The Reachability Problem for Petri Nets is Not Primitive Recursive*

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

Abstract

We present a way to lift up the tower lowerbound of the reachability problem for Petri nets to match the Ackermannian upperbound closing a long standing open problem. We also prove that the reachability problem in fix dimension is not elementary.

*This research has been supported by ANR programme BraVAS (ANR-17-CE40-0028).

1 Introduction

Vector addition systems with states [5], or equivalently vector addition systems [6], or Petri nets are one of the most popular formal methods for the representation and the analysis of parallel processes [3]. The central algorithmic problem is reachability: whether from a given initial configuration there exists a sequence of valid execution steps that reaches a given final configuration. Many computational problems reduce to this reachability problem in logic, complexity, real-time systems, protocols [15, 4].

A *d-dimensional vector addition system* (*d-VASS*, or just *VASS* when the dimension d is not relevant) is a pair $V = (Q, T)$ where Q is a non empty finite set of elements called *states*, and T is a finite set of triples in $Q \times \mathbb{Z}^d \times Q$ called *transitions*. A *configuration* is a pair $(q, x) \in Q \times \mathbb{N}^d$ also denoted as $q(x)$ in the sequel, and an *action* is a vector in $\mathbb{Z}^d$. The semantics is defined by introducing for each transition t the binary relation $\xrightarrow{t}$ over the configurations defined by $p(x) \xrightarrow{t} q(y)$ if $t = (p, y - x, q)$. We also associate to a word $\sigma = t_1 \dots t_k$ of transitions $t_1, \dots, t_k$ the binary relation $\xrightarrow{\sigma}$ over the configurations defined by $p(x) \xrightarrow{\sigma} q(y)$ if there exists a sequence $c_0, \dots, c_k$ of configurations such that:

$$p(x) = c_0 \xrightarrow{t_1} c_1 \cdots \xrightarrow{t_k} c_k = q(y)$$

The reachability problem takes as input a VASS V and two configurations $c_{\text{in}}, c_{\text{out}}$ and it decides if there exists a word σ of transitions such that $c_{\text{in}} \xrightarrow{\sigma} c_{\text{out}}$. After an incomplete proof by Sacerdote and Tenney [14], decidability of the problem was established by Mayr [11, 12], whose proof was then simplified by Kosaraju [7]. Building on the further refinements made by Lambert in the 1990s [8], in 2015, a first complexity upperbound of the reachability problem was provided [9] more than thirty years after the presentation of the algorithm introduced by Mayr [11, 12, 7, 8]. The upperbound given in that paper is cubic Ackermannian. This complexity is obtained by analyzing the computation complexity of the Mayr algorithm. By refining this algorithm and by introducing a new ranking function proving the termination of this refinement, an Ackermannian complexity upperbound was obtained in [10]. This paper also showed that the reachability problem in fixed dimension d is primitive recursive by bounding the length of executions thanks to the fast growing function F_{d+4} of the Grzegorczyk hierarchy.

In this paper we provide an Ackermannian complexity lowerbound for the reachability problem for VASS. Moreover, in fixed dimension $4d + 9$, we prove that the reachability problem is hard for the class F_d. This version of the result is a draft one. A new version will soon be available on arXiv.

2 Fast Growing Functions

We introduce the sequence $(F_d)_{d \in \mathbb{N}}$ of functions $F_d : \mathbb{N} \to \mathbb{N}$ defined by $F_0(n) = n + 1$ for every $n \in \mathbb{N}$, and defined by induction for every $d \geq 1$ and $n \in \mathbb{N}$ by $F_d(n) = F_{d-1}^{n+1}(n)$

Example 1. We have $F_1(n) = 2n + 1$ and $F_2(n) = 2^{n+1}(n + 1) - 1$ for every $n \in \mathbb{N}$. The function F_3 is a tower function which is not elementary. The *Ackermann function* F_ω is defined by $F_\omega(n) = F_{n+1}(n)$. This function is not primitive recursive.

We introduce the submonoid $\mathcal{M}_d$ of $(\mathbb{N}^d \to \mathbb{N}^d, \circ)$ generated by the functions $D_{d,p}$ with $p \in \{1, \dots, d - 1\}$ and $L_{d,p}$ with $p \in \{1, \dots, d - 2\}$ partially defined over the vectors $v \in \mathbb{N}^d$ satisfying $v[p] \geq 1$ by the following equalities for every $1 \leq i \leq d$:

$$D_{d,p}(v)[i] = \begin{cases} v[i] - 1 & \text{if } i = p \\ 2v[i] & \text{if } i = p + 1 \\ v[i] & \text{otherwise} \end{cases} \qquad L_{d,p}(v)[i] = \begin{cases} v[i] - 1 & \text{if } i = p \\ v[i]v[i + 1] & \text{if } i = p + 1 \\ 1 & \text{if } i = p + 2 \\ v[i] & \text{otherwise} \end{cases}$$

Functions $D_{d,p}$ are called *doubling functions*, and functions $L_{d,p}$ are called *lifting functions*.

We denote by $\geq$ the partial order over $\mathbb{N}^d$ defined by $v \geq w$ if $v[i] \geq w[i]$ for every $1 \leq i \leq d$. We first prove some technical lemmas. In those lemmas, we implicitly use several times the fact that functions in $\mathcal{M}_d$ are monotone, i.e. if v is in the definition domain of some $f \in \mathcal{M}_d$ then any $w \geq v$ is also in the definition domain of f and $f(w) \geq f(v)$. Given two vectors $v, w \in \mathbb{N}^d$, we write $v \sqsupseteq w$ if there exists $f \in \mathcal{M}_d$ such that $f(v) \geq w$. Notice that $\sqsupseteq$ is transitive since functions in $\mathcal{M}_d$ are monotonic. Moreover, since $\mathcal{M}_d$ contains the identity function, we deduce that $v \geq w$ implies $v \sqsupseteq w$ for every $v, w \in \mathbb{N}^d$.

Lemma 1. *We have* $(3+k, 2, 3+n, \ldots, 3+n) \sqsupseteq (2+k, 3+n, \ldots, 3+n)$ *for every* $d \geq 3$*, and* $n, k \geq 0$.

Proof. Just observe that $L_{d,1}(3+k, 2, 3+n, \ldots, 3+n) = (2+k, 6+2n, 1, 3+n, \ldots, 3+n)$ and $D_{d,2}^{3+n}(2+k, 6+2n, 1, 3+n, \ldots, 3+n) = (2+k, 3+n, 2^{3+n}, 3+n, \ldots, 3+n) \geq (2+k, 3+n, \ldots, 3+n)$. □

Lemma 2. *We have* $(3+n, \ldots, 3+n) \sqsupseteq (2, 3+F_d(n), \ldots, 3+F_d(n))$ *for every* $d \geq 2$ *and every* $n \geq 0$.

Proof. We prove the lemma by induction.

