
748 rest of this proof, let us suppose that  $\pi$  visits the iteration step  $B - 1$  times. Whenever  $\pi$   
749 goes through the initialisation step or the iteration step, it decrements  $b'$  by 1, while gaining

750 the opportunity to increment the sum  $a + c + t + \Sigma Z$ , that we denote  $\Sigma Z'$  in order to maintain  
 751 Equation (8). As we showed when we studied the good runs, in an ideal scenario  $\Sigma Z'$  is  
 752 quadrupled, which compensates the decrement of  $b'$ , and Equation (8) still holds. We now  
 753 show that the occurrence of a single mistake at any point results in  $\Sigma Z'$  not being quadrupled  
 754 along a step: We suppose that the run  $\pi$  is bad, we list all the possible errors it can commit,  
 755 and show that each one breaks Equation (8):

- 756 ■ If  $\pi$  fails to maximally iterate one of the flat loops at lines 1 or 2 then the sum  $\Sigma Z'$  is  
 757 not quadrupled during the initialisation step: Maximally iterating both loops (that is,  
 758 iterating both of them  $a$  times) increments  $\Sigma Z'$  by  $3 \cdot a$ , which exactly quadruples it since  
 759 initially the other variables occurring in  $S$  have value 0. However, since line 2 increases  
 760  $\Sigma Z'$ , not maximally iterating it results in a smaller value. Moreover, while line 1 has no  
 761 direct effect on  $\Sigma Z'$ , not maximally iterating it reduces the number of times line 2 can be  
 762 iterated, which in turn reduces the value of  $\Sigma Z'$ .
- 763 ■ If  $\pi$  fails to maximally iterate one of the flat loops at lines 5, 6 or 7 then the sum  $\Sigma Z'$  is  
 764 not quadrupled during the corresponding iteration step: Maximally iterating the three  
 765 loops (that is, iterating lines 5 and 6  $a$  times and line 7  $c$  times) increments  $\Sigma Z'$  by  
 766  $3 \cdot (a + c)$ , which exactly quadruples it as long as the other variables occurring in  $\Sigma Z'$  had  
 767 value 0 to start with. However, since lines 6 and 7 increase  $\Sigma Z'$ , not maximally iterating  
 768 them results in a smaller value. Moreover, while line 5 has no direct effect on  $\Sigma Z'$ , not  
 769 maximally iterating it reduces the number of times line 6 can be iterated.
- 770 ■ If  $\pi$  does a non  $Z$ -zeroing call to the program  $\mathcal{P}$ , we differentiate two cases. If this  
 771 happens in the last iteration step we get that  $\pi$  is not  $(Z \cup \{c''\})$ -zeroing as it is not  
 772  $Z$ -zeroing. If this happens in one of the previous iteration steps then the sum  $\Sigma Z'$  is not  
 773 quadrupled in the *next* iterations step: as we just saw the iteration step increases  $S$  by at  
 774 most  $3 \cdot (a + c)$ , which fails to quadruple it if there are nonzero counters in  $Z$ .
- 775 ■ Finally, let us consider the case where the first error committed by  $\pi$  is failing to maximally  
 776 iterate the flat loop at line 8. In this case, we show that the subsequent call to  $\mathcal{P}$  is not  
 777  $Z$ -zeroing, which, as we have just shown, implies that  $\pi$  is not  $(Z \cup c'')$ -zeroing. Since we  
 778 assume that this is the first error committed by  $\pi$ , up to this point,  $\pi$  has behaved as a  
 779 good run. To analyse this situation, let  $\pi'$  be the run that behaves as  $\pi$  up to this point  
 780 but then maximally iterates the flat loop at line 8. By Lemma 10.2, we know that  $\pi'$  enters  
 781 the call to  $\mathcal{P}$  with a counter valuation matching  $\text{TRIPLE}(A \cdot 4^{i+2-F^i(1)}, F^i(1), a, b', c', X)$   
 782 for some  $0 \leq i \leq B - 1$ . In particular, the following equation holds for  $\pi'$ :

$$783 \quad (a + c + t + \Sigma Z) \cdot 4^{b'} = A \cdot 4^{i+2} = a + c + t + \Sigma Z + c'$$

784 However, since  $\pi$  did *not* maximally iterate line 8,  $b'$  will be smaller in  $\pi$  compared to  $\pi'$   
 785 (and  $b$  will be larger - but this has no impact on the following argument since  $b \notin Z$ ).  
 786 Consequently,  $\pi$  will call the program  $\mathcal{P}$  with a counter valuation satisfying:

$$787 \quad (a + c + t + \Sigma Z) \cdot 4^{b'} < a + c + t + \Sigma Z + c'.$$

788 Since  $\mathcal{P}$  is a strong amplifier, this equation still holds at the exit of  $\mathcal{P}$ . In particular, this  
 789 implies that  $c'$  is not 0, thus the call to  $\mathcal{P}$  is not  $Z$ -zeroing. ◀

## 790 **C** Proof of Lemma 10

791 ► **Lemma 10.** *Let  $\mathcal{Q}$  be a program simulating a strong  $F$ -amplifier  $(\mathcal{P}, X, (a, b, c), (a, b', c'), t, Z)$   
 792 without delegating the counters  $a, b, c$  and  $t$ .*

- 793 ■ If  $\mathcal{Q}$  delegates  $b'$  but not  $c'$ , then  $\widetilde{\mathcal{Q}}$  simulates  $(\widetilde{\mathcal{P}}, X \cup \{b'', c''\}, (a, b, c), (a, b'', c''), Z \cup \{c''\})$   
 794 while delegating two input counters  $b''$  and  $c''$  in addition to the counters delegated by  $\mathcal{Q}$ .
- 795 ■ If  $\mathcal{Q}$  delegates  $b'$  and  $c'$ , then  $\overline{\mathcal{Q}}$  simulates  $(\widetilde{\mathcal{P}}, X \cup \{b'', c''\}, (a, b, c), (a, b'', c''), Z \cup \{c''\})$   
 796 while delegating only one input counter  $b''$  in addition to the counters delegated by  $\mathcal{Q}$ .

797 **Proof.** Let  $X$  and  $S$  denote the set of counters of  $\mathcal{Q}$ , respectively its stack alphabet. Let  $\widetilde{h}$   
 798 denote the function  $h_{X, S \cup \{b'', c''\}}$  that transforms configurations of  $\widetilde{\mathcal{Q}}$  into configurations of  
 799  $\widetilde{\mathcal{P}}$ . Similarly, let  $\overline{h}$  denote the function  $h_{X \cup \{c''\}, S \cup \{b''\}}$  that transforms configurations of  
 800  $\overline{\mathcal{Q}}$  into configurations of  $\widetilde{\mathcal{P}}$ . The proof is done in three steps. First, we show that we can  
 801 easily translate the runs of  $\widetilde{\mathcal{Q}}$  and  $\overline{\mathcal{Q}}$  into runs of  $\widetilde{\mathcal{P}}$  with matching source and target. The  
 802 harder part of the proof is to show the reciprocal statement: we consider the good runs of  
 803  $\widetilde{\mathcal{P}}$  described in Lemma 10.2 and we show how to translate them, first into runs of  $\widetilde{\mathcal{Q}}$ , and  
 804 finally into runs of  $\overline{\mathcal{Q}}$ .

805 **Transforming runs of  $\widetilde{\mathcal{Q}}$  and  $\overline{\mathcal{Q}}$  into runs of  $\widetilde{\mathcal{P}}$ .** The program  $\widetilde{\mathcal{Q}}$  is a constrained version  
 806 of  $\widetilde{\mathcal{P}}$ : every line is identical except for the lines with a popping instruction instead of a  
 807 decrement, which are more restrictive since the correct symbol needs to be at the top of  
 808 the stack. As a consequence, for every run  $\widetilde{\pi}$  of  $\widetilde{\mathcal{Q}}$  between two configurations  $x$  and  $y$  we  
 809 immediately get a run  $\pi$  of  $\widetilde{\mathcal{P}}$  between  $\widetilde{h}(x)$  and  $\widetilde{h}(y)$  which uses the same lines as  $\widetilde{\pi}$ .