The base case $d = 2$ is immediate since $D_{2,1}^{n+1}(3+n, 3+n) = (2, 2^{n+1}(3+n)) \geq (2, 3+F_2(n))$. Now, let us assume the case $d-1$ proved for some $d \geq 3$. It follows that for every $n \geq 0$ there exists $f \in \mathcal{M}_{d-1}$ such that $f(3+n, \ldots, 3+n) \geq (2, 3+F_{d-1}(n), \ldots, 3+F_{d-1}(n))$. By incrementing all the indexes occurring in a decomposition of f into doubling and lifting functions, we get a function $g \in \mathcal{M}_d$ such that for every $m \geq 0$ we have $g(3+m, 3+n, \ldots, 3+n) \geq (3+m, 2, 3+F_{d-1}(n), \ldots, 3+F_{d-1}(n))$. Lemma 1 shows that $(3+m, 2, 3+F_{d-1}(n), \ldots, 3+F_{d-1}(n)) \sqsupseteq (2+m, 3+F_{d-1}(n), \ldots, 3+F_{d-1}(n))$. We have proved that $(3+m, 3+n, \ldots, 3+n) \sqsupseteq (2+m, 3+F_{d-1}(n), \ldots, 3+F_{d-1}(n))$ for every $n, m \geq 0$. By induction on m, we deduce that $(3+m, 3+n, \ldots, 3+n) \sqsupseteq (2, 3+F_{d-1}^{m+1}(n), \ldots, 3+F_{d-1}^{m+1}(n))$ for every $m, n \geq 0$. In particular with $n = m$, we get $(3+n, \ldots, 3+n) \sqsupseteq (2, 3+F_{d-1}^{n+1}(n), \ldots, 3+F_{d-1}^{n+1}(n))$ for every $n \in \mathbb{N}$. Since $F_{d-1}^{n+1}(n) = F_d(n)$ we have proved the induction. □

We deduce as a corollary the following lemma that will be useful in Section 8. The two previous lemmas 1 and 2 are no longer used in the sequel.

Lemma 3. *We have* $(3+2n, \ldots, 3+2n) \sqsupseteq (1, F_d(n), \ldots, F_d(n))$ *for every* $d \geq 1$ *and for every* $n \in \mathbb{N}$.

Proof. If $d = 1$ the proof is immediate since $3+2n \geq F_1(n)$. So, assume that $d \geq 2$. Lemma 2 shows that $(3+n, \ldots, 3+n) \sqsupseteq (2, 3+F_d(n), \ldots, 3+F_d(n))$ Since $(3+2n, \ldots, 3+2n) \geq (3+n, \ldots, 3+n)$ and $(2, 3+F_d(n), \ldots, 3+F_d(n)) \geq (1, F_d(n), \ldots, F_d(n))$ we are done. □

In the sequel, functions $D_{d,p}$ and $L_{d,p}$ will be implemented by programs $\mathbf{double}_{d,p}$ and $\mathbf{lift}_{d,p}$ introduced in sections 6 and 7 respectively.

3 Programs

We introduce the formal model of *Minsky programs* and its syntactic restriction called the *Petri programs* that have the same expressive power as VASS.

3.1 Basic Commands

Let C be a finite set of elements called *counters*. A *configuration* is a mapping $\rho : \mathsf{C} \to \mathbb{N}$. We extends valuations over various expressions over the counters the expected way. For instance $\rho(\mathsf{x}_1 + \mathsf{x}_2) = \rho(\mathsf{x}_1) + \rho(\mathsf{x}_2)$ and $\rho(\mathsf{x}_1, \mathsf{x}_2) = (\rho(\mathsf{x}_1), \rho(\mathsf{x}_2))$. We also say that a valuation satisfies a constraint the expected way. For instance ρ satisfies x is even if $\rho(\mathsf{x})$ is even, and ρ satisfies $\mathsf{x}_1 = \mathsf{x}_2$ if $\rho(\mathsf{x}_1) = \rho(\mathsf{x}_2)$. The *zero configuration* is the configuration satisfying the constraint $\mathsf{c} = 0$ for every counter c.

Given a counter c, the *increment command* of c, the *decrement command* of c, and the *test command* of c, are respectively denoted as $\mathsf{c} \mathrel{+}= 1$, $\mathsf{c} \mathrel{-}= 1$, and $\mathsf{c} \mathrel{?}= 0$. Those commands are called *basic commands*. We associate to every basic command $\mathbf{cmd}$ a function $[\![\mathbf{cmd}]\!]$ that maps any configuration ρ onto a set of configurations as follows:

- $[\![\mathsf{c} \mathrel{+}= 1]\!](\rho) = \{\beta\}$ where β is defined by $\beta(\mathsf{c}) = \rho(\mathsf{c}) + 1$ and $\beta(\mathsf{x}) = \rho(\mathsf{x})$ if $\mathsf{x} \neq \mathsf{c}$.
- $[\![\mathsf{c} \mathrel{-}= 1]\!](\rho)$ is the empty set if $\rho(\mathsf{c}) = 0$, and it is equal to $\{\beta\}$ otherwise where β is the configuration such that $\beta(\mathsf{c}) = \rho(\mathsf{c}) - 1$ and $\beta(\mathsf{x}) = \rho(\mathsf{x})$ for every counter $\mathsf{x} \neq \mathsf{c}$.
- $[\![\mathsf{c} \mathrel{?}= 0]\!](\rho)$ is the empty set if $\rho(\mathsf{c}) \neq 0$ and it is equal to $\{\rho\}$ otherwise.

We extend the function $[\![\mathbf{cmd}]\!]$ over sets of configurations S by $[\![\mathbf{cmd}]\!](S) = \bigcup_{\rho \in S} [\![\mathbf{cmd}]\!](\rho)$. We also define $[\![\pi]\!](S)$ where $\pi = \mathbf{cmd}_1 \dots \mathbf{cmd}_n$ is a word of basic commands by $[\![\pi]\!](S) = [\![\mathbf{cmd}_n]\!] \circ \dots \circ [\![\mathbf{cmd}_1]\!](S)$.

3.2 Minsky Programs

A *Minsky program* M over a finite set C of counters is a word $\mathbf{cmd}_1 \dots \mathbf{cmd}_n$ where $\mathbf{cmd}_i$ is either a basic command over C, or a *goto L command* with $L \subseteq \{1, \dots, n+1\}$ and denoted as $\mathbf{goto}\ L$. A *state* of M is a pair (ℓ, ρ) where $\ell \in \{1, \dots, n+1\}$ and ρ is a configuration. We define the binary relation $\Rightarrow$ over the states by $(\ell, \rho) \Rightarrow (\ell', \rho')$ if $\ell \neq n+1$ and in that case:

- $\ell' \in L$ and $\rho' = \rho$ if $\mathbf{cmd}_\ell$ is the goto L command.
- $\ell' = \ell + 1$ and $\rho' \in [\![\mathbf{cmd}_\ell]\!]$ if $\mathbf{cmd}_\ell$ is a basic command.

The binary relation $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$. The binary relation $\xrightarrow{M}$ over the configurations is defined by $\alpha \xrightarrow{M} \beta$ if $(1, \alpha) \Rightarrow^* (n+1, \beta)$. In that case we say that starting from α there exists an execution of the Minsky program that leads to the final configuration β.

It will be convenient to execute in series Minsky programs. Notice that if M_1 and M_2 are two Minsky programs, the concatenation $M_1.M_2$ of the programs seen as words of commands is not correct since the goto L commands of M_2 must be re-indexed. Formally, let us define for a Minsky program M and a natural number $n \geq 0$, the word M^{+n} obtained from M by replacing every goto L command by the goto $L + n$ command where $L + n = \{\ell + n \mid \ell \in L\}$. Given two Minsky programs M_1, M_2, we denote by $M_1; M_2$ the Minsky program $M_1.M_2^{+|M_1|}$. Observe that $\alpha \xrightarrow{M_1;M_2} \beta$ if, and only if, there exists a configuration ρ such that $\alpha \xrightarrow{M_1} \rho \xrightarrow{M_2} \beta$. This construction is associative. Given a sequence $M_1, \dots, M_k$ of Minsky programs, the Minsky program $M_1; \dots; M_k$ is denoted as follow:

$$\begin{array}{l} M_1 \\ \vdots \\ M_k \end{array}$$

Given a Minsky program M, we denote the Minsky program $\mathbf{goto}\ \{2, 2 + |M|\}.M^{+1}.\mathbf{goto}\ \{1\}$ in a more readable way as follows:

loop
M

Denoting by M' that Minsky program, notice that $\xrightarrow{M'}$ is the reflexive and transitive closure of $\xrightarrow{M}$.

We also denote by $\mathsf{c} \mathrel{+}= k$ where $k \in \mathbb{N}$ the Minsky program $\mathsf{c} \mathrel{+}= 1; \ldots ; \mathsf{c} \mathrel{+}= 1$ that repeats k times $\mathsf{c} \mathrel{+}= 1$. The Minsky program $\mathsf{c} \mathrel{-}= k$ is defined symmetrically. Similarly, $\mathsf{c}_1, \ldots, \mathsf{c}_n \mathrel{?}= 0$ denotes $\mathsf{c}_1 \mathrel{?}= 0; \ldots ; \mathsf{c}_n \mathrel{?}= 0$.

The *reachability problem* for Minsky programs takes as input a Minsky program M and a final configuration β and it consists in deciding if there exists an execution of the Minsky program starting from the zero configuration that leads to the final configuration β. This problem is undecidable [13] even if the set of counters C is limited to 2 counters. When bounding the possible values of the counters along executions, the reachability problem becomes decidable and high complexity classes can be defined.

More formally, given a bound $B \in \mathbb{N}$, we say that a configuration is *B-bounded* if it satisfies the constraint $\mathsf{c} \leq B$ for every counter c. We also say that a state (ℓ, ρ) is B-bounded if ρ is B-bounded. Given a Minsky program M, we denote by $\Rightarrow_{\leq B}$ the binary relation over the B-bounded states defined by $(\ell, \rho) \Rightarrow_{\leq B} (\ell', \rho')$ if $(\ell, \rho) \Rightarrow (\ell', \rho')$. The reflexive and transitive closure of that relation is denoted as $\Rightarrow^*_{\leq B}$. We introduce the binary relation $\xrightarrow{M}_{\leq B}$ over the configurations defined by $\alpha \xrightarrow{M}_{\leq B} \beta$ if $(1, \alpha) \Rightarrow^*_{\leq B} (n+1, \beta)$ where $n = |M|$.

To prove that the reachability problem for VASS is hard for the class Ackermann, we provide a reduction from the following problem complete for the class Ackermann of all decision problems that are solvable in time or space bounded by $F_\omega(p(n))$ where p is any primitive recursive function [16], with respect to primitive recursive reductions.

Input A Minsky program M of size n.
Return: Denoting by ρ the zero configuration:

- true if $\rho \xrightarrow{M} \rho$.
- false if we do not have $\rho \xrightarrow{M}_{\leq F_\omega(n)} \rho$.

3.3 Petri Programs

A *Petri program* is a Minsky program such that test commands are only performed at the end. Formally, a Petri program is a Minsky program of the form $M; \mathsf{c}_1, \ldots, \mathsf{c}_n \mathrel{?}= 0$ where M is a Minsky program without test commands.

As expected, the *reachability problem* for Petri programs takes as input a Petri program M and a final configuration β and it consists in deciding if there exists an execution of the Petri program starting from the zero configuration that leads to the final configuration β. The proof of the following lemma is immediate but it provides the link between the reachability problem for VASS, and the reachability problem for Petri programs.

Lemma 4. *When numbers are encoded in unary, the reachability problem for VASS and for Petri programs are interreducible in logspace. Moreover the dimension of the VASS and the number of counters of the Petri programs are equal.*

4 The Good, the Bad, and the Conform

Let us fix a natural number $d \geq 1$. We are going to manipulate configurations provided by counters $\mathsf{x}_0, \ldots, \mathsf{x}_d, \mathsf{b}_1, \ldots, \mathsf{b}_d$ and their corresponding auxiliary counters denoted with a

prime. For instance to express **loop at most** v **times** <*body*> where v is one of the counters $\mathsf{x}_0, \ldots, \mathsf{x}_d, \mathsf{b}_1, \ldots, \mathsf{b}_d$, we employ the auxiliary counter v' to which the value of v is transferred and then transferred back. Provided v' is zero at the start, the body is indeed performed at most v times.

loop
  $\mathsf{v} -= 1 \quad \mathsf{v}' += 1$
loop
  $\mathsf{v}' -= 1 \quad \mathsf{v} += 1$
  <*body*>

We denote by X_i the expression $\mathsf{x}_i + \mathsf{x}'_i$ for every $0 \leq i \leq d$, and we denote by B_i the expression $\mathsf{b}_i + \mathsf{b}'_i$ for every $1 \leq i \leq d$. A configuration is said to be *i-good* with $i \in \{1, \ldots, d\}$ if ρ satisfies the constraint $X_{i-1} = B_i X_i$. A configuration is said to be *good* if it is i-good for every $1 \leq i \leq d$. A *null* configuration is a good configuration satisfying $X_0 = 0$. A configuration ρ is said to be *i-bad* if it satisfies the constraint $X_{i-1} > B_i X_i$, and it is j-good for every $1 \leq j < i$. A configuration is said to be *bad* if it is i-bad for some i. A configuration is said to be *conform* if it is good or bad.

There exist configurations that are not conform. However, programs we consider in the sequel produce only conform configurations from conform configurations. Good and bad configurations will be used as follows. Our programs are non deterministic. However, we will prove that starting from a good configuration, if the program behaves the expected way it will produce a good configuration. If not, the program will produce a bad one. Moreover, starting from a bad configuration, programs will only produce bad configurations.

5 Check Program

We introduce a program $\mathbf{check}_d$ that checks if a conform configuration is good. Intuitively, starting from an initial conform configuration, it terminates successfully if, and only if the initial configuration is good. It will be convenient to introduce the program **flush** v

loop
  $\mathsf{v}' -= 1 \quad \mathsf{v} += 1$

and the symmetric program **load** v

loop
  $\mathsf{v} -= 1 \quad \mathsf{v}' += 1$

We introduce the program $\mathbf{dec}_0$ defined as:

flush x_0
$\mathsf{x}_0 -= 1$

and the program $\mathbf{dec}_p$ for $p \in \{1, \ldots, d\}$ defined by induction as the following program:

flush x_p
$\mathsf{x}_p -= 1$
loop at most b_p **times**
  $\mathbf{dec}_{p-1}$

Intuitively, by executing the loops of $\mathbf{dec}_p$ the maximal number of times, we decrement X_p and all the X_j for $j < p$ in such a way the equality $X_{j-1} = B_j X_j$ is maintained. This idea is formalized as follows.