810 Now given a run  $\overline{\pi}$  of  $\overline{\mathcal{Q}}$  between two configurations  $x$  and  $y$ , translating  $\overline{\pi}$  into a  
 811 run of  $\widetilde{\mathcal{P}}$  is not as direct since the lines 7–16 of  $\overline{\mathcal{Q}}$  are not exactly analogous to the lines 7–8  
 812 of  $\widetilde{\mathcal{P}}$ . However, as we explained in the paper, using some local reshuffling  $\widetilde{\mathcal{P}}$  can reproduce  
 813 any counter update corresponding to the lines 7–16 of  $\overline{\mathcal{Q}}$ . This allows us to transform the  
 814 run  $\overline{\pi}$  into a run of  $\widetilde{\mathcal{P}}$  between  $\overline{h}(x)$  and  $\overline{h}(y)$ .

815 **Transforming runs of  $\widetilde{\mathcal{P}}$  into runs of  $\widetilde{\mathcal{Q}}$ .** Let us suppose that  $\mathcal{Q}$  delegates the counter  $b'$   
 816 but not  $c'$ , and let  $\pi$  be a *good* run of  $\widetilde{\mathcal{P}}$  as described in the proof of Lemma 10.2. We denote  
 817 by  $x$  and  $y$  the starting and ending configuration of  $\pi$ . In order to transfer  $\pi$  to  $\widetilde{\mathcal{Q}}$ , we begin  
 818 by creating an appropriate initial configuration  $x'$  as in the proof of Lemma 9. Formally,  
 819 let  $u_\pi \in \{b', c'\}^*$  be the word listing, in order, the occurrences of the decrements of  $b''$  and  
 820  $c''$  along  $\pi$ . We define the configuration  $x'$  of  $\widetilde{\mathcal{Q}}$  by setting the values of the counters of  $X$   
 821 to the values they have in the starting configuration  $x$  of  $\pi$ , and setting the stack content  
 822 to the reverse of the word  $u_\pi$  (so that the first letter of  $u_\pi$  is at the top of the stack). This  
 823 definition guarantees that  $h(x') = x$ . To conclude, we need to argue that  $\widetilde{\mathcal{Q}}$  can simulate  
 824  $\pi$  starting from  $x'$ . First, remark that the initialisation step is easily simulated since the  
 825 definition of the initial stack content guarantees that the appropriate symbol is at the top of  
 826 the stack whenever needed. We now explain, step by step, how  $\widetilde{\mathcal{Q}}$  simulates the iteration  
 827 steps of  $\pi$ . First, thanks to the definition of the initial stack content the loops on lines 5–7  
 828 can be iterated as in  $\pi$ . Then, we also iterate line 8 as in  $\pi$ . Remark that this disrupts the  
 829 stack by adding a sequence of  $b'$  on top of it. Next comes the call to  $\mathcal{P}$ , and since  $\pi$  is a good  
 830 run we know that this call is *correct*, in the sense that it starts in  $\text{TRIPLE}(A, B, a, b', c', X)$   
 831 (Equation (6)) and ends in  $\text{TRIPLE}(A \cdot 4^{B - \widetilde{F}(B)}, \widetilde{F}(B), a, b, c, X)$  (Equation (5)) for some  
 832  $A, B \in \mathbb{N}_+$ . Therefore in  $\widetilde{\mathcal{Q}}$  we can simulate this call to  $\mathcal{P}$  by a call to  $\mathcal{Q}$ , and since the  
 833 value of  $b'$  is 0 in the ending configuration this implies that the call to  $\mathcal{Q}$  will automatically  
 834 pop all of the  $b'$  that were added on the stack. Therefore we are back with a stack content  
 835 that matches a prefix of our initial stack content, and we can conclude the simulation of the  
 836 iteration step by popping a single  $b''$  from the stack.