Lemma 5. *Let α be a configuration and $p \in \{0, \ldots, d\}$. If α satisfies $X_p = 0$ the program $\mathbf{dec}_p$ is blocked when starting form α. So, assume that α satisfies $X_p > 0$. In that case, every configuration β reachable from α by program $\mathbf{dec}_p$ satisfies $\beta(\mathsf{x}_i, \mathsf{b}_i, \mathsf{x}'_i, \mathsf{b}'_i) = \alpha(\mathsf{x}_i, \mathsf{b}_i, \mathsf{x}'_i, \mathsf{b}'_i)$ for every $i > p$, and $\beta(B_i) = \alpha(B_i)$ for every $1 \leq i \leq d$. Moreover,*

(i) If the initial configuration α is i-good for every $1 \leq i \leq p$, then there exists an execution of $\mathbf{dec}_p$ that leads to a final configuration β satisfying additionally for every $0 \leq i \leq d$:

$$\beta(X_i) = \alpha\left(\begin{cases} X_i - \prod_{i<j\leq p} B_j & \text{if } i \leq p \\ X_i & \text{otherwise} \end{cases}\right)$$

In particular β is i-good for every $1 \leq i \leq p$.

(ii) If the initial configuration is i-good for every $1 \leq i \leq p$ then every final configuration β that does not satisfy the previous conditions is j-bad for some $1 \leq j \leq p$.

(iii) If the initial configuration is i-bad with $i \neq p+1$ then the final configurations is j-bad for some j such that $j = i$ or $1 \leq j \leq \min\{i, p\}$.

Proof. The lemma is proved by induction on p. Just observe that an execution that leads to a configuration satisfying (i) is obtained by iterating the loops the maximal number of times. □

The program $\mathbf{check}_d$ is defined as follows:

loop
  $\mathbf{dec}_d$
$\mathsf{x}_0, \ldots, \mathsf{x}_d, \mathsf{x}'_1, \ldots, \mathsf{x}'_d$?= 0

We prove the correctness of program $\mathbf{check}_d$ as follows.

Lemma 6. *Program $\mathbf{check}_d$ successfully terminates on an initial conform configuration if, and only if, the initial configuration is good.*

Proof. Let us consider an initial good configuration α. If α satisfies $X_d > 0$, Lemma 5 shows that there exists an execution of $\mathbf{dec}_d$ starting from α that leads to a good configuration β such that $\beta(X_d) = \alpha(X_d) - 1$. It follows that by iterating $\alpha(X_d)$ times the loop, and by executing $\mathbf{dec}_d$ the right way, we obtain a final good configuration β satisfying $X_d = 0$. Since the configuration β is good, we derive that β also satisfies $X_i = 0$ for every $0 \leq i \leq d$. It follows that the program $\mathbf{check}_d$ successfully executes the last line on β.

Now, let us consider an initial bad configuration α. Lemma 5 parts (iii) shows that every execution of $\mathbf{dec}_d$ can only produce bad configurations. It follows that before executing the last line, we have an i-bad configuration β. It follows that β satisfies $X_{i-1} > B_i X_i$ and in particular $X_{i-1} > 0$. Hence the last line cannot be executed and the program is blocked. □

6 Doubling Program

We introduce in this section the *doubling* program $\mathbf{double}_{d,p}$ parameterized by $p \in \{1, \ldots, d-1\}$ that intuitively implements doubling function $D_{d,p}$ introduced in Section 2.

The program $\mathbf{double}_{d,p}$ is defined as follows:

1: **flush** b_p
2: b_p −= 1
3: **load** x_p
4: ⋮

```
5:  load x_d
6:  loop
7:      x'_d -= 2    x_d += 1
8:      loop at most b_d times
9:          ⋱
10:         x'_i -= 2    x_i += 1
11:         loop at most b_i times
12:             ⋱
13:             x'_{p+1} -= 2    x_{p+1} += 1
14:             loop at most b_{p+1} times
15:                 dec_{p-1}    dec_{p-1}
16:                 x'_p -= 2    x_p += 2
17: load b_{p+1}
18: loop
19:     b'_{p+1} -= 1    b_{p+1} += 2
```

We prove the correctness of $\mathbf{double}_{d,p}$ thanks to the following lemma.

Lemma 7. *For every good configuration α satisfying $B_p \geq 1$ and X_d is even, there exists an execution of program $\mathbf{double}_{d,p}$ starting from α that leads to a good configuration β satisfying $\beta(B_1, \ldots, B_d) = D_{d,p}(\alpha(B_1, \ldots, B_d))$ and $\beta(X_d) = \alpha(\frac{X_d}{2})$. Moreover, any other final configuration is null or bad.*

The program is blocked when started from a conform configuration satisfying $B_p = 0$. When the program starts from a bad configuration, it leads to bad configurations. When it starts from a good configuration satisfying X_d is odd, it leads to bad configurations.

Proof. Assume first that the initial configuration α is good and it satisfies the two constraints $B_p \geq 1$ and X_d is even. We consider the execution obtained by iterating loops the maximal number of times and where $\mathbf{dec}_{p-1}$ is executed the expected way. It follows that before the first execution of line 6 we have a configuration ρ such that $\rho(x'_j) = \alpha(X_j)$ and $\rho(x_j) = 0$ for every $p \leq j \leq d$.

Loop at line 6 is executed $\alpha(\frac{X_d}{2})$ times, and inside that loop, the **loop at most** b_i **times** with $p < i \leq d$ is executed exactly $\alpha(B_i)$ times. The last loop is executed $\alpha(B_p)$ times. It follows that the final configuration β satisfies $\beta(B_1, \ldots, B_d) = D_{d,p}(\alpha(B_1, \ldots, B_d))$ and for every $0 \leq i \leq d$, we have:

$$\beta(X_i) = \alpha\left(\begin{cases} \frac{X_i}{2} & \text{if } i > p \\ X_i - X_p \prod_{i<j<p} B_j & \text{if } i < p \\ X_i & \text{if } i = p \end{cases}\right) \tag{1}$$

Let us now consider any other final configuration β reachable starting form the good configuration α and let us prove that β is null or bad. Observe that if one of the program $\mathbf{dec}_{p-1}$ is not executed the expected way (by executing the loops the maximal number of times), at some point we get a i-bad configuration for some $i \leq p-1$, and β is necessarily j-bad for some $j \leq i$. So, we can assume that program $\mathbf{dec}_{p-1}$ is always executed the right way. In particular the final configuration β is i-correct for every $i \leq p-1$.

Observe that line 15 is executed at most $\alpha(X_p)$ times thanks to the control given by line 16. If it is executed strictly less that $\alpha(X_p)$ times then $\beta(X_{p-1}) > \alpha(X_{p-1} - X_p)$. As $\beta(X_p) = \alpha(X_p)$, $\beta(B_p) = \alpha(B_p) - 1$, and α satisfies $X_{p-1} = B_p X_p$, we deduce that β satisfies $X_{p-1} > B_p X_p$ and we have proved that the final configuration β is p-bad. So, we can assume that line 15 is executed exactly $\alpha(X_p)$ times. The only way to do so, is by executing the **loop at most**

b_i **times** exactly $\alpha(B_i)$ times. So, we can assume that all loops, except maybe the last one are executed the maximal number of times. It follows that β satisfies for every $0 \leq i \leq d$ the equality (1). Now, just observe that if the last loop is not executed $\alpha(B_{p+1})$ times, then $\beta(B_{p+1}) < 2\alpha(B_{p+1})$. It follows that if $\beta(X_{p+1}) > 0$ then β satisfies $X_p > B_{p+1}X_{p+1}$ and the configuration β is $(p+1)$-bad. Observe that if $\beta(X_{p+1}) = 0$ then β is a null configuration.

Now assume that α is a good configuration such that X_d is odd. In that case, there is no way to execute line 15 exactly $\alpha(X_p)$ times. In fact, line 6 is executed at most $\alpha(\frac{X_d}{2})$ times, it means a strictly smaller number of times. It follows that line 15 is executed strictly less than $\alpha(\frac{X_d}{2}B_d \dots B_{p+1})$ times (notice that for the strictness we use the fact that α is positive). Since α is good, we derive $\alpha(\frac{X_d}{2}B_d \dots B_{p+1}) = \alpha(\frac{X_p}{2})$. In particular the number of times program $\mathbf{dec}_{p-1}$ is executed is strictly less than $2\alpha(\frac{X_p}{2}) = \alpha(X_p)$. As previously shown, in that case, the final configuration is necessarily p-bad.

Finally, let us consider an initial i-bad configuration ρ. Observe that if $i \leq p-1$ we deduce from Lemma 5 that the final configuration is j-bad for $j \leq p-1$. So we can assume that $i \geq p$. Lemma 5 shows that we can assume that the program $\mathbf{dec}_{p-1}$ is always executed the right way since otherwise the final configuration is j-bad for some $j \leq p-1$. So, we can assume that the program $\mathbf{dec}_{p-1}$ is always executed the right way.

If $i = p$ then α satisfies $X_{p-1} > B_pX_p$. Observe that the number of times program $\mathbf{dec}_{p-1}$ is executed is bounded by $\alpha(X_p)$ thanks to the limitation introduced by line 16. It follows that $\beta(X_{p-1}) \geq \alpha(X_{p-1} - X_p)$. Since $\beta(X_p) = \alpha(X_p)$, and $\beta(B_p) = \alpha(B_p)$ we derive that β satisfies $X_{p-1} > B_pX_p$. Therefore the final configuration is p-bad.

If $i > p$ then α satisfies $X_{i-1} > B_iX_i$. Notice that line 10 is executed at most $\alpha(\frac{X_i}{2})$. Hence, line 15 is executed at most $\alpha(\frac{X_i}{2}B_i \dots B_{p+1})$ which is strictly bounded by $\alpha(\frac{X_{i-1}}{2}B_{i-1} \dots B_{p+1})$. Since α is j-correct for every $j < i$, we deduce that α satisfies $\frac{X_{i-1}}{2}B_{i-1} \dots B_{p+1} = \frac{X_p}{2}$. Hence, line 15 is executed strictly less that $\alpha(X_p)$. As previously shown it follows that β is p-bad. □

7 Lifting Program

We introduce in this section the *lifting* program $\mathbf{lift}_{d,p}$ indexed by a parameter $p \in \{1, \dots, d-2\}$ that intuitively implements lifting function $L_{d,p}$ introduced in Section 2.

The lifting program is using the *multiplication* program $\mathbf{mul}_p$ parameterized by $p \in \{1, \dots, d-2\}$ and defined as follows:

```
load b_{p+1}
loop
    b'_{p+1} -= 1
    loop at most b_{p+2} times
        b_{p+1} += 1
```

Such a program is usually called a weak multiplier. The proof of the following lemma is immediate.

Lemma 8. *For every initial configuration α, every reachable configuration β satisfies $\beta(B_{p+2}) = \alpha(B_{p+2})$ and $\beta(B_{p+1}) \leq \alpha(B_{p+1}B_{p+2})$. Moreover, there exists a reachable one satisfying $\beta(B_{p+1}) = \alpha(B_{p+1}B_{p+2})$.*

Now, we introduce the following program $\mathbf{lift}_{d,p}$.

```
1: flush b_p
2: b_p -= 1
3: load x_p
```

4: **flush** b_{p+1}
5: b_{p+1} $-=$ 1
6: **loop at most** b_{p+1} **times**
7: **loop at most** x_{p+1} **times**
8: $\mathbf{dec}_{p-1}$ x'_p $-=$ 1 x_p $+=$ 1
9: b_{p+1} $+=$ 1
10: $\mathbf{mul}_p$
11: **flush** b_{p+2}
12: b_{p+2} $-=$ 1
13: **loop**
14: **loop at most** x_{p+2} **times**
15: x_{p+1} $-=$ 1 $\mathbf{dec}_{p-1}$ x'_p $-=$ 1 x_p $+=$ 1
16: b_{p+2} $-=$ 1
17: b_{p+2} $+=$ 1
18: **load** x_{p+1}
19: **loop at most** x_{p+2} **times**
20: x'_{p+1} $-=$ 1 x_{p+1} $+=$ 1 $\mathbf{dec}_{p-1}$ x'_p $-=$ 1 x_p $+=$ 1

Lemma 9. *For every good configuration α satisfying $B_p \geq 1$, there exists an execution of* $\boldsymbol{lift}_{d,p}$ *starting from α that leads to a good configuration β satisfying $\beta(B_1, \ldots, B_d) = L_{d,p}(\alpha(B_1, \ldots, B_d))$ and $\beta(X_d) = \alpha(X_d)$. Moreover, any other final configuration produced by the program is null or bad.*

The program is blocked when started from a conform configuration satisfying $B_p = 0$. When the program starts from a bad configuration, it leads only to bad configurations.

Proof. Let us first consider an initial good configuration α satisfying $B_p \geq 1$. We consider an execution such that program $\mathbf{dec}_{p-1}$ is always executed the right way (see Lemma 5) and such that loops are executed the maximal number of times. We are going to explain why we get a good configuration satisfying the lemma. In fact, line 8 is executed $\alpha((B_{p+1}-1)X_{p+1}) = \alpha(X_p - X_{p+1})$, line 15 is executed $\alpha((B_{p+2}-1)X_{p+2}) = \alpha(X_{p+1} - X_{p+2})$. Let ρ be the configuration we obtain before executing line 17. It follows that $\rho(X_{p+1}) = \alpha(X_{p+1} - (X_{p+1} - X_{p+2})) = \alpha(X_{p+2})$. Hence, the line 20 is executed $\alpha(X_{p+2})$ times. It follows that the program $\mathbf{dec}_{p-1}$ is executed $\alpha((X_p - X_{p+1}) + (X_{p+1} - X_{p+2}) + (X_{p+2})) = \alpha(X_p)$ and we get a conform configuration satisfying the lemma since β satisfies $\beta(B_1, \ldots, B_d) = L_{d,p}(\alpha(B_1, \ldots, B_d))$, and for every $1 \leq i \leq d$:

$$\beta(X_i) = \alpha\left(\begin{cases} X_i - X_p \prod_{i<j<p} B_j & \text{if } i < p \\ X_{i+1} & \text{if } i = p+1 \\ X_i & \text{otherwise} \end{cases}\right) \tag{2}$$