837 **Transforming runs of  $\widetilde{\mathcal{P}}$  into runs of  $\overline{\mathcal{Q}}$ .** Let us suppose that  $\mathcal{Q}$  delegates both  $\mathbf{b}'$  and  $\mathbf{c}'$ ,  
 838 and let  $\pi$  be a *good* run of  $\widetilde{\mathcal{P}}$ , as described in the proof of Lemma 10.2. We denote by  $x$  and  
 839  $y$  the starting and ending configuration of  $\pi$ . We show how to construct a run  $\bar{\pi}$  of  $\overline{\mathcal{Q}}$  that  
 840 simulates  $\pi$ . First, remark that we have a single possibility for the starting configuration of  $\bar{\pi}$ :  
 841 In the starting configuration of  $\pi$  only the values of  $\mathbf{a}$ ,  $\mathbf{b}''$  and  $\mathbf{c}''$  are nonzero (Equation (3)),  
 842 and  $\overline{\mathcal{Q}}$  only delegates  $\mathbf{b}''$  among these three counters. Therefore the initial stack content will  
 843 just be a sequence of  $\mathbf{b}''$  of the appropriate length. Then, simulating the initialisation step of  
 844  $\pi$  is easy: one  $\mathbf{b}''$  is popped from the stack and the other counters are updated as in  $\pi$ .

845 To conclude, we show how to simulate the iteration steps visited by  $\pi$ . Let  $\pi_1\pi_2\pi_3$  be  
 846 a subrun of  $\pi$  corresponding to an iteration step of  $\widetilde{\mathcal{P}}$ , where  $\pi_2$  stands for the call to the  
 847 program  $\mathcal{P}$ . As we showed in the proof of Lemma 10.2, every call to  $\mathcal{P}$  along  $\pi$  is *correct*, in  
 848 the sense that it starts in some configuration  $\text{TRIPLE}(A, B, \mathbf{a}, \mathbf{b}', \mathbf{c}', \mathbf{X})$  (Equation (6)) and  
 849 ends in  $\text{TRIPLE}(A \cdot 4^{B-\widetilde{F}(B)}, \widetilde{F}(B), \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{X})$  for some  $A, B \in \mathbb{N}_+$  (Equation (5)). As a  
 850 consequence, since  $\mathcal{Q}$  simulates  $\mathcal{P}$ , there exists a run  $\bar{\pi}_2$  of  $\mathcal{Q}$  that simulates  $\pi_2$ , but this  
 851 run requires a starting stack content corresponding to some specific shuffle  $u_{\pi_2}$  of the word  
 852  $(\mathbf{b}')^B(\mathbf{c}')^A \cdot (4^B - 1)$ . Fortunately, as we explained in the paper, the lines 7–16 of  $\overline{\mathcal{Q}}$  allow to  
 853 push any shuffle of  $\mathbf{b}'$  and  $\mathbf{c}'$  on the stack. In particular, there exists a subrun  $\bar{\pi}_1$  of  $\overline{\mathcal{Q}}$   
 854 that simulates  $\pi_1$  and pushes the word  $u_{\pi_2}$  on the stack. As a consequence,  $\bar{\pi}_1\bar{\pi}_2$  simulates  
 855 truthfully the subrun  $\pi_1\pi_2$  with no impact on the stack:  $\bar{\pi}_1$  pushes  $u_{\pi_2}$ , which is then popped  
 856 by  $\bar{\pi}_2$ . Therefore, we can simulate the iteration step  $\pi_1\pi_2\pi_3$  by starting with  $\bar{\pi}_1\bar{\pi}_2$ , and then  
 857 adding  $\bar{\pi}_3$  which pops a single  $\mathbf{b}$  from the stack to simulate the decrement of  $\mathbf{b}$  occurring in  
 858  $\pi_3$ . ◀