Now, let us consider any other final configuration β that we can obtain after executing the program starting from α. If program $\mathbf{dec}_{p-1}$ is not executed the right way then Lemma 5 shows that β is j-bad for some $j \leq p-1$. So, we can assume that $\mathbf{dec}_{p-1}$ is always executed the right way. Notice that if one of the loops (excluding $\mathbf{mul}_p$, and the **flush** macros) are not executed the maximal number of times, we deduce from the previous paragraph that $\mathbf{dec}_{p-1}$ is executed strictly less than $\alpha((X_p - X_{p+1}) + (X_{p+1} - X_{p+2}) + (X_{p+2})) = \alpha(X_p)$. It follows that $\beta(X_{p-1}) > \alpha(X_{p-1}) - \alpha(X_p)$. Since α satisfies $X_{p-1} = B_p X_p$, and $\beta(B_p) = \alpha(B_p) - 1$, we deduce that β satisfies $X_{p-1} > B_p X_p$. Hence β is p-bad. So, we can assume that all the loops (excluding the ones of $\mathbf{mul}_p$) are executed the maximal number of times. We deduce that β is a configuration satisfying equality (2) for every $i \leq 0 \leq d$. In particular β is j-correct for every $j \neq p+1$, $\beta(X_p) = \alpha(X_p)$ and $\beta(X_{p+1}) = \alpha(X_{p+2})$. As α is good, we get $\alpha(X_p) = \alpha(X_{p+2} B_{p+1} B_{p+2}) = \beta(X_{p+1})\beta(B_{p+1} B_{p+2})$. Observe that if $\beta(X_{p+1}) = 0$ then β is a

null configuration and we are done. So, we can assume that $\beta(X_{p+1}) > 0$. Lemma 8 shows that if program $\mathbf{mul}_p$ is not executed the expected way, then $\beta(B_{p+1}) < \alpha(B_{p+1}B_{p+2})$. We deduce that β is $(p+1)$-bad.

Finally, let us consider an execution from an i-bad configuration α that leads to a final configuration β. Observe that if $i \leq p-1$, from Lemma 5 we deduce that β is bad as well. So we can assume that $i \geq p$. From Lemma 5 we can assume that program $\mathbf{dec}_{p-1}$ is always executed the right way since otherwise β is bad. If $i = p$, then α satisfies $X_{p-1} > B_pX_p$. Notice that the number of times $\mathbf{dec}_{p-1}$ is called is limited by $\alpha(X_p)$. It follows that $\beta(X_{p-1}) \geq \alpha(X_{p-1} - X_p)$. Since $\beta(B_p) = \alpha(B_p) - 1$, $\beta(X_p) = \alpha(X_p)$, we deduce that β satisfies $X_{p-1} > B_pX_p$. Hence β is p-bad. If $i = p+1$, then α satisfies $X_p > B_{p+1}X_{p+1}$. Notice that line 8 is executed at most $\alpha((B_{p+1} - 1)X_{p+1})$ times, and lines 15 and 20 are in total executed at most $\alpha(X_{p+1})$ times. It follows that $\mathbf{dec}_{p-1}$ is executed strictly less than $\alpha(X_p)$ times. It follows that β is p-bad. If $i = p+2$ then α satisfies $X_{p+1} > B_{p+2}X_{p+2}$. Observe that line 8 is executed at most $\alpha((B_{p+1} - 1)X_{p+1}) = \alpha(X_p - X_{p+1})$. Moreover lines 15 and 20 are in total executed less than or equal to $\alpha(B_{p+2}X_{p+2}) < \alpha(X_{p+1})$. It follows that program $\mathbf{dec}_{p-1}$ is executed strictly less than $\alpha(X_p)$. It follows that β is p-bad as shown previously. If $i \geq p+3$ it follows that α satisfies $X_{i-1} > B_iX_i$. Since X_{i-1}, B_i, X_i are constant, notice that β satisfies $X_{i-1} > B_iX_i$. Since α is j-correct for every $j \leq p+2$, the behavior of the program is the same as starting from a good configuration. It follows that if the execution is not performed the right way, the configuration β if bad. It the execution is performed the right way, notice that β is j-correct for every $j < i$. As β satisfies $X_{i-1} > B_iX_i$, it follows that β is i-bad. □

8 Ackermann Program

We are now ready to introduce a program parameterized by $n \in \mathbb{N}$ that is computing for every $\ell \in \mathbb{N}$ good configurations satisfying $B_d \geq F_d(n)$ and $X_d = \ell$.

We introduce the following initialization program $\mathbf{init}_{d,n}$ where $n \geq 0$ is any natural number:

1: $\mathsf{b}_1 \mathrel{+}= 3 + 2n \ldots \mathsf{b}_d \mathrel{+}= 3 + 2n$
2: **loop**
3: $\quad \mathsf{x}_0 \mathrel{+}= (3+2n)^d \ldots \mathsf{x}_d \mathrel{+}= (3+2n)^0$

The proof of the following lemma is trivial by executing the loop ℓ times.

Lemma 10. *Program $\mathbf{init}_{d,n}$ produces only good configurations from the zero configuration. Moreover, for every $\ell \in \mathbb{N}$ it produces a good configuration satisfying $(B_1, \ldots, B_d) = (3 + 2n, \ldots, 3 + 2n)$ and $X_d = \ell$.*

The Ackermannian program $\mathbf{ack}_{d,n}$ is defined as follows:

```
init_{d,n}
loop
    loop
        double_{d,1}
    ⋮
    loop
        double_{d,d-1}
    loop
        lift_{d,1}
    ⋮
    loop
        lift_{d,d-2}
```

Clearly, from Lemmas 7, 9, and 10, we deduce that program $\mathbf{ack}_{d,n}$ produces only conform configurations. Let us now prove that it can produce large good configurations.

Lemma 11. *For every function f in the monoid $\mathcal{M}_d$ and for every $\ell \in \mathbb{N}$, program $\boldsymbol{ack}_{d,n}$ can produce a good configuration satisfying $(B_1, \ldots, B_d) = f(3+2n, \ldots, 3+2n)$ and $X_d = \ell$.*

Proof. We prove the lemma by induction on the number of basic functions that generates the function f. Observe that if f is the identity function, the proof is immediate. So, let us assume the lemma proved for some function f, and let us prove that the lemma for $D_{d,p} \circ f$ for every $1 \leq p \leq d-1$, and for $L_{d,p} \circ f$ for every $1 \leq p \leq d-2$. Let $\ell \in \mathbb{N}$.

Let us first consider the case $D_{d,p} \circ f$. We let $v = f(3+2n, \ldots, 3+2n)$ and $w = f_p(v)$. By induction, there exists an execution of $\mathbf{ack}_{d,n}$ that produces a good configuration α satisfying $(B_1, \ldots, B_d) = v$, and $X_d = 2\ell$. Lemma 7 shows that there exists an execution of $\mathbf{double}_{d,p}$ that produces from α a good configuration β satisfying $(B_1, \ldots, B_d) = w$, and $X_d = \frac{2\ell}{2}$. So we are done in that case.

Finally, let us consider the case $L_{d,p} \circ f$. We let $v = f(3+2n, \ldots, 3+2n)$ and $w = g_p(v)$. By induction, there exists an execution of $\mathbf{ack}_{d,n}$ that produces a good configuration α satisfying $(B_1, \ldots, B_d) = v$, and $X_d = \ell$. Lemma 9 shows that there exists an execution of $\mathbf{lift}_{d,p}$ that produces from α a good configuration β satisfying $(B_1, \ldots, B_d) = w$, and $X_d = \ell$. So we are done also in that case.

We have proved the lemma by induction. □

We deduce the following lemma.

Lemma 12. *For every $\ell \in \mathbb{N}$, program $\boldsymbol{ack}_{d,n}$ can produce a good configuration satisfying $B_d \geq F_d(n)$ and $X_d = \ell$.*

Proof. Lemma 3 shows that there exists a function $f \in \mathcal{M}_d$ such that $f(3+2n, \ldots, 3+2n) \geq (1, F_d(n), \ldots, F_d(n))$. Now, just apply Lemma 11 with f. □

9 Ackermannian Weak Amplifier

In [2] we introduced the notion of B-amplifier defined as a Petri program that produces when it successfully terminates configurations satisfying $\mathsf{b} = B$ and $\mathsf{y} = \mathsf{bx}$ for some counters $\mathsf{y}, \mathsf{b}, \mathsf{x}$. Such an amplifier is then used for simulating Minsky programs with counters bounded by B. In this section, we introduce a similar notion called weak Amplifiers.

A *weak amplifier* is a Petri program that contains the counters $\mathsf{y}, \mathsf{b}, \mathsf{x}$ and such that every configuration reached by a successful execution from the zero configuration satisfies $\mathsf{y} = \mathsf{bx}$. A *weak amplifier* is said to be *B-strong* for some $B \in \mathbb{N}$ if for every $\ell \in \mathbb{N}$, there exists a successful execution that leads to a configuration satisfying $\mathsf{b} \geq B$ and $\mathsf{x} = \ell$.

We now provide a weak amplifier that is $F_d(n)$-strong. In order to obtain that Petri program, we first slightly modify $\mathbf{ack}_{d,n}$ by adding three counters $\mathsf{y}, \mathsf{b}, \mathsf{x}$ that intuitively simulate X_{d-1}, B_d, X_d. More formally, after every command $\mathsf{x}_{d-1} \mathrel{+}= 1$ and $\mathsf{x}'_{d-1} \mathrel{+}= 1$ in $\mathbf{ack}_{d,n}$ we insert the command $\mathsf{y} \mathrel{+}= 1$, and so one. We denote by $\mathbf{ack}'_{d,n}$ the program we obtain that way. We now introduce the program $\mathbf{amp}_{d,n}$ defined as follows:

$\mathbf{ack}'_{d,n}$
$\mathbf{check}_d$

Notice that $\mathbf{ack}'_{d,n}$ is a Petri program since test commands are only at the end of the program.

Lemma 13. *Program* $\boldsymbol{amp}_{d,n}$ *is a* $F_d(n)$*-strong weak amplifier.*

Proof. Let us consider an execution of the program that successful terminates on a configuration β. Let α be the configuration we obtain just after $\mathbf{ack}'_{d,n}$. Lemma 12 shows that α is a conform configuration. If this configuration is not good, Lemma 6 shows that $\mathbf{check}_d$ cannot successfully terminate and we get a contradiction. It follows that α is good. In particular α satisfies $\mathsf{y} = \mathsf{bx}$. Since the program $\mathbf{check}_d$ do not modify counters y, b and x, it follows that β satisfies also that constraint. We have proved that $\mathbf{amp}_{d,n}$ is a weak amplifier.

Now, let us prove that it is $F_d(n)$-strong. Let $\ell \in \mathbb{N}$. Lemma 12 shows that there exists an execution of $\mathbf{ack}_{d,n}$ that leads to a good configuration satisfying $B_d \geq F_d(n)$ and $X_d = \ell$. By simulating that execution in $\mathbf{ack}'_{d,n}$ we get a good configuration α satisfying $\mathsf{b} \geq F_d(n)$ and $\mathsf{x} = \ell$. Since α is good, Lemma 6 shows that there exists a successfully execution of $\mathbf{check}_d$ starting from α that leads to a configuration β. Since the program $\mathbf{check}_d$ do not modify the counters $\mathsf{y}, \mathsf{b}, \mathsf{x}$, it follows that β satisfies also $\mathsf{b} \geq F_d(n)$ and $\mathsf{x} = \ell$. We have proved the lemma. □

10 Simulating Tests

We now provide a Petri program $W \rhd M$ that simulates thanks to a B-strong weak amplifier W, a Minsky program M with counters bounded by a value at least as large as B. The construction is exactly the same as the one given in [2]. Properties satisfied by the construction, starting from a weak amplifier, and not a strong one as in [2] are new.

Under the stated assumptions, we now define a construction of the Petri program $W \rhd M$. The idea is to turn each counter c of M into one through supplementing it by a new counter $\hat{\mathsf{c}}$ and ensuring that the invariant $\mathsf{c} + \mathsf{c}' = R$ for some large natural number R is maintained, so that a test command of c can be replaced by loops that R times increment c and then B times decrement c. Counter b provided by W is employed to initialise each complement counter c', whereas x and y are used to ensure that if y is zero at the end of the run then all the loops in the simulations of the tests iterated R times as required. Concretely, the program $W \rhd M$ is constructed as follows:

(i) By modifying W we can assume that all the counter except $\mathsf{y}, \mathsf{b}, \mathsf{x}$ of W are in a test command at the end of W, i.e $W = W'; \mathsf{x}_1, \ldots, \mathsf{x}_n \mathrel{?=} 0$ where W' is a Petri program without test commands, and the disjoint union of $\{\mathsf{x}_1, \ldots, \mathsf{x}_n\}$ and $\{\mathsf{y}, \mathsf{b}, \mathsf{x}\}$ is the set of counters of W;

(ii) counters are renamed if necessary so that no counter occurs in both W and M;

(iii) letting $\mathsf{c}_1, \ldots, \mathsf{c}_l$ be the counters of M, new counters $\mathsf{c}'_1, \ldots, \mathsf{c}'_l$ are introduced and the following code is inserted at the beginning of M:

```
loop
    c'_1 += 1   ···   c'_l += 1
    b −= 1   y −= 1
x −= 1
```

(we shall show that executions necessarily iterate this loop R times, i.e. until counter b becomes zero);

(iv) every $\mathsf{c}_i \mathrel{+}= 1$ command in M is replaced by two commands

$$\mathsf{c}_i \mathrel{+}= 1 \quad \mathsf{c}'_i \mathrel{-}= 1;$$

(v) every $\mathsf{c}_i \mathrel{-}= 1$ command in M is replaced by two commands

c_i −= 1 $\quad$ c'_i += 1;

(vi) every c_i ?= 0 command in M is replaced by the following code:

```
loop
    c_i += 1    c'_i −= 1
    y −= 1
x −= 1
loop
    c_i −= 1    c_i += 1
    y −= 1
x −= 1
```

(we shall show that complete runs necessarily iterate each of the two loops R times, i.e. they check that c_i equals 0 through checking that c'_i equals R by transferring R from c'_i to c_i and then back);

(vii) the Petri program $W \rhd M$ is defined as $W'; M'; \mathsf{x}_1, \ldots, \mathsf{x}_n\mathsf{y}$?= 0 where M is the program M modified as stated.

?= 0.

We remark that simulating test commands of counters bounded by some R using transfers from and to their complements is a well-known technique that can be found already in Lipton [1]; the novelty here is the cumulative verification of such simulations, through decreasing appropriately the two counters y and x whose ratio is R, and checking that y is zero finally.

10.1 Correctness

Correctness. The next proposition states that the construction of $W \rhd M$ is correct. In one direction, the proof proceeds by observing that $W \rhd M$ can simulate faithfully any execution of M. In the other direction we argue that although some of the loops introduced in steps (vi) may iterate fewer than R times and hence erroneously validate a test, the ways in which counters x and y are set up by W and used in the construction ensure that no such run can continue to a complete one. Informally, as soon as a loop in a simulation of a test iterates fewer than R times, the equality $\mathsf{y} = \mathsf{x} \cdot R$ turns into the strict inequality $\mathsf{y} > \mathsf{x} \cdot R$ which remains for the rest of the run, preventing counter y from reaching zero.

Proposition 14. *Let α be the zero configuration, β be any configuration, and assume that W is B-strong.*

- *If $\alpha \xrightarrow{M}_{\leq B} \beta$ then $\alpha \xrightarrow{W \rhd M} \beta$.*
- *If $\alpha \xrightarrow{W \rhd M} \beta$ then $\alpha \xrightarrow{M} \beta$.*

Proof. The first item is straightforward: from an execution of M with counters bounded by B, let ℓ be the number of test commands performed along the execution, we obtain an execution of $W \rhd M$ with the same final valuation of counters by

- running W to terminate with a configuration satisfying $\mathsf{b} \geq B$, $\mathsf{c} = 2\ell + 1$, $\mathsf{y} = \mathsf{x} \cdot \mathsf{b}$ and all the other counters are equal to 0, where the latter counters will remain untouched for the rest of the run and hence satisfy the requirement to be zero finally (cf. step (vii) of the construction), let R be the valuation of b at that point,
- iterating the loop in step (iii) R times to initialise each complement counter c'_i to R, which also subtracts R and 1 from y and x (respectively) as well as decreases y to 0, and

- in place of every test command in M, iterating both loops in step (vi) R times, which subtracts $2R$ and 2 from y and x (again respectively), eventually decreasing them both to 0.

For the second item, consider an execution of $W \rhd M$ from the zero configuration to a configuration β. We let ρ be the configuration we obtain just after executing W'. Firstly, by step (vii) and the fact that counters $\mathsf{y}_1, \ldots, \mathsf{y}_m$ are not used after executing the part of code from W, we have that ρ satisfies $\mathsf{y} = \mathsf{bx}$. Let us introduce $R = \rho(\mathsf{b})$. Extracting an execution of M with the same final configuration β is easy once we show that, for each simulation of a c_i ?= 0 command by the code in step (vi) of the construction, the values of c_i at the start and at the finish of the code are 0.

After the code in step (iii) we therefore have that $\mathsf{c}_i + \mathsf{c}'_i \leq R$ for all i. We infer that

$$\mathsf{x}_i + \mathsf{x}'_i \leq R \text{ for all } i, \text{ and } \mathsf{y} \geq \mathsf{x} \cdot R$$

is an invariant that is maintained by the rest of the run.

Now, due to step (vii) again, y is zero finally, and so the inequality $\mathsf{y} \geq \mathsf{x} \cdot R$ is finally an equality. Therefore, x is zero finally as well. Arguing backwards through the execution, we conclude that in fact $\mathsf{y} = \mathsf{x} \cdot R$ has been maintained and that, for each simulation of a x_i ?= 0 command, each of the two loops has been iterated exactly R times, and hence the values of c_i at its start and at its finish have been as required. Also, the loop introduced in step (iii) has been iterated R times, and b is zero finally. □

11 Conclusion

By applying Proposition 14 with the Ackermannian weak amplifier, we deduce the following complexity result.

Theorem 15. *The vector addition system with states reachability problem is hard for* Ackermann. *Moreover, in fixed dimension $4d + 9$ the reachability problem is F_d-hard.*

Proof. Just observe that $\mathbf{amp}_{n+1,n}$ is a weak amplifier that is $F_{n+1}(n)$-strong. Since $F_{n+1}(n) = F_\omega(n)$, by applying Proposition 14 we deduce that the reachability problem is hard for Ackermann.

For the fix dimension, just observe that we just need to simulate Minsky machines M with two counters. In total with the weak amplifier $\mathbf{amp}_{d,n} \rhd M$ is using 4 counters for the Minsky machine, $4d + 1$ counters corresponding to $(\mathsf{x}_i)_{0 \leq i \leq d}$, $(\mathsf{b}_i)_{1 \leq i \leq d}$, $(\mathsf{x}'_i)_{1 \leq i \leq d}$, $(\mathsf{b}'_i)_{1 \leq i \leq d}$ (since we do not use x'_0), the three counters $\mathsf{y}, \mathsf{b}, \mathsf{x}$, and the auxiliary counter b' (we do not need auxiliary counters for x and y in the construction). So, in total, we have $4 + 4d + 1 + 3 + 1 = 4d + 9$ counters. □

It follows that the complexity of the reachability problem is closed since additionally, we have the following result.

Theorem 16 ([10])**.** *The vector addition system with states reachability problem is easy for* Ackermann. *Moreover, in fixed dimension d the reachability problem is F_{d+4}-easy.*

Notice that in fixed dimension $4d + 9$ we still have a gap between F_d and F_{4d+13}.

References

[1] E. Cardoza, R. J. Lipton, and A. R. Meyer. Exponential space complete problems for petri nets and commutative semigroups: Preliminary report. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 50–54. ACM, 1976. doi: 10.1145/800113.803630.

[2] W. Czerwiński, S. Lasota, R. Lazić, J. Leroux, and F. Mazowiecki. The reachability problem for petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 24–33. ACM, 2019. doi: 10.1145/3313276.3316369.

[3] J. Esparza and M. Nielsen. Decidability issues for petri nets - a survey. *Bulletin of the European Association for Theoretical Computer Science*, 52:245–262, 1994.

[4] M. H. T. Hack. *Decidability questions for Petri nets.* PhD thesis, MIT, 1975. URL `http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-161.pdf`.

[5] J. E. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoritical Computer Science*, 8:135–159, 1979.

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

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

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

[9] J. Leroux and S. Schmitz. Demystifying reachability in vector addition systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE Computer Society, 2015. doi: 10.1109/LICS.2015.16.

[10] J. Leroux and S. Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi: 10.1109/LICS.2019.8785796.

[11] E. W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 238–246. ACM, 1981. doi: 10.1145/800076.802477.

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

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

[14] G. S. Sacerdote and R. L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 61–76. ACM, 1977. doi: 10.1145/800105.803396.

[15] S. Schmitz. The complexity of reachability in vector addition systems. *SIGLOG News*, 3 (1):4–21, 2016. URL `https://dl.acm.org/citation.cfm?id=2893585`.

[16] Sylvain Schmitz. Complexity hierarchies beyond elementary. *TOCT*, 8(1):3:1–3:36, 2016. URL `http://doi.acm.org/10.1145/2858784`.